KONGSBERG

Bachelor's thesis

Kromium - KONGSBERG Remote Operated Mechanical Image Utilizing Machine

University of
South-Eastern Norway

Faculty of Technology, Natural Sciences and
Maritime Sciences
Campus Kongsberg

This page was intentionally left blank.

Course: TS3000 Bacheloroppgave
Date: June 8, 2024
Title: Kromium

Project group: 3
Group members: Shahin Ostadahmadi
Aditi Deshpande
Oscar Melby
Adrian Elias Haugjord
Henrik Bertelsen

Internal supervisor: Hieu T. Nguyen
External supervisor: Qui-Huu Le-Viet (01.01.2024-17.04.2024)
Merethe Gotaas (18.04.2024-)

Project partner: Kongsberg Maritime

Acknowledgements

We would like to express our gratitude to our external supervisors, Merethe Gotaas and Qui-Huu Le-Viet, for their constant support, reassurance, invaluable guidance, and continuous encouragement throughout this project. Their expertise and insights have been instrumental in shaping our research and ensuring its successful completion.

We also would like to convey our sincere appreciation to our internal supervisor, Hieu T. Nguyen, for his insightful and precise feedback, constructive criticism, and steady belief in our capabilities. His mentorship has been invaluable in keeping us focused, on track and motivated throughout the project's development.

Finally, we thank the University of South-Eastern Norway for providing us with the opportunity to undertake this research and for creating such a stimulating academic environment. We also acknowledge the support of the faculty, who have provided us with the resources and knowledge necessary to succeed.

Abstract

Offshore operations have recently undergone a technological revolution, boosting efficiency, safety, and cost-effectiveness to new heights. One untapped opportunity lies in the adoption of remote operations. This bachelor's thesis explores the integration of virtual reality, artificial intelligence, and robotics to enhance offshore operations. The system uses virtual reality for human-robot interaction and receives visual information from the robot which is analysed through object detection models. It also receives information about the status of the robot. The software architecture is modular, dividing tasks to quicken development and ensure efficient operation.

The successful implementation of the integration of virtual reality, artificial intelligence, and robotics demonstrates the feasibility and potential of using these fields in revolutionizing offshore operations for improved safety, efficiency, and cost-effectiveness.

This page was intentionally left blank.

Contents

Acknowledgements **3**

Abstract **4**

List of Figures **14**

List of Tables **22**

Glossary **23**

Nomenclature **29**

1 Introduction **30**
1.1 About the project . . . 30
1.2 Group members . . . 30
1.3 Problem domain . . . 31
1.3.1 Introduction . . . 32
1.3.2 Virtual Reality . . . 32
1.3.3 Artificial Intelligence . . . 33
1.3.4 Robotics . . . 34
1.3.5 Robotic arms . . . 35
1.3.6 Integration of VR, AI, and Robotics . . . 36
1.4 Project overview . . . 36
1.4.1 The customer . . . 36
1.4.2 Project goal . . . 37
1.5 Requirements . . . 37
1.5.1 Definition of priority order . . . 37
1.5.2 Additional requirements . . . 38
1.5.3 Validation . . . 38
1.5.4 Related work: Object detection . . . 38

2 Project Management & Risk Assessment **39**
2.1 Organizational structure . . . 39
2.1.1 Member responsibilities . . . 39
2.1.2 Team building . . . 40
2.2 Project tools . . . 41
2.2.1 ChatGPT . . . 41
2.3 Website . . . 41
2.4 Hardware used . . . 41
2.4.1 Virtual reality headset . . . 41
2.4.2 Development platform . . . 42
2.5 Software tools . . . 42
2.5.1 Git & GitHub . . . 42
2.5.2 ROS 2 . . . 42
2.5.3 Operating system . . . 43
2.5.4 Docker . . . 43
2.5.5 Unity . . . 44
2.5.6 Programming languages . . . 44
2.5.7 Google Colab . . . 44
2.5.8 Integrated development environment . . . 44

2.5.9 Doxygen . . . 44
2.5.10 Sphinx . . . 45
2.5.11 GitHub Workflows & Pages . . . 45
2.5.12 Python standards . . . 45
2.5.13 GitHub CoPilot . . . 45
2.5.14 Miscellaneous tools . . . 46
2.6 Project development process . . . 46
2.6.1 GitHub workflow . . . 46
2.7 Mechanical development tools . . . 46
2.7.1 Computer-aided design . . . 47
2.7.2 SolidWorks simulation . . . 47
2.7.3 UltiMakrer Cura . . . 47
2.8 Electronics . . . 47
2.8.1 Tools used for electrical development . . . 47
2.9 Project model . . . 47
2.10 Risk Assessment: Rapid Risk Ranking (RRR) . . . 48
2.10.1 Consequence severity . . . 49
2.10.2 Probability levels . . . 49
2.10.3 Risk Matrix . . . 50
2.10.4 Risk levels . . . 50
2.10.5 Hazardous events . . . 50
2.10.6 Risk analysis form . . . 50

3 Design overview **52**
3.1 Project design overview . . . 52
3.2 System functionalities . . . 53
3.2.1 VR application features . . . 53
3.2.2 Operator Information Features . . . 54
3.3 Hardware design . . . 55
3.3.1 Robot specifications . . . 55
3.3.2 Structure of the robot . . . 55
3.3.3 Building the robot . . . 56
3.3.4 Easy accessible power switch . . . 56
3.3.5 Easily accessible electronics . . . 56
3.3.6 How to change the battery . . . 57
3.3.7 Turning on the power . . . 57
3.3.8 Wheels . . . 58
3.4 Electrical design . . . 58
3.4.1 Battery design . . . 58

4 Software Implementation **61**
4.1 Software setup . . . 61
4.1.1 Network configuration . . . 61
4.1.2 VR application environment . . . 61
4.1.3 Robot & AI environment . . . 61
4.2 The VR application . . . 62
4.2.1 Visual overview of the application . . . 63
4.2.2 Communication between VR and the robot . . . 65
4.2.3 Robot visual view . . . 67
4.2.4 VR application and robot modes . . . 70

4.2.5 Controlling the robot car . . . 74
4.2.6 Controlling the robot arm . . . 79
4.2.7 Robot status: information display . . . 82
4.2.8 Emergency control . . . 83
4.3 Robot . . . 85
4.3.1 Using ROS 2 . . . 86
4.3.2 Node architecture . . . 87
4.3.3 Communicating with the Meta Quest 3 . . . 88
4.3.4 Video stream . . . 89
4.3.5 Interacting with the expansion board . . . 89
4.3.6 Handling modes and safety features . . . 90
4.3.7 Changing modes . . . 91
4.3.8 Driving the robot . . . 93
4.3.9 Logging functionality . . . 95
4.3.10 Sending of robot data & status . . . 95
4.4 Robotic arm . . . 97
4.4.1 Arm movement concept . . . 97
4.4.2 Servo and arm limitations . . . 98
4.4.3 End effector placement and arm dimensions . . . 98
4.4.4 Deciding on an arm manipulation method . . . 100
4.4.5 Inverse kinematics . . . 101
4.4.6 Implementation . . . 104
4.4.7 Simulation . . . 109
4.5 Artificial Intelligence: Object Detection . . . 109
4.5.1 Neural networks . . . 110
4.5.2 Loss functions and gradient descent . . . 111
4.5.3 Convolutional Neural Networks . . . 113
4.5.4 How convolution works . . . 113
4.5.5 Architectural components of data transformation in CNNs . . . 114
4.5.6 Putting together a simple CNN . . . 116
4.5.7 Transfer learning . . . 117
4.5.8 CNNs for edge devices . . . 118
4.5.9 Object detector CNNs . . . 121
4.5.10 Loss functions in object detectors . . . 122
4.5.11 Framework for object detection . . . 123
4.5.12 TensorFlow object detection with MediaPipe . . . 123
4.5.13 Preparing the dataset . . . 124
4.5.14 Dataset augmentation . . . 126
4.5.15 Training the model . . . 127
4.5.16 Displaying object information . . . 128
4.6 Additional work . . . 129
4.6.1 Getting depth data . . . 129
4.6.2 Showing depth data in the VR Space . . . 136
4.6.3 Digital twin of the robot arm . . . 137
4.6.4 Detection of bolts using a self-made dataset . . . 143
4.6.5 Automated fastening of bolts . . . 145
4.6.6 Reversing camera . . . 145
4.6.7 Backup logging . . . 146
4.6.8 Integration of voice commands in the VR application . . . 147
4.6.9 Arm safety limits . . . 148

4.6.10 Optimizing the video stream . . . 149
4.6.11 Caching of object detection boxes . . . 150
4.6.12 Displaying latency of the robot . . . 151
4.6.13 Internal logging . . . 151
4.6.14 Database integration with object detection . . . 152

5 Electrical **155**
5.1 Electrical components . . . 155
5.1.1 Yahboom Battery . . . 155
5.1.2 Custom built battery . . . 155
5.1.3 DC motors . . . 156
5.1.4 Servos . . . 156
5.1.5 USB 3.0 HUB expansion board . . . 156
5.1.6 ROS robot expansion board . . . 156

6 Mechanical hardware development **157**
6.1 Given equipment . . . 157
6.2 Robot design requirements . . . 158
6.3 Robot design ideas . . . 158
6.4 Robot iteration one . . . 159
6.4.1 Robot iteration one, assembly . . . 159
6.5 Robot iteration two . . . 160
6.5.1 Iteration two assembly . . . 162
6.6 Robot iteration three . . . 163
6.6.1 Robot iteration three redesigns . . . 163
6.6.2 Robot iteration three assembly . . . 164
6.7 Robot iteration four . . . 164
6.7.1 Robot iteration four redesigns . . . 165
6.7.2 Robot iteration four assembly . . . 165
6.8 Iteration five design . . . 165
6.8.1 Robot iteration five redesigns . . . 166
6.8.2 Robot iteration five assembly . . . 166
6.9 Battery design . . . 167
6.9.1 Battery iteration one . . . 167
6.9.2 Battery iteration two . . . 168
6.9.3 Battery iteration three . . . 169
6.9.4 Battery iteration four . . . 170
6.9.5 Battery iteration five . . . 170
6.9.6 Battery iteration six . . . 171
6.10 Exporting the model for the digital twin . . . 173
6.11 Production . . . 175
6.11.1 Robot car sheet floors . . . 175
6.11.2 Robot car cover . . . 175
6.11.3 Post processing . . . 177
6.11.4 3D printed components . . . 180

7 Testing & Results **181**
7.1 Integration testing . . . 181
7.2 Object detection model performance . . . 181
7.2.1 People detection . . . 181
7.2.2 Bolt detection . . . 184

7.2.3 Analyzing the results . . . 186
7.3 Running inference . . . 186
7.3.1 Pre-processing . . . 187
7.3.2 Detection and post-processing . . . 187
7.4 Production mode . . . 191
7.5 Unit tests . . . 191
7.5.1 Writing an unit test . . . 191
7.5.2 Ignoring messages in wrong mode . . . 192
7.5.3 Battery percentage verification . . . 193
7.6 robot-test-client . . . 194
7.7 Simulations . . . 195
7.7.1 Testing of 520 DC motor brackets . . . 195
7.7.2 Arm camera bracket . . . 195
7.7.3 Wall attachment bracket assembly . . . 196
7.8 Visual physical testing . . . 197
7.8.1 Robot iteration one testing . . . 197
7.8.2 Robot iteration two testing . . . 198
7.8.3 Robot iteration three testing . . . 200
7.8.4 Robot iteration four testing . . . 201
7.8.5 Robot iteration five test . . . 202
7.9 Battery testing . . . 203
7.9.1 Test of battery cells . . . 203
7.9.2 Test of battery iterations . . . 205
7.10 User survey testing . . . 208

8 Future Work 209
8.1 Software . . . 209
8.1.1 3D mapping using depth information . . . 209
8.1.2 Autonomous movement, object avoidance and operation . . . 210
8.1.3 Avoid arm collisions with MoveIt . . . 210
8.1.4 Simultaneous driving and arm manipulation . . . 210
8.1.5 Increase performance with Ubuntu OS . . . 211
8.1.6 Offload heavy computation . . . 212
8.1.7 Tracking of object detection boxes . . . 212
8.1.8 3D object recognition with graph CNNs . . . 213
8.2 Mechanical . . . 213
8.2.1 Increase lifting capacity of the arm . . . 213
8.3 Electrical . . . 214
8.3.1 Upgrade hardware . . . 214
8.3.2 Battery . . . 214

9 Challenges 215
9.1 Non-technical challenges . . . 215
9.1.1 Electrical engineer could not continue . . . 215
9.1.2 Previous report being exempt from public . . . 215
9.1.3 Carbon CNC mill broken . . . 215
9.1.4 External supervisor changing job during the project . . . 216
9.2 Technical challenges . . . 216
9.2.1 Corrupting one of our microSD cards . . . 216
9.2.2 Frying an expansion board diode . . . 216

9.2.3 Problems with downloading tflite-model-maker . . . 217
9.2.4 Difficulties in using the PyTorch library . . . 218
9.2.5 Robot randomly going unresponsive . . . 219
9.2.6 Issues with MoveIt . . . 219
9.2.7 Trouble transferring ∼300,000 points of depth data . . . 220
9.2.8 Frying a BMS . . . 222

10 Conclusion **223**

11 References **224**

A User stories **235**

B Use cases **241**

C System requirement specification **246**

D Global requirements **250**

E Software testing documentation **252**

F Estimated project timeline **269**

G Budget **271**

H Rapid risk ranking **274**

I Schematic diagrams **282**
I.1 Expansion board schematics . . . 282

J Battery **285**
J.1 Introduction . . . 285
J.2 Design . . . 285
J.2.1 General design . . . 285
J.2.2 Components . . . 286
J.2.3 Research . . . 286
J.2.4 Design . . . 288
J.3 Building the battery with BMS . . . 294
J.4 Building the battery without BMS . . . 296
J.5 Test of battery cells . . . 296
J.5.1 Introduction . . . 296
J.5.2 Equipment . . . 296
J.5.3 Testing . . . 297
J.5.4 Results . . . 298
J.6 Testing of batteries . . . 298
J.6.1 Battery with BMS . . . 298
J.6.2 Battery without BMS . . . 299
J.6.3 Discussion . . . 299
J.7 Using the battery . . . 300
J.7.1 General use . . . 300
J.7.2 Assembly . . . 300
J.7.3 Swapping cells . . . 303

J.7.4 Charging . . . 303

K Initial software implementation for arm 304

L Electronics interface description 307
L.1 Motor expansion board YB-ERF01-V1.0 interface description . . . 307

M Earlier iterations of object detection 308
M.1 First iteration . . . 308
M.2 Second iteration . . . 308
M.3 Testing of the custom model . . . 310
M.4 Previous bolt detection dataset . . . 310

N Implementation of detection caching 312

O Robot: Detailed Implementation and Configuration 316
O.1 Example code a ROS 2 node . . . 316
O.2 Example of Rosmaster Library code modifications . . . 317
O.3 Installing Docker . . . 319
O.4 Permissions to use Docker . . . 320
O.5 Dockerfile . . . 320
O.6 requirements.txt . . . 321
O.7 Expansion board symlink . . . 321
O.8 MongoDB database implementation . . . 321
O.8.1 Installing the Docker image . . . 321
O.8.2 Running the container . . . 322
O.9 Measuring the speed of the robot . . . 322
O.9.1 Speed at 0% . . . 322
O.9.2 Speed at 50% . . . 322
O.9.3 Speed at 100% . . . 323
O.9.4 Errors with estimation . . . 324
O.10 Finding arm camera angle . . . 324
O.11 Finding and drawing end effector point . . . 325
O.12 Mapping servo ids to names . . . 326
O.13 Mesh plotting code . . . 327
O.14 Dealing with different paths . . . 328
O.15 Flask video stream implementation . . . 329
O.16 Missing packages for the Astra driver . . . 329
O.17 Docker robot container options . . . 329
O.18 Building ROS 2 from source . . . 329
O.19 Building code documentation . . . 330
O.20 Robot repository timeline & lines of code . . . 332

P VR application 334
P.1 Unity Scene and Game Object Elements . . . 336
P.1.1 Scene . . . 336
P.1.2 Game objects . . . 336
P.1.3 Prefab . . . 336
P.2 First Two Iterations . . . 337
P.2.1 First iteration: Initial design . . . 337
P.2.2 Conceptualization . . . 337

P.2.3 Second Iteration: New feature and improvements 339

Q Calculations **340**
Q.1 Degrees of freedom for robot arm 340
Q.2 Battery duration calculation 341

R Code statistics **343**
R.1 Repositories timeline 343

S Netron Object detection model analyzation **345**
S.1 Bolt detection 346
S.2 People detection 356

T Technical drawings **366**
T.1 Robot design iteration one 366
T.1.1 Arm cover iteration one 366
T.1.2 Car cover iteration one 368
T.1.3 Camera holder iteration one 370
T.1.4 Shock absorber iteration one 373
T.2 Electronics 374
T.2.1 Motor expansion board YB-ERF01-V1.0 dimensions 374
T.3 Final parts technical drawings 375
T.3.1 Floors 375
T.3.2 Walls 377
T.3.3 Expansion board brackets 381
T.3.4 Raspberry Pi case bracket 383
T.3.5 Magnet brackets 384
T.3.6 Brackets 386
T.3.7 Battery drawer 392
T.3.8 Yahboom battery case 393
T.3.9 KROMIUM battery 394
T.4 Subassembly drawings 395
T.5 Floor assemblies 403
T.6 Wall assemblies 407
T.7 Main assembly 409
T.8 Production drawings 413

U Robot code documentation **414**

V Object-detection code documentation **472**

W Transfer learning training code documentation **484**

X VR headset code documentation **495**

Y Working hours **704**

List of Figures

1.1 A picture of a VR headset with hand controllers [1] . . . 32
1.2 Fields and subfields of AI . . . 34
2.1 Project organization chart . . . 39
2.2 "Raspberry Pi connector for PCIe" [2] . . . 43
2.3 Kromium's GitHub workflow . . . 46
2.4 Typical week at Kromium **before** Easter . . . 48
2.5 Typical week at Kromium **after** Easter . . . 48
3.1 Human-robot interaction through virtual reality . . . 52
3.2 Human-robot interaction with controls . . . 52
3.3 Interaction Features for Robot Control from the operator . . . 53
3.4 Features provided to the operator . . . 54
3.5 Robot exploded view. More detailed version T.7. . . . 56
3.6 Changing electronics . . . 57
3.7 Changing battery . . . 57
3.8 Some of the robot movement options. Source: [3] . . . 58
3.9 How a 2P3S configuration looks like on paper . . . 59
4.1 Main components in the VR application . . . 63
4.2 Overview of the application (Disconnected from the robot) . . . 64
4.3 Overview of the application left-hand side when in action . . . 64
4.4 Network manager VR application . . . 65
4.5 Network manager interface examples . . . 66
4.6 Data distribution from network manager to the application . . . 67
4.7 Camera monitor inside the VR application . . . 68
4.8 Camera monitor in Drive mode (Camera 1) detecting a person . . . 69
4.9 Camera monitor in Arm mode (Camera 2) detecting a bolt . . . 69
4.10 Use case diagram - Robot operator . . . 70
4.11 State machine diagram - Mode states in VR application . . . 71
4.12 Drive scene detecting user's hand movements . . . 72
4.13 Arm scene detecting user's hand movements . . . 72
4.14 Visual interface for changing mode, connect to the robot and get depth data 72
4.15 Activities that occur when changing mode . . . 73
4.16 Scene for controlling the robot car . . . 74
4.17 Controlling the robot car components . . . 75
4.18 Default tracking is when the hand is not detected. . . . 76
4.19 Tracking area highlighted when the hand is detected. . . . 76
4.20 Activities in drive mode . . . 77
4.21 Tracking area in the application drive mode . . . 78
4.22 Tracking area conceptualized . . . 78
4.23 Tracking area divided by sections . . . 78
4.24 Calculation of speed . . . 79
4.25 Scene for controlling the robot arm . . . 80
4.26 Default tracking is when the hand is not detected - arm scene . . . 80
4.27 Tracking area highlighted when the hand is detected - arm scene . . . 80
4.28 Illustration of the robot arm and user's hand - Back side . . . 81
4.29 Illustration of the robot arm and user's hand - Front side . . . 81
4.30 Illustration of the robot arm and user's hand - Front side . . . 81
4.31 Activities in Arm Mode . . . 82
4.32 Robot status information panel . . . 83

4.33 Data distribution and updating process in the Head-Up Display 83
4.34 Emergency button in the VR application 84
4.35 Emergency button in action 85
4.36 Example of ROS 2 Nodes [4] 86
4.37 All the ROS nodes in the robot system 87
4.38 Validating commands based on mode 90
4.39 Drive camera view 92
4.40 Arm camera view with end effector indication 92
4.41 Predefined arm movement when switching from drive to arm 93
4.42 Normal drive mode 94
4.43 Precision drive mode 94
4.44 Reverse drive mode 94
4.45 Logger node topic subscriptions 95
4.46 Linear battery estimation plot 96
4.47 GeoGebra linear estimation of the speed 97
4.48 Kinematics black box 98
4.49 Robotic Arm Servo Limitations [5] 98
4.50 End-effector placement 99
4.51 Lengths of each arm link 99
4.52 Two valid arm configurations in the same point 100
4.53 3R Arm 101
4.54 inverse kinematics - 3 DOF robotic arm 102
4.55 VR operator's hand and corresponding JSON data received 104
4.56 Coordinate system showing the YZ-plane with the point (0, 0.5, 0.5) . . . 105
4.57 Angles related to X value 106
4.58 Precision differences in x to angle calculation 106
4.59 ϕ function based on Z-coordinate 107
4.60 Elbow down is illegal for this point 108
4.61 Simulation of arm angles in Python 109
4.62 Object detection black-box 109
4.63 High-level representation of AI detection in the system 110
4.64 A simple neural network with two inputs (the boxes), one hidden layer of two units and two outputs (circle 5) [6, p. 804] 111
4.65 Convolution example with 2 steps 114
4.66 Padding example 115
4.67 Example of average pooling from 4 adjacent values 115
4.68 Fully connected layer 116
4.69 A simple CNN 116
4.70 Frozen and trainable layers in Fine tuning 118
4.71 Standard Convolution vs. depthwise separable convolution 119
4.72 Bottleneck layer visualisation [7, p. 4] 120
4.73 Residual block 120
4.74 Full MobileNet architecture [7] 121
4.75 Example architectures of two-shot and single-shot detectors 121
4.76 Binary Cross-entropy versus Focal Loss [8, p. 1] 123
4.77 MediaPipe architecture [9] 124
4.78 Labelling an image with a person [10] 125
4.79 The Pascal VOC directory structure and XML file format 126
4.80 Difference without and with data augmentation [10] 127
4.81 Training process 128

4.82 Astra Pro Plus Diagram [11] . . . 129
4.83 Astra Pro Plus RGB picture (1280x720) . . . 131
4.84 Depth data plot of 255,888 points (640x480) . . . 132
4.85 Huffman encoding difference . . . 133
4.86 Plot seen from front . . . 134
4.87 Plot seen from the right-hand side . . . 134
4.88 Compressed depth data plot of 3,412 points . . . 134
4.89 Sequence of activities between involved classes for mapping depth data . 136
4.90 Conceptual illustrations of depth data visualization . . . 137
4.91 Real-time depth data visualization in the VR application showing a slightly opened door. . . . 137
4.92 Visualization of the Digital Twin in the VR Application . . . 138
4.93 Digital twin rotation points and pinch . . . 139
4.94 Examples of the digital twin alongside the real robot arm . . . 140
4.95 Workflow of the Digital Twin Data Visualization . . . 141
4.96 Dynamic content gathering of ROS 2 messages . . . 142
4.97 Data Distribution by the Digital Twin Controller . . . 143
4.98 The different distance perspective . . . 144
4.99 Top - side - front perspective . . . 144
4.100Arm camera perspective of fastening without bolt detection . . . 145
4.101Reversing camera . . . 146
4.102Wit.ai application interface . . . 148
4.103The different entities available [12] . . . 148
4.104Illustration of box containing illegal points . . . 149
4.105Caching of detection results . . . 150
4.106Ping command showing latency in milliseconds . . . 151
4.107Internal logging monitor inside VR-application . . . 152
4.108Example JSON data used . . . 153
5.1 Final battery design complete . . . 156
6.1 The robot at the start and end of the project . . . 157
6.2 Equipment from previous projects . . . 157
6.3 Some of the design ideas . . . 159
6.4 Robot at 02.03.2024 . . . 159
6.5 Iteration two SOLIDWORKS model . . . 160
6.6 Models downloaded from *GrabCad* . . . 160
6.7 Robot arm assembly from *Yahboom* . . . 161
6.8 Simple models motor and case . . . 161
6.9 Expansion board models and brackets . . . 162
6.10 Iteration two assembled . . . 162
6.11 Robot iteration three design . . . 163
6.12 New expansion board brackets . . . 163
6.13 Simple robot arm model . . . 164
6.14 Robot iteration three assembly . . . 164
6.15 Robot iteration four design . . . 165
6.16 Robot iteration four assembly . . . 165
6.17 Robot iteration five design . . . 166
6.18 Robot iteration five assembly . . . 167

6.19 A part of the first iteration. Arrow 1 points towards a block representing the attachment between the charger wires and wires leading to the BMSs. Arrow 2 points towards a hole where the wires from the end will appear. Arrow 3 points towards one of the attachment holes for the cover 168
6.20 The second design iteration of the battery 169
6.21 In figure a: Arrow 1 points towards the holder for the mechanical switch. Arrow 2 points towards the magnetic connector. Arrow 3 points towards the mechanical switch 169
6.22 Arrow 1: area for connector plates. Arrow 2: wedging mechanism 170
6.23 Battery iteration five 171
6.24 Producing battery iteration five 171
6.25 Final battery design 172
6.26 Inside of final battery 172
6.27 Configure links and joints 173
6.28 Robot arm moved during export 174
6.29 Aluminium sheet parts 175
6.30 CNC assembly in solidworks and after cut 176
6.31 Before and after adjusting the roof for water jet production 177
6.32 Cover parts after production 177
6.33 Burn marks on edge of front cover 178
6.34 Post processing tools 178
6.35 Before and after clean up 179
6.36 Drilling the holes for the roof 180
7.1 Precision and Recall of people detection model 182
7.2 Loss functions 183
7.3 Total losses 184
7.4 AP and AR (Bolt) 184
7.5 Loss functions 185
7.6 Total losses 186
7.7 Simple inference process 187
7.8 Pre- and post-processing steps 187
7.9 The annotated frame after post-processing 188
7.10 Person 189
7.11 Object detection recognition and displaying of information 189
7.12 Full activity diagram of testing the model 190
7.13 The fixed angle bracket. The hole is square to remove the need for support when 3D printing the part. 196
7.14 This way the cable for the arm camera does not hook itself to the robot. . 196
7.15 This is how the pieces fit together. 197
7.16 Iteration one testing 198
7.17 Iteration two sections to remove 199
7.18 Holes to find the best camera position. 200
7.19 Iteration three sections to fix 201
7.20 Iteration four sections to fix 202
7.21 Weight of the robot 203
7.22 Test of battery with BMS. The red light on the BMSs indicates that the cells are charging. 206
8.1 3D mapped environment with OctoMap [13] 209
8.2 Ubuntu support as of 9th of February 2024 [14] 211
8.3 Ubuntu support as of 6th of May 2024 [14] 212

9.1 Both pictures show the back side of the expansion board. 217
9.2 YOLO real-time detection without NMS 218
9.3 Mesh seen from front 221
9.4 Mesh seen from the right-hand side 221
9.5 3D mesh of depth data with highlighted objects 221
9.6 The BMS with the fried IC 222
H.1 Consequence severity 275
H.2 Probability levels 276
H.3 Risk Matrix 276
H.4 Risk levels 276
H.5 Hazardous events allocated to stakeholders 277
H.6 Possible hazards and description 278
H.7 RRR-main form 1/2 279
H.8 RRR-main form 2/2 280
H.9 RRR-revision overview 281
I.1 Schematic of the robot expansion board YB-ERF01-V1. 283
I.1 Schematic of the robot expansion board YB-ERF01-V1. 284
I.2 Crop out from YB-ERF01-V1. D3 diode is observable in the bottom right corner. 284
J.1 This is how a 3S2P battery is connected 286
J.2 This is how a 2P3S battery is connected 286
J.3 Functional diagram of the charging module [15] 287
J.4 A part of the first iteration. Arrow 1 points towards a block representing the attachment between the charger wires and wires leading to the BMSs. Arrow 2 points towards a hole where the wires from the end will appear. Arrow 3 points towards one of the attachment holes for the cover 289
J.5 The mounting plate for the battery cell holders for the first battery design . 289
J.6 Battery iteration 2. Arrow 1 points towards the holder for the mechanical switch. Arrow 2 points towards the magnetic connector. Arrow 3 points towards the mechanical switch 290
J.7 Mounting plate for battery cell holders, for the second design 291
J.8 The cover for the second iteration 291
J.9 The mechanical switch. Arrow 1 points towards the space for the connectors. Arrow 2 points towards the taps that keep the switch attached to the battery box. 292
J.10 The red markings show the added cut to the holes for the switch 292
J.11 The mounting plate had to be made longer for this iteration because the fork switch had to be slid in the length direction. The small cut in the bottom right corner is to make space for the notch in the battery cover lid. 293
J.12 The circuit for the battery with BMS in a 2P3S configuration. Wires for charging are not drawn into this circuit. 293
J.13 The battery cover for design iteration four 294
J.14 Mounting plate for the final design iteration 294
J.15 The switches used for battery iteration four, in this position the switch is turned on 295
J.16 The switches used for battery iteration four, in this position the switch is turned off 296
J.17 The voltage test was a success. The white ribbon under the battery cells is used to detach the battery cells easily. 299
J.18 Step one of assembling the battery 301

J.19 Insert the battery into the battery box . . . 301
J.20 Close-up picture of the back side of the connector, from the datasheet for the magnetic connector[16] . . . 302
J.21 . . . 302
J.22 The finished battery . . . 303
K.1 VR hand control area . . . 304
L.1 Motor expansion board YB-ERF01-V1.0 interface description . . . 307
M.1 Object detection with the COCO dataset and MobilenetV2 . . . 308
M.2 System architecture for Transfer Learning . . . 309
M.3 Testing the custom-made model . . . 310
M.4 Example images from dataset . . . 311
O.1 50% speed: start time . . . 322
O.2 50% speed: end time . . . 323
O.3 100% speed: start time . . . 323
O.4 100% speed: end time . . . 324
O.5 Enter Caption . . . 325
O.6 Enter Caption . . . 326
O.7 Commits timeline [17] . . . 332
O.8 Lines of code [18] . . . 333
P.1 High-level system architecture . . . 335
P.2 Default Unity empty scene [19] . . . 336
P.3 Different game objects: animated character, a light, a tree, and an audio source [20] . . . 337
P.4 Tree prefab example in a scene [21] . . . 337
P.5 Initial iteration of VR application development . . . 338
P.6 Command conversion from Hand tracking . . . 338
P.7 Second iteration of VR application development . . . 339
P.8 Hand tracking version one . . . 340
P.9 Hand tracking version two . . . 340
P.10 Speed control . . . 340
Q.1 Robot arm simple drawing of joints and bodies. Joints (J) = blue & bodies (N) = Red . . . 341
R.1 transfer-learning-training commits timeline [22] . . . 343
R.2 obj-detection-pi commits timeline [23] . . . 344
T.1 Car to arm interface technical drawing . . . 366
T.2 Car to arm interface cover technical drawing . . . 367
T.3 Car to arm interface assembly technical drawing . . . 367
T.4 Top of the temporary cover . . . 368
T.5 Rear part of the temporary cover . . . 368
T.6 Rear side piece of the temporary cover . . . 369
T.7 Front side part of the temporary cover . . . 369
T.8 Assembly of the temporary cover . . . 370
T.9 Base for temporary camera support . . . 370
T.10 Holder for the temporary camera . . . 371
T.11 Legs for the temporary camera . . . 371
T.12 Stabilizer for the camera legs . . . 372
T.13 Spacer for the camera legs . . . 372
T.14 Temporary camera support assembly . . . 373
T.15 Shock absorber front short . . . 373
T.16 Shock absorber front long . . . 374

T.17 Caption . . . 374
T.18 Robot car bottom floor . . . 375
T.19 Robot car middle floor . . . 376
T.20 Robot car top floor . . . 376
T.21 Robot car roof . . . 377
T.22 Robot car front wall . . . 377
T.23 Robot car left wall . . . 378
T.24 Robot car right wall . . . 378
T.25 Robot car rear wall . . . 379
T.26 Raspberry Pi house large wall . . . 379
T.27 Raspberry Pi house open wall . . . 380
T.28 Wall behind robot arm camera . . . 380
T.29 Raspberry Pi house middle wall . . . 381
T.30 Motor expansion board snap bracket bottom . . . 381
T.31 Motor expansion board snap bracket top . . . 382
T.32 USB hub snap bracket top . . . 382
T.33 USB hub snap bracket bottom . . . 383
T.34 Raspberry Pi bracket top . . . 383
T.35 Raspberry Pi bracket bottom . . . 384
T.36 Magnet bracket wall bottom . . . 384
T.37 Magnet bracket wall top . . . 385
T.38 Magnet bracket car back . . . 385
T.39 Magnet bracket car front . . . 386
T.40 L bracket . . . 386
T.41 Top floor opening to wall bracket . . . 387
T.42 Top floor other bracket . . . 387
T.43 Wall attachment bracket female . . . 388
T.44 Wall attachment bracket male . . . 388
T.45 520 DC motor bracket . . . 389
T.46 Arm cam holder fixed angle . . . 389
T.47 Cam cable holder . . . 390
T.48 Female magnet connector bracket . . . 390
T.49 Power switch bracket left side . . . 391
T.50 Power switch bracket right side . . . 391
T.51 Battery drawer rails . . . 392
T.52 Battery drawer . . . 392
T.53 Yahboom battery case lid . . . 393
T.54 Yahboom battery case . . . 393
T.55 KROMIUM battery case . . . 394
T.56 KROMIUM battery cell separator holder . . . 394
T.57 KROMIUM battery case lid . . . 395
T.58 Motor assembly Front left & back right . . . 395
T.59 Motor assembly Front right & back left . . . 396
T.60 KROMIUM battery assembly . . . 396
T.61 KROMIUM battery drawer assembly . . . 397
T.62 Yahboom battery drawer assembly . . . 397
T.63 Magnet car mount assembly . . . 398
T.64 Magnet wall mount assembly . . . 398
T.65 Motor expansion board snap bracket assembly . . . 399
T.66 Power switch bracket assembly . . . 399

T.67 Raspberry Pi bracket assembly 400
T.68 USB hub snap bracket assembly 400
T.69 Wall attachment bracket assembly 401
T.70 Yahboom battery case assembly 401
T.71 Bottom floor assembly 403
T.72 Middle floor assembly 404
T.73 Top floor assembly 405
T.74 Robot roof assembly 406
T.75 Front wall assembly 407
T.76 Left wall assembly 407
T.77 Right wall assembly 408
T.78 Rear wall assembly 408
T.79 Robot assembly without walls 410
T.80 Robot assembly with shell 411
T.81 Robot assembly bill of materials 412
T.82 Parts layout for CNC machining 413
T.83 Parts layout for water jet cutting 413

List of Tables

1.1 Team Members 31
3.1 The two groups of battery cell clusters used 60
4.1 Summary of camera and machine learning configurations across different operational modes. 70
4.2 Velocity at percentages 97
4.3 Big-endian 131
4.4 Little-endian 131
7.1 Battery cell test results. Cell number 11 was dead. 205
7.2 Operational time based on 30 min operation, and calculations 207
J.1 The resistance (in kilo Ω) for the different charging currents [24] 288
J.2 Battery cell test results. Cell number 11 was dead. 298
J.3 The two groups of battery cell clusters used for the battery 300
M.1 Example of the dataset CSV file looks like 309

Abbreviations

AI Artificial Intelligence. 30, 32–34, 37, 38, 41, 42, 45–47, 52, 55, 61, 109, 110, 123, 181, 223

ANN Artificial Neural Network. 111–113

AP Average Precision. 181, 184, 186

API Application Programming Interface. 316

AR Average Recall. 181, 184–186

BMS Battery management system. 59, 155, 156, 167, 205, 206, 214, 222, 285, 286

CAM Computer aided manufacturing. 176

CD Continuous Deployment. 40

CI Continuous Integration. 40

CNC Computer numerical control. 175, 176

CNN Convolutional Neural Network. 38, 112–114, 116–118, 122, 213, 218, 308

CPU Central processing unit. 42, 43, 123, 216, 219

CSS Cascading Style Sheets. 41

CSV comma-separated values. 309

DBMS Database Management System. 153

DOF degrees of freedom. 36, 54, 340

DPST Double Pole Singe Throw. 155, 170, 293

EOL End-of-life. 42

FOSS free and open-source software. 44

FPS Frames per second. 89, 150, 219

GPS Global Positioning System. 209

GPU Graphics processing unit. 42, 44, 123, 127, 218, 219

GUI Graphical user interface. 219, 220

HTML HyperText Markup Language. 41, 44, 45

HTTP Hypertext Transfer Protocol. 149, 329

humble Humble Hawksbill. 42

IC Integrated circuit. 222, 286, 287

IDE Integrated Development Environment. 44, 62, 318

IoU Intersection over Union. 181, 185, 186, 218

IP Internet Protocol. 61, 88

IR Infrared. 129

JSON JavaScript Object Notation. 65, 87, 88, 135, 141, 142, 145, 151, 153, 191

Kromium KONGSBERG Remote Operated Mechanical Image Utilizing Machine. 39

LCDs Liquid-crystal displays. 32

Li-ion Lithium-ion. 285

LiPo Lithium Polymer. 285

LTS Long term support. 42, 211

ML Machine Learning. 44, 54, 68, 69, 123–126, 148, 186, 308

NaN Not a Number. 131

NMS Non-Maximum Supression. 18, 188, 189, 218, 219

OS Operating System. 43, 44, 211, 217, 219

PCIe Peripheral Component Interconnect Express. 43

PDF Portable Document Format. 44, 45

PID Proportional-Integral-Derivative. 210

pip package installer for Python. 321

PLA Polylactic acid. 162, 195

PoC Proof of Concept. 30, 33, 36, 66, 143, 150, 209, 220, 223

QoS Quality of Service. 316

QRA Quantitive Risk Analysis. 49

RAM Random Access Memory. 42

ReLU Rectified Linear Unit. 111, 117, 119

RGB Red Green Blue. 114, 129, 187, 210

ROS Robot Operating System. 42–44, 62, 86–89, 100, 110, 129, 130, 140–142, 156, 191–193, 211, 219, 310, 316, 329

RP Rapid Prototyping. 197

RPi Raspberry Pi. 42, 43, 46, 56, 61, 62, 85, 88, 89, 118, 123, 156, 211, 216–220, 308, 310, 327, 329, 330

RPiOS Raspberry Pi OS. 43, 44, 61, 211, 319, 329

RRR Rapid Risk Ranking. 48

SCARA Selectively compliant arm for robotic assembly. 35

SDK Software development kit. 148, 337

SSD Single-Shot Detector. 122

SSH Secure Shell. 44, 46, 61, 62, 95, 219, 220

TCP Transmission Control Protocol. 88, 95, 132, 135, 142, 146, 150, 151, 220

TFLite TensorFlow Lite. 123, 124, 188, 218, 219, 308, 310

UDP User Datagram Protocol. 150, 222

UGV Unmanned ground vehicle. 35

URDF Unified Robotics Description Format. 173, 210, 213

URL Uniform Resource Locator. 69

USB Universal Serial Bus. 89, 156, 216, 329

VR Virtual Reality. 14, 16, 30, 32, 33, 37, 38, 41, 42, 44, 46, 47, 52–55, 61–63, 65, 68, 70–72, 74, 75, 79, 82, 83, 85, 87–91, 93–98, 104, 106, 108, 110, 129, 131, 132, 135–142, 145–152, 181, 191, 194, 208–210, 212–214, 220, 222, 223, 304, 312, 329, 334, 336–338

VSC Visual Studio Code. 44, 45, 62

WSL Windows Subsystem for Linux. 46, 217

Description of Terms

GrabCad A open-source website for 3D models and engineering. 160

SOLIDWORKS A program used for 3D-design and testing. 47, 160, 161

3D-printing Also known as additive manufacturing. Construction of a three-dimensional object layer by layer following a 3D model. 47

18650 A battery cell that has a nominal voltage of 3.7 volts and got its name after the diameter and length, 18 mm in diameter and 65 mm in length. 285

additive manufacturing Also known as 3D-printing. Construction of a three-dimensional object layer by layer following a 3D model. 46

artificial neural network A node-based, trainable machine learning model inspired by neurons in the brain [6, p. 727]. 110, 123

C# A high-level, multi-paradigm programming language developed by Microsoft that encompasses strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. 337

closed-source also known as proprietary software "is software whose author owns all rights to use, modify, and copy it. Software products that do not meet the requirements for open-source software are generally categorized as closed-source software." [25]. 88

computer-aided design The use of computer software in the creation, optimization, testing, simulation, and modification of a design. 47

delamination A type of failure where the material fractures into layers. 176, 215

docstring "Python docstrings are the string literals that appear right after the definition of a function, method, class, or module" [26]. 40, 44, 45, 318, 331

end effector A point which denotes the final location attained upon configuring an arm's angles. 80, 98–103, 107

epoch One entire passing of data while training a model. 310

finite element method A method of numerically solving differential equations in mathematics and engineering. Can be used to simulate the deformations on a part as a result of applied load. 47

git Version control program which makes collaboration and keeping code up-to-date easier. 42

GitHub A well-known developer platform owned by Microsoft [27] used for project management, code hosting, workflow, and access control. 40, 42, 44–46, 104, 122, 124, 129, 151

ground truth The true answer we are asking our model to predict [6]. 122, 181

hyperparameter The settings that oversee the training process of a machine learning model [28]. 127, 144

Interaction SDK A library of modular, composable components that allows developers to implement a range of robust, standardized interactions (including grab, poke, raycast, and more) for controllers and hands. Interaction SDK also includes tooling to help developers build their hand poses. 337

Jetson Nano A small, powerful computer for embedded applications and artificial intelligence Internet of Things that delivers the power of modern artificial intelligence [29]. 42

Meta Quest 3 A VR headset developed by Meta Platforms, Inc. (formerly known as Facebook). The headset supports both virtual reality and augmented reality through the use of cameras. 41, 44, 52, 61, 208, 214, 337

metadata Information that is given to describe or help you use other information [30]. 124, 127, 188

natural language processing the branch of computer science that involves giving computers the ability to interpret and produce language [30] . 33, 41, 147

overfitting A concept in machine learning that happens when the model is paying too much attention to the data it is training on, thus making subpar predictions on unseen data [6, p. 701]. 126, 127, 184–186, 311

pipeline A set of steps that help automate and organize the process of creating, training, testing, and using machine learning models [31]. 117, 124

planar Two-dimensional space where all the movements of a manipulator are restricted to a single flat plane. 101

pre-preg A type of composite material where the fibres are pre-impregnated with a partially cured matrix. 175

rapid prototyping Producing a prototype quickly, often utilizing CAD models and additive manufacturing instead of manufacturing process planning, tooling, or fixtures. 46

raspberry Pi A small computer made by Raspberry Pi Ltd. https://www.raspberrypi.com/for-home/. 308

Raspberry Pi OS An operating system for Raspberry Pi (previously called “Raspbian”), which is based on Linux [32]. 44

scrum Project tool used for agile software development. 47

Ubuntu A popular Linux operating system which can run on the Raspberry Pi [14]. 42–44, 216, 217

Unity A game engine made by Unity Technologies which supports VR game and app development. 19, 44, 61, 66, 88, 148, 336

wholly-owned "relating to a company that is completely owned by another company or organization" [30]. 36

wrapper "A wrapper is a programming language function for encapsulating and organizing elements within a well-defined interface." [33]. 89, 129

Nomenclature

Φ	The orientation of the end-effector compared to the start point of a manipulator
θ, α, β	Angle
C	Constraints between two rigid bodies
J	The number of joints
L	Loss Function
m	The degrees of freedom of a single body
N	The number of bodies, including ground
x, y, z	Position

1 Introduction

1.1 About the project

OM | *AD*

In recent years, the realm of offshore operations has witnessed a transformative wave of technological innovation, steering the industry towards unprecedented efficiency, safety, and cost-effectiveness. One avenue which is yet to be explored is the use of remote operations. Interactions through Virtual Reality (VR) systems in combination with human-machine interaction such as hand movements, could be groundbreaking for the industry. These cutting-edge technologies could provide the immersion an onsite operator needs, all while harnessing human decision and precision.
This paper considers developing a Proof of Concept (PoC) prototype for Kongsberg Maritime, which utilizes VR, Artificial Intelligence (AI), and hand tracking for the possibility of carrying out remote oil rig operations.

1.2 Group members

AD | *SO*

Our bachelor group was formed in May 2023, and we had all worked together previously on an interdisciplinary student project named Hydroplant [34]. As a result, we knew each other well and had a good idea of each other's work methods. A desire to challenge ourselves and address complex issues in today's world led us to form this multidisciplinary group. In October 2023 we had our first meeting with our project partner Kongsberg Maritime, and subsequently, we received this project from them. We held several meetings with our external supervisor to define the project before the start of the bachelor thesis.

Although this project is a group effort, each group member gets an individual evaluation. Therefore we thought it was important to define the authorship of each section in the report clearly. Inspired by the project reports of previous groups [35], we chose to specify the initials of the member responsible for writing the section, and also the member who proofread it. The proofreader's responsibility was to ensure that the text was grammatically error-free and consistent with the entire report. This is displayed beside each section, with the writer's initials on the left of a vertical line and the proofreader's initials on the right. The table 1.1 contains information about each group member, their initials, and their role within the project.

	Name	**Shahin Ostadahmadi**
	Initials	SO
	Discipline	Software engineer - Cyber-physical systems
	Role	Project Leader and Systems Engineer
	Name	**Aditi Deshpande**
	Initials	AD
	Discipline	Software engineer - Cyber-physical systems
	Role	Software engineer
	Name	**Oscar Melby**
	Initials	OM
	Discipline	Software engineer - Cyber-physical systems
	Role	Software engineer
	Name	**Adrian Elias Haugjord**
	Initials	AEH
	Discipline	Mechanical engineer - Product Development
	Role	Mechanical engineer
	Name	**Henrik Bertelsen**
	Initials	HB
	Discipline	Mechanical engineer - Product Development
	Role	Mechanical engineer

Table 1.1: Team Members

1.3 Problem domain

1.3.1 Introduction **OM, SO** | *AEH*

Working in offshore environments, such as oil and gas rigs, presents significant safety and operational challenges. The tragedy of the "Alexander L. Kielland" platform on 27th March 1980, where 212 personnel working on the platform fell into the cold North Sea and only 89 survived the disaster [36], underscores these challenges. Traditional operations on offshore platforms require workers to be physically present for daily duties and maintenance, often involving technicians being transported back and forth to work on specific operations.

This project outlines challenges with the traditional operations in offshore platforms identified by Kongsberg Maritime. It aims to explore the world of AI and VR combined with robotics to enable remote execution of these operations from land-based offices. The goal is to replicate the operations typically done by humans on offshore rigs through a combination of VR, AI, and robotics from land-based facilities.

This paper will primarily research the technical aspects of the challenges. Due to time and resource limitations - economic, safety, environment and sustainability aspects will not be part of our problem domain. This section highlights the core technologies of VR, AI, and robotics, their current situation, and specific applications in addressing operational challenges in offshore rigs.

1.3.2 Virtual Reality **SO, AD** | *AEH*

Definition: The definition of Virtual Reality (VR) as a concept is that it is a set of images and sounds produced by a computer that seems to represent a real place or situation [30]. A popular way to realize virtual reality is through simulations. A person observing these simulations acquires an immersive feeling of a virtual world. To fully immerse oneself in the virtual world, one can use head-mounted displays such as a VR headset. These headsets utilize a vast range of sensors such as accelerometers, gyroscopes, infrared sensors, depth sensors, and so on. They also tend to have Liquid-crystal displays (LCDs), or organic light-emitting diodes (OLEDs) for the visuals [37]. These headsets tend to come with hand controllers for interaction. Figure 1.1 shows a picture of a VR headset with hand controllers.

Figure 1.1: A picture of a VR headset with hand controllers [1]

VR differs from Augmented Reality in the way that VR creates a completely immersive digital environment that replaces the user's real-world surroundings, whereas augmented

reality overlays digital elements onto the real world [38].
Status of VR in the industry: Virtual Reality moves beyond games and consumer applications. Several application areas have been investigated in recent years and research has shown that usage of VR applications can be beneficial compared to traditional approaches [39]. In this project, we are narrowing the scope to research how we can utilize VR combined with AI and robotics to develop a PoC to enhance offshore operations.

Application in our project: Several ideas and suggestions have been discussed with the customer. The ideas that our customer found most interesting involve the use of VR:

- **Object detection:** Detect objects in the robot's environment, display information about these objects, and provide this information to the operator through the VR headset.
- **AI assistant:** Assist the robot for mimicking hand movements when the operator uses the VR headset to control the robot's arm. The purpose is to enable precise imitation of human hand motion.
- **AI drive assistant**: Helps the robot follow the operator's hand movements for driving, and suggests routes.

Our project's potential to incorporate these applications depends on time, resources, and progress.

1.3.3 Artificial Intelligence **AD** | *SO*

Definition The book *Introduction to Artificial Intelligence* [40] gives eight different definitions of Artificial Intelligence (AI), classified by thinking humanly, acting humanly, thinking rationally, and acting rationally. As this is an engineering project, it is optimal to present the definition by acting rationally - "**Artificial Intelligence** is the study of the design of intelligent agents." An agent is something that acts and in the digital world, computers are used as agents. These computer agents are expected to act in various ways: operate autonomously, perceive their environment, adapt to change, and create goals. They should be set to achieve the best outcome or, when there is uncertainty, the best-expected outcome [6, p. 4]. In the field of AI, there is a big focus on using computer agents in fields like reasoning, analytics, learning, perception, support for robotics, and much more.

Current trends in AI AI is a very relevant topic in various industries. Businesses use predictive analytics to increase sales and generate more revenue, healthcare providers use machine learning to understand diseases and improve treatments, autonomous vehicles use machine perception to navigate safely, and AI is playing a crucial role in advancing natural language processing, enabling human-computer interactions through voice assistants and chatbots. In the realm of robotics, AI is being used to enhance the capabilities of robots in manufacturing, and domestic settings, making them more autonomous, and capable of managing complex tasks [41]. In our project, we have focused specifically on machine perception, machine learning, and support for robotics.

Machine perception is a field of AI that uses sensors such as cameras to receive data and interpret it in a manner that is similar to the way humans use their senses to relate to the world around them. The end goal is to enable the computer system to use hardware

and software to recognize images, sounds, and even touch in a way that improves the interaction between human operators and machines. [42]. The computer then presents this data in a way that is perceivable to the user and also takes actions based on the data. This particular method is also called **computer vision** [43].

Machine learning is also a field of AI. A machine is learning if it improves its performance on future tasks after making observations about the world [6, p. 693]. Machine learning focuses on data and algorithms to imitate the way that humans learn, gradually improving its accuracy. There are usually three methods of conducting learning, but we focus on supervised learning, where the data that is used to train algorithms to classify data or predict outcomes is labelled [44].

AI in our project We will now present how we are going to use AI in our project. Machine learning and machine perception can be combined to create a technology called **object detection**. Figure 1.2 shows the different fields and subfields of AI and what object detection derives from. Here, an image is taken and classified as a dog, person, etc, and is predicted to belong to a particular class. In addition, the image class is also displayed as an image or video and tracked in real-time [43]. Our goal in the project is to use object detection on the camera placed on the robot to analyze the robot's surroundings and detect objects of interest. In addition, we want to analyze the objects further and display possible information we have about them.

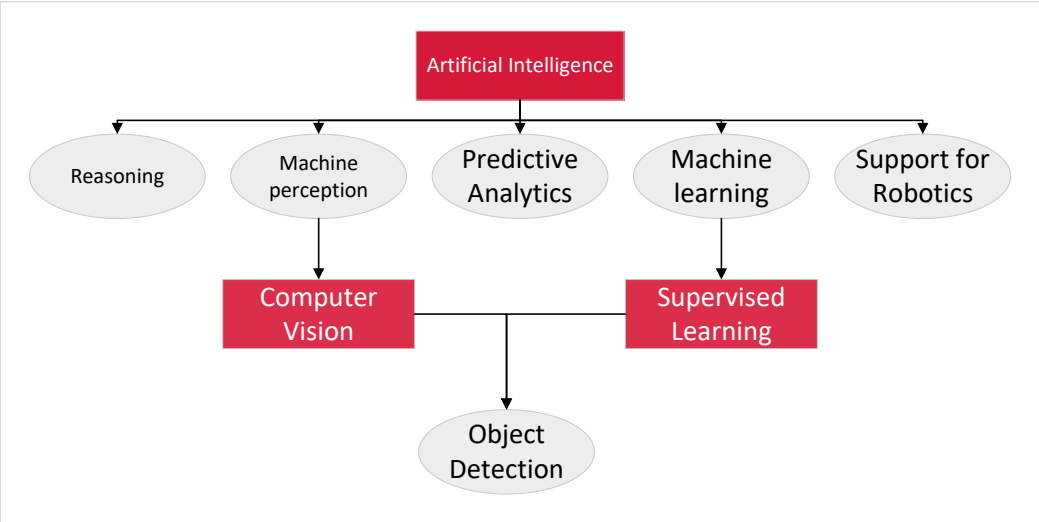


Figure 1.2: Fields and subfields of AI

1.3.4 Robotics

HB | *AD*

Robotics is a huge field within the technology and engineering world. Our project will mainly focus on combining the branches of AI and VR to control a car, which also has a mechanical arm that will be used to do different simple tasks.

Definition and usage in our project:

"Robotics is the intersection of science, engineering, and technology that produces machines, called robots, that replicate or substitute for human actions " [45]. A robot's five major components are effectors, perception, control, communications, and power. An **effector** is an appendage of a robot that can move or make the robot move, it can be legs, arms, necks, and wrists. Within effectors, we have a subsection called **manipulators**. These manipulators are something that can be used like arms or hands. **Perception** is

how the robot can "see" and "feel" the world around it. **Control** is a bunch of computations that maximize the chances of success. **Communication** is how the robot interacts with other agents. **Power** is other functions that can be enabled, such as sound and lights [46, p. 39-40].

Robotics in our project: The effectors on the robot in our project are the omnidirectional wheels, while the mechanical arm is the manipulator. The perception of the robot will be through cameras mounted on the robot itself, and the robotic arm. The control of the robot is the recognition of the different hand gestures, which will send the correct signals to the robot regarding the driving and hand operation. It also concerns the object detection that takes place from the camera. The communication is how the VR headset is "talking" with the robot by sending hand gesture signals to the robot, the robot then calculates what the hand gesture means and sends live images from the camera back to the VR headset so the operator can see where they drive and how they control the robotic arm. The power functions in our robot will perhaps be some lights and some sort of sound. There are three main categories of robots: air, ground, and sea. Our robot will be an Unmanned ground vehicle (UGV). Within the UGV field we have three categories again.
The three categories for UGV are:

- **Humanoid:** A robot that imitates humans and has human-like features.
- **Mobile:** Any robot that can move.
- **Motes:** A small stationary robot.

Our robot is a mobile robot, and here we have three more categories:

- **Man-packable:** The robot fits inside one or two backpacks, where one backpack has the controller and batteries, and the other backpack has the robot itself.
- **Man-portable:** The weight of the robot requires two people to carry it.
- **Maxi:** A Robot so large and heavy that it is too heavy to be carried by people.

Here, our robot falls under the category of man-packable [46, p. 39-40].

1.3.5 Robotic arms **AEH** | *HB*

At the start of this project, the team had no prior experience working with robotic arms. This chapter will therefore explore what categories of robotic arms exist in the industry, what category the arm we were given at the start of this project belongs to, and if there are more suitable options.
Robotic arms are as the name implies simply different types of mechanical arms working independently or manually to perform certain tasks. There are many types of robotic arms for various amount of use-cases.
Some distinguishable categories of robotic arms we found are:

Selectively compliant arm for robotic assembly (SCARA) robots which are small, fast, and precise as a result of their rigid builds. The rigidity of the SCARA comes in part from the fact that it cannot move vertically. This is a suitable choice for flat working environments such as PCB soldering but a poor choice for a project like ours.

Cartesian robots utilize three axes and are mostly used for operations confined within

a rigid frame or environment such as 3D printers, plasma cutters, etc.

Cylindrical robots are built up of a rotary joint at the base, a linear joint that controls the height, and a linear joint for the length of the arm.

Delta robots are similar to Cartesian robots in that they are encapsulated in a work environment. They are built up of multiple arms connected from the main operational head up to different motors, the different motors acting on each arm move the head to different positions. This is used in pick and place, 3D printing, and similar quick and precise operations.

Lastly, we have **articulated arms** which is a broad term for multiple types of arms where the one we currently have is called a six-axis robot. As the name implies our robot has six-axis also called six degrees of freedom (DOF). More about what DOF in chapter Q.1.

There is certainly an argument to be made if other robotic arms could be used in the project. However, one of the ideas discussed with the client is to map each joint of the operator and assign it to a designated joint on the robot arm. Going for another design could close the possibility of testing and implementing this.
If the dimensions and design of the arm were to be changed, the calculations and kinematics behind the arm would also need to be redone. This could take a lot of time from the actual task of researching different ways of integrating Virtual reality with Artificial intelligence and robotics in different ways.

1.3.6 Integration of VR, AI, and Robotics **SO** | *AD*

The fusion of Virtual Reality (VR), Artificial Intelligence (AI), and robotics forms the backbone of this project. This integration aims to enhance offshore operations through land-based facilities, providing innovative solutions to existing challenges. By combining these technologies, the project seeks to develop a PoC that demonstrates the potential for more efficient, safe, and effective remote operations.

1.4 Project overview

1.4.1 The customer **AD** | *OM*

Kongsberg Maritime is a wholly-owned subsidiary of Kongsberg Gruppen (KONGSBERG). KONGSBERG is a leading global technology corporation delivering mission-critical solutions with extreme performance for customers that operate under extremely challenging conditions. Kongsberg Maritime supplies the technology, equipment, and services needed for sustainable maritime operations today and in the future. Kongsberg Maritime is a big multinational company with 117 offices in 32 countries and more than 7,000 employees. [47]
As of today, Kongsberg Maritime incurs significant expenses and time in sending personnel to offshore rigs to carry out operations. A safety risk is also posed to the personnel that should be considered. As a result, there is a great emphasis on remote control and autonomy for offshore operations. [48, p. 33]

1.4.2 Project goal **AD** | *OM*

In the previous section 1.3, we explored the individual capabilities of VR, AI, and robotics, and their transformative impact on various industries. Building on these topics, we look back to the pressing challenge within the offshore industry. Given the demanding nature of offshore operations and the necessity for safety and efficiency, our project is driven by a critical question: How can we utilize VR and AI, in combination with robotics, to enhance and improve offshore operations? This question lies at the centre of our research, aiming to use these technologies for innovative solutions in offshore environments.
This thesis aims to design and implement a robot prototype that an operator can navigate through the use of an VR headset. The approach includes hand gesture recognition to translate the operator's movements to robot commands, applying AI to enable the robot to identify objects in its surroundings and display information about the objects, ensuring the robot's design is functional and aesthetically pleasing, and facilitating easy battery replacement.

1.5 Requirements **AD** | *AEH*

The requirements for the project were defined from the project description given by Kongsberg Maritime. During our meetings with them, they talked about the idea they had in mind. We then defined the user stories and use cases that covered the objectives of the product. The user stories can be found in Appendix A and the use cases can be found in Appendix B. From the user stories and use cases, we created a list of requirements which we presented to our external supervisor. We received confirmation that the requirements fit the idea that Kongsberg Maritime had in mind. The requirement table can be seen in Appendix C.
In the table, the user story ID and name are first mentioned, followed by the ID and the name of use cases that were defined from the user story. This is because during the process of creating them, we observed that a user story could have several use cases, and a use case could have several requirements. The fifth and sixth columns represent the requirements, followed by their priority that was assigned based on the wishes of Kongsberg Maritime. Also included is the technical performance of a requirement that contains any predefined constraints and criteria. The test method column explains how the requirement will be tested, and the verification column confirms that the product meets the requirement. A test can test several requirements, and a requirement can be tested with several tests. Therefore, we have created a test report ID column where one or more test reports can be linked to a requirement. Lastly, we have the status of the completion of the requirement.

1.5.1 Definition of priority order **SO** | *AD*

The requirements of this project all have priorities ranging from A to C. Here is the definition of the order of the priorities:

Priority A - Highest priority: These are the critical requirements that the project must fulfill for successful completion. These are core functionalities that we must meet to achieve our objectives. For example, integration between VR, AI, and the robot for basic operational functionalities.

Priority B - Should have: Important but not critical for the main functionality of the system. They enhance overall performance and user experience but are not essential for the

prototype to function. For example, additional AI capabilities, or advanced hand gesture recognition fall into this category.

Priority C - Nice to have: Beneficial features that are not essential and can be developed if time and resources allow. An example might be additional capabilities of the robot's arm such as rotate tool to rotate an object.

1.5.2 Additional requirements **SO** | *OM*

In addition to the original requirements (A, B, and C), the group has addressed further tasks requested by the customer during the project. The implementation details of this additional work are documented in a separate section 4.6.

1.5.3 Validation **AD** | *AEH*

Validation is about constructing the right system. Here we need to make sure that the system does what it is supposed to do in its intended environment and satisfies the needs of the stakeholders [49, p. 4]. Validation is a critical stage as even if the end product is impressive, it cannot be considered a success if it is not what the stakeholder wanted. Therefore, we conducted weekly meetings with our external supervisor to make sure we were always on the right track.

1.5.4 Related work: Object detection **AD** | *OM*

There has been endless research on the object detection field due to advancements in the different AI fields. The paper *"Object Detection with Deep Learning: A Review"* [50] by Zhong-Qiu Zhao et al. provides a comprehensive overview of the evolution of object detection methods, highlighting the transition from traditional approaches to the dominance of deep learning-based models. This shift has been driven by the remarkable success of Convolutional Neural Network (CNN)s (explained further in 4.5.3) in extracting meaningful features from images and accurately localizing objects within them.
The paper *"Implementation of ROS-Based Mobile Robots with Few Shot Object Detection Using TensorFlow API"* explored the integration of object detection into a robotic application [51]. Their work focused on enabling robots to detect objects from a limited number of examples and on edge devices, showcasing the potential for real-time object recognition in robotic applications.
The paper *"Real-time object detection and tracking in mixed reality using Microsoft HoloLens"* [52] presents a mixed reality system using the headset Microsoft HoloLens. It incorporates object detection and tracking with lighter models, but the camera used to detect the objects is from the HoloLens itself, not from a robot placed remotely. It also uses complex 2D to 3D mapping that is too complex for our system.
Kromium's system will use object detection models to analyse the surroundings of the robot. This is visualised by a camera and the camera feed will be displayed on the VR application.

2 Project Management & Risk Assessment

This section outlines our organizational project management approach and describes the tools, applications, and project models used in our project. This section includes how administrative and operational tasks were dealt with; including communication with Kromium's internal and external supervisor. In addition, we have written about the development process, categorically discipline-specified development methods for software, mechanical, as well as electronics. Choosing a clear and well-defined development process helps the project progress toward the success of developing a product that satisfies the customer's requirements and expectations. Finally, we have written about the risk assessment made concerning the project using Rapid Risk Ranking (RRR).

2.1 Organizational structure

SO | *AD*

Our organizational structure is illustrated in Figure 2.1 below. As the project had various software and mechanical aspects, a natural approach to the organization was adopted, where each team member took primary responsibility for a specific subsystem or component. Additionally, the project leader took the role of a systems engineer and an administrator.

Smaller tasks, like note-taking, research on a specific topic, and other duties, were assigned to different group members each time.

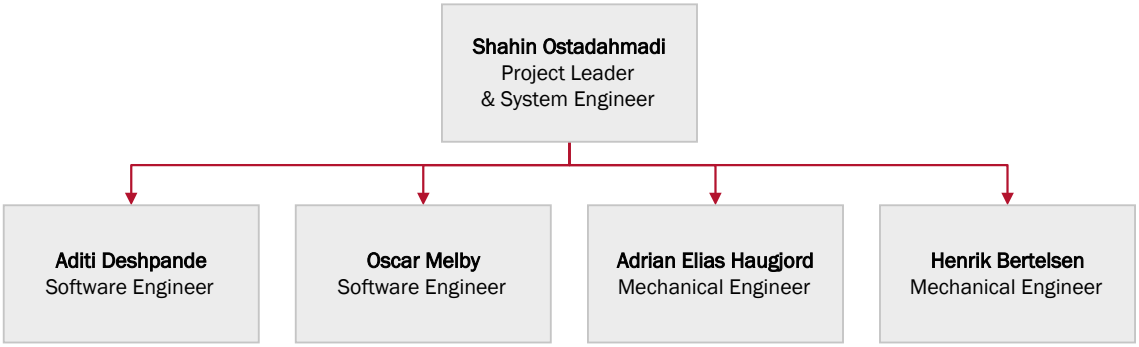


Figure 2.1: Project organization chart

2.1.1 Member responsibilities

OM | *AD*

Each group member is assigned a specific role, which entails responsibility for certain aspects of the project. However, this division of roles does not necessarily imply that the individual solely performs all tasks associated with their role. Instead, they are accountable for ensuring that the assigned tasks are completed and that applicable standards are followed.

Shahin Ostadahmadi

- **Requirement-driven development process** - ensure the project stays on track and tasks being done by members belong to requirements
- **Interfaces** - managing interfaces between different subsystems and components
- **System integration** - ensure components of the system get integrated, both software and hardware

- **Code integration** (CI/CD) - ensure the team is integrating new parts and testing them

Aditi Deshpande

- **High-level software design** - responsible for the high-level software design and ensuring that this is followed
- **Interfaces for software** - define software interfaces

Oscar Melby

- **GitHub workflow** - ensure the team follows the workflow (see 2.3) and responsible for GitHub actions
- **Code standards** - define formatting, linting and code style standards
- **Git** - fix merge conflicts, ensure code is available on GitHub
- **Code documentation** - define documentation tools, design, and docstring formats
- **Unit tests** - relevant parts of the codebase should have unit tests

Adrian Elias Haugjord

- **Robot design** - design, testing and construction of robot
- **Requirement overview** - overview of requirements for mechanical engineering
- **Order and production** - order and production of aluminium and carbon fibre parts.

Henrik Bertelsen

- **Risk assessment** - responsible for assessments of risks
- **Technical drawings up to standard** - make sure the technical drawings for all parts are up to standard
- **3D drawings up to standard** - make sure that all of the 3D modelled parts are up to standard
- **Battery** - design, produce, and test the battery

2.1.2 Team building **SO** | *AD*

Team building is part of our initiative to enhance the work environment and cooperation within the group. We began our first team-building activity in December 2023 with an escape room in Oslo, solving *The Mad Inventor* escape challenge, where our mission was to "Save humanity from complete idiocy by completing the Mad Inventor's Machine" [53] in one hour. Throughout the project, we plan to organize an event before and after each presentation, involving simple activities such as playing games with the Meta Quest VR headset, dining together, and solving other escape rooms among other things.

2.2 Project tools

OM | *AD*

A plethora of tools and applications were used when working on this project. The tools mentioned here are the most essential ones in terms of management.
Atlassian's Jira and Confluence were used for keeping track of tasks, time, and meetings and for writing smaller documents. Additionally, various applications under the "Office Suite"-umbrella were used for sharing files, communication, and keeping track of working hours.
The paper was written using LaTeX on Overleaf and Zotero was utilized for tracking references and keeping them up-to-date. Discord was used for communication internally in the group and for sharing resources.

2.2.1 ChatGPT

AD | *AEH*

Our group has leveraged the advanced capabilities of ChatGPT, an AI natural language processing model developed by OpenAI, to optimize both the report's overall language and the project's technical aspects. However, we have collected the sources that it uses, read over the source, and referred to the source as a reference.

2.3 Website

OM | *AD*

The Kromium website was made using a HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) template from free-css.com called "lodge" [54]. The template was modified to have fitting colours, layout, and content. The website is static and hosted on USN's servers. The domain name kromium.no was bought and works as a redirect.

2.4 Hardware used

OM | *AD*

This section describes the hardware components used and the considerations taken into account before purchasing them.

2.4.1 Virtual reality headset

SO | *OM*

Choosing the right VR headset was one of the initial steps to ensure the development process could proceed without delays. Several popular VR headsets and glasses were considered, focusing on both gaming and industrial applications. Industrial headsets, while robust, were quite expensive (over 15,000 NOK). Our customer requested two options.

We were recommended the Meta Quest 3 and the *Microsoft HoloLens*. There is a lot of open-source documentation available for application development with these two headsets. Both are supported by platforms such as Unity and Unreal Engine, which offer extensive documentation and community support. A comparative analysis was conducted using resources like vr-compare.com, which was shared with the customer. Ultimately, the Meta Quest 3 was selected based on the customer's evaluation, offering a suitable balance of cost and functionality for our project needs. The headset was purchased before Christmas of 2023 to be ready for the project start and will continue to be used for further development by the customer.

2.4.2 Development platform **OM** | *AD*

Kongsberg Maritime provided the group with hardware previously utilized by earlier bachelor groups. It was emphasized that any hardware bought would benefit future bachelor groups. The group knew running AI models is a demanding task, which requires hardware with enough resources. The required resources vary depending on the nature of the AI models and their required processing speed. There were no requirements for local execution on the robot, but this was assumed.

We received a Raspberry Pi (RPi) 2B and a Jetson Nano 4GB B01. The RPi 2B, introduced in 2015, features specifications that are outdated by current standards, with a Central processing unit (CPU) clocked at 900MHz and 1GB of Random Access Memory (RAM) [55]. In contrast, the Jetson Nano B01 4GB appears to have been released in 2020, with a CPU operating at 1.43GHz, 4GB of RAM, and a 128-core Maxwell Graphics processing unit (GPU) [29]. It runs Ubuntu 18.04 Long term support (LTS), released in 2018 [56], with non-upgradable soldered memory.

Given the group's engagement with cutting-edge technologies and a newly launched VR headset, we sought the latest hardware for compatibility and potential future development.

After careful consideration, the RPi 5 8GB was opted for, featuring a 2.4GHz CPU and 8GB of RAM, said to have "2–3× the speed of the previous generation" [55]. Two units were bought at a cost of 1,224 NOK each, inclusive of transportation [57]. The decision to purchase two units was motivated by the need for redundancy in case of damage and to be able to develop in parallel.

2.5 Software tools **OM** | *AD*

This section lists generic or more essential technologies used by the group for developing software.

2.5.1 Git & GitHub **OM** | *AD*

Git was chosen as our version control program, as this was what the group was most familiar with. GitHub is where all of Kromium's code is hosted. A GitHub organization was created, where all group members were added. Only members of this organization can edit and manage the repositories without having to ask for approval. Smaller and less relevant repositories for the thesis were kept private, which means only members of the organization can access and view them. The more essential and relevant parts of the codebase are publicly available on the organization's GitHub page [58].

2.5.2 ROS 2 **OM** | *AD*

Robot Operating System (ROS) 2 "is a set of software libraries and tools for building robot applications" [59]. ROS uses nodes, which is an intuitive way to structure the software. This works well when collaborating, as one person can be responsible for one node and only know the external interfaces, while another person works on a different node. By defining the interfaces beforehand, development can be done in parallel.
There are multiple ROS distributions available, such as Foxy Fitzroy (foxy), Rolling Ridley (rolling), Humble Hawksbill (humble) and Iron Irwini (iron). humble was the chosen distribution as it has LTS [60] and the most distant End-of-life (EOL) date, as well as it

is stable. Additionally, a member of the group had previous experience with this distribution.

2.5.3 Operating system **OM** | *AD*

The RPi needs to run an Operating System (OS) for it to be able to run software. The RPi 5 has a 64-bit Arm CPU [2] and can run both arm32 and arm64 architectures [32]. The two most popular RPi OSs are Raspberry Pi OS (RPiOS) and Ubuntu. For our group, it is important that they are popular, as this means more resources and information are available online in case of problems.

ROS 2 is supported on RPiOS, but "Raspberry Pi OS is based on Debian which receives Tier 3 support, but it can run Ubuntu docker containers for Tier 1 support. [...] Tier 1 support means distribution-specific packages and binary archives are available, while Tier 3 requires the user to compile ROS 2 from source" [59]. Compiling ROS 2 from source involves many steps, and due to the RPi 5 being newly released, encountering issues are likely. Additionally, there is no official guide available for compiling ROS 2 from source on the RPi 5. Although Ubuntu has Tier 3 support, previous experience has shown that camera support might be limited, especially when using the Peripheral Component Interconnect Express (PCIe) connector. It was uncertain whether the PCIe connector would need to be used, but having it available was considered optimal. Therefore, the chosen OS was RPiOS 64-bit, as it can run ROS 2 through Docker containers.

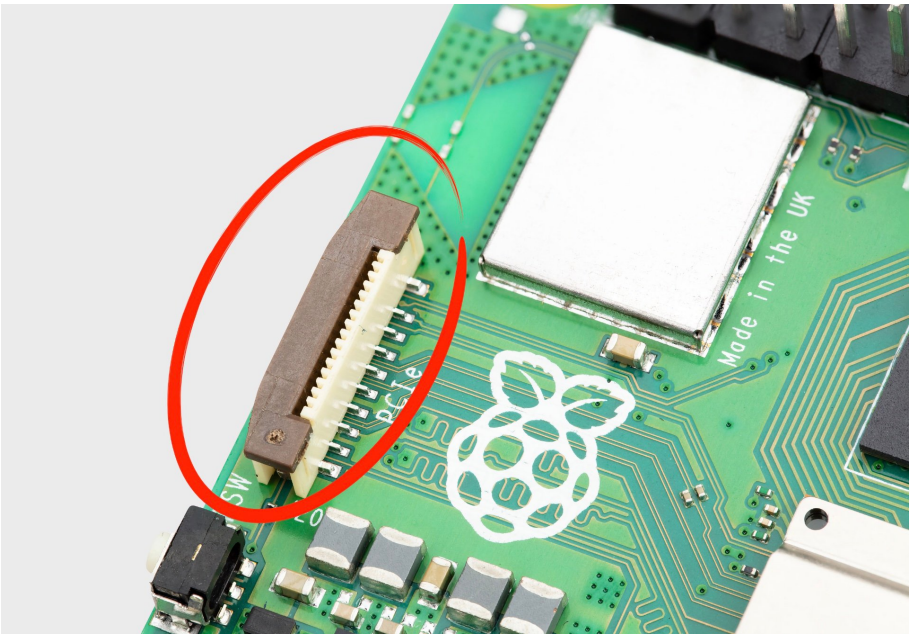


Figure 2.2: "Raspberry Pi connector for PCIe" [2]

2.5.4 Docker **OM** | *AD*

Docker is a platform which enables the containerization of code and environments. "Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run" [61].

Docker was used for running ROS 2 on the Raspberry Pi OS 64-bit installation, as there are no official binary archives available for this OS [59]. Docker is also supported for RPiOS 32-bit [62], but there was uncertainty surrounding ROS 2 Docker images for this architecture. Additionally, Docker can run on Ubuntu.

2.5.5 Unity

OM | *AD*

Developing for the Meta Quest 3 there are two viable options, Unreal Engine or Unity. Unreal Engine uses C++, while Unity uses C# [63], none of them are completely free and open-source software (FOSS). Unity was chosen as there were more resources and documentation available online, and it seemed to have more VR functionalities than Unreal Engine.

2.5.6 Programming languages

OM | *AD*

C# and Python were chosen as the primary languages for this project. No one in the group had previous experience with C#, as this is the language Unity uses [63]. Nevertheless, the software members of the group had previous experience with C++, so it was thought that this transition to C# would not be too challenging.
Python, being a high-level language was selected as the primary language outside of the Unity application development. The reasoning for this was due to the vast amount of tools already available, no requirements for speed, shorter development time, and previous experience with the language.

2.5.7 Google Colab

AD | *OM*

Google Colab was used to make our own custom Machine Learning (ML) model. It is a free, cloud-based Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs. Colab is especially well suited to ML [64]. As it provides free access to a GPU, our group decided to use it to reduce the training time.

2.5.8 Integrated development environment

OM | *AD*

Unity Hub was used for developing the VR application. Unity Hub allows users to "create and open projects in the Unity Editor, access resources..." [65]. Additionally, Visual Studio Code (VSC) was used for writing C# code.

VSC was chosen as our main Integrated Development Environment (IDE) for several reasons. It supports numerous extensions, including *flake8*, *black*, and GitHub CoPilot (see 2.5.13). Another significant factor was its remote Secure Shell (SSH) integration. This feature simplifies programming for the development platform, allowing code writing, running, and command execution on the device itself remotely using a local instance of VSC.

2.5.9 Doxygen

OM | *AD*

For documenting the C# code we used Doxygen. Doxygen can build diagrams, HTML and Portable Document Format (PDF) documentation from code comments and docstrings.

2.5.10 Sphinx **OM** | *AD*

Sphinx, much like Doxygen, can build HTML and PDF documentation from code comments. Sphinx was used for all Python code documentation using Google's Python standard with type annotations [66]. Built-in Sphinx extensions such as autodoc [67] and Napoleon [68] made it possible to automatically build documentation using these docstrings. The HTML documentation shown on our website was built using Pradyun Gedam's *furo* theme [69].

2.5.11 GitHub Workflows & Pages **OM** | *AD*

GitHub Workflows can be set up to run every time code is changed or a new release is created. These workflows can be configured to run custom code on GitHub's servers to perform actions. Examples of actions could be running tests, checking if formatting is correct and building code for various platforms. Workflows run in their virtual machine instance, so needed packages and dependencies need to be defined [70]. There is one workflow on both the "robot" and "transfer-learning-training" repositories, which is configured to build the Sphinx code documentation. The workflow runs every time a file gets changed on the "main" branch. When the documentation gets built, it is deployed to GitHub Pages.

GitHub Pages are static websites hosted by GitHub for free, typically available under the github.io domain. GitHub Pages have been used for hosting the code documentation for the "robot" and "transfer-learning-training" repositories. The documentation is kept up-to-date using a GitHub Workflow. Any time a file is changed, the documentation gets rebuilt and deployed to the static website. Both of these documentation pages are linked to on our website.

2.5.12 Python standards **OM** | *AD*

A list of packages was used when developing Python code. These were chosen due to prior experience, which accelerated development and reduced errors. For formatting, the group used *black*, which was configured in VSC to run automatically whenever a file was saved. *black* ensures consistent code appearance throughout the file, enhancing readability.

For linting, the group chose *flake8*. A linter performs static checks for code standards, such as the correct order of imported packages, proper use of operators, consistent quotation style, and identification of unused variables. These checks help users write consistent code and catch issues early.

The group selected *unittest* as the Python package for unit testing. It was known from previous experience that unit testing would be beneficial for validating, testing, and writing code. Unit tests enabled the group to test parts of the code without having to physically test the system each time, as explained in 7.5.

2.5.13 GitHub CoPilot **OM** | *AD*

GitHub CoPilot is an AI-powered application developed by GitHub, which can autocomplete and predict more than a typical language server. CoPilot is available as a VSC extension and can automatically complete text, common code snippets, and scripts [71].

CoPilot has significantly sped up the process of writing code documentation and unit tests, in addition to generating code for graph plotting in this thesis.

2.5.14 Miscellaneous tools **OM** | *AD*

Windows Subsystem for Linux (WSL) was used for SSH access to the RPi and committing local changes or files to GitHub. FileZilla was utilized for transferring multiple files between the RPis and our local machines.

2.6 Project development process **AD, SO** | *OM*

The development process of the project across all disciplines was an iterative incremental process. The process is based on two methods, iterative and incremental. The iterative method allows us to develop the system through repeated cycles, while the incremental method divides the system into smaller portions at a time. This approach enabled us to develop smaller parts of the system through repeated cycles allowing us to gain new knowledge, experiment with AI and VR technology, and demonstrate and validate our progress along the way.

2.6.1 GitHub workflow **SO, OM** | *AD*

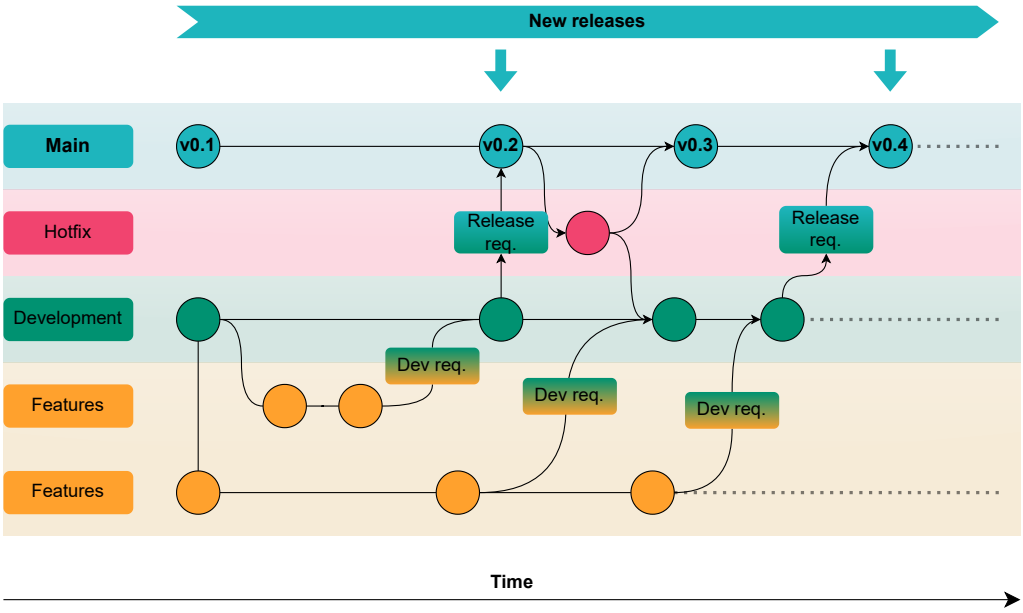


Figure 2.3: Kromium's GitHub workflow

The workflow consists of branches that should be used when adding new functionalities. However, hotfixes are allowed directly on the main branch to quickly address bugs and issues, making the workflow more agile. The diagram emphasizes the continuous nature of software development. New features are constantly added, developed, and released.

2.7 Mechanical development tools **AEH** | *HB*

During development, the goal was always to have a usable robot for testing and demonstrations. For the development and testing cycles, we utilized rapid prototyping and additive manufacturing.

2.7.1 Computer-aided design **AEH** | *HB*

All the computer-aided design parts are produced in the program *SOLIDWORKS* before being tested or produced.

2.7.2 SolidWorks simulation **AEH** | *HB*

SolidWorks simulation uses the finite element method for digital testing of 3D models to simulate if a part is strong enough to withstand different loads.

2.7.3 UltiMakrer Cura **AEH** | *HB*

The preparation for 3D-printing parts is done on the free slicing program UltiMaker Cura. The program determines all the factors that go into producing the part. After plotting the necessary values you get what is called a G-code which is all the information the 3D printer needs to produce the part layer by layer.

2.8 Electronics **HB** | *AEH*

Because the team does not have an electrical engineer, the responsibility for the electronics will be divided between the team.

2.8.1 Tools used for electrical development **HB** | *AD*

The tools used for the electrical development are Excel and "Circuit Diagram - A Circuit Diagram Maker". Excel was used to log results for the test of the battery cells, listing which battery cells can be used for the battery, and calculating a rough estimate for the battery's operational time. "Circuit Diagram - A Circuit Diagram Maker" is a webpage where it is possible to create and download circuit diagrams for free. The URL for this webpage is circuit-diagram.org.

2.9 Project model **SO** | *AD*

Our project involves the research and development of a product integrating technologies such as VR and AI, focusing on their application in remotely controlling a robot. We recognized the need to learn and evolve through multiple iterations, enabling us to achieve a better result. This understanding led us to choose an agile methodology combined with an iterative and incremental development approach.

Our project model was a simplified scrum. "Scrum is a framework for organizing and managing work. The Scrum framework is based on a set of values, principles, and practices that provide the foundation to which your organization will add its unique implementation of relevant engineering practices and your specific approaches for realizing the Scrum practices. The result will be a version of Scrum that is uniquely yours" [72, p. 123]. We practised the following essentials of scrum:

- **Sprint**: A time-boxed period during which specific work has to be completed and made ready for review.
- **Sprint Planning**: Meetings where the team plans the work to be completed during the sprint.

- **Sprint Reviews**: Meetings at the end of each sprint to demonstrate what has been accomplished.
- **Daily Stand-up Meeting**: Short daily meetings to discuss progress and obstacles.
- **Retrospective**: Meetings to reflect on the sprint and identify improvements for future sprints.
- **Backlog**: A prioritized list of tasks and requirements for the project.

A typical week at Kromium is illustrated in Figure 2.4 and Figure 2.5, showcasing typical weeks before and after Easter respectively.

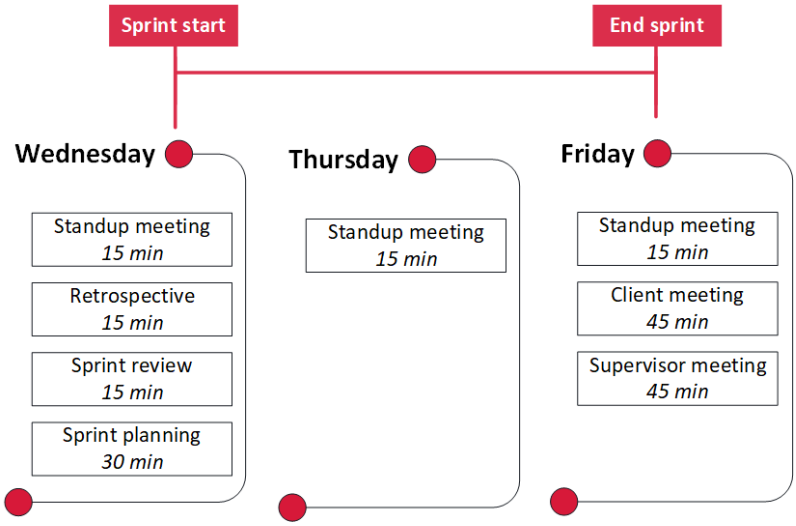


Figure 2.4: Typical week at Kromium **before** Easter

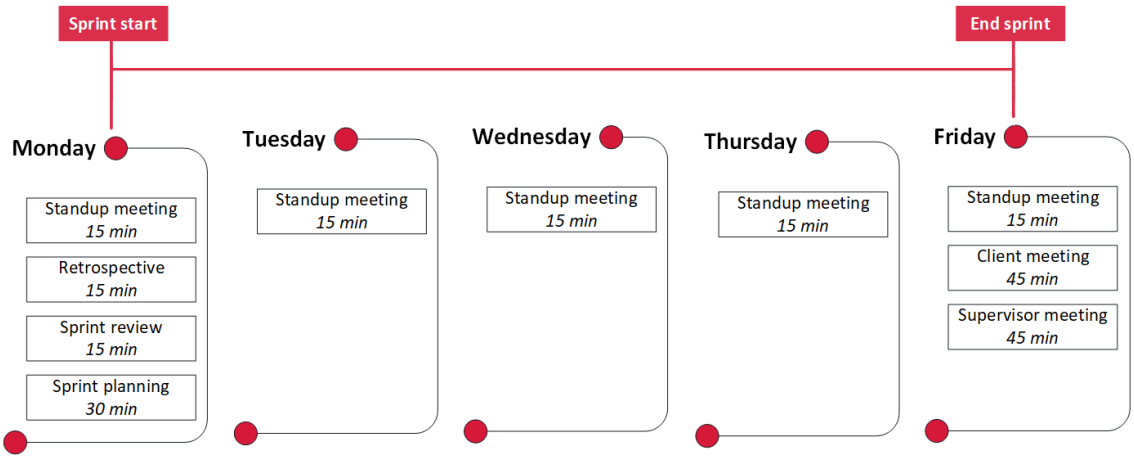


Figure 2.5: Typical week at Kromium **after** Easter

2.10 Risk Assessment: Rapid Risk Ranking (RRR) **HB** | *OM*

Rapid Risk Ranking (RRR) is a type of risk assessment. This method is used to predetermine possible hazardous events that may occur, so mitigating measures can be deployed to reduce the actual risk. Maybe even remove the risk completely.

"RRR is a simpler approach than a full Quantitive Risk Analysis (QRA). The reason for using RRR is that the information available at this stage is not detailed enough for a full QRA. Rapid Risk Ranking (RRR) with the use of risk matrixes will yield the desired results for evaluation of the different concepts." [73, p. 3]

To perform the risk analysis, it was decided that the group should use the same methodology which Norsk Hydro ASA and DNV used when they created an RRR for hydrogen filling stations in 2002. It consists of five different tables where the values and definitions are dependent on the project itself. This means that the tables the group uses must be redefined and rewritten so they will fit the project. The reason the group chose the same methodology as Norsk Hydro ASA and DNV, was because two of the group participants had already used this methodology in a couple of subjects during their bachelor's degree.

The different tables are:

- Consequence severity
- Probability levels
- Risk matrix
- Risk levels
- Hazardous events
- Risk analysis form

When all of the tables have been rewritten such that, they fit the project. The different hazardous events, their probability of occurring, and what degree of hazard they have can now be found. When all of this is done, the main reporting form can be filled in, and the decision of what risk the different hazardous events have can be made.

2.10.1 Consequence severity **HB** | *OM*

The consequence severity table in a risk analysis, describes the different areas that can be affected by an event. Each area such as "Environment" or "People", has a different description for each severity level. This is because the severity levels can change depending on what area is looked at. The different areas located in the risk analysis are the developers, the environment, Kongsberg Maritime, material values, and people. The reason the project group chose to have these areas in the risk analysis, is based on the group participants' opinions on what areas need to be a part of the risk assessment.

2.10.2 Probability levels **HB** | *OM*

A probability level table in a risk analysis consists of different levels of probabilities for an unwanted event to occur. In this risk analysis, the levels are arranged from A to E, where A represents the lowest probability, and E represents the highest probability. The amount of levels is based on how accurate the risk analysis has to be, considering the project. In this risk analysis, it was chosen to use five different levels, since this gives an ability to be quite accurate, and the table from Norsk Hydro and DNV had this setup as well.

2.10.3 Risk Matrix **HB** | *OM*

A risk matrix uses the consequence severity table and the probability level table to make a new table. A risk matrix uses these two tables to create a cross-product that represents a risk. In this case, the risks have been divided into three different sections, LOW, MEDIUM, and HIGH. There is no problem with dividing the risks into more sections, but for this project, it is not necessary.

Explanation of the different sections in the Risk Matrix:

- **Low**: Mitigating measures don't need to be considered
- **Medium**: Mitigating measures should be considered
- **High**: A mitigating measure must be carried out

It is important to note that the different levels may vary from project to project, but they are often described as something like this.

2.10.4 Risk levels **HB** | *OM*

The risk level table in a risk analysis describes the different risk levels in a risk matrix. In this risk matrix the first letter of the risk section (L, M or H) and a matching colour to keep the table clear and easy to use. The risk table that is used first describes the letter for each risk section, then it describes the risk level itself. If the risk level is any other than LOW, it is also described how important it is for one or more mitigating measures to be carried out.

2.10.5 Hazardous events **HB** | *AD*

The table "Hazardous event allocated to our stakeholders" is a list of unwanted events that are believed to possibly occur during the project. These events can be ranked between very common too highly unlikely to occur. But every event that is listed on the table is thought to be reasonable.

To begin with, it was brainstormed a lot of different unwanted events. Then the different events were divided between the different areas that could be affected. It is important to note that one unwanted event can affect several areas. Therefore, the same event can be found in several areas. To make sure that the events will not be misinterpreted, and that the people reading the report have a better understanding of the different hazards, the table with the different hazardous events has a description section to better describe each event. Even if some of the events are quite self-explanatory, there is a small description of the event.

2.10.6 Risk analysis form **HB** | *AD*

The risk analysis form is the last step before the risk analysis can be completed. In this step, everything is put together, and the final result becomes visible and easy to understand. It was decided to divide the table into two sections, to make it easier to read. This was because the writing became too small to read if the whole table was on one page. In addition to the sections area/ID, hazard, and cause, the first table consists of the consequence severity and the risk before the mitigating measures. The second part of the table consists of the mitigating measures, the consequence, the probability, and the risk after

the mitigating measures have been applied, in addition to the area/ID, hazard, and cause.

To use and find the results of the risk analysis, the consequence of the different hazards is defined for each area. If the hazard does not affect the area, the consequence box for this area has a dash placed in it. Then the probability of the hazard to occur is defined. With these two values, the risk for the hazardous event can now be found. If the mitigating measures are considered, the consequence and the probability often change. This should give a lower risk than the original risk. If the risk is still at the same level or not low enough, other mitigating measures should be considered in addition to the original mitigating measures.

3 Design overview

This section provides a high-level overview of the Kromium system's design. It outlines the functionalities and features of the robot from a user perspective. We describe what users can do with the application, how they interact with the system, and the useful information the application provides to them. Detailed implementation specifics for each feature are covered in the following section.

3.1 Project design overview

SO, AD | *OM*

Kromium's system given the requirements and purpose of the system is to enhance offshore operations using VR, AI and robotics. As illustrated in Figure 3.1, Kromium's system is designed to facilitate human-robot interaction through virtual reality. The group uses the Meta Quest 3 as VR-headset to create an immersive experience, which allows the operator to control and monitor the robot.

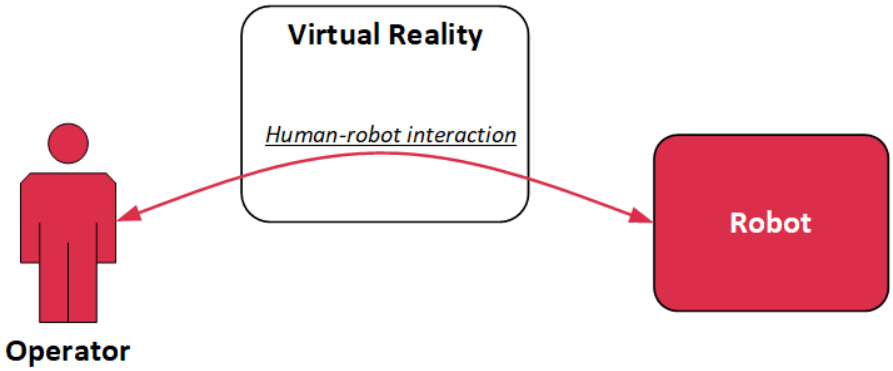


Figure 3.1: Human-robot interaction through virtual reality

For an operator to remotely control a robot through a VR application and perform tasks, they need to interact with the system and receive feedback, such as video feeds of the robot's environment and the robot's status. Figure 3.2 illustrates this interaction. The red path shows how the operator uses the VR application to interact with virtual objects and send commands to the robot, such as driving forward, driving backward, or taking a picture. The blue path represents the communication from the robot to the operator, providing visual information such as video feeds and status updates (e.g., connection status, speed, and latency). This system ensures an intuitive and natural way for the operator to remotely control the robot and obtain environmental information through the robot's cameras, facilitating comfortable and efficient task performance in offshore environments.

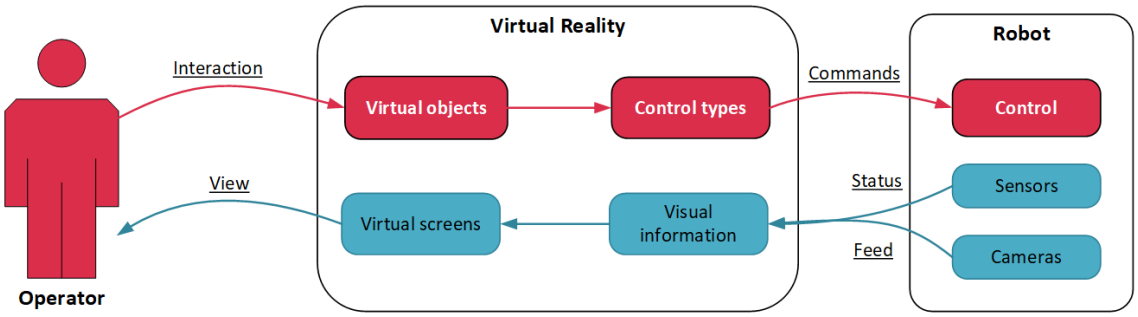


Figure 3.2: Human-robot interaction with controls

The operator interacts with the robot by manipulating virtual objects in the VR application, which can be controlled in various ways. For example, the user can perform hand movements within a virtual area to drive the robot to a desired destination. The operator sees what the robot sees on a virtual screen that displays the visual information from the robot's cameras, providing an immersive interaction experience.

3.2 System functionalities

SO | *AD*

The system functionalities are divided into two categories: "Interaction Features" and "Operator Information Features".
Interaction Features describe the actions the operator can perform with the application, including interacting with virtual objects to control the robot and perform tasks in the robot's environment.
Operator Information Features provide the operator with useful information, such as video feeds from the robot's cameras and status updates, to aid in task performance.
In this subsection, we will provide a graphical overview of these functionalities before delving into the details in the following sections.

3.2.1 VR application features

SO, AD | *OM*

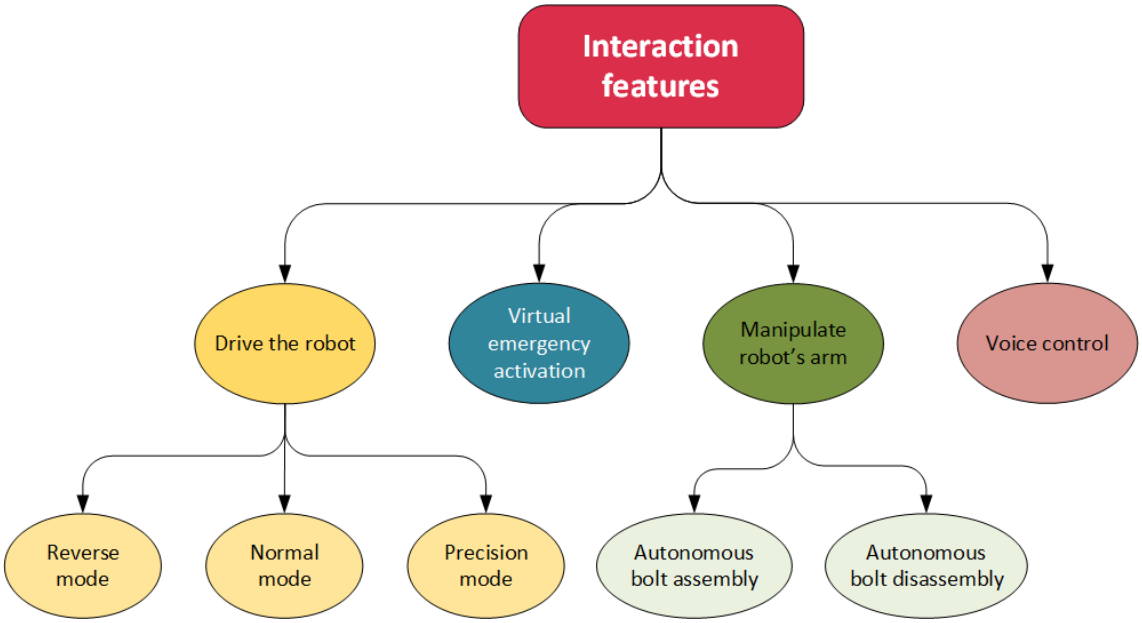


Figure 3.3: Interaction Features for Robot Control from the operator

Figure 3.3 illustrates the different ways the operator can interact with the robot using the VR application. These interaction features can be broadly categorized into the following:

- **Drive the robot:** This feature allows the operator to drive the robot with hand movements interacting with virtual objects, like driving a car. This feature includes three driving modes:
  - **Normal mode:** Standard movement, allowing the user to drive in all directions with speed control up to the robot's full capability.
  - **Precision mode:** Fine-grained movement for delicate manoeuvres, such as driving to a corner.

  - **Reverse mode:** Allows the operator to drive backward.

- **Virtual emergency activation:** In critical situations, the operator can put the robot into an emergency mode, requiring a physical inspection by a technician. This is activated through a virtual emergency button.
- **Manipulate the robot's arm:** This feature allows the operator to control the robot's 6 DOF arm, enabling it to perform tasks such as picking up objects and assembling components. It can also be used autonomously for tasks like assembling and disassembling bolts.
- **Voice control:** This feature allows the operator to issue commands to the robot using voice interactions. ML models are used to recognize voice commands, which can be particularly useful when the operator needs to keep their hands free.

3.2.2 Operator Information Features **SO, AD** | *OM*

In 3.2.1, the interaction features from the operator's side were discussed. In this section, the features provided to the operator are discussed. Figure 3.4 shows all the features and information that Kromium's system provides to the operator through the VR application. These features and functionalities provide useful and important information to the user operating the robot in the robot's environment.

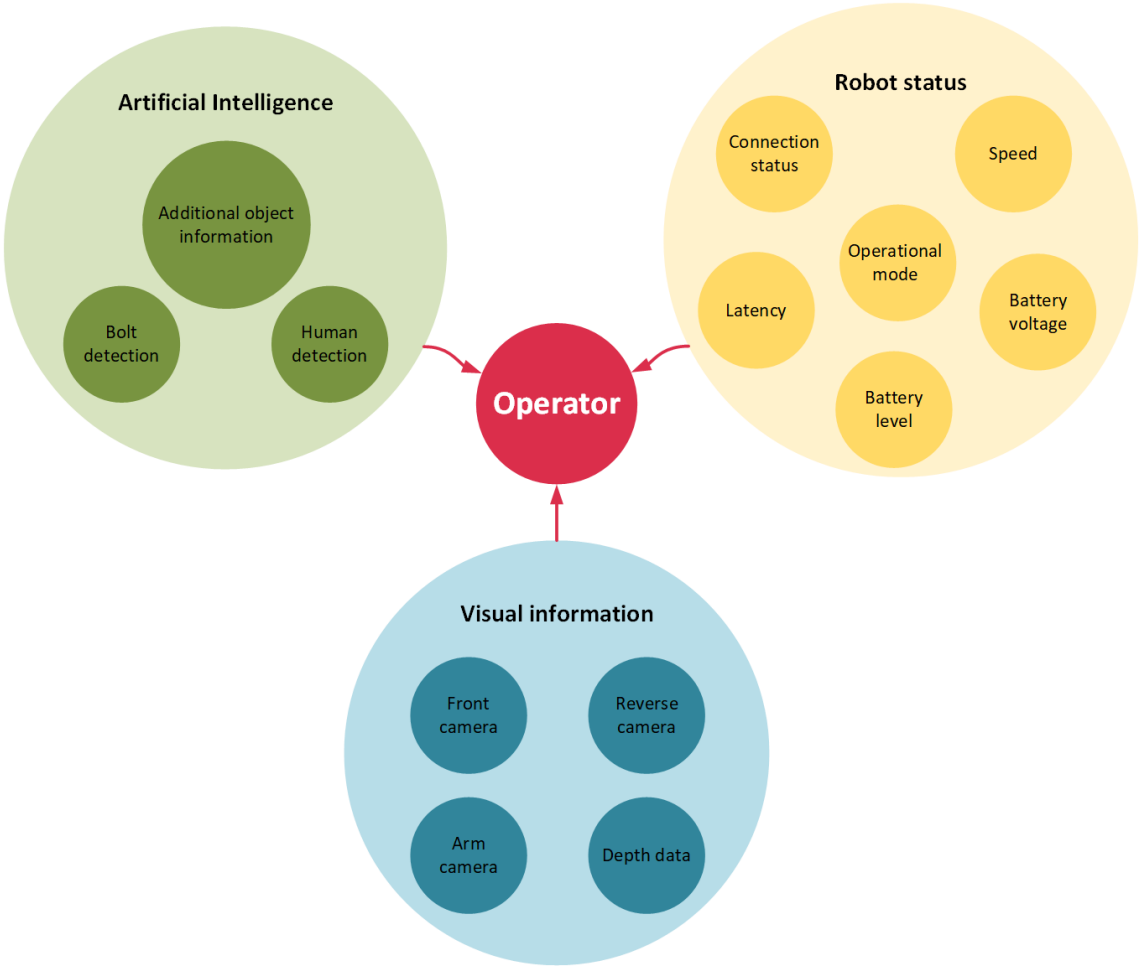


Figure 3.4: Features provided to the operator

The features are divided into three main channels:

1. **AI:** The AI models used in the project can detect two things: humans and bolts. This is sent to the operator along with additional information about these objects.
2. **Robot status:** As the robot is controlled remotely, it is useful to know its different characteristics. This is the connection status between the VR application and the robot, the robot speed, the current operational mode, the battery level and voltage, and lastly the latency between the VR application sending a command and the robot receiving it.
3. **Visual information:** This includes the video feeds from the robot's front and reverse cameras, as well as the arm camera. Additionally, depth data is transmitted, offering spatial awareness.

3.3 Hardware design

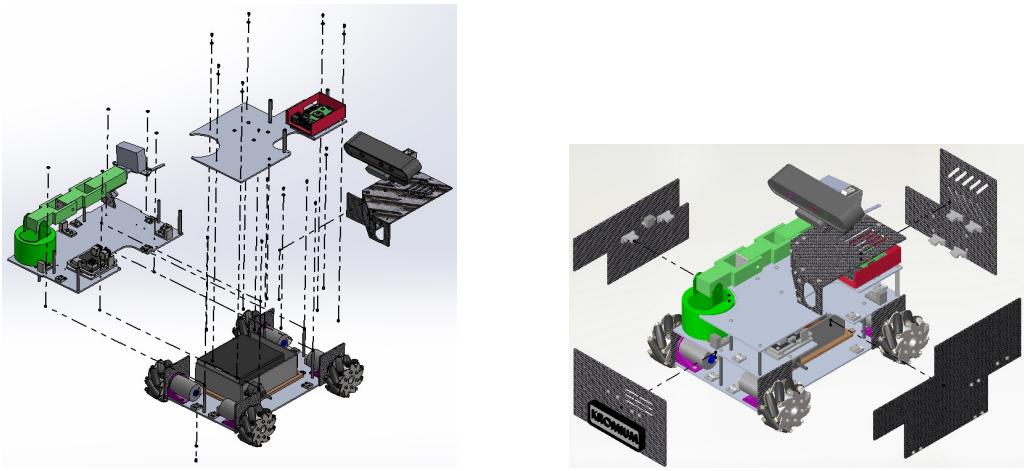
This chapter discusses the different specifications of the robot, which physical functionalities the robot has, and how to use these functionalities.

3.3.1 Robot specifications **HB, AEH** | *OM*

- **Speed**: The car has a maximum speed of 80 cm/s (see O.9)
- **Weight**: The car has a weight of 5.4 kg (see 7.8.5)
- **Number of battery packs**: The robot has two compatible battery packs, one of them with changeable cells.
- **KROMIUM battery pack**: The KROMIUM battery pack can power the robot for approximately 2 hours and 45 minutes, and 2 hours and 25 minutes, depending on which battery cluster is used. For the calculations see appendix Q.2. It is important to note that this is just an approximation, and the time might deviate from this.
- **Yahboom battery pack**: The Yahboom battery pack can power the robot for several hours. This is known because the Yahboom battery pack has a capacity of 9600 mAh, which is more than double the KROMIUM custom battery pack.
- **Degrees of freedom car**: The car has three degrees of freedom, and can move in every direction in a two-dimensional plane while facing the same way.
- **Degrees of freedom arm**: The robot arm has six degrees of freedom (see Q.1)
- **Strength of the arm**: The arm can lift 200 grams when fully stretched out and up to 500 grams when clamping and handling otherwise. [74]

3.3.2 Structure of the robot **HB** | *AEH*

The robot has a modular design. In this case, modular design means that the majority of the parts are easily accessible and can be changed if necessary. To make the robot as modular as possible, the main walls of the robot can be removed by simply lifting them. This way every component on the robot is possible to reach without too much work. The main electronics are also secured by brackets that slide in and out of the robot by rails. The robot consists of three different floors, where every floor stores one or more components used to control the robot.



(a) Robot internal structure exploded view **(b)** Robot external exploded view

Figure 3.5: Robot exploded view. More detailed version T.7.

3.3.3 Building the robot

AEH | *HB*

If the need to access, modify or build part of the robot arises we have made assembly drawings for each subassembly, floor assembly, wall assembly and main assembly. How to assemble the subassemblies can be found in appendix T.4, the floor assemblies in appendix T.5, the wall assemblies in appendix T.6 and the main assemblies in appendix T.7.

3.3.4 Easy accessible power switch

AEH | *HB*

A visible and easy-to-access power switch is on the roof at the back of the robot. This switch is serially wired directly between the battery and the expansion board. You can shut off the system by flipping the switch mounted directly on the expansion board or simply by using the rocker switch on the roof. Since it is wired serially between the battery and the expansion board, which supplies power to the rest of the system, this power switch also functions as an easily accessible emergency shut-off.

3.3.5 Easily accessible electronics

HB | *AEH*

One of the main reasons for the robot to have easily detachable walls is that the electronic components should be easily accessible. In addition to this, the RPi, expansion board and the USB hub all have mounts that make them easily detachable by sliding them in and out of the robot. The battery is also easily accessible. The battery is placed in a drawer which can be dragged out of the robot. Then the battery is lifted out of the drawer because the connector between the robot and the battery is a magnet connector.

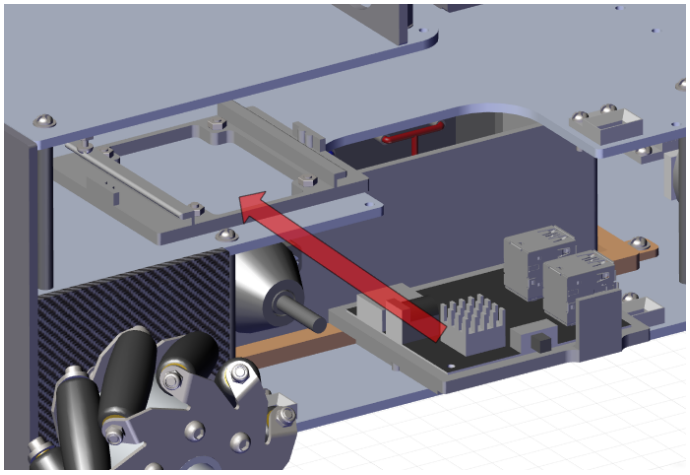


Figure 3.6: Changing electronics

3.3.6 How to change the battery **HB** | *AEH*

To change the battery, make sure that the main power switch is in the off position. If it is not, flick it over so it is. Then, remove the right side wall. Grab the drawer on the bottom floor and pull it out carefully. With the drawer out, grab the battery pack and remove it from the magnet, this might require that the battery is angled gradually while disconnecting. When the battery is removed, grab the new battery, and insert it.

When inserting the new battery, it is easier to tilt the battery a bit, make the connector between the battery and the robot touch and then level off the battery until the pins in the drawer are inserted into the holes in the battery. Make sure that the battery is properly attached. Then slide the drawer in carefully. Attach the right side wall, and turn the power on.

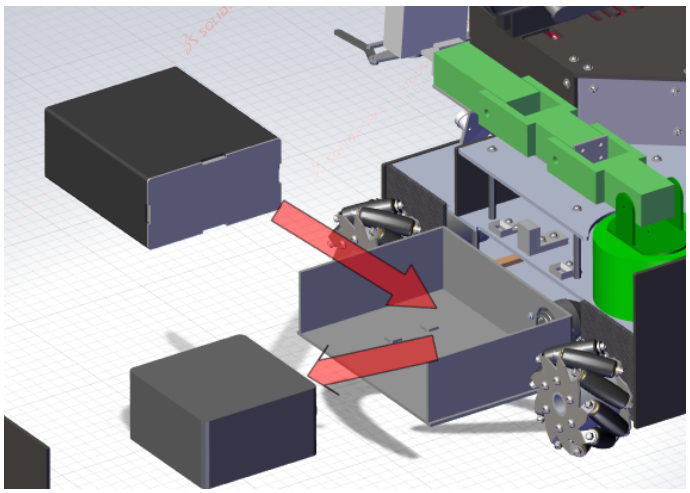


Figure 3.7: Changing battery

3.3.7 Turning on the power **HB** | *AEH*

To turn on the power on the robot, some steps need to be followed. The first step is to make sure that the battery inside the robot has enough charge for the robot to be operational for the desired time or that the battery is fully charged. The second step is to make sure the connector between the battery and the robot is properly attached. Then, make sure that the on/off switch on the expansion board is in the on position. Then, make sure that the on/off

switch on the USB hub is also in the on position. When all of these steps are checked, the main power switch can be activated.

3.3.8 Wheels **HB** | *AEH*

The wheels used on the robot car are omnidirectional. These wheels allow the car to move more freely. The steering mechanism of the car is the wheels themselves in addition to one motor per wheel. What makes this possible is that the wheels are covered in small rollers which has a 45-degree angle to them. The rollers are attached in such a way that they can create a cross in the middle of the car if a line is drawn normally too the rollers' direction. The disadvantage of these wheels is that they create a lot of vibrations during driving and each wheel needs at least one motor each. The advantages on the other hand are that the car can drive in every direction in a two-dimensional plane while it is facing the same direction, and it can also turn around its own axis. This is possible because the wheels have the rollers and each wheel has its own motor. The car will either turn or move in a specific direction depending on which wheel is turning and what direction they are turning. Because of the rollers on the wheels, this type of wheel has one more degree of freedom than normal wheels, which have two degrees of freedom [75, p. 2]. This extra degree of freedom gives the car the ability to move the way it does. Below is an illustration of how the car will move depending on which direction the different wheels turn, and where the centre point of the turn will be.

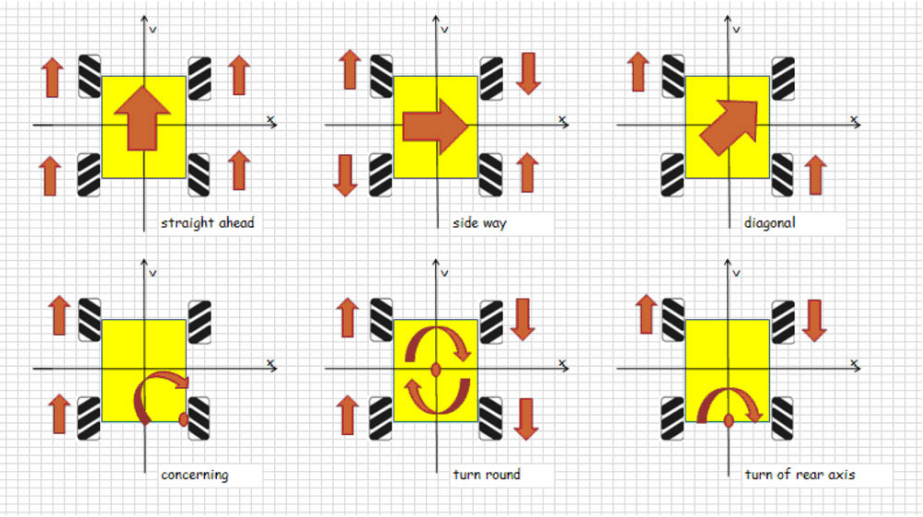


Figure 3.8: Some of the robot movement options. Source: [3]

The reason omnidirectional wheels were used is because the car had them when the project group received it. At the same time, the car should be able to operate in tight spaces where traditional steering could be difficult. At the same time, the budget could be used on other things than wheels.

3.4 Electrical design

3.4.1 Battery design **HB** | *OM*

The battery design process had several design iterations before the battery could be put to use. The reason for this was that the earliest designs consisted of design flaws, that were discovered either during the design phase or the testing phase.

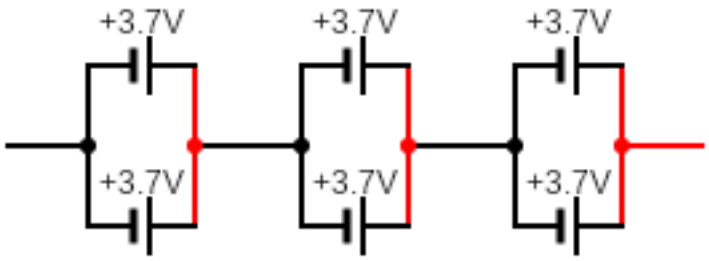


Figure 3.9: How a 2P3S configuration looks like on paper

The battery has the configuration 2P3S, which means that there are three sets of battery cells in pairs, connected in series, see figure 3.9 for reference. Because the cells have a nominal voltage of 3.7V, the nominal voltage of the battery will be 11.1V. However, because the maximum voltage of the battery cells is 4.20V, the maximum voltage of the battery is 12.60V. Because the battery ended up not having a Battery management system (BMS), the capacity of the battery is equal to the capacity of the cell group with the least amount of capacity. If the battery had a BMS designed for a 2P3S configuration, the battery capacity would have been the average of the capacity of the cell pairs. This is because one of the jobs a BMS has is to make sure that the voltage drop is equal in every cell. The capacity of each pair of cells can be calculated like this.

$$mAh_1 + mAh_2 = mAh_{pair} \tag{3.1}$$

This means that the lowest theoretical capacity of the battery can be 4085 mAh for the battery with "Battery cluster 1" and the lowest capacity of the battery with "Battery cluster 2" is 4024 mAh. See 7.1 for the battery cell capacity. Because the cells are not connected to a BMS, the same amount of electricity is being pulled from each cell, this is why the capacity of the battery is the capacity of the pair with the least amount of mAh. The reason the cells are divided into different clusters is to prevent the cells from completely discharging, which can happen if cells that have a very different capacity are used in a battery without a BMS. The other reason that the battery cells are divided into two battery clusters, is to make sure that the robot can always be operational (except when switching out the battery). This way two of the batteries can charge while one is being used. For the two different clusters of battery cells, the twelve best-performing battery cells were chosen, where the top six were in cluster number one, and the ones that came in seventh to twelfth place were in cluster number two. See the tables 3.1 below for the cells used in each cluster. For all of the results from the battery cell test see table 7.1.

Battery cluster 1	
Cell №:	Capacity
8	2060mAh
10	2047mAh
12	2091mAh
13	2038mAh
15	2086mAh
16	2082mAh
Avg:	2067mAh

Battery cluster 2	
Cell №:	Capacity
4	2031mAh
6	2031mAh
9	2023mAh
17	2028mAh
18	2005mAh
21	2019mAh
Avg:	2023mAh

Table 3.1: The two groups of battery cell clusters used

4 Software Implementation

In this section, we delve into the design and implementation aspects of all software components and their functionalities. We begin by describing the setup of the development environment for each software component. Following this, we provide detailed sections on the VR application, the robot, the robotic arm and the AI object detection. Due to the extensive implementation process involved with the robotic arm, we have dedicated a separate subsection to it for the sake of readability. The final section discusses the additional work outlined in 1.5.2, which extends beyond the original project requirements. For clarity and coherence, this section is organized to reflect the sequence of functionalities as they are designed and implemented.

4.1 Software setup **OM** | *AD*

This section explains the steps taken to set up and configure the software development environment.

4.1.1 Network configuration **OM** | *AD*

A local network was configured to facilitate communication between the devices in our system. This network setup enables SSH access, which is blocked on the "eduroam" network. SSH access is crucial for an efficient and streamlined development process. This allowed us to access the RPi remotely. This eliminates the need for physical interaction with the devices, saving significant time and effort. Additionally, static Internet Protocol (IP) addresses were assigned to both RPis, ensuring consistent and straightforward SSH access by maintaining constant IP addresses.

4.1.2 VR application environment **SO** | *AD*

The setup for the VR application environment involved configuring the VR headset, the Unity editor, and the integration software between the VR headset and Unity for developing applications. Given the complexity and numerous steps involved in this process, detailed instructions are not included in this report. However, comprehensive step-by-step guides can be found in the official documentation of the Meta Quest 3 and Unity on their respective websites.
For detailed setup instructions, refer to the following resources:

- **Set up development environment and headset, [76]**
- **Get started with Meta Quest 3 development in Unity [77]**

These resources provide thorough guidance on setting up the development environment for VR applications.

4.1.3 Robot & AI environment **OM** | *AD*

Installing Raspberry Pi OS The installation of RPiOS (64-bit) was performed using the Raspberry Pi Imager [57] and an SD card converter to transfer the ISO image onto the microSD card. This microSD card functions as the hard drive for the RPi.

Configuring the Raspberry Pi After installing and booting the RPi, SSH access was enabled to facilitate remote access, allowing code editing on the RPi using VSC's remote SSH feature. Additionally, a symlink was created for the expansion board, as its name would change randomly (see O.7).

Installing Docker Docker was installed on the RPi using the script shown in O.3. By default, this requires the user to use `sudo` every time the Docker command is executed. Configuring permissions for the Docker group allows this requirement to be bypassed (see O.4).

Creating the Docker Image To run a Docker container, an image is needed. This image should include installation instructions, other dependent Docker images, necessary packages, user permissions, a default working directory, and so on. These instructions are typically listed inside a Dockerfile, the contents of which are shown in O.5. ROS 2 is installed here, with the Dockerfile depending on the `arm64v8/ros:humble` Docker image, where `arm64v8` denotes the system architecture. All required Python packages, such as numpy and mediapipe, are listed in a `requirements.txt` file, which is copied into the image and installed. If a new dependency is needed, the Docker image must be rebuilt. Once the Docker image is built, it is installed and ready to run.

Configuring the Docker container "Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine" [78]. There are several considerations and options to take into account when running a Docker container, as shown in O.17.

- **Sharing files** A volume is specified to allow for bidirectional sharing of files. This means that changes made outside the container are reflected inside it, and vice versa. This enables the use of an IDE outside the Docker container to edit files within it. Without this setup, changes to files would need to be made from within the container, risking loss once the container is terminated.
  Additionally, the `/dev/` directory is shared for easier access to devices such as connected cameras. While the `--device` argument could be used for this purpose, it led to issues.
- **User permissions** A "ros" user is created in the Dockerfile without root access. By entering the container as this user, files within the shared volume (`robot/` folder) do not have root permissions. This allows a user outside the Docker container to edit files created by this non-root user without elevated permissions, and vice versa. This ensures seamless code editing.
- **Network** The container's network is set to use the host's network. This configuration enables ports inside the Docker container to be accessible outside without individual mapping. This simplifies communication with the VR headset and database.

4.2 The VR application **SO** | *OM*

In this section, we delve into the design and implementation aspects of the VR application. The primary objective of using the VR application is to provide an interface for the operator to control the robot. The implementation details are divided into different sections, each explaining various functionalities.

The VR application comprises several main components, as depicted in Figure 4.1. Each component is designed to fulfil specific functionalities derived from the project requirements; thus addressing the A, B, and C requirements 1.5. These components are major groupings, each consisting of various smaller units that collaboratively achieve the intended functionality. Beyond the initial specifications, additional components have been integrated to meet extra requirements provided by the customer beyond the original scope. In Figure 4.1, the components are colour-coded: purple indicates the original A, B, and C requirements, green denotes the additional requirements, and red highlights the components essential for the development process.
In this section, we will provide a visual overview of the VR application to clarify its layout and design. Following this, each component will be described in detail, highlighting their contributions to the overall functionality. Additional functionalities added beyond the initial plan will be addressed in the 'Additional Work' section 4.6.

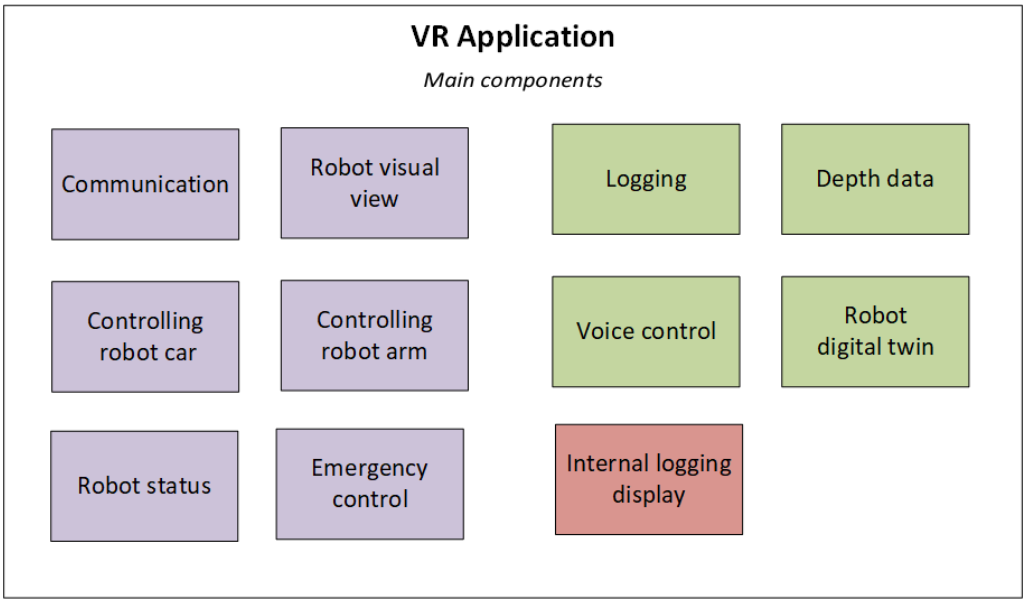


Figure 4.1: Main components in the VR application

4.2.1 Visual overview of the application **SO** | *OM*

Before delving into the detailed functionalities of the VR application, let us examine its visual elements from a broad perspective. The application's interface varies depending on its connection status to the robot. Figure 4.2 illustrates the visual elements when the application is not connected to the robot, while Figure 4.3 depicts the interface when the application is in action.

Each visual element serves a unique function and responds to user interactions through the VR headset, providing both textual and visual information to users of Kromium's application. These elements are labelled in the figures for clarity. In the following sections, we will describe the implementation details of each element.

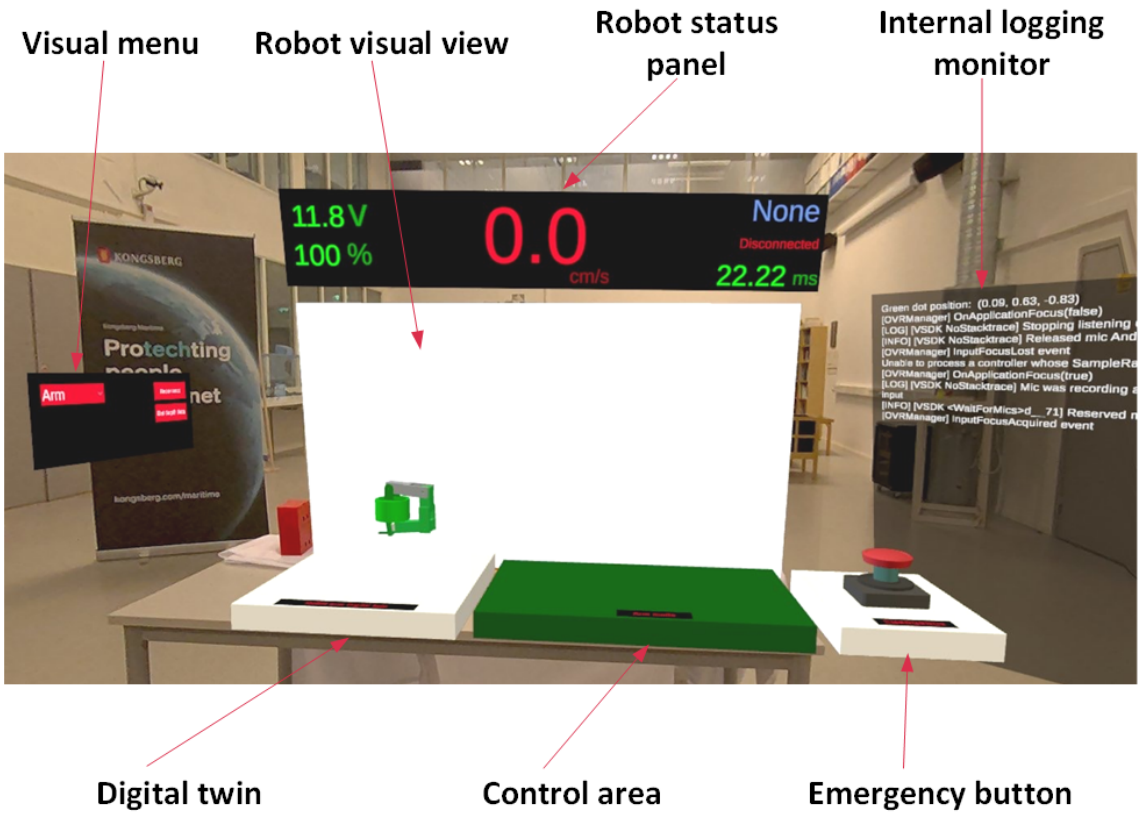


Figure 4.2: Overview of the application (Disconnected from the robot)

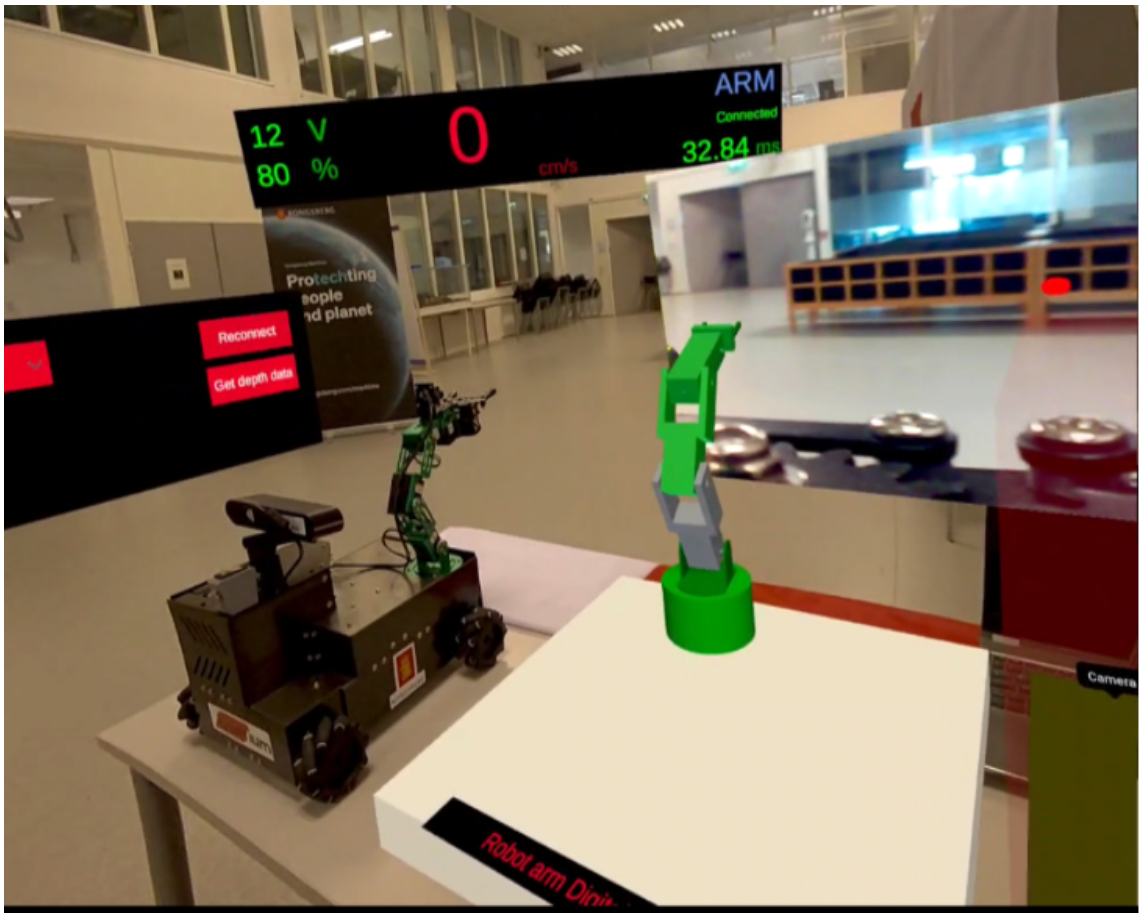


Figure 4.3: Overview of the application left-hand side when in action

4.2.2 Communication between VR and the robot **SO** | *OM*

Integrating the VR application with the robot was a crucial first step in the development process. This integration was segmented by communication method and the message type understood by both the robot and VR, which we define as the interface between them. The **Network Manager** is the sole component responsible for managing communication between the robot and the VR application. All communication, both incoming and outgoing, is handled through the Network Manager, which processes and distributes the received messages throughout the rest of the application. Furthermore, data sent from the application is processed by the Network Manager before being forwarded to the appropriate destination, as depicted in Figure 4.4. The Network Manager adheres to a singleton design pattern, making it a static object that any other object within the application can request for sending messages to the robot, while incoming data is managed by the Network Manager itself.

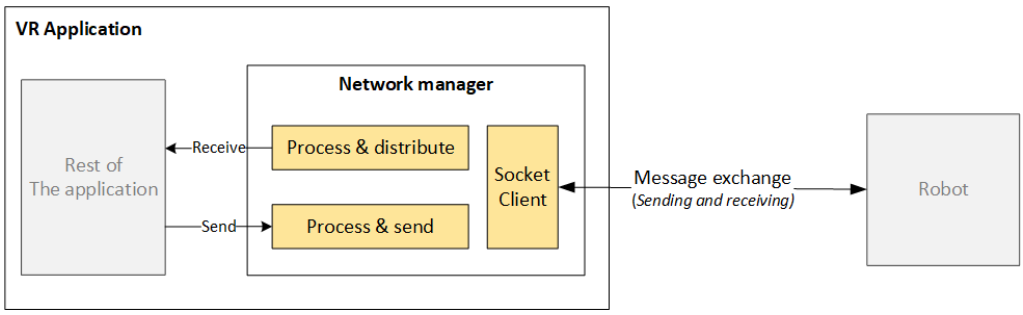


Figure 4.4: Network manager VR application

Network manager socket client and interfaces

The Network Manager communicates with the robot through a socket client connected to a socket server hosted on the robot side. The Network Manager has two interfaces for receiving data; one for JavaScript Object Notation (JSON) format and another for *Brotli* compressed data, which will be elaborated on later in this report 4.6.1. Examples of messages are illustrated in Figure 4.5. The JSON interface is the primary communication protocol, where each message is notated with a key: *"type"* that indicates specific informational data, and the rest of the JSON data contains specific application data. The interface for *Brotli* is marked with a custom header, *'bro'*, added at the beginning of the message. The first three bytes indicating the *Brotli* data are ignored before it is decompressed and distributed to the rest of the application.

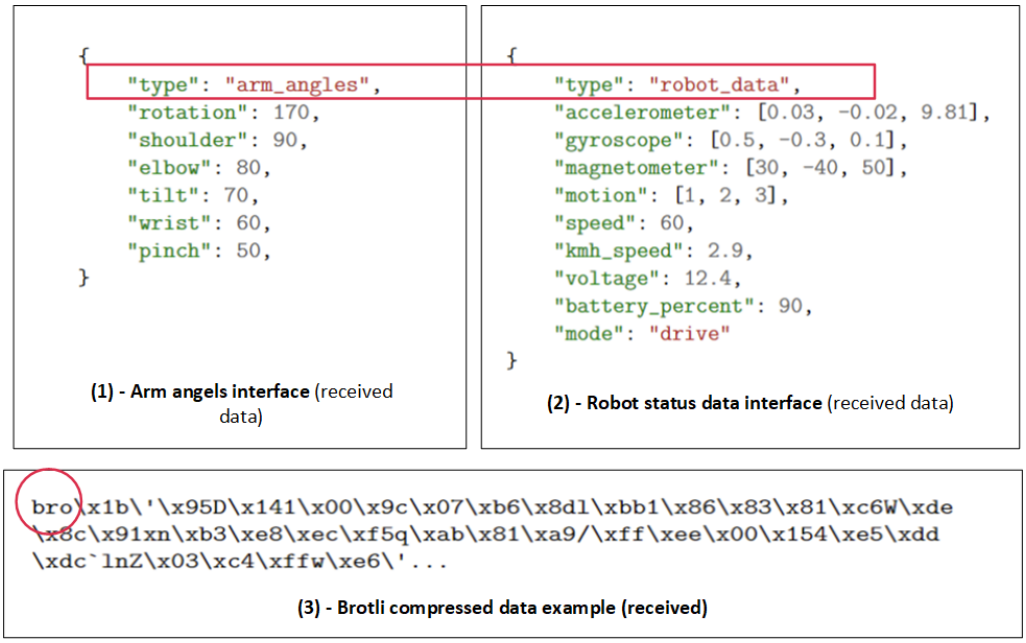


Figure 4.5: Network manager interface examples

Distribution of received information As indicated in Figure 4.4, the Network Manager processes the incoming data and distributes it to the rest of the application. The incoming data to the socket client is managed by a component called the *Received Thread*, which continuously listens for incoming data from the socket client. This component reads the incoming bytes, converts them to UTF-8 strings, and appends them to the *Received Data Buffer*. This buffer is a data structure which contains all incoming information waiting to be processed and sent to specific objects within the application. The *Process Received Data* component retrieves the oldest data from the buffer and processes it according to the type of data described in Figure 4.5, triggering the corresponding event within the Unity event system.

The Unity event system is a framework used to manage and dispatch events, allowing different components within the application to communicate with each other efficiently. It operates on a subscriber model where components can listen for specific events and react to them when they occur. This system enhances modularity and decoupling by allowing objects to interact without needing to know the intricate details of each other's operations. Using this system, components subscribing to a specific event will receive the new data and utilize it accordingly. The Unity event system is used for the distribution of incoming information, as shown in Figure 4.6.

Priority queue was considered to process incoming information based on priority levels. Taking into account our requirements to build a PoC the priority queue was not prioritised for implementation.

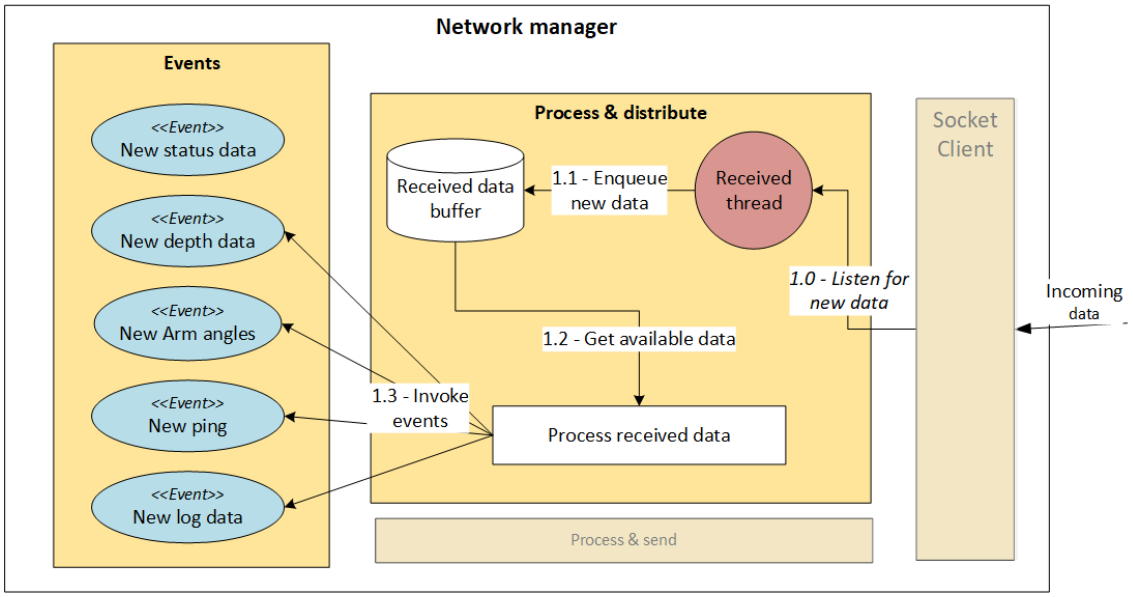


Figure 4.6: Data distribution from network manager to the application

The triggered events are not restricted to a single component; rather, multiple components can subscribe to the same event and respond to the data in varying ways. For instance, upon receiving "new status data" from the robot, visual elements dynamically update themselves, showcasing information like the robot's speed on a text element. Simultaneously, an internal logging component listens for the same event, refreshing the logging screen accordingly. At this stage, the processed incoming data is distributed throughout the application via events, leaving it to other components to utilize this information. In the coming subsections, we will look into how this information is utilized in the application.

4.2.3 Robot visual view

SO | *OM*

The robot visual view is a vital component of Kromium's application, designed to enhance remote operational capabilities. It provides the operator with a first-person perspective of the robot's environment, enabling real-time visual feedback. This feature is crucial as it allows the operator to see through the robot's "eyes", facilitating precise and effective control during operations. Below, Figure 4.7 illustrates the monitor that displays the camera feed within the application.

Figure 4.7: Camera monitor inside the VR application

Functionality:
Providing operators live video feeds from cameras that are strategically placed on the robot (see 6.2). These cameras are positioned to maximize the quality of the visual data they collect, which helps enhance how operators control the robot. This setup enables operators to navigate the robot and manipulate its arm effectively, providing them with a clear view of their surroundings.

The functionality of the cameras extends beyond basic navigation. The cameras provide front, rear, and arm-mounted views. This range of camera angles gives operators comprehensive visual coverage, making it easier to carry out complex tasks with the robot's arm.

Additionally, the video feeds are processed using ML algorithms that recognize and identify objects around the robot. This extra layer of information helps operators make better decisions by giving them a clearer understanding of the environment. More details on these ML enhancements are available in the object detection section on 4.5. In Figure 4.8 and 4.9 you can see examples of different views of the camera inside the VR application.

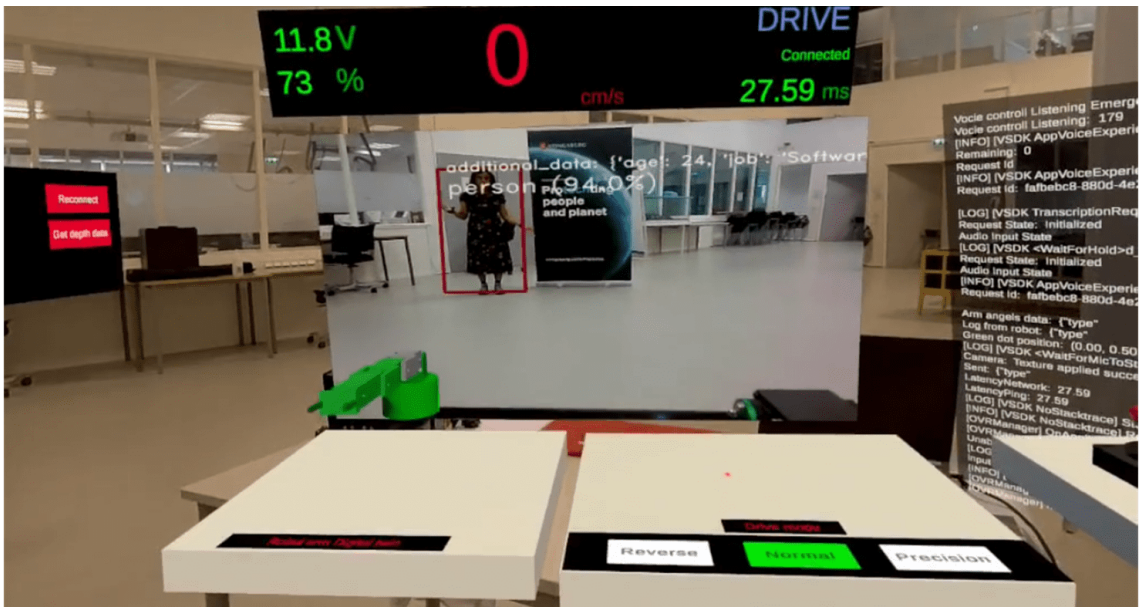


Figure 4.8: Camera monitor in Drive mode (Camera 1) detecting a person

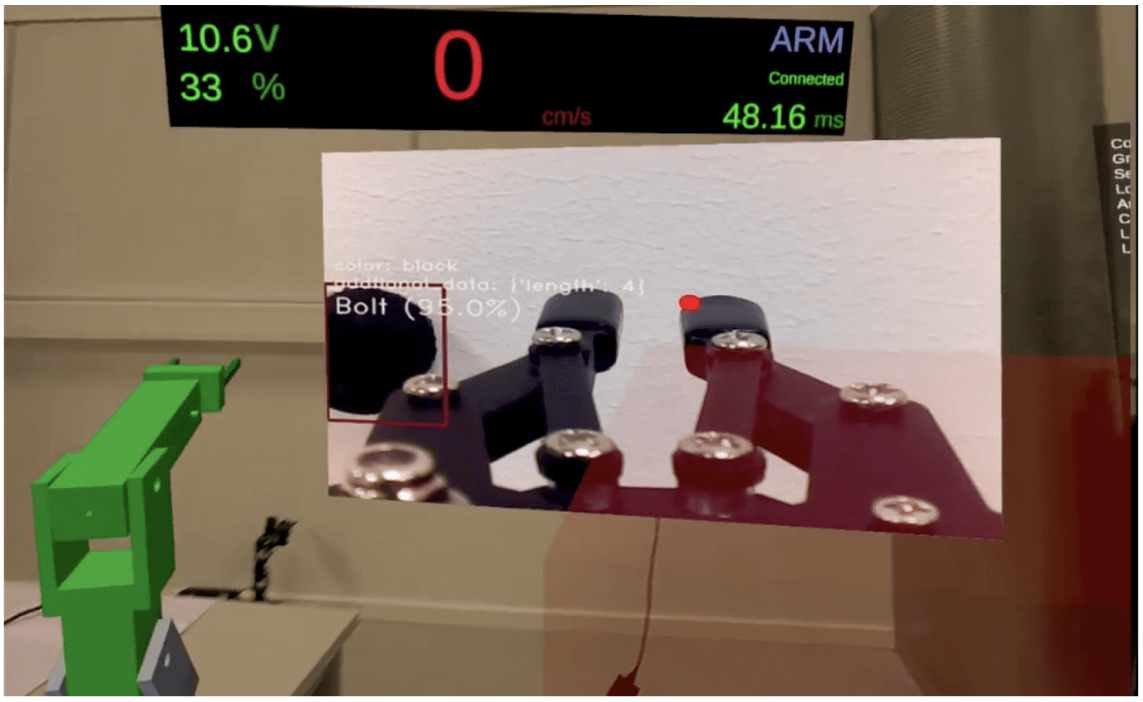


Figure 4.9: Camera monitor in Arm mode (Camera 2) detecting a bolt

Receiving video feed:
The video feed from the robot's cameras is transmitted via websockets. The application continuously listens on a specific Uniform Resource Locator (URL) and port 5000 to receive this camera feed, updating the camera monitor displayed in the application as shown in Figure 4.7. The switch between camera views is automated, depending on the robot's operational mode (see 4.2.4). The robot handles the changes in the camera feed and decides which ML algorithm to use for the video feed transmission, with more details written about in section 4.3 and 4.5. Additionally, the video stream was optimized for better performance in the latter part of the project, as detailed in 4.6.10. The coordination of the robot's cameras and the ML algorithms active in each mode is summarized in the table below.

Operational Mode	Active Camera	Active ML Algorithm
Driving Mode	Camera 1	People Detection model
Arm Mode	Camera 2	Bolt Detection model
Driving Backward	Camera 2 (rotated)	People Detection model

Table 4.1: Summary of camera and machine learning configurations across different operational modes.

4.2.4 VR application and robot modes

SO | *AD*

Kromium's system offers a variety of operational modes that enable the operator to customize the control of the robot according to specific requirements. These modes are accessible through the VR application, allowing the operator to select the most suitable system mode for different scenarios. The available modes are designed to address a range of use cases, each facilitating custom interactions and control. The use case diagram in Figure 4.10 illustrates the operator's ability to switch between these different operational modes, which have been derived from our requirements.

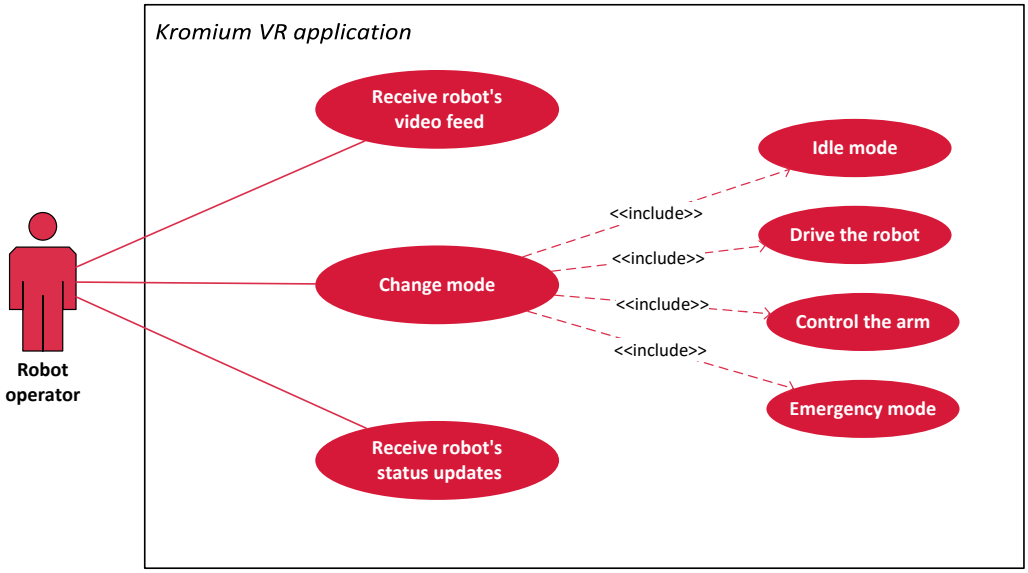


Figure 4.10: Use case diagram - Robot operator

Mode descriptions:

- **Idle mode:** In this mode, the robot does not react to any commands from the operator, nor does the application send any commands. This feature is important for safety, as it allows the operator to do other duties without accidentally sending commands with their hands.
- **Drive the robot:** The operator can use their hand movements to drive the robot. The application constantly sends commands to the robot in response to the operator's appropriate hand movements.
- **Control the arm:** Similar to driving the robot, this mode allows for the control of the robot's arm using different hand movements.
- **Emergency mode:** Similar to the Idle mode, it requires physical inspection from a technician to verify that the robot can continue to operate.

Handling operational modes

Operational modes in the VR application are managed through a state-machine architecture. This setup ensures that each mode is distinct with specific controls and interactions tailored to that state. The application dynamically adjusts the available controls and visual interactions according to the active mode to enhance usability and reduce errors.

- **State machine logic:** The operational mode handling functions like a state machine within the application. Each state corresponds to a different mode of robot operation, ensuring that the control interface adapts accordingly as the mode changes.
- **Visual and interactive controls:** In different modes, the application alters the visibility and functionality of user interface elements. For example, in Idle mode, all controls that could command the robot to move or manipulate its arm are hidden to prevent operations by accident. In contrast, in Driving mode, controls and visual controls become available, allowing the operator to drive the robot.
- **Mode transition:** When the operator switches modes, the application communicates this change to the robot, ensuring that it receives the correct data to respond appropriately.

Figure 4.11 illustrates the transitions between the various states within the application. Upon initialization, the application enters Idle mode. From there, it is designed to allow transitions to any of the other operational modes. Each mode incorporates specific logic that governs its unique state behaviour within the application, ensuring tailored interactions and responses according to the mode's functional requirements. Figure 4.13 shows the arm scene (see P.1.1) in Arm Mode, and Figure 4.12 shows the drive scene in Drive Mode, both of which are further detailed in the following sections 4.2.5 and 4.2.6.

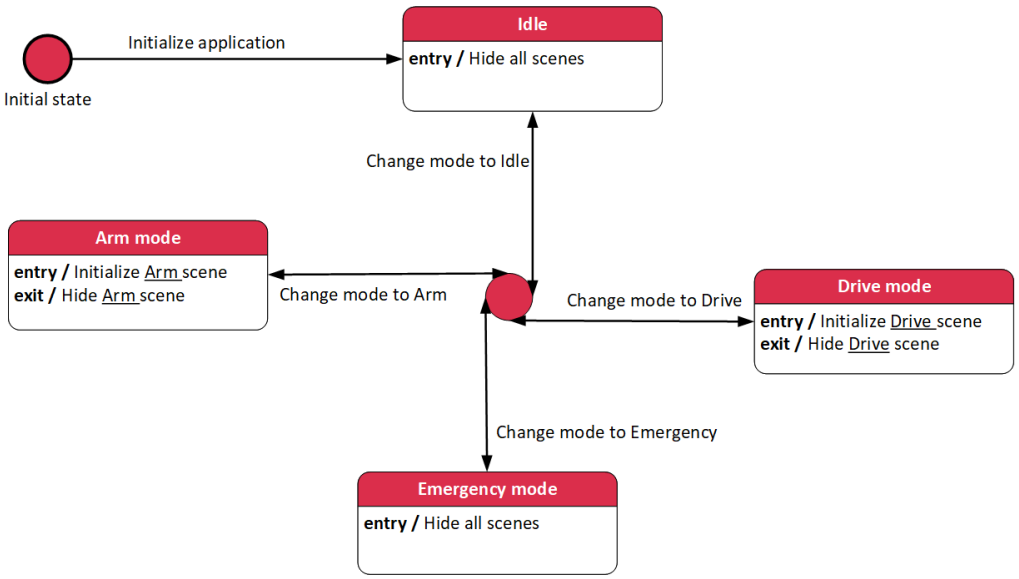


Figure 4.11: State machine diagram - Mode states in VR application

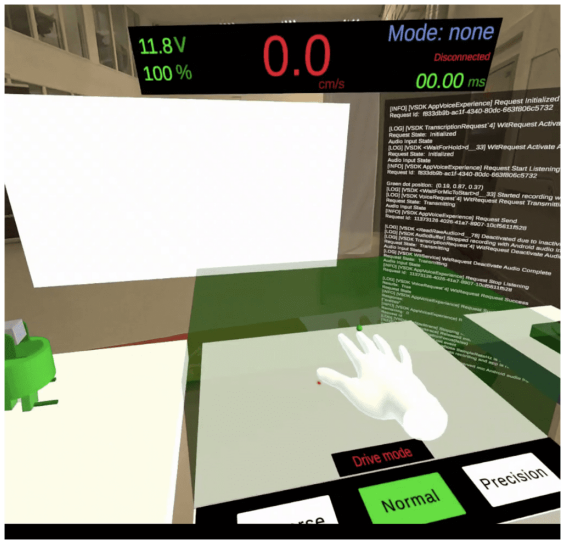


Figure 4.12: Drive scene detecting user's hand movements

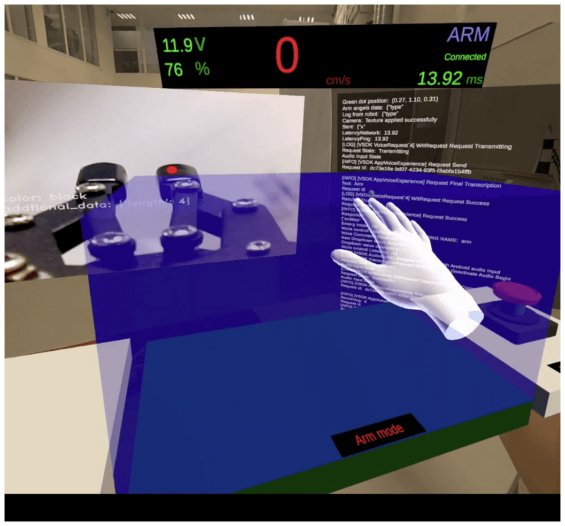


Figure 4.13: Arm scene detecting user's hand movements

Mode synchronization and control interfaces:
Changing operational modes is facilitated by two interfaces:

- **Visual menu:** Operators can change modes using a graphical interface within the VR application, designed for intuitive and quick switching of modes, illustrated in Figure 4.14.
- **Voice control:** For hands-free operation, voice commands are also enabled, allowing operators to switch modes without physical interaction, which is especially useful in scenarios where manual control is impractical. Detailed use cases for voice commands are explored further in section 4.6.8.

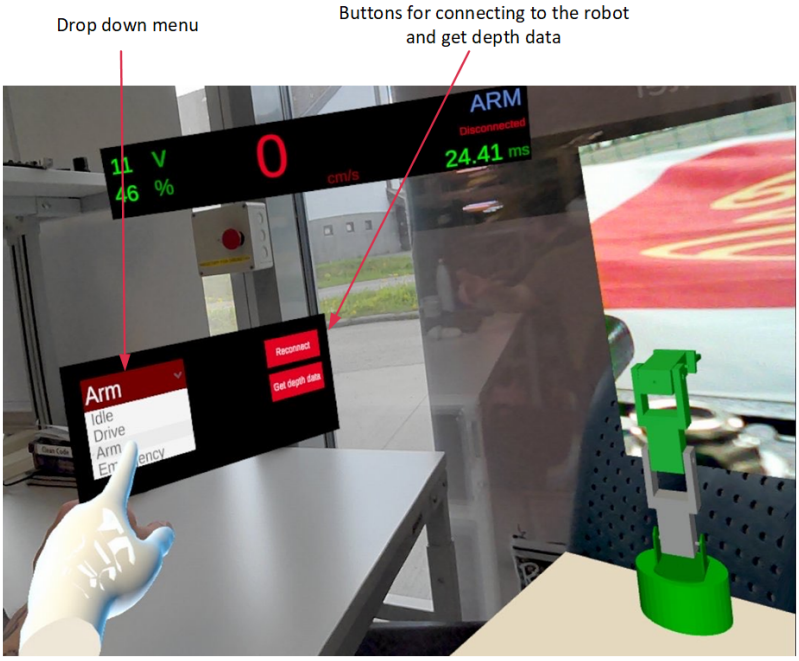


Figure 4.14: Visual interface for changing mode, connect to the robot and get depth data

Synchronization: Synchronizing the mode between the application and the robot is critical. It ensures that commands issued in one mode are appropriate and that transitions into other modes are recognized and implemented without disruption to ongoing tasks. Figure 4.15 details the sequence of activities triggered when a user switches operational modes. Specifically, upon activating the "Enable drive scene" or "Enable arm scene," the application initiates multiple background processes tailored to each mode. For instance, the "Enable drive scene" starts processes such as hand tracking, drive command calculations, and communication with the robot to relay new command data. Concurrently, all game objects (see P.1.2) related to the drive scene become visually available to the user. These processes are elaborated in section 4.2.5. Following the selection of a mode, three parallel activities occur:

- **Update the robot:** The application sends a mode change request to the robot.
- **Update status panel:** The application updates the status panel to display the selected application mode. This panel does not reflect the robot's actual mode, which is instead communicated by the robot itself and displayed on the robot status panel (4.2.7), allowing users to see the robot's current mode.
- **Notify the user(Voice notification):** The application uses a voice notification to inform the user about the mode change.

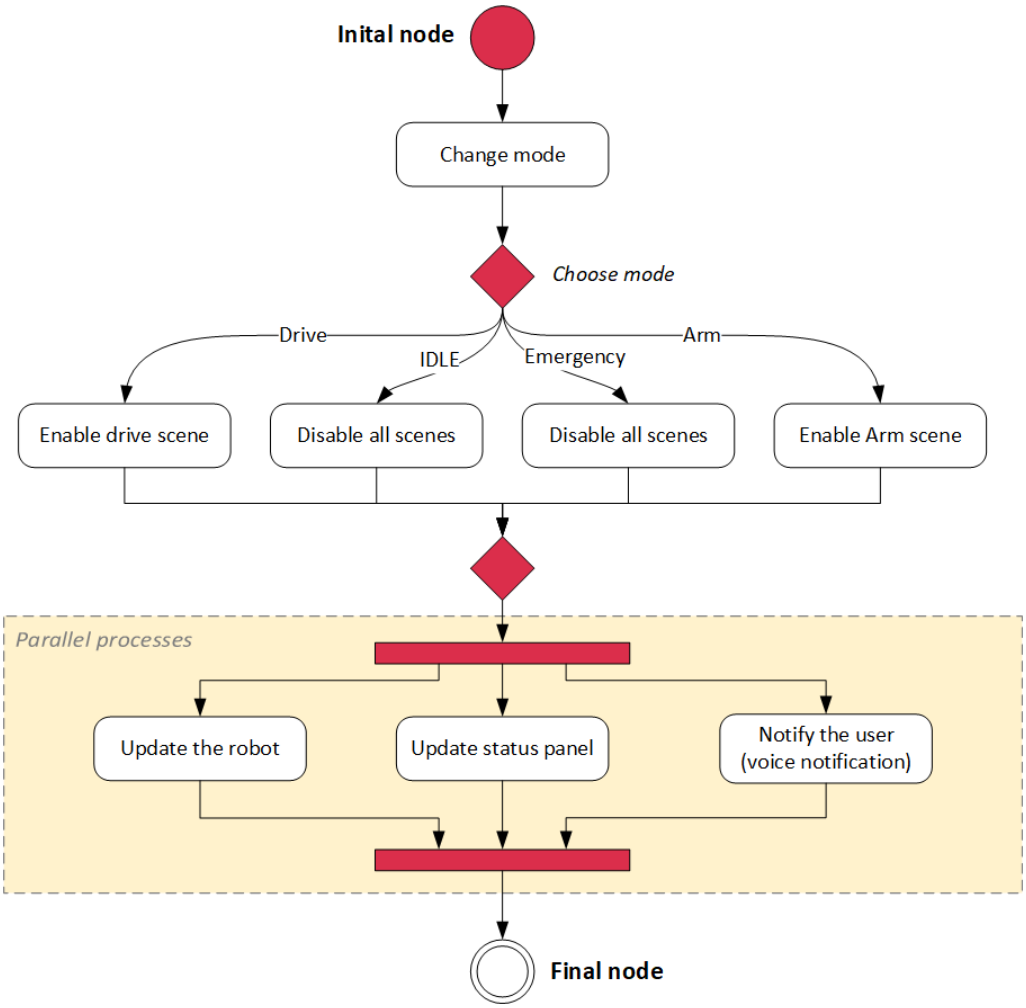


Figure 4.15: Activities that occur when changing mode

4.2.5 Controlling the robot car **SO** | *AEH*

One of our requirements is that the user can drive the robot car through the VR application. The controlling component of the robot car is responsible for all aspects of driving, including the calculation of directions (forward, left, right, backward), driving modes (reverse, normal, precision), and speed control. The application together with the robot, provides an intuitive and precise control mechanism for virtually driving the robot through the VR headset. See the folder **Driving Robot** in the attachments for a demonstration.

The scene for controlling the robot, including all the game objects, is shown in Figure 4.16. As described in section *changing operational modes* 4.2.4, the scene shown here only appears to the user when the application is in drive mode.

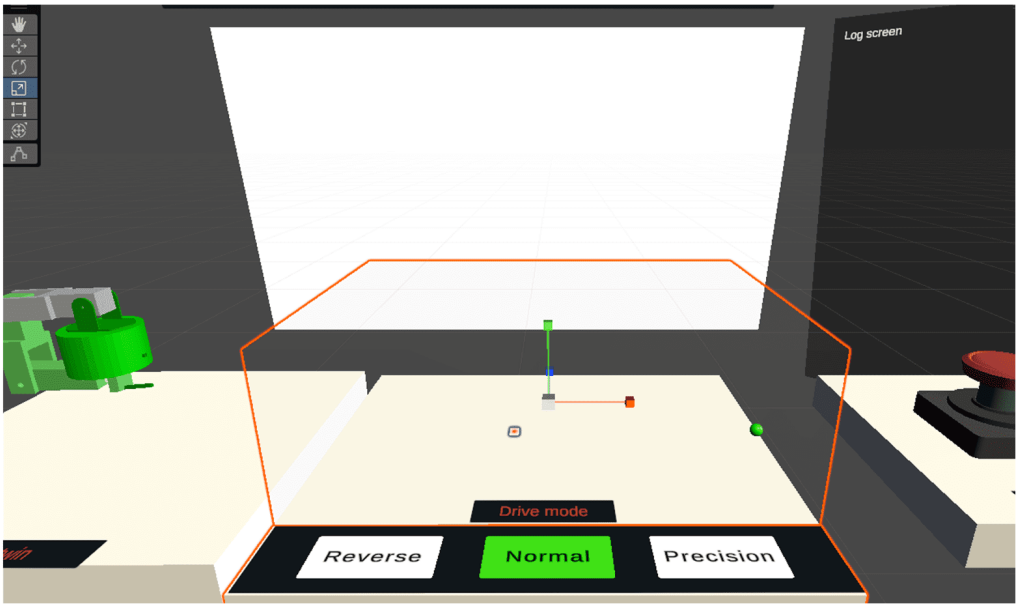


Figure 4.16: Scene for controlling the robot car

The implementation of the driving mode has undergone several iterations and tests to refine an intuitive and precise control mechanism. The components involved in the implementation are illustrated in Figure 4.17. These components, which include both game objects and logic controllers, work together to fulfil the functionality required by our stakeholders (See Appendix C). The game objects in the scene are the visual elements that the user interacts with, while the logic controllers are software components responsible for calculating the correct driving commands and sending them to the robot. In this section, we will describe the roles of game objects and logic controllers and how they interact with each other to create the driving logic.

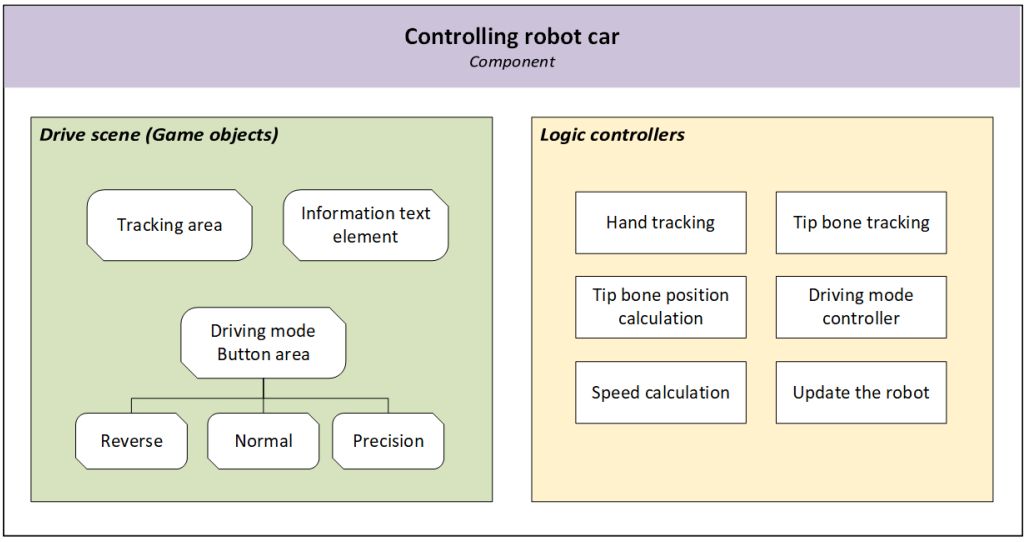


Figure 4.17: Controlling the robot car components

Game objects in the drive scene

The development of the VR-driven control system for the robot car begins with a foundational concept: translating user hand movements into precise drive commands. We built upon this idea by creating a driving scene where the user can interact with a set of game objects to generate driving commands for the robot. In this scene, specific game objects are designed to detect and interpret the user's hand movements. These objects act as the interface between the VR environment and the physical movements of the user, converting hand gestures into actionable commands that are sent to the robot car.

Tracking Area The Tracking Area is an empty rectangular cube located in front of the user's right hand. It serves as the primary interactive zone for hand tracking, initiating tracking when the user's hand enters this space. This object is crucial for detecting hand movements that control the robot car. You can see the non-highlighted tracking area when the hand is absent in Figure 4.18, and the highlighted version when the hand is detected in Figure 4.19.

Figure 4.18: Default tracking is when the hand is not detected.

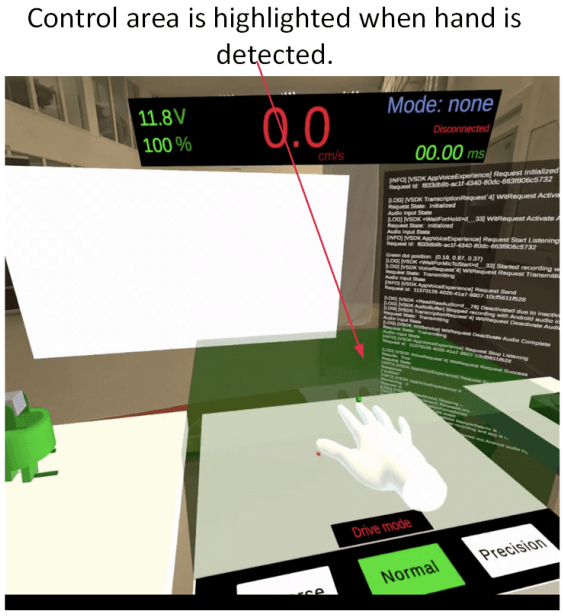


Figure 4.19: Tracking area highlighted when the hand is detected.

Information Text The information text is a text display within the scene that indicates the scene name "Drive mode".

Driving Mode Button Area Located at the back of the tracking area 4.2.5, the Driving Mode Button Area contains several buttons: Reverse, Normal, and Precision. These buttons allow the user to select the desired driving mode. Interaction with these buttons is captured by the logic controllers, and transmitted to the robot, which then processes these inputs to adjust the robot car's driving mode accordingly.

Drive command generation in Normal Drive Mode
The application begins generating drive commands when it enters the *drive* mode. A sequence of activities is illustrated in Figure 4.20. The application tracks the user's hand through the *Hand Tracking* component, which continuously monitors the user's hand position. However, command generation does not commence until the user's *tip bone* enters the tracking area. From this point, the *tip bone* is tracked, and drive commands are generated and sent to the robot. The frequency of this calculation is controlled by the calculation interval, currently set to **0.1 seconds**.

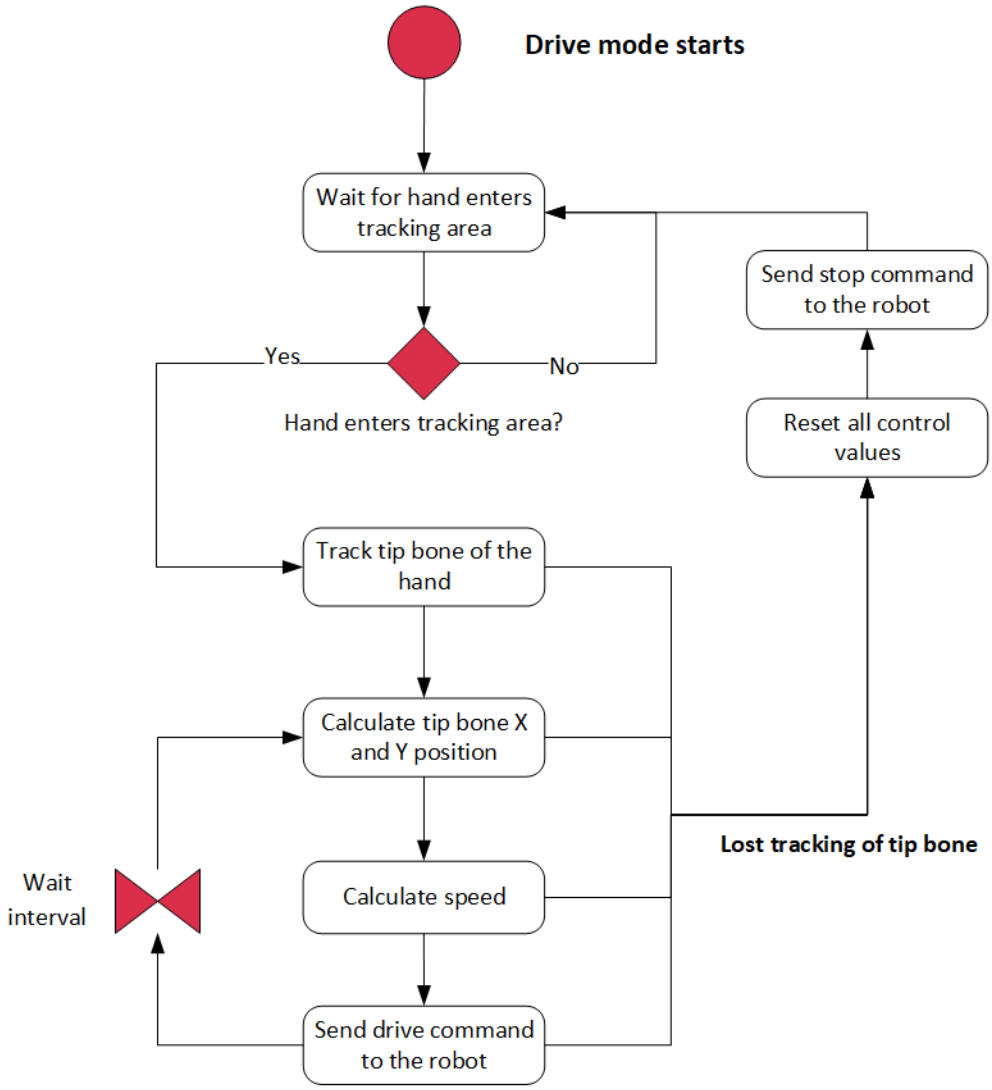


Figure 4.20: Activities in drive mode

Command generation involves calculating the position of the user's *tip bone* relative to the tracking area and mapping the values between $X \in [-1, 1]$ and $Z \in [0, 1]$. Figure 4.21 shows the application scene where the user interacts and Figure 4.22 provides a mathematical diagram illustrating how the relative position of the user's *tip bone* in the tracking area is calculated. Points *p1* and *p2* in the diagram represent examples of the user's *tip bone* position. Initially, the calculations focus on finding a precise and intuitive method for the user to drive the robot. Subsequently, the user's *tip bone* position is converted into directional driving commands on the robot side, as shown in Figure 4.23. The tracking area is divided into directional sections. Movements of the user's hand within these sections send signals to the robot, directing it to drive in specific directions. The translation of these signals by the robot is detailed in 4.3.8.

Calculation of Speed Control

During the development process, we discovered that controlling the robot without speed control, or with a static speed, did not offer a natural feel. The lack of speed control led to uncontrolled and rapid changes in the robot's direction, and less control and precision. Consequently, speed control functionality was integrated into the application to provide simultaneous control of direction and speed through hand movements. The speed control is implemented within the same Tracking Area, where the distance between the user's

tip bone and the edge of the corresponding direction is used as the speed. The speed increases and decreases based on the distance between the user's *tip bone* and the edge of the tracking area. The edge The corresponding directions are mapped as follows: left area to left edge, right area to right edge, forward to front edge. Figure 4.24 illustrates how speed is calculated in a range between 30 to 100, where the speed range represents the percentage of the robot's maximum speed. Speeds between 0 and 30 are disregarded as the robot's movement is too slow to react effectively in this range.

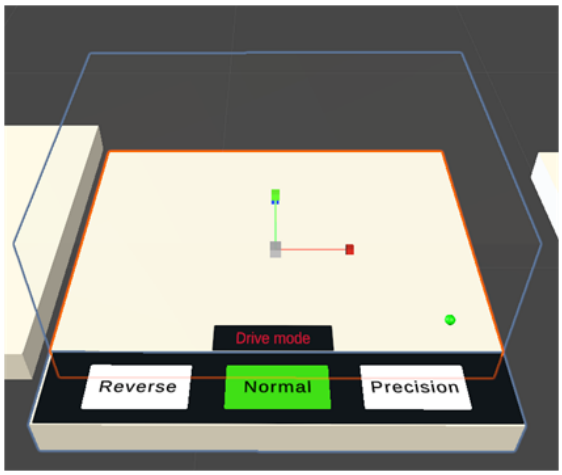


Figure 4.21: Tracking area in the application drive mode

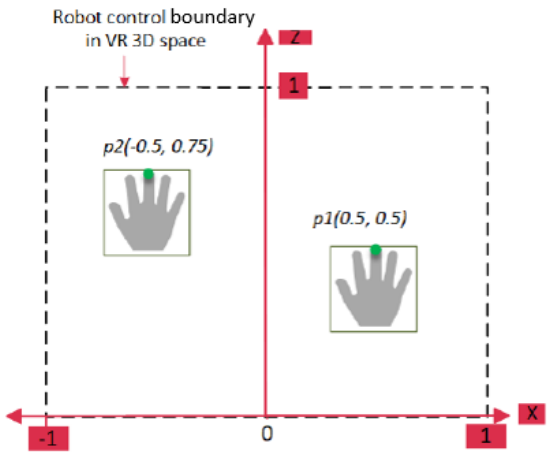


Figure 4.22: Tracking area conceptualized

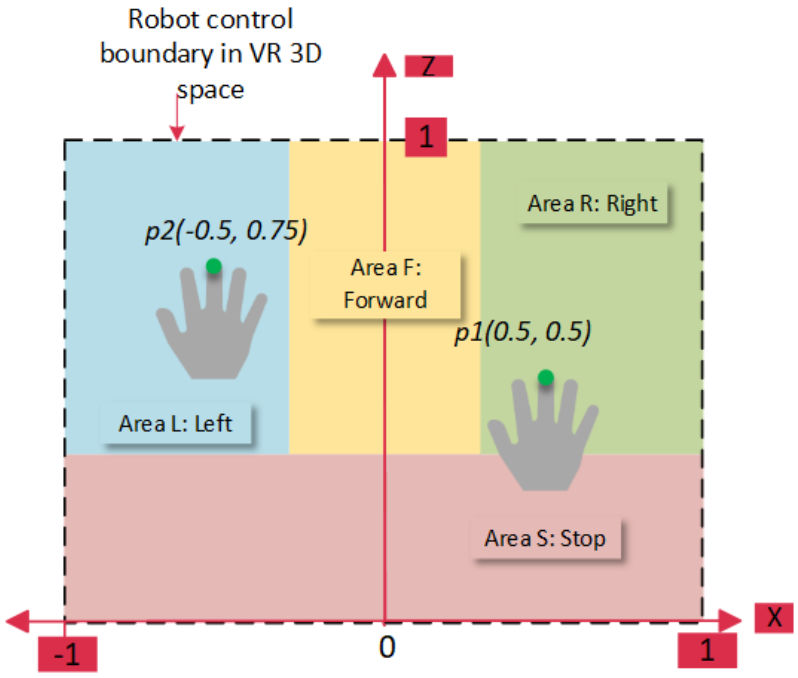


Figure 4.23: Tracking area divided by sections

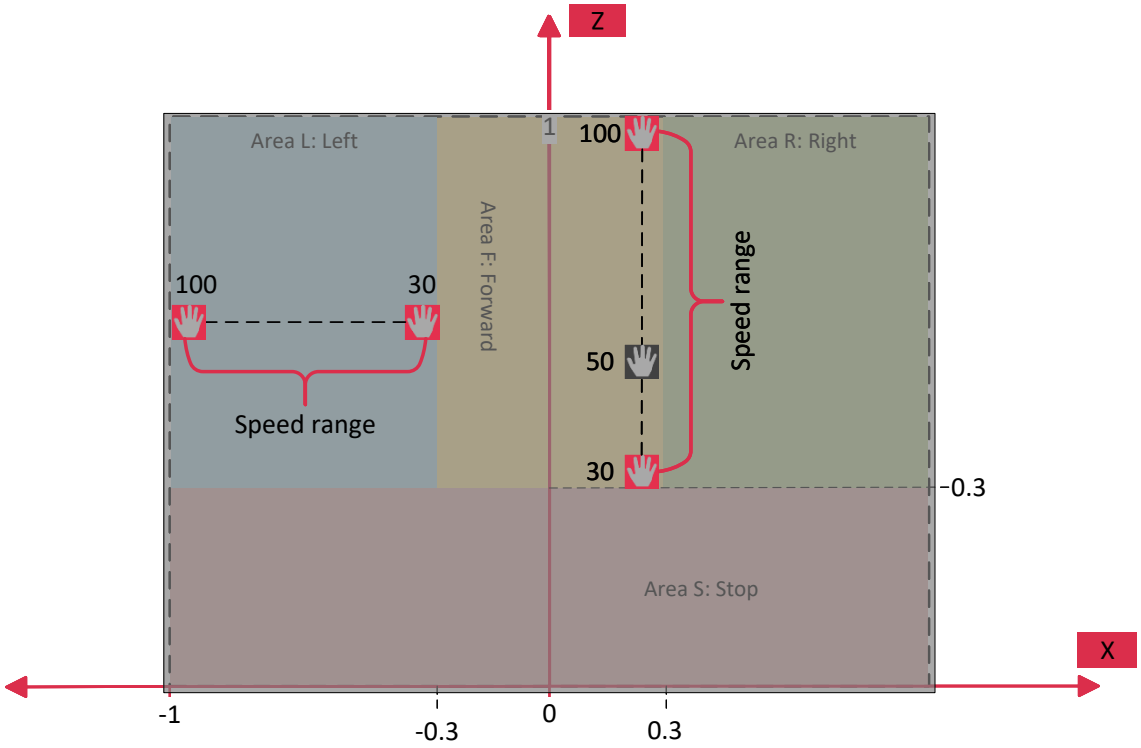


Figure 4.24: Calculation of speed

Precision and Reverse Drive Control Mode

In the Precision and Reverse Mode, the control mechanism remains the same as in Normal mode. The distinctions arise on the robot side, as discussed in 4.3.8. However, when in Reverse Mode, the robot's visual view switches to a different camera, allowing the user to see backwards, as detailed in 4.3.8.

4.2.6 Controlling the robot arm

SO | *AD*

Controlling the robot's arm is one of the primary functionalities of the application, allowing users to manipulate the robot arm through the VR headset. While the control mechanism is analogous to driving a car, the robot's arm interface requires a different set of data. The scene for arm control is shown in Figure 4.25. Users control the physical robot arm through hand movements within the white rectangular cube shown in the scene. The control area, similar to the driving scene, begins tracking and sending commands when the user's hand enters the control area. Figures 4.26 and 4.27 show the application tracking the user's hand presence within the control area and the absence of such tracking, respectively.

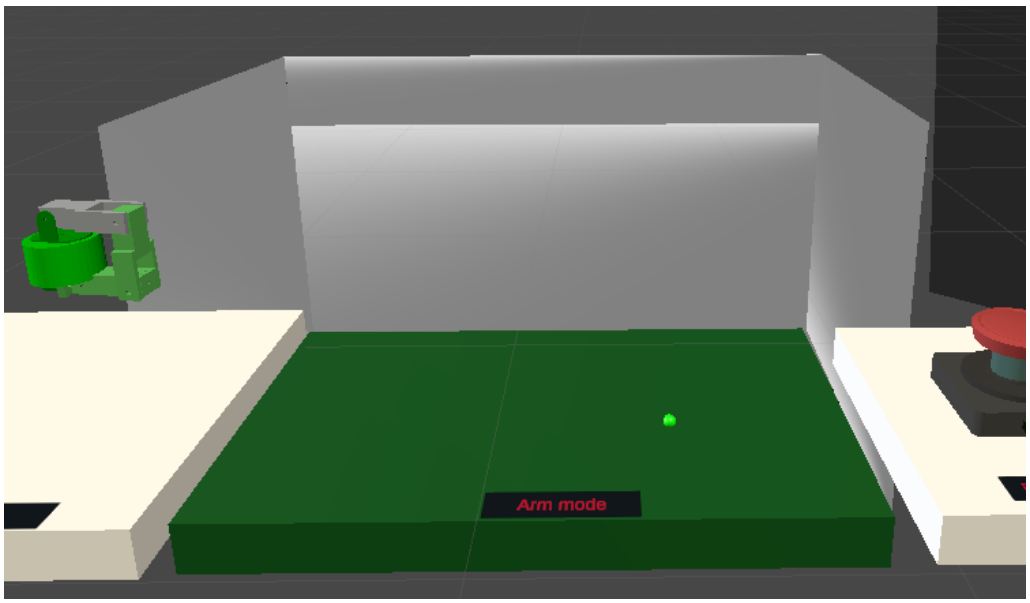


Figure 4.25: Scene for controlling the robot arm

Figure 4.26: Default tracking is when the hand is not detected - arm scene

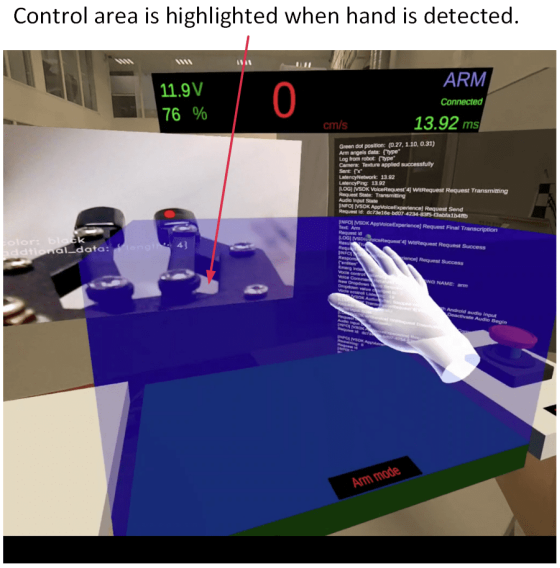


Figure 4.27: Tracking area highlighted when the hand is detected - arm scene

The application tracks the user's *tip bone* within the cube, calculating the x, y, and z positions relative to the cube and mapping the values to x in [-1, 1], y in [0, 1], and z in [-1, 1]. The values for x, y, and z are normalized to correspond to the dimensions of the physical robot arm. For instance, if the user's *tip bone* is at position p(0,0,1), the physical arm will be fully stretched upward. The *tip bone*'s position is mapped to the robot arm's end effector. Hand movements inside the cube translate into control values, which are sent to the robot. The robot employs inverse kinematics to convert the x, y, and z data into angles for each arm servo, moving the end effector to the desired position. Further details on how the robot translates this data are discussed in section 4.4. Examples of these controls are illustrated in Figures 4.28, 4.29, and 4.30.

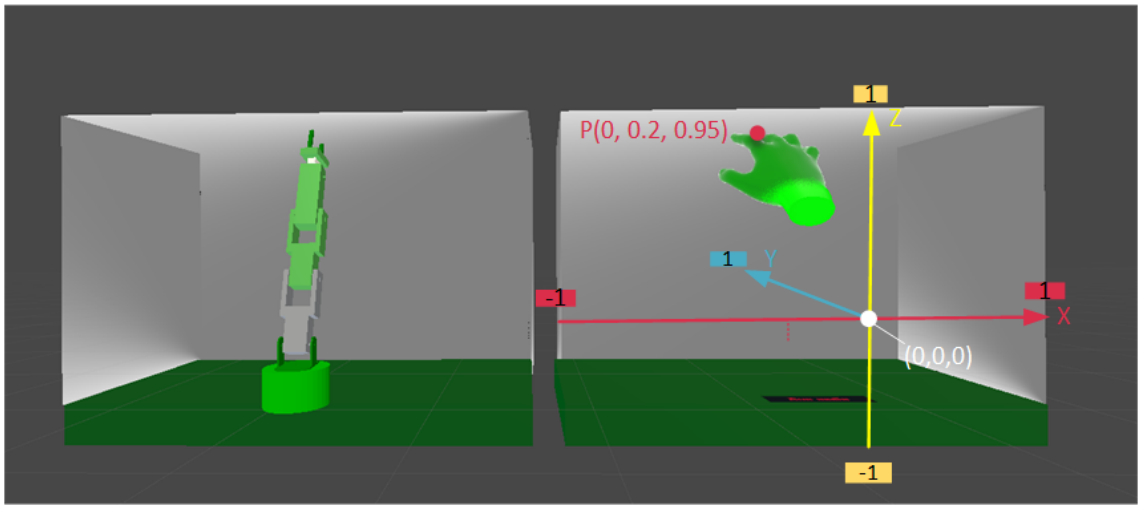


Figure 4.28: Illustration of the robot arm and user's hand - Back side

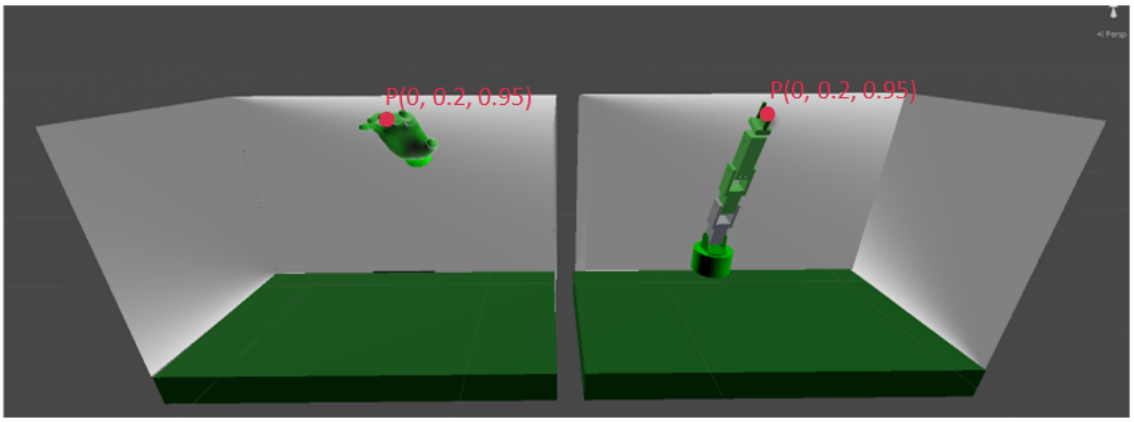


Figure 4.29: Illustration of the robot arm and user's hand - Front side

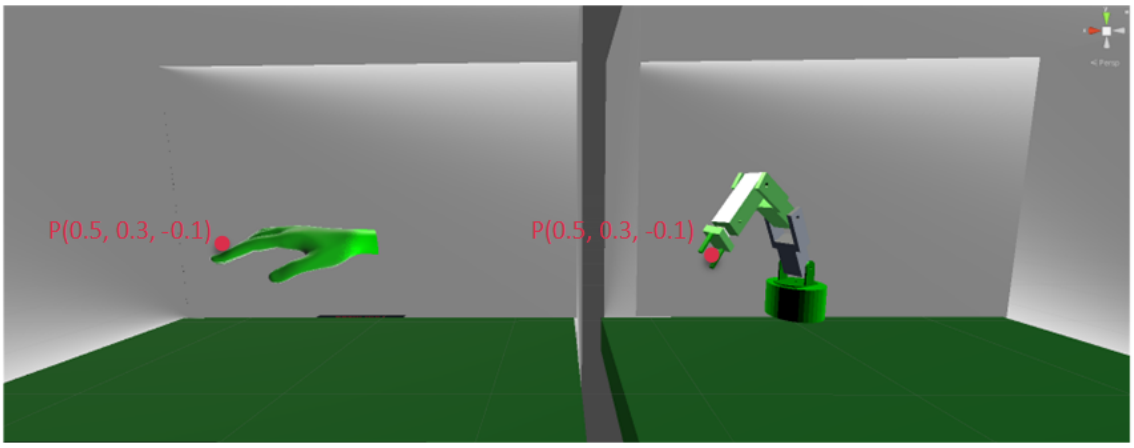


Figure 4.30: Illustration of the robot arm and user's hand - Front side

The sequence of activities when the user engages in Arm Mode is illustrated in the activity diagram 4.31. Unlike the car driving mode, when the user's hand exits the tracking area, the system does not reset the values or send them back to the robot. Instead, it is designed to maintain the arm in the same position as it was when the user's hand left the tracking area. The application then returns to a waiting state until the hand re-enters the tracking area.

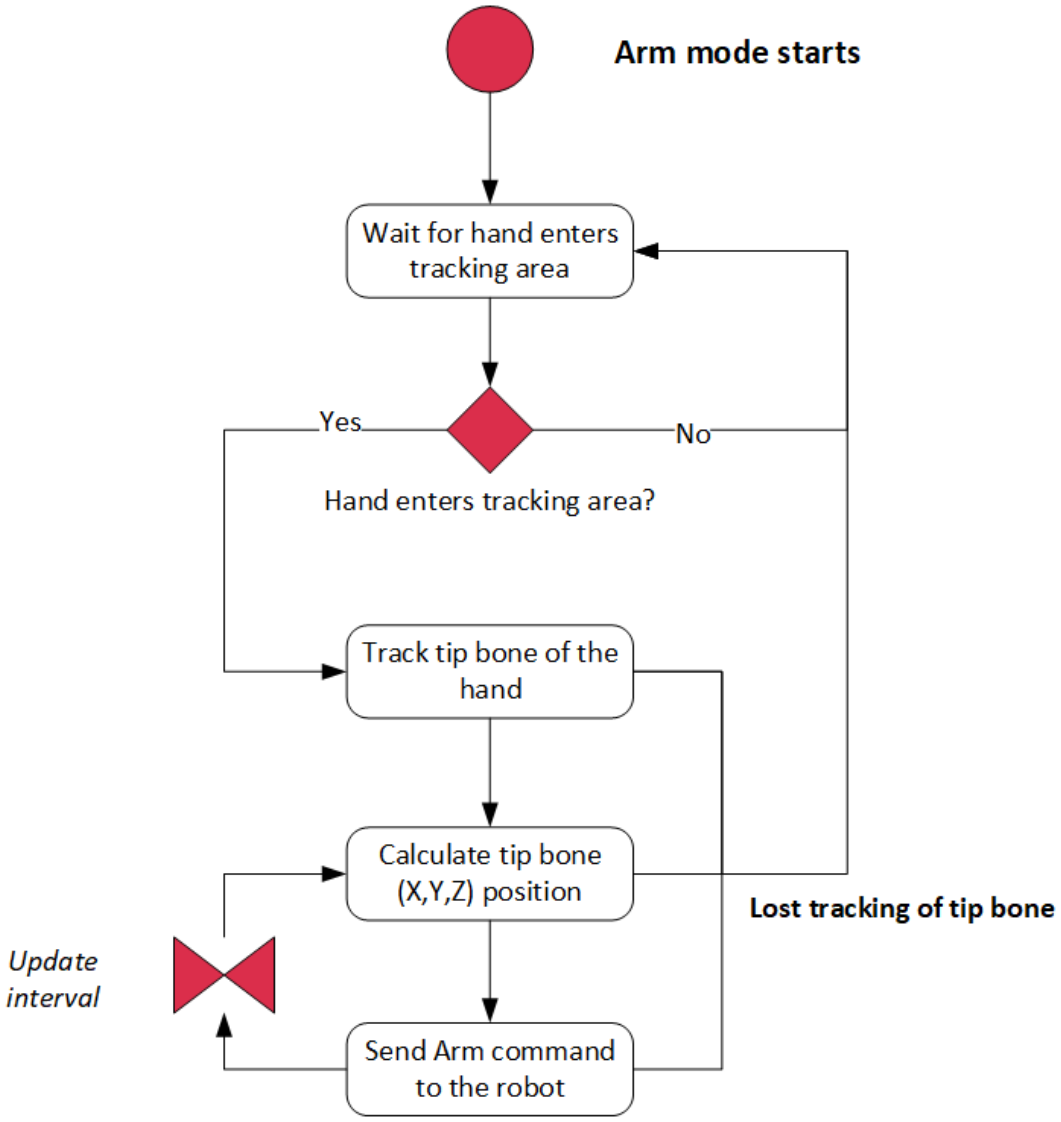


Figure 4.31: Activities in Arm Mode

4.2.7 Robot status: information display

SO | *OM*

The display of status information is a key functionality of the VR application, providing users with real-time updates about the robot. The visual element used to display this information is depicted in Figure 4.32. Each piece of information represents the current status of the robot and is frequently updated to ensure accuracy. The information panel is designed to follow the user's head movement, allowing the user to view the status regardless of their head direction within the virtual environment.

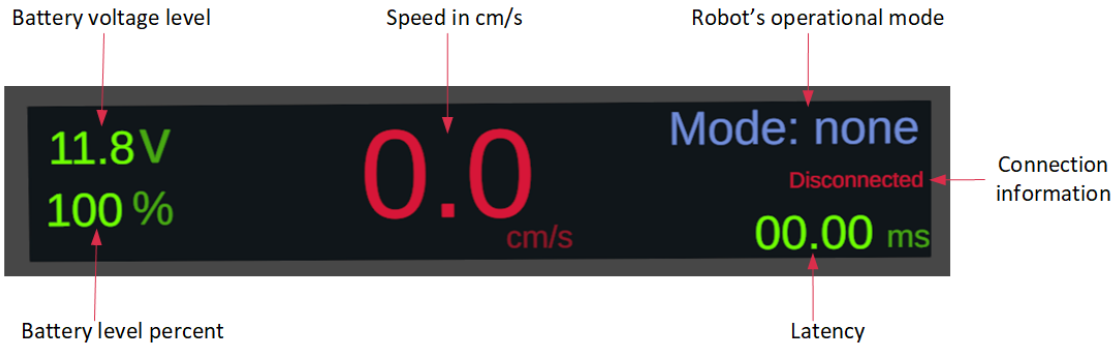


Figure 4.32: Robot status information panel

As outlined in the Network Manager diagram 4.6, the Network Manager triggers events when new Robot and Ping data become available. The robot status information panel is managed by the *Head Up Controller*, which subscribes to Network Manager events such as *New status data* and *New ping data*. This controller processes the incoming data to update the corresponding text elements in the application shown in Figure 4.32. The distribution and updating of this data are illustrated in Figure 4.33.

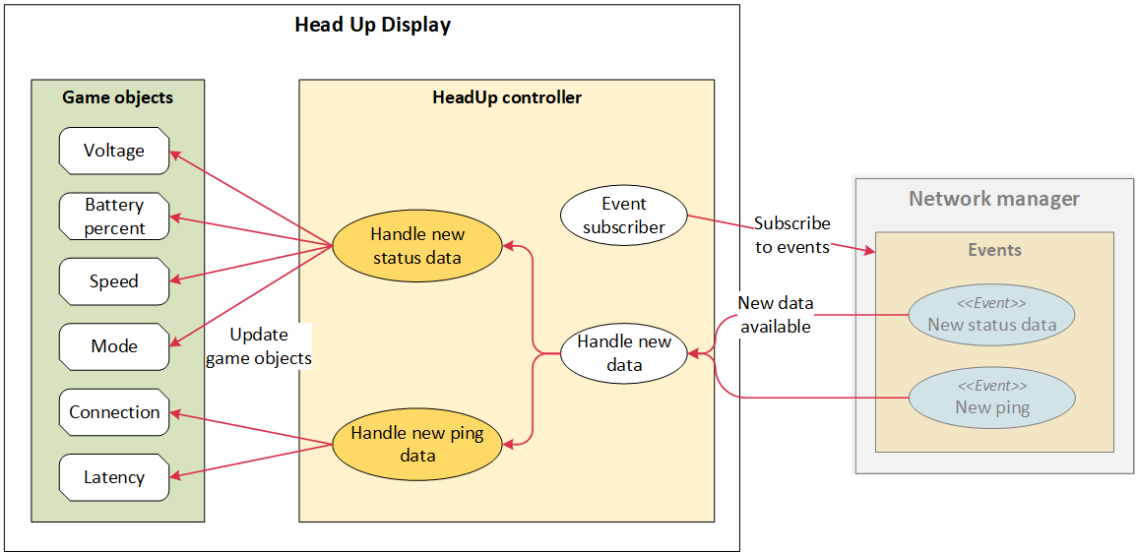


Figure 4.33: Data distribution and updating process in the Head-Up Display

4.2.8 Emergency control

SO | *AEH*

A critical requirement is the inclusion of a virtual emergency button in the VR application, allowing the user to activate the emergency mode of the robot. This button, shown in Figure 4.34, mimics the standard red emergency button widely recognized in the industry. Upon activation, all control scenes within the application are hidden, and an emergency message is sent to the robot. In emergency mode, the robot does not respond to any commands and must be manually reactivated for operational safety.

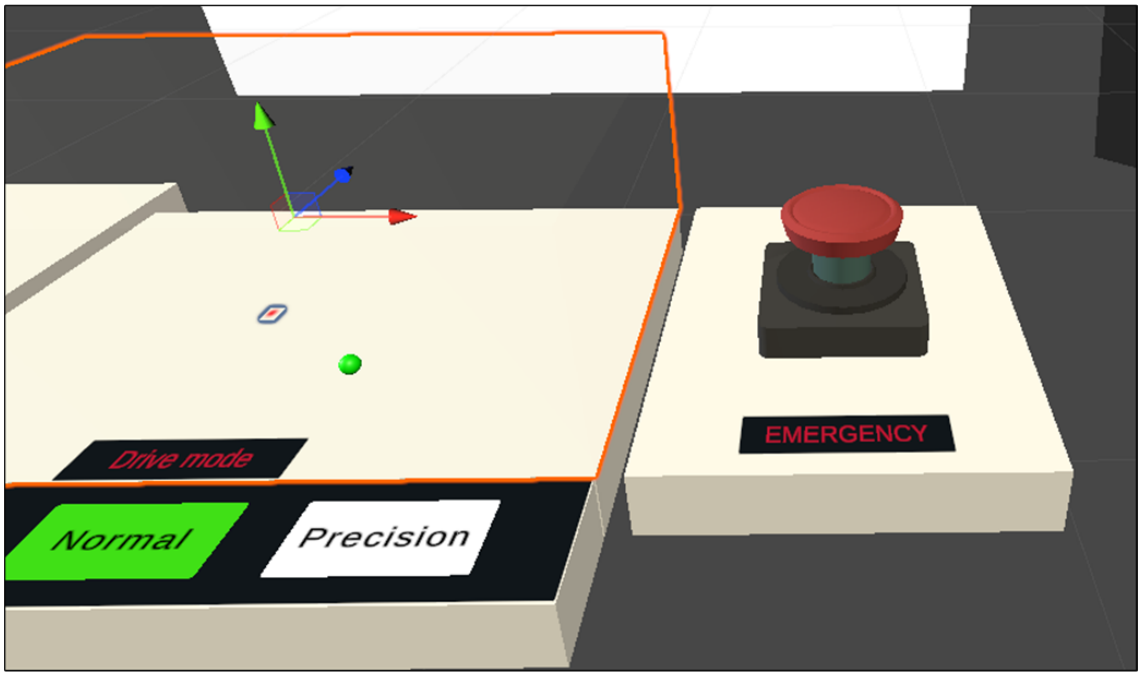
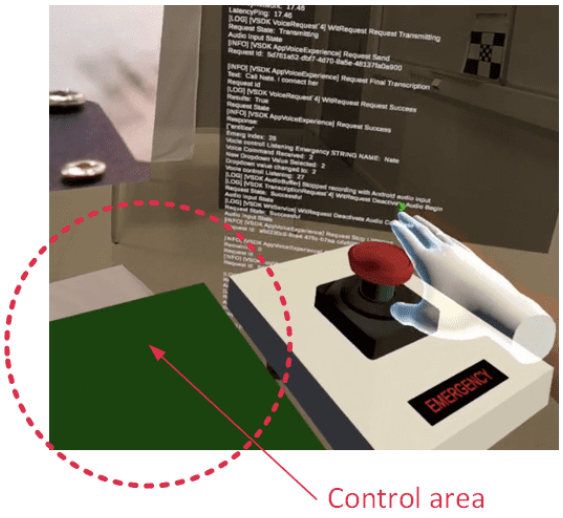
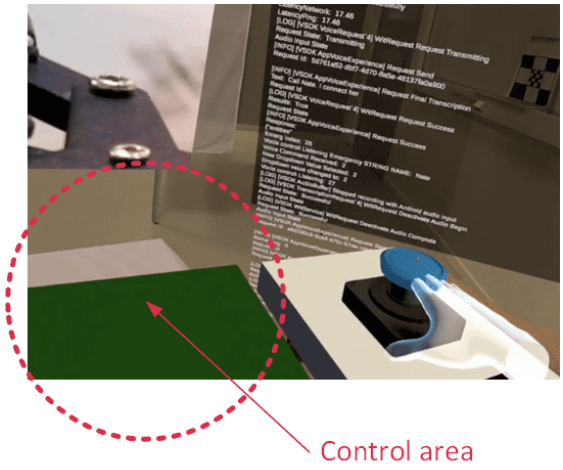


Figure 4.34: Emergency button in the VR application

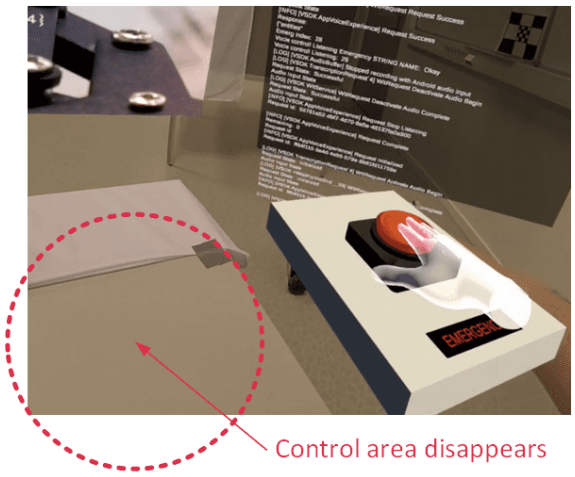
Examples of the emergency button in action within the VR application are shown in Figures 4.35a, 4.35b, and 4.35.



(a) Emergency button in the VR application



(b) Hand detected on the button



(c) Emergency button triggered

Figure 4.35: Emergency button in action

Additionally, the emergency mode can be triggered through a visual menu (see Figure 4.14) or by a voice command detailed in 4.6.8. The user can initiate this mode by selecting the appropriate option in the menu or by saying “Emergency” in English. Both methods lead to the same emergency activation process as pressing the emergency button.
The emergency button is strategically placed on the right-hand side of the control scenes, ensuring easy access for the user to activate it in case of an emergency.

4.3 Robot

OM | *AD*

The term “robot” collectively refers to all software running on the RPi mounted on the physical robot. This software facilitates communication with the VR headset and enables control of the robot. While the robotic arm (refer to 4.4) and object detection (refer to 4.5) are integral parts of this software, they are presented in separate sections in this thesis due to their size and complexity. For details on how the robot code documentation was built, see O.19.

4.3.1 Using ROS 2 **OM** | *AD*

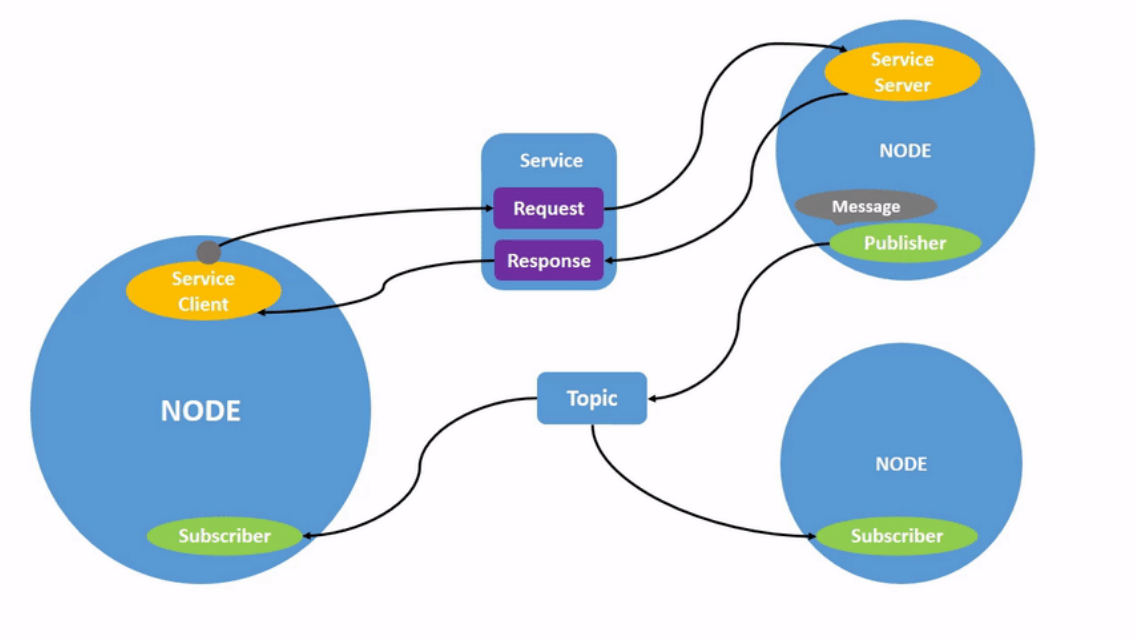


Figure 4.36: Example of ROS 2 Nodes [4]

Basics ROS 2 employs nodes that use topics to publish and subscribe to messages or services to communicate between different nodes. Our implementation primarily uses topics for data transmission, with limited use of services. A ROS node "should be responsible for a single, modular purpose, e.g. controlling the wheel motors or publishing the sensor data from a laser range-finder [...] In ROS 2, a single executable (C++ program, Python program, etc.) can contain one or more nodes" [4]. However, in our implementation, each node is contained within its own executable file. All nodes are written in Python, except for the **Astra camera** node (see 4.6.1). A node can subscribe to or publish on one or more topics, each requiring a specified interface.

Interfaces Interfaces facilitate communication between nodes by being filled and sent as messages. These interfaces can be predefined ROS interfaces or custom ones. For example, the custom-defined message interface **VRDrive.msg** is shown below:

```
float64 x
float64 y
int64 speed
string drive_mode "normal"
```

Each value is separated by a space, and it is possible to have up to three values. The first value defines the data type, the second value is the key for accessing the data when reading or writing the message, and the third value defines the default value if it is not specified when filling in the message.

File structure The nodes are organized into packages. In our implementation, each node is contained within its own package for easy differentiation. However, the example below demonstrates two nodes within the same package:

```
robot/src/
    my_package/
        my_package/
            __init__.py
            example_subscriber.py <-- node
            example_publisher.py <-- node
        resource/
            my_package
        test/
            ... test files
        package.xml
        setup.cfg
        setup.py

    ... more packages
```

To build and run these nodes, they must be structured correctly, as shown above. The code for these two basic Python ROS 2 nodes is shown and explained in O.1.

Building Before a ROS node can be run, the code needs to be built. **Colcon** is used for this purpose, and the code must be rebuilt each time it is changed. However, this does not apply to JSON files, as mentioned in 7.4.

4.3.2 Node architecture

OM | *AD*

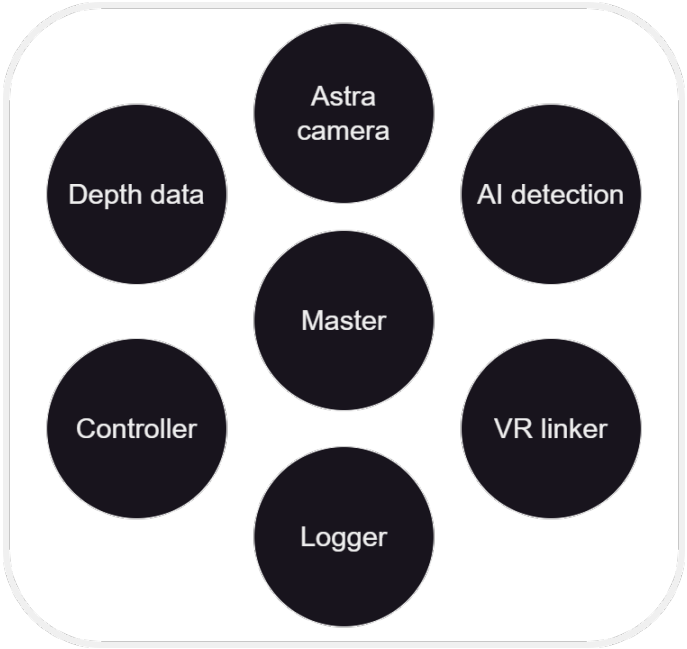


Figure 4.37: All the ROS nodes in the robot system

There are a total of seven nodes running on the robot, each serving a specific purpose. Below is a brief explanation of each node:

- **VR linker node** handles communication with the VR headset, facilitating both the reception and transmission of data. All data exchanged between the robot and the VR headset passes through this node, except for the video stream.

- **Controller node** is the only node which interacts with the expansion board (see 4.3.5), which controls the physical motors and servo. It is within this node that the robotic arm calculations (see 4.4) and driving mechanisms are implemented.
- **Master node** manages the robot's operational modes and adds safety to the system. In specific modes, the robot ignores incoming commands to ensure that driving and controlling the arm simultaneously is not allowed.
- **AI detection node** enables streaming video to the VR headset (see 4.3.4) and implements object detection (see 4.5).
- **Logger node** saves logging information and implements backup logging (see 4.6.7). Logs are primarily used for debugging purposes.
- **Astra camera node** publishes depth information from the depth camera at the highest possible rate. The raw data published by this node is not used directly.
- **Depth data node** listens for raw depth data, processes and compresses it, and eventually sends it to the VR headset through the **VR linker node** (see 4.6.1).

4.3.3 Communicating with the Meta Quest 3 **OM** | *SO*

One of the first tasks undertaken was establishing communication with the VR headset. Initially, we had little to no knowledge about this process and were uncertain if it was even feasible. These doubts stemmed from Meta's software being closed-source and uncertainty regarding what was publicly available. After some research, the Unity package "ROS-TCP-Connector" was discovered, which could connect to an external IP address. However, since the last commit was over two years old [79] and there were difficulties getting it to work, it was decided to develop our own Transmission Control Protocol (TCP) socket implementation. This approach also allowed for greater control over the communication.
It was decided that the RPi would host a ROS node acting as the server (**VR linker node**), with the VR headset serving as the client. This setup seemed more logical, as it made sense for the VR headset to connect to the robot rather than the other way around.

JSON was chosen as the preferred communication format due to its dynamic nature, the group's prior experience with it, and its widespread implementation across various programming languages [80]. JSON uses a key-value format, simplifying the process of adding more data to the message. The receiver can choose to ignore irrelevant data. Below is an example of JSON data:

```
{
    "x": 0.8821,
    "y": 0.1965,
    "speed": 70
}
```

An example of a more compact format is shown below.

```
0.8821,0.1965,70
```

While this format takes fewer bytes, it is less explicit about the data's meaning. This format could lead to issues with type conversion and difficulties in adding new data to the message.

4.3.4 Video stream **OM** | *SO*

The video stream provides visual information, making it easier for the operator to navigate the robot within its environment. It is implemented using a WebSocket server with the "websockets" Python package [81]. The drive camera captures images at a rate of 30 Frames per second (FPS) [82], which are then transmitted over the WebSocket to the VR headset. The specific camera used depends on the robot's current mode, as detailed in 4.3.7. Additionally, each captured image is analyzed using object detection, as explained in detail in 4.5. The initial implementation of this video stream did not use WebSocket and is discussed in 4.6.10.

4.3.5 Interacting with the expansion board **OM** | *SO*

The robot we are using and building on top of is the ROSMASTER X3 PLUS made by Shenzhen Yahboom Technology Co., Ltd. [74] (referred to as "Yahboom"). When referring to the "expansion board", the circuit board made by Yahboom, which is connected to the electronics of the robot (sensors, servos, motors, etc.) is implied. The expansion board can take commands from the RPi over a Universal Serial Bus (USB) serial connection and act on these commands.

Yahboom has developed the firmware running on this expansion board. Additionally, they have made a library called "Rosmaster Library" [83], which works as a driver and can communicate with the expansion board using the *pyserial* Python package. Built on top of this, Yahboom has implemented software that can for instance track a line, use voice control, randomly move the arm and more. These examples are implemented in ROS 1 and/or ROS 2.

The quality of the code and documentation varies, as it has been translated into English. Our client recommended not reinventing the wheel and using what is already available, but building on top of this code or examples was not considered. This would most likely take way more time than building the ROS 2 environment and logic from scratch.

However, the "Rosmaster Library" is used, but there have been made adjustments to it. The driver includes addresses for specific functions, values and methods for sending bytes over serial to the expansion board. On top of this library, a wrapper class has been implemented, making it easier to develop and add more advanced functionality. An example of the modifications that were done is shown in O.2.

A code snippet of this wrapper is shown below.

```
@in_production_mode
def set_arm_elbow(self, angle: int) -> None:
    """Sets the arm elbow angle.

    Args:
        angle: angle
    """
    self.ros_master.set_uart_servo_angle(3, angle)

def reset_arm_elbow(self) -> None:
    """Resets the arm elbow to its default position."""
    self.set_arm_elbow(Preset.ARM_ELBOW_ANGLE)
```

This approach makes interaction with the expansion board more straightforward and explicit. Additionally, users do not have to remember default values or the `servo_id` each time they interact with it. A `@in_production_mode` decorator was also made which throws an exception if the robot is not in production mode (see 7.4).

4.3.6 Handling modes and safety features **OM** | *AD*

The robot can operate in four modes, described below. Note that while production mode ends with "mode," it is not a standalone mode (see 7.4).

- **Idle mode** In this mode, the robot remains stationary and ignores incoming commands. Although commands are received, their handling is terminated prematurely, so the robot does not move or perform calculations.
- **Drive mode** In this mode, the operator can drive the robot. There are three driving sub-modes: normal, precision, and reverse, which are detailed in 4.3.8.
- **Arm mode** In this mode, the operator can control the robot's arm.
- **Emergency mode** In this mode, the robot becomes unresponsive to all commands. Unlike the other modes, switching out of emergency mode requires a reboot, necessitating a physical inspection. This mode is intended for emergencies only.

The master node keeps track of the robot's current mode by storing it in a class variable. Since the robot acts on commands from the VR headset, which can theoretically send any type of command at any time, the robot is programmed to ignore commands that are not appropriate for its current mode. For example, in **Idle mode**, the robot will ignore both drive and arm commands.

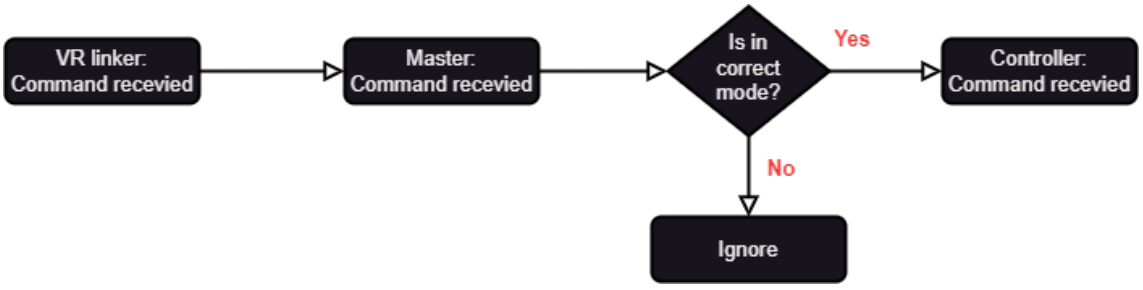


Figure 4.38: Validating commands based on mode

This functionality is implemented in the master node using a function decorator to ensure that the mode matches the expected one.

```
def matches_mode_and_not_emergency(mode: Mode):
    def decorator(func):
        def wrapper(self, msg):
            if self.mode == Mode.EMERGENCY:
                print("In emergency mode, aborting...")
                return

            if mode != self.mode:
                print("Mode mismatch, aborting...")
```

```
                return

            return func(self, msg)
        return wrapper
    return decorator

class Master(Node):
    ...

    @matches_mode_and_not_emergency(Mode.DRIVE)
    def handle_untrusted_vr_drive(self, msg) -> None:
        # code will only run if not in emergency mode
        # AND in drive mode
        ...
```

If the mode does not match the expected mode, the method will terminate prematurely, making the method call ineffective. For instance, this prevents the robotic arm from being controlled while in drive mode. This works because the **VR linker node** publishes messages to an "untrusted" topic, a concept based on naming conventions where untrusted topics start with an underscore.

```
_vr_arm (untrusted)
vr_arm (trusted)
```

Other nodes listen only to "trusted" topics, and the master node only publishes to the trusted topic if the mode matches.
Building code documentation directly from this code can lead to issues, as discussed in O.19. Unit tests for this functionality are detailed in 7.5.2.

Additionally, a function was implemented to make the robot beep if it had not received a command in the last five seconds. This helps indicate whether the VR application is sending commands or if there is a mode mismatch. However, the robot also sends its current mode, as explained in 4.3.10.

4.3.7 Changing modes **OM** | *AD*

When switching between different modes, certain actions are triggered to minimize collisions and maximize the camera view for the application.

Switching camera In most modes, the drive camera is used. However, in arm mode and when reversing (see 4.6.6), the arm camera is used. This ensures that the operator has optimal control. When switching between cameras, the video stream will momentarily freeze for the VR operator.

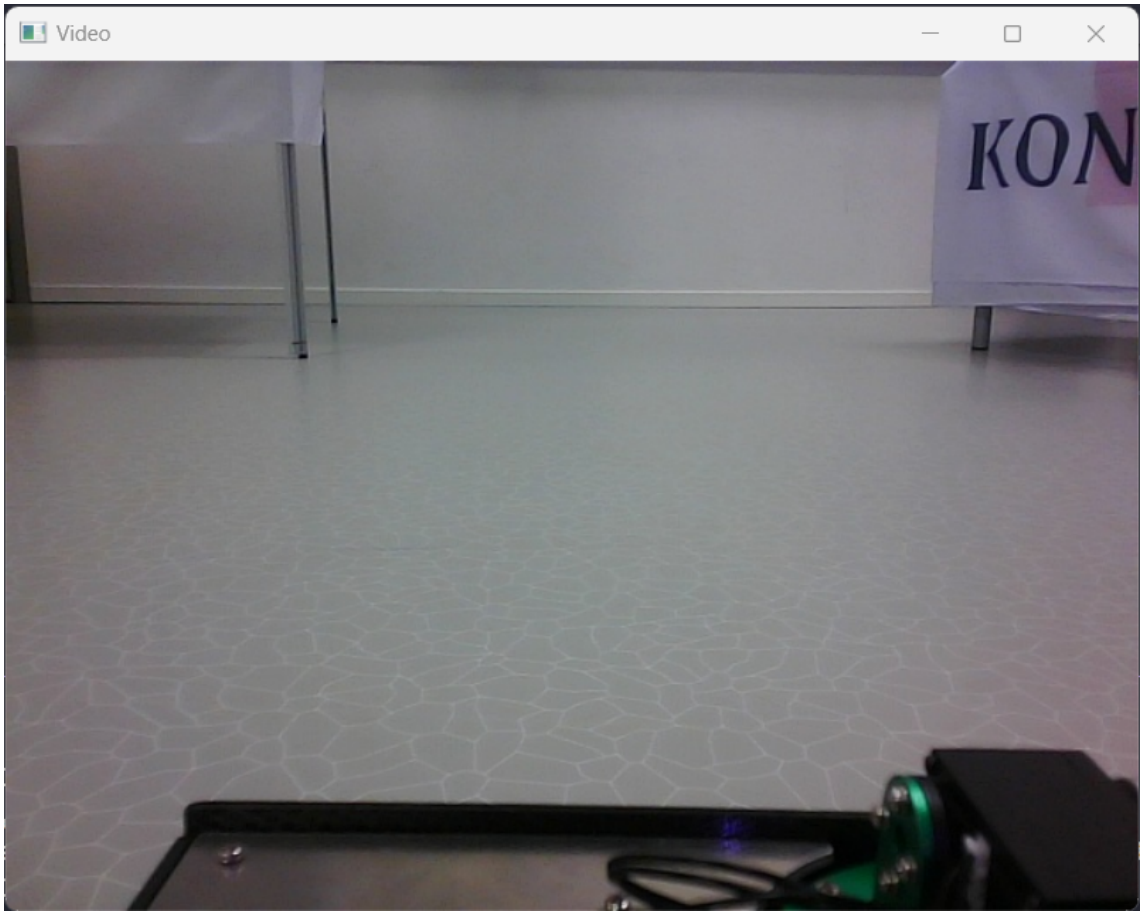
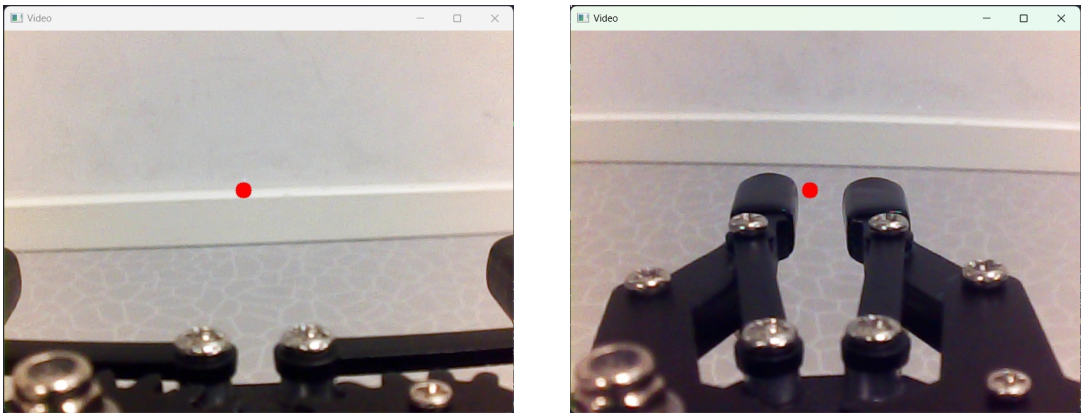


Figure 4.39: Drive camera view



(a) Unpinched **(b)** Pinched

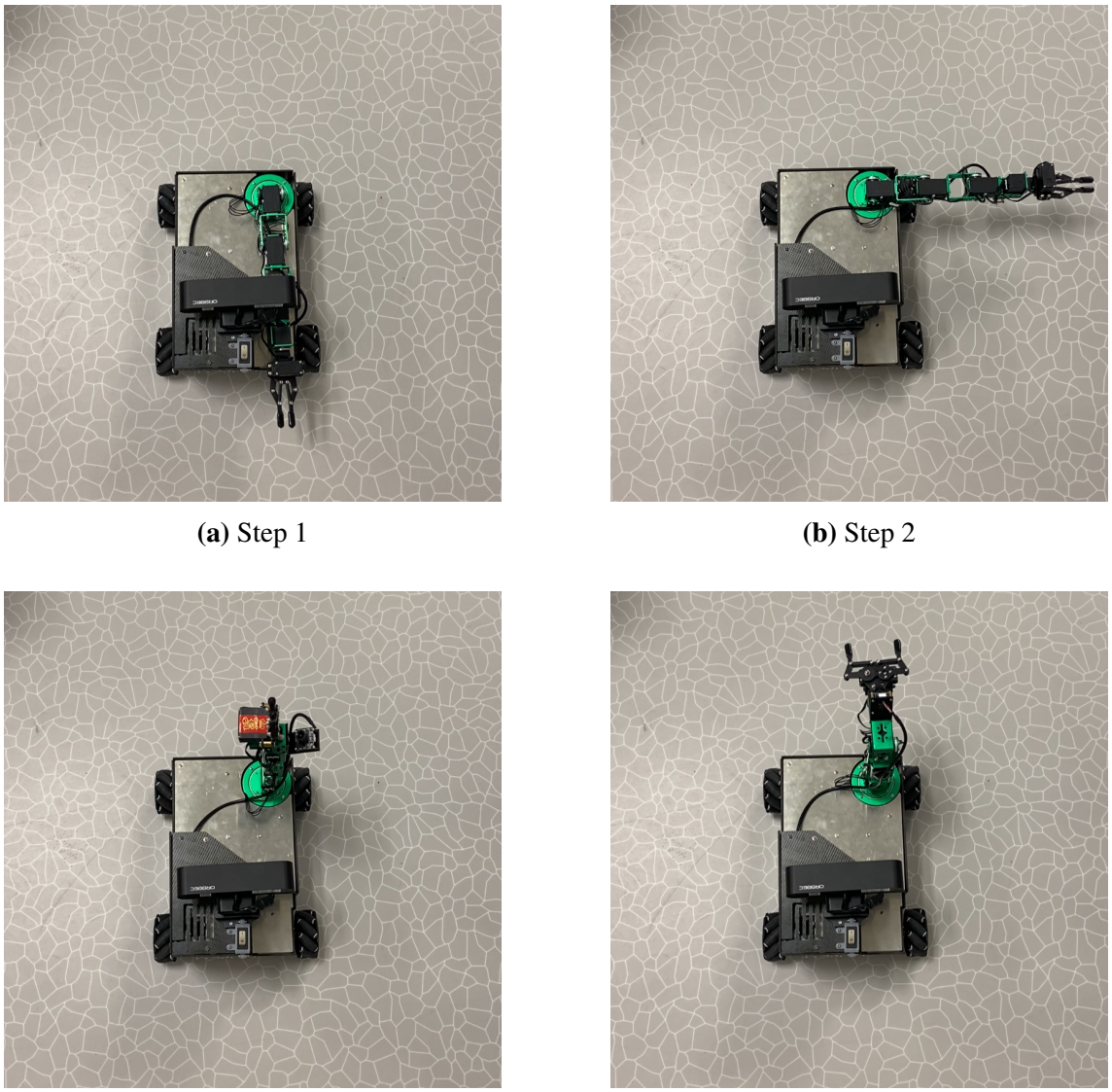
Figure 4.40: Arm camera view with end effector indication

In the arm mode, a red dot is displayed to indicate the position of the end effector, helping the operator understand where an object will be pinched and making arm control easier. The process of determining this point is explained in Section O.11.

Predefined arm movements When switching between the drive and arm modes, the robotic arm is moved. Since there is no requirement to drive and control the arm simultaneously, the arm is positioned away and under the camera when driving. This ensures that the camera view is not obstructed and reduces the risk of damaging the arm in case

of a collision.

When in drive mode, the arm is placed under the camera. In arm mode, the arm is positioned in front. Due to a problem with obtaining the angle of servo 2 (see 4.49), it is always assumed that the arm is under the camera to avoid collisions. This precaution is also applied when rebooting the robot; if the robot is shut down with the arm under the camera, it will not collide upon restarting. Additionally, the robotic arm only moves if the mode changes, preventing unnecessary movements if the VR operator re-sends the current mode.



(a) Step 1 **(b)** Step 2

(c) Step 3 **(d)** Step 4

Figure 4.41: Predefined arm movement when switching from drive to arm

This sequence of movements also occurs when switching from arm to drive mode but in reverse order.

4.3.8 Driving the robot **OM** | *SO*

The robot is equipped with mecanum wheels, allowing it to move in various directions. To provide the operator with optimal control for different scenarios, three drive modes have been implemented.

Figure 4.42: Normal drive mode

Normal In the normal drive mode, the robot behaves like a typical car. It can drive forward, turn right, turn left, and stop. This is the default driving mode because it is the most intuitive. The robot's speed depends on the distance of the operator's hand from a specified point. For example, if the hand is all the way to the left, the robot will turn left as quickly as possible. This allows for both fast and slow driving, providing more precise control. The VR operator's point of view is from the normal forward-facing drive camera.

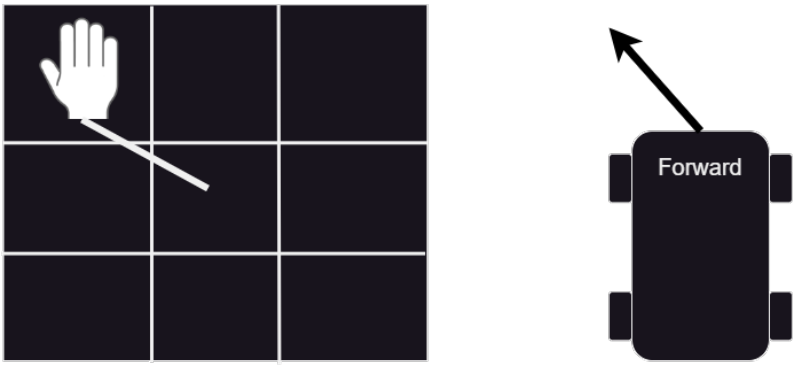


Figure 4.43: Precision drive mode

Precision In precision mode, the robot can move north, south, east, west, northwest, northeast, southwest, and southeast, and it can also stop. The speed is predefined to be 40% of the highest speed, which is relatively slow. This mode allows the operator to drive with high precision, ideal for navigating close to areas where arm operations will take place. The VR operator's point of view remains the forward-facing camera, similar to normal mode.

Figure 4.44: Reverse drive mode

Reverse In reverse mode, the controls are similar to those in normal mode, but the actual directions are inverted. When the operator's hand is in the forward box, the robot drives backward. The same inversion applies to turning left and right; for instance, if the hand is forward left, the robot drives backwards right (from an overhead perspective). This mode is akin to reversing a real car. In reverse mode, the VR operator's camera feed switches to the arm camera, which faces backward acting as a reversing camera (see 4.6.6).

4.3.9 Logging functionality **OM** | *AD*

The logger node subscribes to all topics considered important for logging.

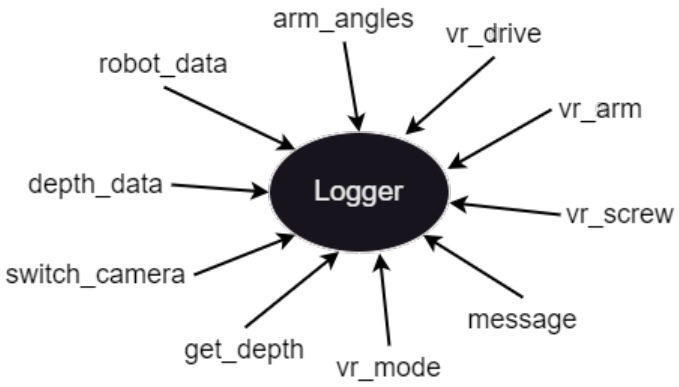


Figure 4.45: Logger node topic subscriptions

The logging is used solely for debugging purposes, allowing the review of data and behavior during or after testing. By retracing the logs, it is possible to identify when and where issues occurred and potentially understand why. These logs are saved to a local `.log` file on the robot, which can be accessed remotely via SSH. Additionally, backup logging is also implemented, as discussed in 4.6.7. An excerpt from one log file is shown below.

```
15/05/2024 19:27:45.734 - [CRITICAL]: Socket connection to VR closed.
15/05/2024 19:27:47.667 - [INFO]: Using settings is_production=Tru...
15/05/2024 19:27:54.814 - [DEBUG]: topic='arm_angles': rotation=90...
15/05/2024 19:27:54.815 - [DEBUG]: topic='robot_data': x=0.0239257...
```

All logs begin with the timestamp and severity level, followed by the topic received and the message contents.

4.3.10 Sending of robot data & status **OM** | *AD*

The robot sends feedback data to the VR operator, including information such as the robot's current mode, battery voltage, speed, and more. This data is transmitted using the TCP socket in the **VR linker** node. The structure of the transmitted data is shown below.

```
{
    "type": "robot_data",
    "accelerometer": [0.03, -0.02, 9.81],
    "gyroscope": [0.5, -0.3, 0.1],
    "magnetometer": [30, -40, 50],
    "motion": [4, 2, 3],
```

```
    "speed": 100,
    "cms_speed": 80.0,
    "voltage": 12.6,
    "battery_percent": 100,
    "mode": "drive"
}
```

This data is retrieved from the expansion board. Values such as `battery_percent` and `cms_speed` are estimated calculations. While not all of the data sent to the VR headset is utilized, the option to use it is available.

Battery percentage estimation It was assumed that at 12.6V the battery is fully charged (100%), and the robot stops if the battery drops below 9.6V (0%) [84]. Assuming a linear relationship between voltage and battery percentage, the graph shown below was derived. There was no requirement for an accurate battery estimation.

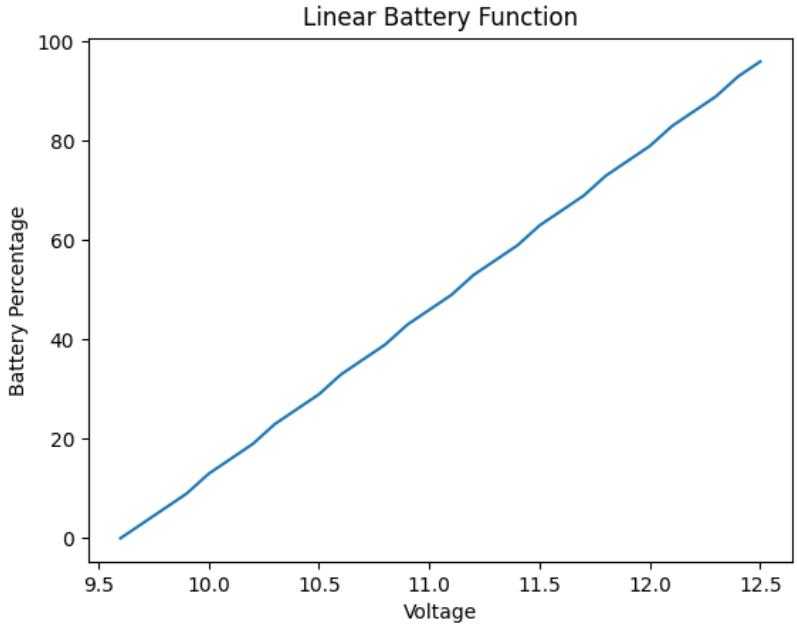


Figure 4.46: Linear battery estimation plot

The equation shown below was used for estimating the battery percentage:

$$Percentage(voltage) := \frac{voltage - 9.6}{12.6 - 9.6} \cdot 100\% \tag{4.1}$$

Before physically testing the Python functionality using the expansion board, unit testing was conducted using the code presented in 7.5.3.

Speed estimation When retrieving the speed from the expansion board, only a percentage is returned, ranging from 0 to 100. The speed of the robot was physically measured, as detailed in O.9, and estimated to be:

Percentage	Velocity (cm/s)
0	0
50	34.72
100	80

Table 4.2: Velocity at percentages

The calculations were done using centimeters per second (cm/s) rather than the more common meters per second (m/s) or kilometers per hour (km/h) units due to the small values involved. The system appeared more responsive when there were larger differences in speed changes.

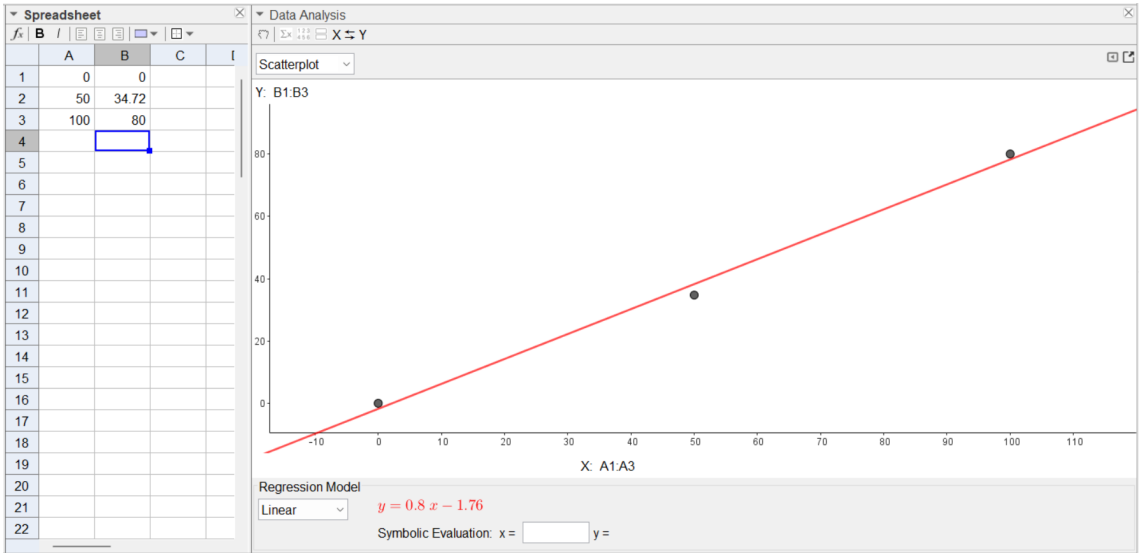


Figure 4.47: GeoGebra linear estimation of the speed

$$Speed(percentage) := 0.8 \cdot percentage - 1.76 \tag{4.2}$$

Due to the output being negative for smaller values, the output is clamped in the Python code.

4.4 Robotic arm **OM** | *AD*

The robotic arm is attached to the robot and can execute tasks such as turning a valve or pressing a button. It is controlled by a VR operator through hand movements. Each time the operator moves their hand, a new coordinate is transmitted to the robot. This coordinate is used to determine and move the arm to the specified location. While there are no precision requirements (see C), there are constraints on the weight and size the arm can lift, as well as maximum latency requirements.

4.4.1 Arm movement concept **OM** | *AD*

The robotic arm is manipulated by the VR operator's hand in VR space. The arm consists of six servos, each requiring a specific angle to provide the operator with sufficient control.

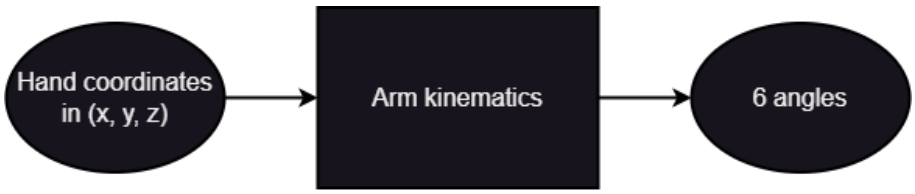


Figure 4.48: Kinematics black box

The coordinate point provided by the VR application represents an absolute point in VR space. This point must be interpreted to correspond to a coordinate in the robot's space.

4.4.2 Servo and arm limitations

OM | *AEH*

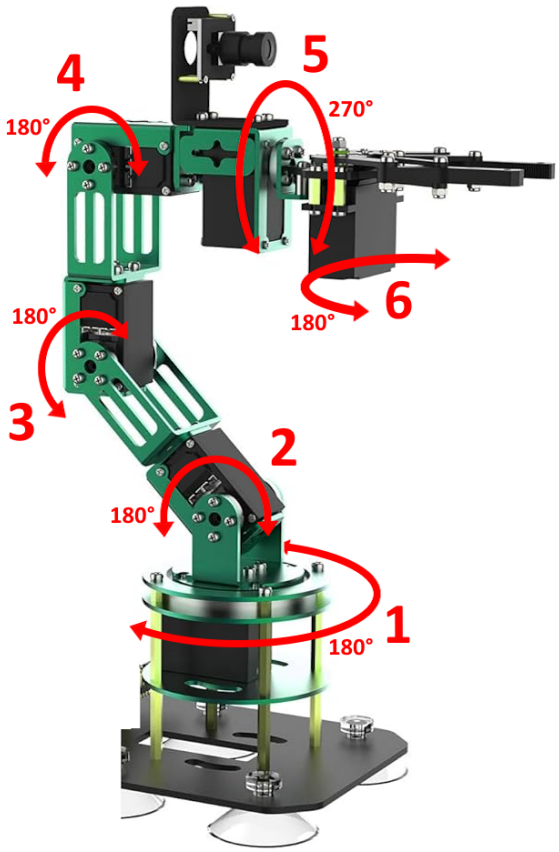


Figure 4.49: Robotic Arm Servo Limitations [5]

The robotic arm includes six servos, each of which is software-limited to a movement range of 180 degrees, except for servo 5, which can rotate 270 degrees. Servo 5 specifically controls the rotation of the pinching segment around its own axis. Servos 2, 3 and 4 can be considered a three revolute (3R) planar manipulator.

4.4.3 End effector placement and arm dimensions

OM | *AD*

To ensure correct angle calculations, a reference point, known as the end effector, is crucial. The placement of this point significantly influences the arm's behavior.

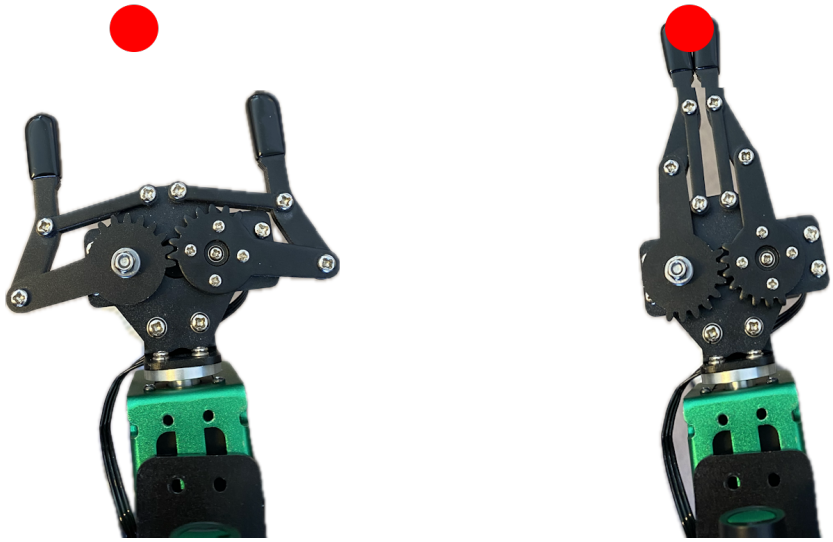


Figure 4.50: End-effector placement

When the arm's "fingers" pinch, the "fingertips" move forward. Therefore, the end effector point is positioned at the tip of the pinching mechanism when it is closed. This placement ensures that the other segments of the arm do not need to retract when pinching an object. This end effector point remains fixed, regardless of whether the pinching mechanism is closed or not. Additionally, accurate measurements of the arm's links are essential for determining the servo angles.

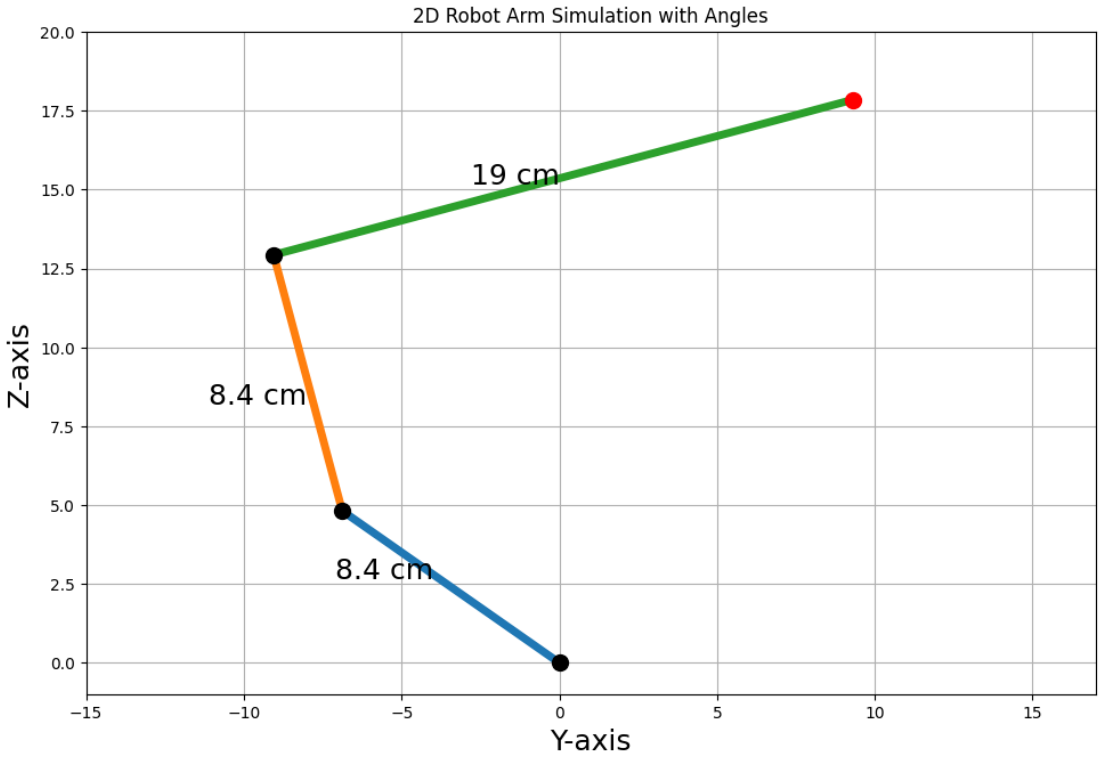


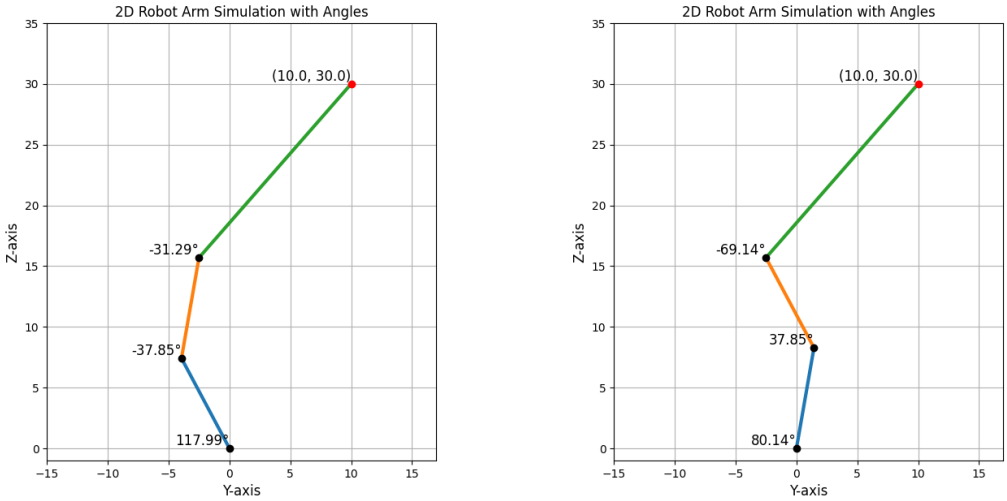
Figure 4.51: Lengths of each arm link

Measurements were taken using a ruler and are approximate. Arm links one and two each measure 8.4 cm from joint to joint. The third and final arm link is 19 cm long from the

joint to the chosen end effector point. The total arm length is 35.8 cm, representing its maximum reach.

4.4.4 Deciding on an arm manipulation method **OM** | *AD*

Determining the angles for each servo introduces complexity, as adjusting one servo's angle subsequently affects the angle of the next servo. Additionally, there may be multiple valid angle configurations for the same point.



(a) Elbow up configuration **(b)** Elbow down configuration

Figure 4.52: Two valid arm configurations in the same point

There are different methods to determine arm angles. Some of these methods are outlined below.

Forward kinematics "is used to calculate the position and orientation of the end effector when given a kinematic chain with multiple degrees of freedom" [85]. In other words, this is the opposite of what we want, as the goal is to determine the angles from a point, not to calculate a point from the given angles.

Inverse kinematics "is a method of solving the joint variables when the end-effector position and orientation (relative to the base frame) of a serial chain manipulator and all the geometric link parameters are known" [86].

Using a framework MoveIt is to our knowledge the most popular and widely used robotics manipulation framework. MoveIt is described as "an open source robotics manipulation framework for developing new applications, prototyping designs, and benchmarking algorithms" [87]. It is used by agencies and corporations such as NASA, Google, and Microsoft [88], and "have received funding from the European Union's Horizon 2020 research and innovation programme" [89]. Implementations are available for both ROS 1 and ROS 2.

Initially, it was thought that the optimal and easiest way to handle arm movements was to use the MoveIt framework. However, attempts to implement this framework led to issues discussed in 9.2.6. Consequently, the group decided to take on the challenge of

implementing inverse kinematics ourselves, which provided the added benefit of greater control and a more challenging development process.

4.4.5 Inverse kinematics

AD | *OM*

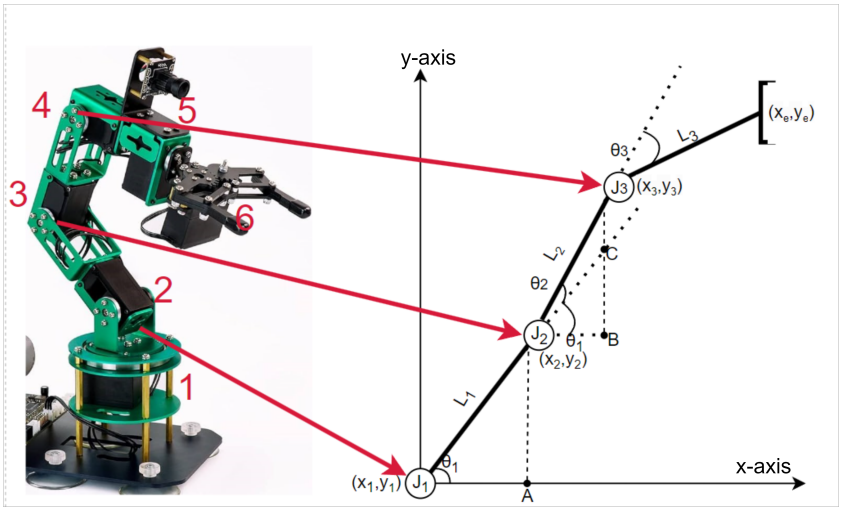


Figure 4.53: 3R Arm

Kinematics for the 3R planar manipulator involves identifying the desired end position or the joint angles. To explain how this is done, one needs to understand forward kinematics.

As seen in the figure 4.53, the servos 2, 3, and 4 in our robotic arm are considered the three joints J_1, J_2, J_3 respectively, and the distance between them have been referred to as L_1, L_2, and L_3, where the L stands for link. The end position coordinates of the arm in the X-Y plane are denoted as (x_e, y_e) and the angle ϕ is the end orientation of the end effector compared to the start point of our arm. Lastly, the joint angles are denoted as θ_1, θ_2, and θ_3. The end position of the end effector is calculated by calculating the coordinates of the three joints.
The coordinates of J_1 is $(0,0)$. The position of J_2 is calculated with the help of trigonometry. If we take the right-angled triangle formed using point A, J_1 and (x_2, y_2) as vertices, we can state that :

$$\cos(\theta_1) = \frac{x_2}{L_1}$$

from which we can calculate x_2:

$$x_2 = L_1 \cdot \cos(\theta_1) \tag{4.3}$$

Similarly, by using the $\sin(\theta_1)$ we can calculate y_2:

$$y_2 = L_1 \cdot \sin(\theta_1) \tag{4.4}$$

Finding (x_3, y_3) involves a bit more trigonometry because it depends on the angles and lengths of both the first and second segments of the arm. This is because looking at the figure 4.53, we can see that point B, J_2 and J_3 can all create a right-angled triangle. The angle $\angle BJ_2J_3$ is the summation of θ_1 and θ_2($\triangle BJ_2C$ is a triangle similar to $\triangle AJ_1x_2$). We

use this to calculate x_3 and y_3:

$$\begin{aligned} \cos(\theta_1+\theta_2) &= \frac{x_3-x_2}{L_2} \\ \Rightarrow x_3 &= \cos(\theta_1+\theta_2)\cdot L_2 + x_2 \\ y_3 &= \sin(\theta_1+\theta_2)\cdot L_2 + y_2 \end{aligned} \tag{4.5}$$

Replacing the values of x_2 and y_2 from the equations 4.3 and 4.4, we have:

$$x_3 = L_2\cdot\cos(\theta_1+\theta_2) + L_1\cdot\cos(\theta_1) \tag{4.6}$$

$$y_3 = L_2\cdot\sin(\theta_1+\theta_2) + L_1\cdot\sin(\theta_1) \tag{4.7}$$

Using the same method, we can find (x_e, y_e):

$$\begin{aligned} x_e &= L_3\cdot\cos(\theta_1+\theta_2+\theta_3) + L_2\cdot\cos(\theta_1+\theta_2) + L_1\cdot\cos(\theta_1) \\ y_e &= L_3\cdot\sin(\theta_1+\theta_2+\theta_3) + L_2\cdot\sin(\theta_1+\theta_2) + L_1\cdot\sin(\theta_1) \end{aligned} \tag{4.8}$$

Each joint rotation affects the overall orientation of the end effector. The total orientation ϕ is therefore the sum of the individual joint angles.

$$\phi = \theta_1+\theta_2+\theta_3 \tag{4.9}$$

Here are the final equations for finding (x_e, y_e), using equation 4.9:

$$\begin{aligned} x_e &= L_3\cdot\cos(\phi) + L_2\cdot\cos(\theta_1+\theta_2) + L_1\cdot\cos(\theta_1) \\ \Rightarrow x_e &= L_3\cdot\cos(\phi) + x_3 \end{aligned} \tag{4.10}$$

$$\begin{aligned} y_e &= L_3\cdot\sin(\phi) + L_2\cdot\sin(\theta_1+\theta_2) + L_1\cdot\sin(\theta_1) \\ \Rightarrow y_e &= L_3\cdot\sin(\phi) + y_3 \end{aligned} \tag{4.11}$$

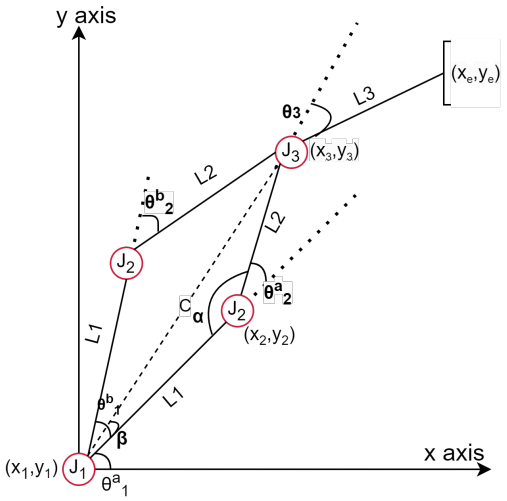


Figure 4.54: inverse kinematics - 3 DOF robotic arm

The principles of inverse kinematics can now be explored. Here we are given (x_e, y_e) and ϕ, and the motive is to find θ_1, θ_2, and θ_3. Initially, from (x_e, y_e), we can find (x_3, y_3) with the help of equation 4.10 and 4.11:

$$x_3 = x_e - L_3 \cdot \cos(\phi)$$

$$y_3 = y_e - L_3 \cdot \sin(\phi)$$

It is important to note that there are two possible configurations that an arm could be in to reach a given end position- "elbow-up" or "elbow-down". These terms describe the position of the manipulator's elbow joint (J_2) relative to the rest of the arm when reaching out for the specified end position. **"Elbow-up"** configuration refers to the position where the elbow joint is above the line drawn between the base and the end effector whereas **"elbow-down"** is where the elbow joint is below the line drawn between the base and the end effector [90]. Therefore we will get two sets of possible angles for the joints. We use the notation θ_1^a, θ_2^a, and θ_3^a for "elbow-down" angles and θ_1^b, θ_2^b, and θ_3^b for "elbow-up" angles.
To mathematically determine the joint angles, we introduce two auxiliary angles, α and β, which represent angles within the triangle formed by the links L_1, L_2 and the line segment C formed by x_1, y_1 and x_3, y_3 as shown in figure 4.54. The length of C is calculated by using the Pythagorean theorem.

$$C = \sqrt{x_3^2 + y_3^2}$$

We find α and β by using the law of cosines, and the law of sines [91].

$$\begin{aligned} \cos\alpha &= \frac{L_1^2 + L_2^2 - C^2}{2 \cdot L_1 \cdot L_2} \\ \alpha &= \cos^{-1}\left(\frac{L_1^2 + L_2^2 - C^2}{2 \cdot L_1 \cdot L_2}\right) \\ \sin\beta &= \frac{L_2 \cdot \sin\alpha}{C} \\ \beta &= \sin^{-1}\left(\frac{L_2 \cdot \sin\alpha}{C}\right) \end{aligned} \tag{4.12}$$

Looking at the figure 4.54, we can see that:

$$\begin{aligned} \alpha &= \pi - \theta_2^a \\ \Rightarrow \theta_2^a &= \pi - \alpha \end{aligned} \tag{4.13}$$

As θ_2^b is measured in the direction of the L_2's movement from the straight-line extension L_2, its value can calculated in this way:

$$\theta_2^b = -(\pi - \alpha) \tag{4.14}$$

To find θ_1^a and θ_1^b, we can look at the triangle formed by x_3, y_3 and C in figure 4.54.

$$\tan(\beta + \theta_1^a) = \frac{y_3}{x_3}$$

$$\Rightarrow \theta_1^a = \tan^{-1}\left(\frac{y_3}{x_3}\right) - \beta \tag{4.15}$$

$$\text{Similarly, } \theta_1^b = \tan^{-1}\left(\frac{y_3}{x_3}\right) + \beta$$

Finally, θ_3^a and θ_3^b can be calculated. As we know from equation 4.9, all the three angles are equal to ϕ , so we just solve for θ_3^a and θ_3^b.

$$\theta_3^a = \phi - \theta_1^a - \theta_2^a$$

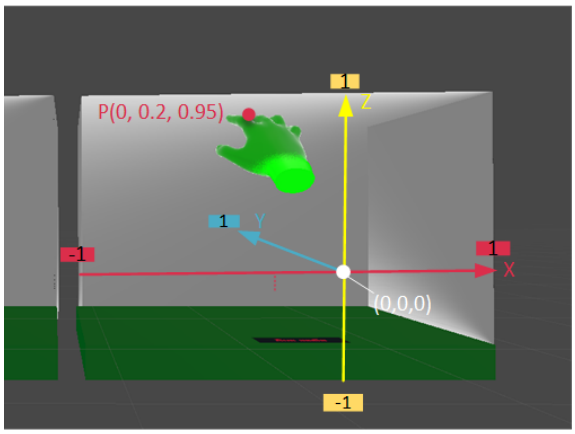
$$\theta_3^b = \phi - \theta_1^b - \theta_2^b \tag{4.16}$$

4.4.6 Implementation

OM | *AD*

The inverse kinematics are implemented in Python using the numpy package. For details on the initial implementation presented in our second presentation, see J.7.4. The entire implementation is available on our GitHub repository [92].

Inputs & outputs When the operator moves their hand in VR space, the robot receives data as shown in the example below.



(a) Operator's coordinate space


```
{
    "x": 0.0000,
    "y": 0.2000,
    "z": 0.9500,
    "strength": 0.4113
}
```


(b) JSON data received

Figure 4.55: VR operator's hand and corresponding JSON data received

In this data, x and z range from $[-1, 1]$, while y and *strength* are in the interval $[0, 1]$. The x, y, and z values represent the coordinates of the operator's hand in VR space, and *strength* indicates how strongly the operator is pinching their fingers, with 0 meaning the fingers are far apart and 1 meaning the fingers are touching. Based on this input, the system should output an angle for each servo, resulting in six angles in total. The inverse kinematics calculation requires three inputs and produces three outputs.

If we imagine that x affects only the rotation of the arm, the y and z values serve as two of the three inputs for the inverse kinematics, corresponding to x_e and y_e, respectively.

The final input, ϕ, is explained later in this section. These inputs represent the desired y position, z position, and "attack angle" of the arm with respect to the y-axis.

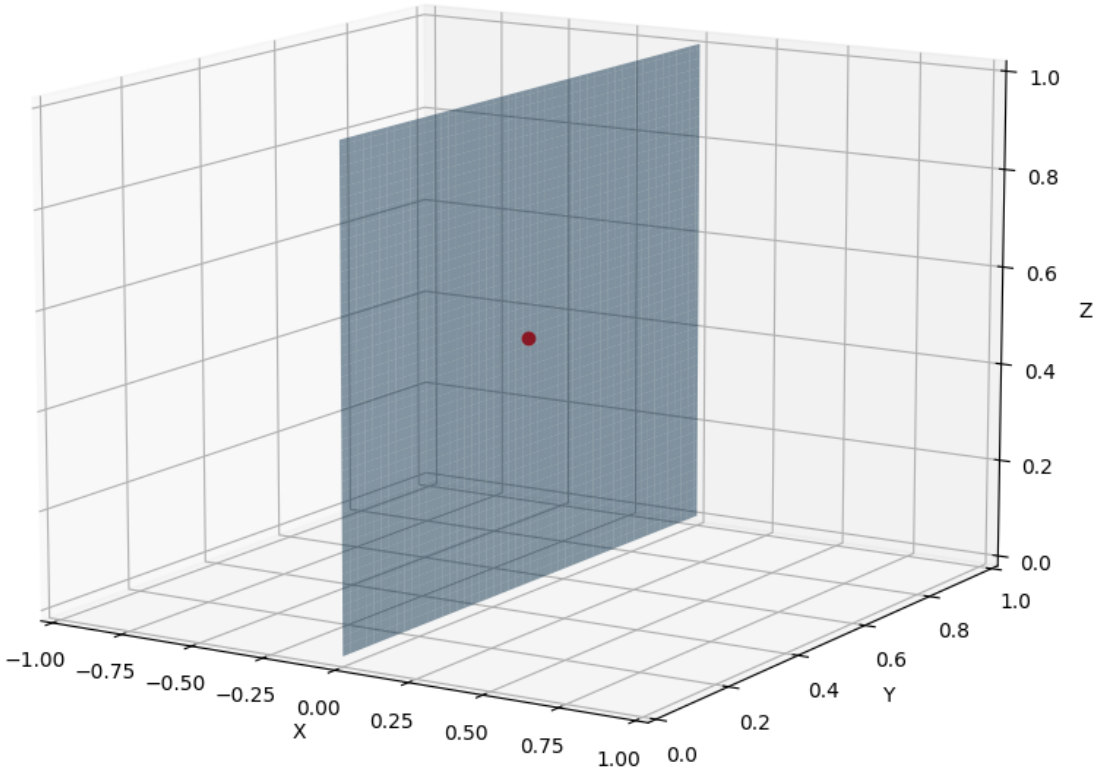


Figure 4.56: Coordinate system showing the YZ-plane with the point (0, 0.5, 0.5)

The y and z coordinates are converted to physical coordinates by multiplying the values by the length of the arm.

```
y: 0.2000 * 35.8 = 7.16
z: 0.9500 * 35.8 = 34.01
```

Since the length of the arm is defined in centimetres, the coordinates are now also specified in centimetres. This conversion simplifies working with the coordinates and defining safety limits (see 4.6.9). The three outputs θ_1, θ_2, and θ_3 correspond to servos 2, 3, and 4, respectively (see 4.53).

Rotating the arm The rotation of the arm is determined by the x value, which has to fall within the interval [0, 180]. This calculation is performed prior to any inverse kinematics operations to ensure responsive arm movement. However, it is important to note that there are many points that the arm cannot reach, which will be explained later. Since the x coordinate ranges from -1 to 1, it needs to be converted to degrees. The conversion process is illustrated below:

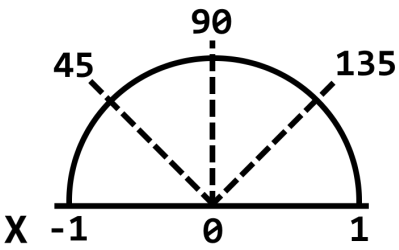
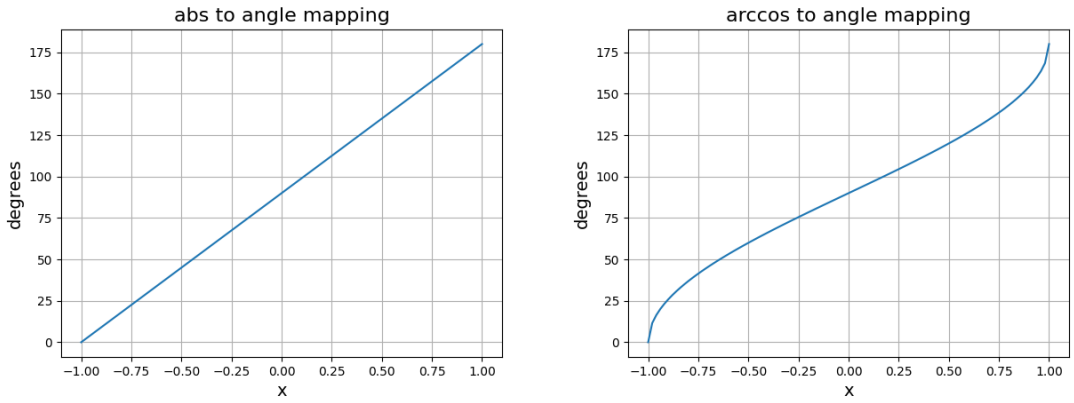


Figure 4.57: Angles related to X value

This conversion can be implemented in Python using either of the following methods:

```
abs(-90 * x - 90)
# or
np.rad2deg(np.pi - np.arccos(x))
```

The two implementations yield slightly different results, as shown in the plots below:



(a) Not using arccos **(b)** Using arccos

Figure 4.58: Precision differences in x to angle calculation

The linear option was chosen as the better fit because the x coordinate received is precise. This choice leads to greater precision at the extrema angles (close to 0 and 180).

Calculating the attack angle Due to the lack of a reliable method for extracting the angle of the hand in the VR space, the ϕ angle is calculated solely based on the Z-coordinate (height) of the operator's hand. The ϕ value is determined according to the plot below.

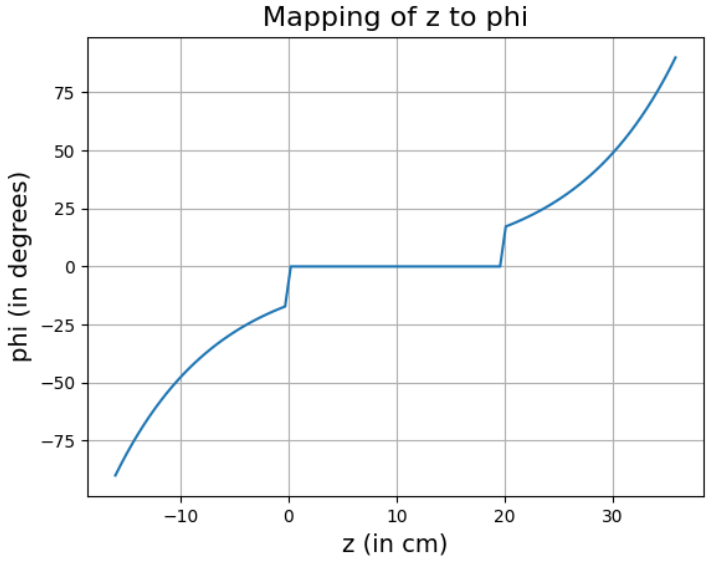


Figure 4.59: ϕ function based on Z-coordinate

This mapping can be interpreted as follows: the lower the Z-value, the more the end effector points towards the ground (down to -90 degrees relative to the Y-axis). The higher the Z-value, the more the end effector points towards the sky (up to 90 degrees relative to the Y-axis). When the Z-value is between these extrema, the end effector points horizontally forward (0 degrees relative to the Y-axis).

The reasoning behind this is practical: if the arm needs to reach something on the ground (low Z-value), the end effector should point downwards to see the object and grasp it from above. Conversely, for a higher Z-value, the end effector should point upwards. For most interactions, the end effector will point forward, making it easier to handle objects that require rotation.

Calculating the θs The calculation of the three theta angles relies solely on the y and z values, along with the previously calculated phi angle. The output of these calculations varies based on arm configuration and can fall into one of several cases:

- Mathematically impossible with real numbers
- Physically impossible
- Impossible due to servo limitations
- Possible

Each case must be accounted for and handled. Additionally, two extra cases are checked: whether the robot is fastening or if the end effector point is illegal, as elaborated in 4.6.5 and 4.6.9.

After obtaining the ϕ angle, a partial point is calculated to determine the correct end effector position:

```
partial_point_y = y - LINK_THREE_LENGTH * np.cos(phi)
partial_point_z = z - LINK_THREE_LENGTH * np.sin(phi)
```

Using these values, the elbow-up configuration can be computed:

```
elbow_up_degrees = calculate_elbow_up_degrees(
    partial_point_y, partial_point_z, phi
)

if not any_nan(elbow_up_degrees) and not out_of_range(elbow_up_degrees):
    return elbow_up_degrees
```

Only if the calculation is impossible will the elbow-down configuration be computed, this is due to how the arm is designed and mounted on the robot. Both configurations are calculated to optimize the number of possible points, considering that there are already many impossible points due to the arm's design.

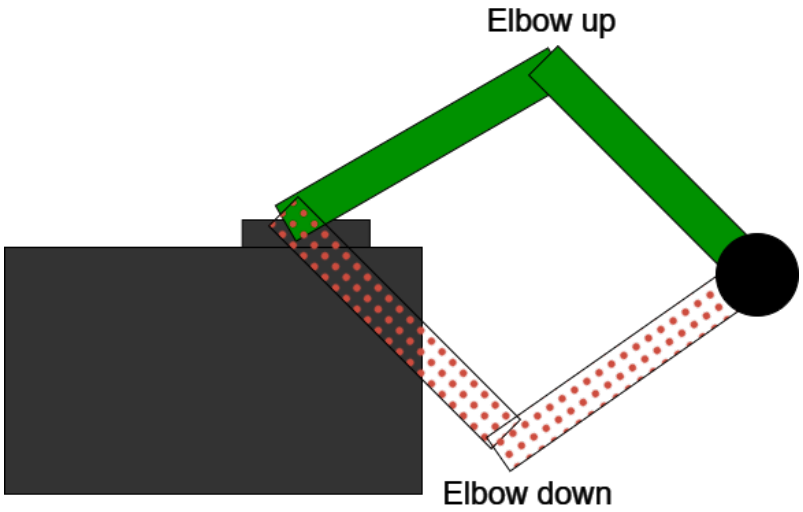


Figure 4.60: Elbow down is illegal for this point

If neither configuration yields feasible angles, `(None, None, None)` is returned, indicating that the arm will not move and will wait for the next hand coordinate. Conversely, if feasible angles are found, they must be converted to match the values expected by the servos.

Converting the angles Two of the three θ angles cannot be directly set to the servos because their values fall outside the expected range. The servos expect positive integers within [0, 180], while the output from the calculation yields ± 90 angles as float values. However, θ_1 can be converted to an integer and set directly to the servo as it falls within the interval due to its relation to the y-axis.

For θ_2 and θ_3 angles, 90 degrees need to be added to the output, and then the result is converted to a whole number.

Pinching The servo angle of the pinching mechanism is determined by the strength value received from the VR application, which varies within [0, 1]. Calculating the angle is straightforward:

```
int(strength * 180)
```

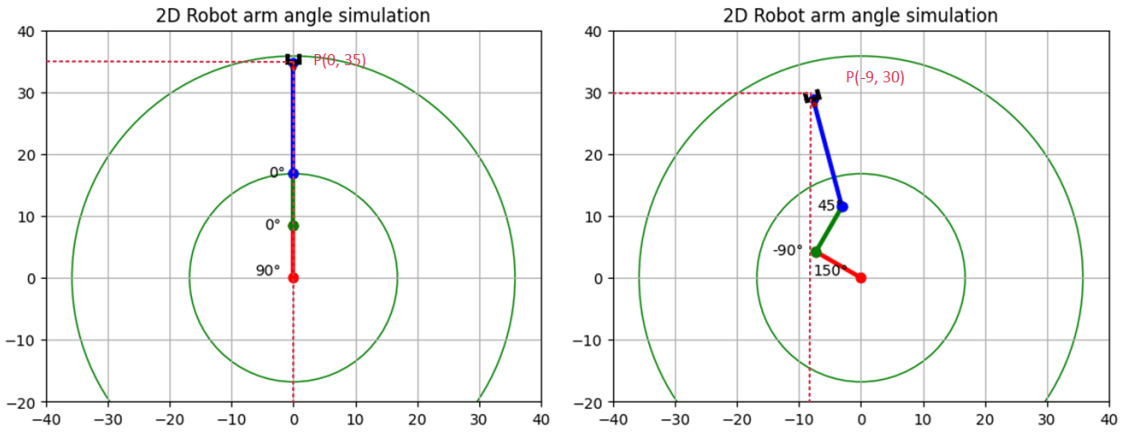
However, this particular servo tends to malfunction if the angle is less than 45 degrees, even though 0 is technically an allowed angle. To prevent damage of the servo, the angle is clamped within the [45, 180] interval.

4.4.7 Simulation

SO | *OM*

To visualize and test the inverse kinematics code, we developed a Python script using the matplotlib library. This script visualizes the output from the inverse kinematics calculations, which determine the angles for joints 1, 2, and 3 based on given (y, z) coordinate. The resulting angles are then fed into the simulation to ensure that the robot's end-effector moves accurately to the specified (y, z) points.

Figure 4.61a shows the simulation result for point p(0, 30), and Figure 4.61b for point p(-9, 30). We conducted the simulation with several (y, z) coordinates and animated the sequence to demonstrate the robot arm's response to each set of test angles.



(a) Simulation at point: p(0, 30)

(b) Simulation at point: p(-9, 30)

Figure 4.61: Simulation of arm angles in Python

4.5 Artificial Intelligence: Object Detection

AD | *OM*

Object detection is one of the most useful and practical applications of AI. It is a difficult technique as one needs to do two things: classify the object from an image and localise it, *i.e.*, determine exactly where the object is in the image. Figure 4.62 shows a simple black box of this process.

Figure 4.62: Object detection black-box

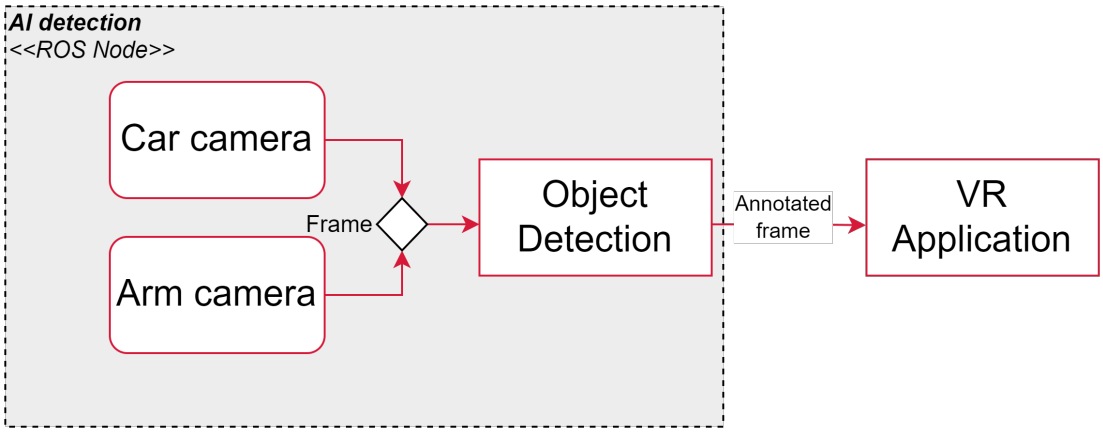


Figure 4.63: High-level representation of AI detection in the system

Figure 4.63 illustrates how the AI module interacts with the broader system. The ROS node responsible for AI detection receives one frame at a time, from either the car camera (when in "Drive" mode) or the arm camera (when in "Arm" mode). This frame is analysed through object detection models, which return an annotated frame with the detected objects and their locations. In addition, information about the object is extracted from a database and added to the frame (see 4.6.14). This frame is then sent to the VR application (see 4.3.4).

To implement object detection in our project, a thorough understanding of how the technology works is required. Therefore, this subsection deals with how it is integrated into the system, the explanation of neural networks which are the building bones of a detection model, types of object detectors, our implementation, and the database connection to display information about the detected objects of interest. To see the early iterations of how the object detection model was made and tested, see Appendix M.

4.5.1 Neural networks

AD | *OM*

Object detection employs supervised learning as referred to in 1.3.3. One of the algorithms typically used in this technology is called artificial neural networks (ANN). Directed links connect the nodes in the neural network. Every node receives a form of activation from the nodes it is connected to and each link has a weight that influences how strong or weak the activation passed from one node to another is. The function that processes the sum of the inputs is called an **activation function**. It decides how a node should react to the sum of its inputs [6, p. 751]. If a node n outputs a value a_n, and the weight between n and another node m is denoted by $w_{n,m}$, then we have the equation:

$$a_n = g_n(\Sigma_m w_{n,m} \cdot a_m) \tag{4.17}$$

where g_n is the activation function [6, p. 752].
Another feature of a network is a **bias**. A bias represents the relevance of a node [93]. A bias term, often denoted as $w_{0,n}$ for node n, allows the network to adjust the activation of a node independently of its inputs. Equation 4.18 is the updated representation of the output value a_n for a node n with a bias b_n:

$$a_n = g_n(\Sigma_m w_{n,m} \cdot a_m) + w_{0,n} \tag{4.18}$$

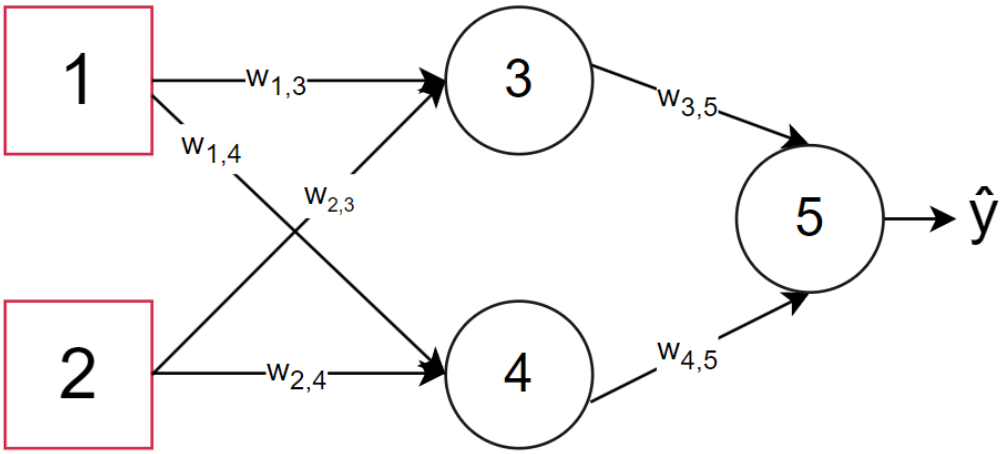


Figure 4.64: A simple neural network with two inputs (the boxes), one hidden layer of two units and two outputs (circle 5) [6, p. 804]

An elementary neural network can be seen in figure 4.64. The network in Figure 4.64 has two input layers, two hidden layers and one output layer. **Hidden layers** are the intermediary stages between a neural network's input and output layers. The depth of a network refers to the number of hidden layers it contains, whilst the width refers to the number of neurons or nodes in each hidden layer. A network with multiple hidden layers is called a **deep** neural network [94]. The activation function allows neural networks to make complex decisions because they are nonlinear. The output one tries to predict does not change linearly with the inputs. Therefore the activation functions are important to neural networks [6, p. 803]. An example of an activation function is the Rectified Linear Unit (ReLU) function which interprets the positive part of its argument:

$$\text{ReLU}(x) = \max(0, x) \tag{4.19}$$

An expression of the output $\hat{y}$ that 5 produces can be expressed as follows:

$$\hat{y} = g_5(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \tag{4.20}$$

$$\Rightarrow g_5(w_{0,5} + w_{3,5}g_3(w_{0,3} + w_{1,3}1 + w_{2,3}2) + w_{4,5}g_4(w_{0,4} + w_{1,4}1 + w_{2,4}2) \tag{4.21}$$

4.5.2 Loss functions and gradient descent

AD | *OM*

After the Artificial Neural Networks (ANNs) have been formed, we need to train it so that it can learn to perform the task it has been designed for. The training process also allows the ANN to discover patterns and relationships in the data. When we start training the network, the weights w_i are all randomized. The key to this learning process lies in adjusting the network's weights so that its predictions closely match the known "correct" and labelled data. This is where loss functions and the method of gradient descent are used.

The loss function is a formula that quantifies the difference between the network's predictions and the actual target values [95]. Many loss functions are used in training an ANN. An example of a loss function is the Cross-Entropy Loss or Categorical Cross Entropy Loss. If we assume that we have a set of data for which the target value for a point is y, and the output value produced by the network is $\hat{y}$, then cross-entropy loss states:

$$L = -y \cdot \log(\hat{y}) \tag{4.22}$$

And the combined loss function for the whole network:

$$TotalL = \sum_{i=1}^{N} -y_i \cdot \log \hat{y}_i \tag{4.23}$$

where N is the total number of data points [96].

The goal is to minimize this loss function as much as possible. This is achieved through **gradient descent**, an optimization algorithm that iteratively adjusts the weights of the neural network to move towards the minimum of the loss function [97]. Each adjustment step is scaled by a **learning rate**, which controls the magnitude of the change. The learning rate can be decided when designing the network. To calculate the gradient of the loss concerning the weights, we need to use the chain rule as seen in equation 4.24 [6, p. 805].

$$\frac{\delta g(f(x))}{\delta x} = g'(f(x))\frac{\delta f(x)}{\delta x} \tag{4.24}$$

$$\frac{\delta}{\delta w_{4,5}} L = \frac{\delta \hat{y}}{\delta w_{4,5}} \cdot L'$$

The calculations in 4.25 show us the gradient of how the loss function will change if we change $w_{4,5}$ from the neural network in Figure 4.64. The notation Δ_5 is equal to $L' \cdot g_5'(in5) \cdot a_4$, to represent the "perceived error" in an easier way. If Δ_5 is negative, that means we need to increase $w_{4,5}$ to reduce the loss. The product $\Delta_5 \cdot a_4$ gives us the direction and the magnitude of the weight adjustment for $w_{4,5}$. a_4 acts as a scaling factor for the gradient. If a_4 is big, then the gradient contribution from $w_{4,5}$ is significant, emphasizing that larger activations magnify the impact of changes in weights. The bias is also a parameter that is learnt along with the weights [93].

$$= \frac{L'}{\delta w_{4,5}} \cdot \delta(g_5(in5)) \tag{4.25}$$

$$= \frac{L' \cdot g_5'(in5)}{\delta w_{4,5}} \cdot \delta(w_{0,5} + a_3 w_{3,5} + a_4 w_{4,5}) \tag{4.26}$$

$$= L' \cdot g_5'(in5) \cdot a_4 \Rightarrow \Delta_5 \cdot a_4 \tag{4.27}$$

If we follow the same calculation logic from equations in 4.25 to calculate Δ_3, then we see that it is:

$$\Delta_3 = \Delta_5 \cdot w_{3,5} \cdot g_3'(in3)$$

We observe that the "error" from 5 is passed backwards. This is why this process is called **back-propagation** for the way that the error at the output is passed back through the network [6, p. 806]. By iteratively adjusting the weights of the network based on the calculated gradients, the network is pushed towards minimizing the loss function and making more accurate predictions. The back-propagation algorithm enables the efficient computation of these gradients, propagating error information backwards through the network's layers.

While the fundamental principles outlined above provide a solid foundation, it is important to note that numerous advanced optimizations and variations of gradient descent exist. For those seeking a deeper understanding, Chapter 19 of "Artificial Intelligence: A Modern Approach" offers an extensive exploration of this topic [6, p. 669].

There are many types of ANNs. The type of ANN that is most relevant to our project, and the one that is most used in object detection is called CNN, which is explained in 4.5.3.

4.5.3 Convolutional Neural Networks **AD** | *OM*

Several key parameters must be considered when choosing an ANN for analyzing images and detecting objects. An important aspect to consider when analyzing an image is that it cannot be treated as a mere vector of pixel values for input due to the significance of pixel adjacency in making accurate predictions. For example, let us take a picture of a dog and observe the area depicting its eyes. That area is not just a collection of random pixels but an arrangement where each pixel is contextually linked to the pixels around it. Another important feature of images is that the same features can be present in several parts of the image. Continuing with the example of a dog, the eyes appear in different parts of the picture but the network should still be able to recognize them as eyes. This is known as translation/spatial invariance [6, p. 811].

So how does one tackle this when we want to recognize objects in an image or classify an image? That is where we use an ANN called **CNN** which is a deep network that learns features, such as an eye with the help of **kernels/filters** that systematically traverse the entire image. A filter is just a pattern of weights (represented as a matrix of weights) repeated across multiple regions and each filter has a bias [6, p. 811]. This solves the spatial invariance problem as the filter is applied to the whole image and also solves the issue of pixel adjacency as the filter is applied to a small, local region of an image at a time. The process of applying filters to an image is called **convolution**, hence the name of CNN [6, p. 811]. These characteristics make CNNs very powerful for tasks like object detection, where understanding visual patterns is essential.

4.5.4 How convolution works **AD** | *SO*

We can try to understand the process of convolution with an example. As a filter slides over an image, it computes the dot product between the filter and the local regions of the input, creating a feature map called the output. An image is a matrix of pixels. Let us assume that the image is black-and-white and a simple 5 x 5 matrix where each entry in the matrix represents the intensity of a pixel, where values typically range from 0 (black) to 255 (white). Let us also have a 3 x 3 matrix filter that contains randomized weights at the start.
The first local region of the image corresponds to the top-left 3 x 3 segment of the image matrix. This is then multiplied with the filter matrix. The region is then shifted right and multiplied by the filter matrix again. When at the end of the row the region starts at the next row and the same steps are followed. The result of the finished multiplication is a feature map of dimension 3 x 3 [96]. The first two steps of the convolution process are shown in 4.65.

$$\text{Image Matrix}(I) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 7 & 15 & 0 \\ 0 & 9 & 14 & 8 & 0 \\ 0 & 12 & 17 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{Filter Matrix}(F) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10 & 7 \\ 0 & 9 & 14 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 50 & \\ & \end{bmatrix}$$

(a) First convolution step

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 7 & 15 & 0 \\ 0 & 9 & 14 & 8 & 0 \\ 0 & 12 & 17 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 10 & 7 & 15 \\ 9 & 14 & 8 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 50 & 0 \\ & \end{bmatrix}$$

(b) Second convolution step

Figure 4.65: Convolution example with 2 steps

This example shows how the process is done for a grey-scale image. Colour images are different as they have three matrices (often referred to as channels) representing the Red Green Blue (RGB) components of the image. For colour images, a filter is applied to each matrix independently. The filter matrix for colour images thus has the dimensions of h x w x 3, where h and w are the height and width of the network respectively. Each matrix's results are combined to produce a single two-dimensional feature map [96].

4.5.5 Architectural components of data transformation in CNNs **AD** | *OM*

In the realm of CNNs various architectural components play important roles in transforming input data into desirable outputs. These components systematically modify the input image to preserve important features while reducing dimensionality or adjusting the focus to specific traits of the input. Listed below are some of the components.

Padding: A process of adding layers of zeroes or other values outside the input matrix. The main purpose is to ensure that the feature map also has the same size as the original input matrix [96]. Figure 4.66 shows an example of padding.

$$\text{(I) without padding} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\text{(I) with padding layer of zeroes} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.66: Padding example

Stride: A parameter that determines the way that the filter moves across the image. The stride determines how many units the filter shifts at each step. So a stride of two moves the filter two regions at a time [96].

Pooling: Pooling is the process of summarizing a set of adjacent values with a single value. This operation is fixed, not learned. One way of pooling is called average pooling (Figure 4.67), where the statistical mean is taken from all the inputs [6, p. 813].

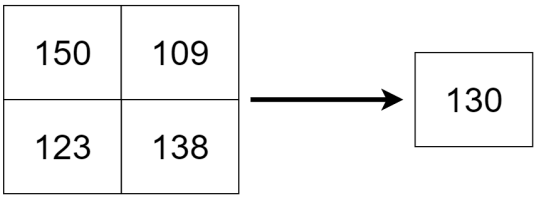


Figure 4.67: Example of average pooling from 4 adjacent values

Fully connected layers: Fully connected means that all nodes in one layer are connected to the outputs of the next layer. This layer is where the input matrix is flattened to create a single vector which is the input for the next layer [96].

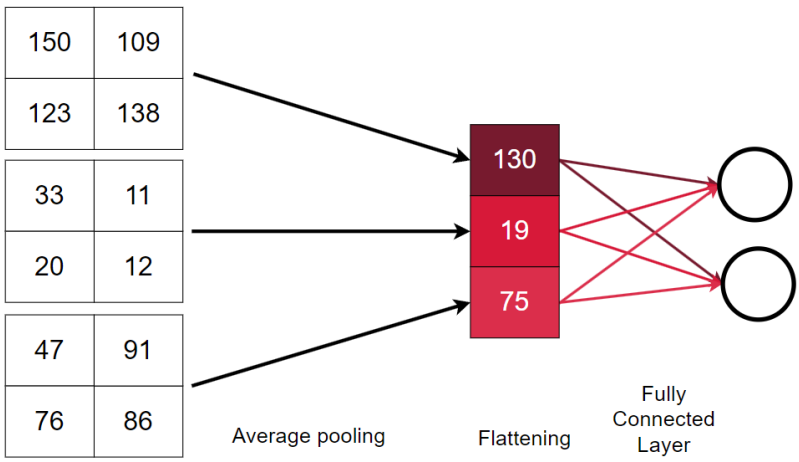


Figure 4.68: Fully connected layer

Softmax: The layer where decimal probabilities to each class are assigned. If one has several classes in their network, this process assigns probabilities to each class, which all need to add up to 1.0. Equation 4.28 shows the equation of the softmax:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.28}$$

where z_i is the probability for a particular score i, z is the matrix containing the raw output scores for each class from the fully connected layer, K is the total number of classes in the network, and e is the base of the natural logarithm. The denominator represents the summation of the exponentials of all the output scores to normalize the probability value [98].

4.5.6 Putting together a simple CNN

AD | *OM*

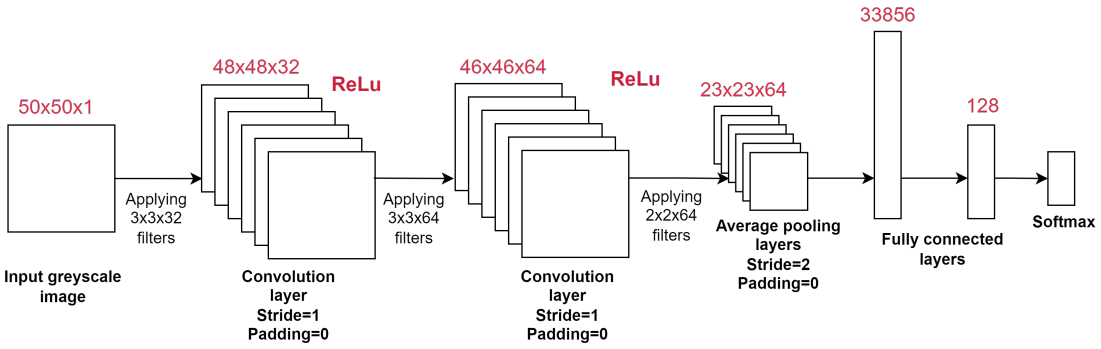


Figure 4.69: A simple CNN

Figure 4.69 illustrates the architecture of a very simple CNN. The formula to calculate the dimensions of a feature map:

$$(n \times n) * (f \times f) = (\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1) \tag{4.29}$$

where n is the size of the input, f is the size of the feature map, p is the padding size, and s is the stride size. We input a grey-scale image with dimensions 50 x 50 x 1 and apply a series of 32 filters with size 3 x 3, which creates a feature map with size 48 x 48 x 32. After each convolutional layer, the activation function ReLU is used.
The second convolution layer uses 64 filters with dimensions 3 x 3, resulting in a feature map of 46 x 46 x 64. It is average pooled using 64 2 x 2 filters, flattened into a single vector of size 33,856. This serves as input to a fully connected layer of 128 nodes. Lastly, the softmax function converts the output to probability nodes based on the number of predefined classes.
This is a very simple CNN. Modern CNNs used in object detection have multiple hidden layers and use different ways of convolution, which is explained in 4.5.8.

4.5.7 Transfer learning **AD** | *OM*

Building an accurate CNN requires a huge amount of data. It also takes a long time when one starts from random data to make the filters detect edges, patterns, features, etc. Therefore in practice, it is rare to build an entire CNN from scratch. Instead, it is common to use an already trained network that has been trained on a very large dataset.
Training large-scale networks on expansive image datasets such as the Common Objects in Context (COCO) dataset often necessitates weeks-long computational efforts. This dataset comprises 80 distinct object classes and encompasses a training corpus of approximately 330,000 images [99]. A network that has been trained on that much data would have useful filters such as edge detectors, pattern detectors, etc. that can be applied to other image domains. For example, a network trained to detect apples and oranges can be used to detect other similar fruits. This entire concept is called **Transfer Learning**, and the network we build on top of is called a **pre-trained** or **base model** [96]. The word model is a term that encompasses the entire system designed to detect objects [100]. It is where we use pre-trained models in other domains to achieve faster training times and great accuracy.

There are two major types of transfer learning, feature extraction and fine-tuning. Both are quite similar, however, the transfer learning pipeline provided by the framework that was used in our project (explained in 4.5.12) uses fine-tuning. **Fine-tuning** involves modifying and retraining a pre-trained model to adapt it to a specific task. It involves adjusting the parameters of the pre-trained model to fit the end task. The first few layers or the lower layers of the pre-trained model are frozen, which means that it is not changed. This is because the lower layers of the CNN are the layers where it learns to detect edges, patterns, blobs etc which is what we wish to retain. However, the top layers are where it learns the specifics of the objects it is trained to detect which we update to suit our dataset. Figure 4.70 shows an example of what the frozen and trainable layers could look like when using fine-tuning.

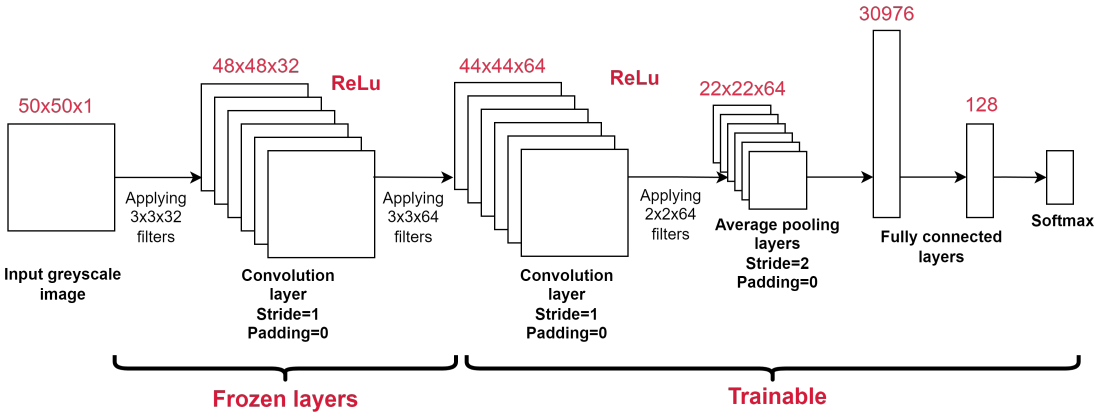


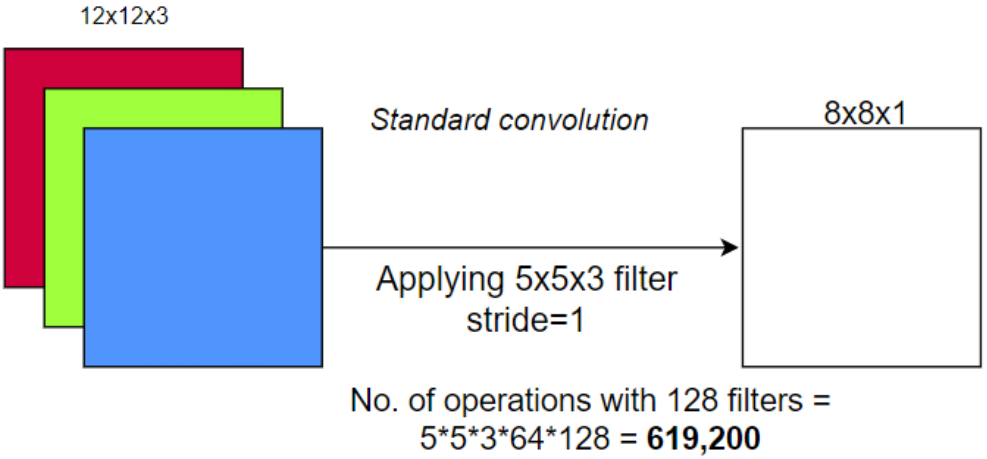
Figure 4.70: Frozen and trainable layers in Fine tuning

4.5.8 CNNs for edge devices

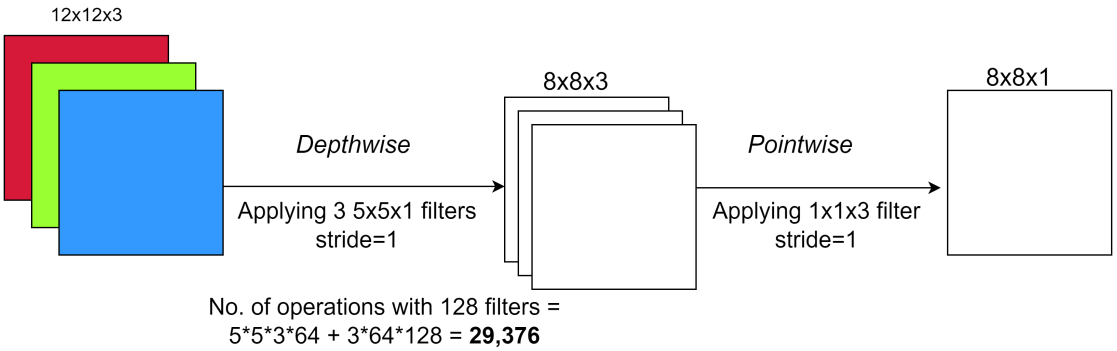
AD | *OM*

Our project utilizes a RPi which is an edge device. If we want to deploy a custom-made object detection model using transfer learning, we cannot use a CNN as described in 4.5.6 as they are computationally intensive [96]. A CNN whose architecture is designed for edge devices needs to be used.
MobileNet Version 2 (V2) is a CNN whose architecture is designed for mobile and edge devices. It builds upon the original MobileNet architecture [7]. It has special features that make it suitable for usage in edge devices. To achieve this, MobileNetV2 introduces several key architectural innovations, listed below.

A cornerstone of MobileNetV2's efficiency is the use of **depthwise separable convolutions**. This is a way to perform convolutions that greatly reduce the number of operations that the network needs. It has two parts, depthwise and pointwise. Depthwise convolution is a spatial convolution performed independently over each channel of an input, followed by a pointwise convolution, which is a 1 x 1 convolution. Figure 4.71 shows the difference between the standard convolution operation and depthwise separable convolution, taking a simple 12 x 12 x 3 colour image as an example.



(a) Standard convolution



(b) Depthwise separable convolution

Figure 4.71: Standard Convolution vs. depthwise separable convolution

ReLU6 as activation function: MobileNet V2 uses ReLU6 as the activation function, which modifies equation 4.19. The difference is that the maximum output of the function is 6. The reason this is used is because of its robustness in low-precision settings [7].

$$\text{ReLU6}(x) = \min(\max(0,x),6) \tag{4.30}$$

Linear Bottleneck layer: This is a layer of convolutional blocks where some of the blocks do not use an activation function (recall that activation functions are used to introduce non-linearity). This reduces the computational power needed by ensuring the dimensionality of information flow within certain network layers remains relatively low. Figure 4.72 shows the blocks in the linear bottleneck layer from the paper published introducing MobileNet V2 [7]. It can be observed that the blocks before and after depthwise separable convolutions do not use ReLU6.

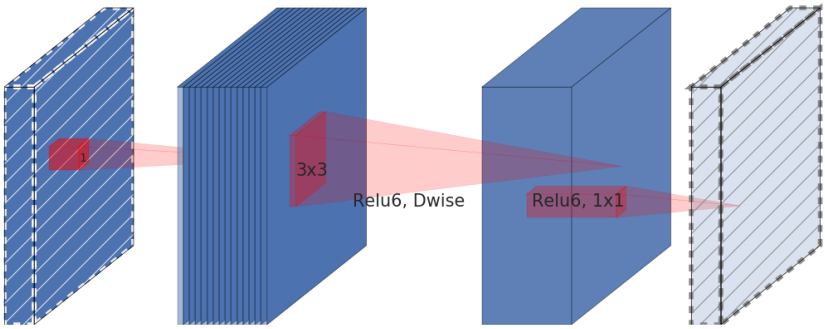


Figure 4.72: Bottleneck layer visualisation [7, p. 4]

Residual blocks: Another feature of MobileNet V2 is the incorporation of residual blocks. When training a deep neural network, it can sometimes struggle to learn complex features or even forget what it learned earlier. During back-propagation, the gradients that update the weights can become progressively smaller, or exactly zero. This can make it extremely difficult to adjust the weights in the early layers of the network. Therefore the model may get stuck. This is known as the vanishing gradients problem [6, p. 756].
Residual blocks are used to solve this problem, as they create a shortcut/skip connection. This connection allows the network to bypass one or more layers, enabling the gradient to flow more directly through the network during training. As a result, residual blocks help to mitigate the vanishing gradient problem and facilitate the training of deeper networks [7]. Figure 4.73 shows an example of the residual block with a linear bottleneck on the right side. It is a visualisation of the custom model created for the project, using an application called Netron [101]. Note that the visualiser shows four dimensions n x h x w x c, where n represents the number of images processed. The full visualisation of all the convolutions and steps can be found in the Appendix S.

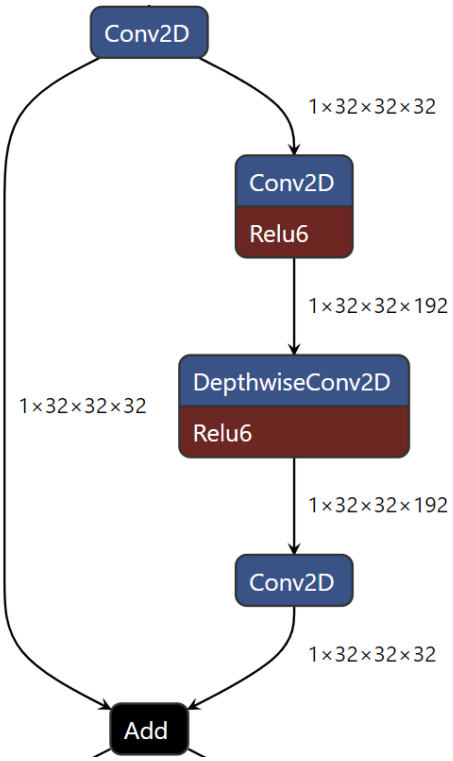


Figure 4.73: Residual block

Figure 4.74 represents the architecture of MobileNet V2. The first column represents

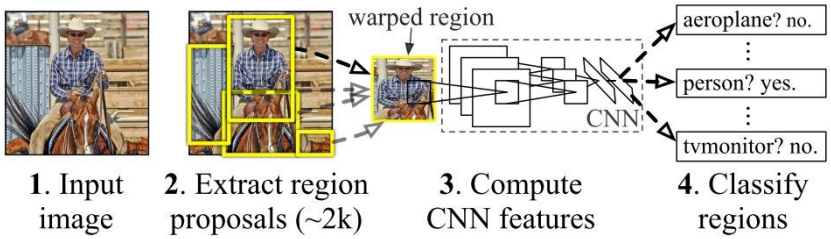
the dimensions of the input and the second is the different operations or layers in the network. The third column shows the expansion factor t, which represents the number of channels that are expanded in the layer before applying depthwise-separable convolution and c denotes the number of output channels from each layer. n represents the number of bottleneck layers that have been repeated, and s represents the stride.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

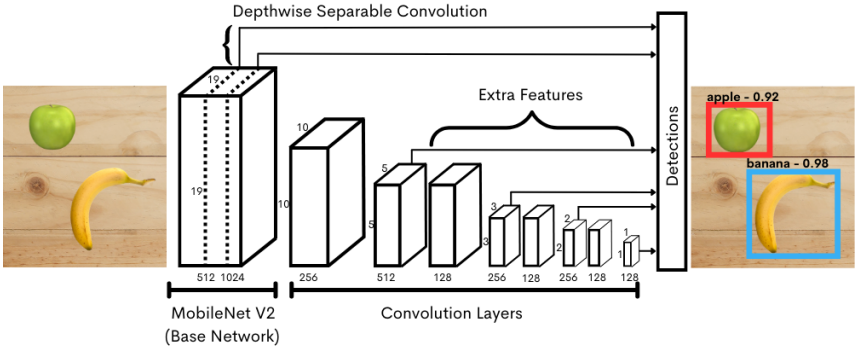
Figure 4.74: Full MobileNet architecture [7]

4.5.9 Object detector CNNs

AD | *OM*



(a) Faster R-CNN, a two-shot detector [50]



(b) MobileNet V2, a single-shot detector [102]

Figure 4.75: Example architectures of two-shot and single-shot detectors

As seen from the black box in Figure 4.62, we wish to input a frame and get the classification and localisation of the object in the frame. The term used to denote the category of an object is referred to as the label, while its spatial extent is known as a bounding box, representing the object's boundary. Additionally, the model provides a probability score indicating its confidence level in the correctness of the classification [103].

There are two types of object detectors: single-shot and two-shot. A two-shot detector does the localization first and then the classification. In contrast, a Single-Shot Detector (SSD) does both simultaneously with one single CNN. This makes SSDs much faster than two-shot detectors. They are however less accurate when using the models on powerful hardware, but when it comes to deploying these detectors on edge devices, SSDs outperforms the two-shot detectors [104]. Therefore, our group decided to use SSDs for our project, namely MobileNet V2.

4.5.10 Loss functions in object detectors **AD** | *OM*

Since object detection is divided into two predictions, namely classification and localisation, it is natural to think that there are two loss functions [105]. Usually, for localisation, the median absolute error (or L1 loss) function is used [95], and for classification, the focal loss is used [8] [106]. Focal loss is used because for long SSDs had a problem of class imbalance, which is a problem where the number of background examples (or negative examples) significantly outweighs the number of foreground examples (or positive examples) in the training dataset [8, p. 1]. This happens especially if the positive examples are small compared to the size of the image. To solve this issue, the focal loss function is used.

Median Absolute Error (MAE): This loss function calculates the average absolute differences between predicted values and the actual target values. The formula is as follows:

$$MAE = \frac{1}{n} \cdot \Sigma_{i=1}^{n} |\hat{y}_i - y_i| [95] \tag{4.31}$$

where n is total number of data points.
To understand focal loss, Binary Cross-Entropy Loss needs to be understood.

Binary Cross-Entropy Loss: This is an extension of the cross-entropy loss from equation 4.22. This loss function quantifies the difference between the predicted probabilities generated by the model ($\hat{y}$) and the actual labels (y which is 0 or 1). Specifically, for each data point, the binary cross-entropy loss is calculated as:

$$L = -y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y}) \tag{4.32}$$

This loss function penalizes incorrect predictions, with the penalty being more severe for predictions made with high confidence that turn out to be wrong. **Focal loss** builds on the binary cross-entropy loss. It essentially tells the model to give more weight to learning the wrongly classified examples.

$$FL(\hat{y}, y) = -\alpha \cdot (1-\hat{y}_t)^{\gamma} \cdot (y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) [8, p.\ 3] \tag{4.33}$$

where α is a weighting factor to balance positive/negative examples, γ is a focusing parameter that controls how much to down-weight of correctly classified examples and $\hat{y}_t$ is the adjusted probability; if $y = 1$, then $\hat{y}_t = \hat{y}$, if $y = 0$, then $\hat{y}_t = 1 - \hat{y}_t$. If γ increases, then the focus is more on learning the misclassified examples, and if α increases, the model gives more weight to the positive examples. Figure 4.76 shows the loss concerning the ground truth for different γ values. Notice that when γ is 0, the function is the binary cross-entropy loss.
The pipeline that has been used in our implementation lists more loss functions that are built on top of the above loss functions, for further reading see the implementation in GitHub [106].

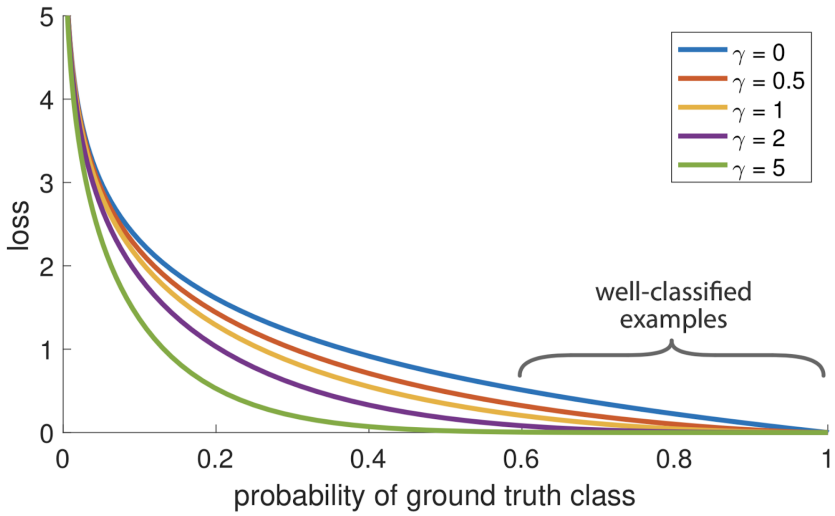


Figure 4.76: Binary Cross-entropy versus Focal Loss [8, p. 1]

4.5.11 Framework for object detection

AD | *SO*

It is beneficial a framework to create an ML model. An AI framework, or a ML framework is the building block that aids in creating, implementing, and deploying models. They come with pre-built functions and libraries that one can use instead of building a AI model from scratch. Other advantages are that these frameworks are well-documented, tested and they make integration of AI models to other platforms relatively straightforward.

Some of the popular frameworks in the field are TensorFlow and PyTorch. Both are open-source and free and provide libraries mainly in Python and C++. They also support computations via the CPU and the GPU. **PyTorch** is a framework developed by Meta AI and is based on the Torch library. The advantages of using PyTorch are that it is flexible, its deep integration with the Python language (thereby making it easier to use if one is familiar with Python), and strong community support from the researchers at Meta and other developers. **TensorFlow** is a framework developed by the Google Brain team for Google's internal use in ML and artificial neural networks. Its advantages are that it makes deployment of AI models on other platforms like RPis easier and provides multiple levels of abstraction [107].

Our group started this project by using the TensorFlow libraries in Python as it was known that it had the libraries and tools necessary for not only making a model for object detection but also exporting it to a lighter version called **TensorFlow Lite (TFLite)** so that it would be easier to deploy it on a mobile device or a single-board computer such as the RPi [108]. However as this is a research project, the group has also experimented with using the PyTorch library to explore alternative approaches and compare their performance. Ultimately, TensorFlow was used for the final implementation due to reasons mentioned in the Challenges section, section 9.2.4.

4.5.12 TensorFlow object detection with MediaPipe

AD | *OM*

When using the TensorFlow library for making our custom object detection model, our group came across *TensorFlow Lite Model Maker*, a library that simplifies the process of creating a custom TFLite model using transfer learning [109]. However, many problems

were encountered with the library, which has been discussed in 9.2.3. This led us to look for other options, and that is where our group came across a GitHub issues page for the official TensorFlow repository which addressed the issues with *TFLite Model Maker* library by advising users to use a pipeline called MediaPipe instead [110]. A ML pipeline is a set of steps that help automate and organize the process of creating, training, testing, and using ML models [31].
MediaPipe has also been developed by Google and falls under the TensorFlow framework. It is also built specifically for on-device ML, and it exports the model in TFLite format [111]. Therefore, the project went forward with running this pipeline in Google Colab.

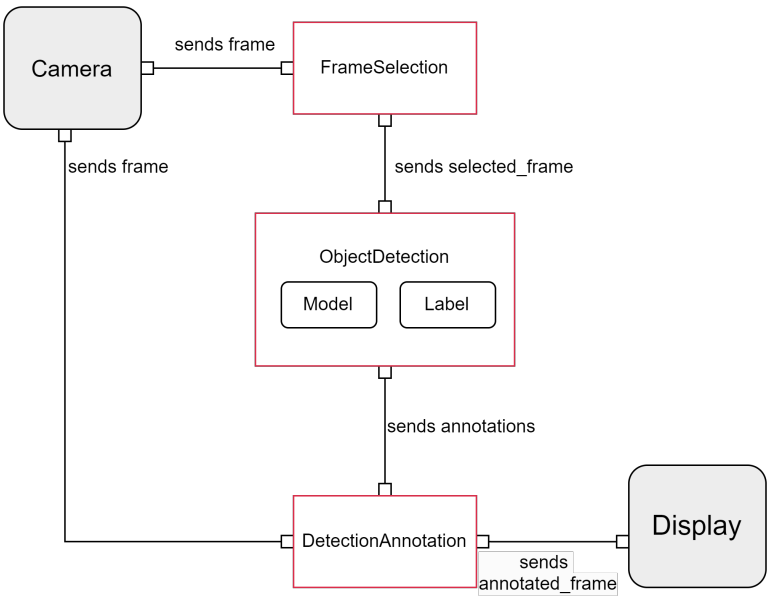


Figure 4.77: MediaPipe architecture [9]

MediaPipe operates using a graph-based architecture. These graphs are composed of nodes and edges, where each node represents a computational step, referred to as a calculator, and edges define the data flow. Calculators are the building blocks within MediaPipe graphs. Each calculator is designed to perform a specific task such as video decoding, image transformations, executing a portion of a ML model, etc. Calculators can be easily reused and connected in various configurations within a graph [9].
The data within a MediaPipe graph flows in the form of packets. Packets can contain different data types, such as raw video frames, audio samples, ML models or metadata. The design ensures that each packet is timestamped, supporting the synchronization of multiple data streams within the graph.
Figure 4.77 shows the architecture of object detection deployment using MediaPipe. The transparent boxes represent calculators in a graph, the solid boxes represent external input/output to the graph, and the lines entering and exiting the calculators represent the input and output streams respectively. The small round rectangles inside a calculator are the input side packets [9].

4.5.13 Preparing the dataset **AD** | *OM*

The group prioritized the detection of people as a fundamental capability for our custom model. By accurately identifying and locating people within the camera's field of view, the robot can potentially assist in collision avoidance, emergency response, etc. We de-

cided to use images already available on websites such as RoboFlow, Kaggle, and Open Datasets. These sites have a lot of pre-made datasets that can serve as a solid foundation for ML projects. After evaluating our options, RoboFlow was chosen as it provided a platform not just for accessing a diverse array of images but also for labelling our dataset and exporting it in various formats.
Such a dataset was found in the RoboFlow open-source computer vision projects [10]. This particular dataset was also luckily labelled with the bounding boxes. It contained 1327 images of three classes: person, person-like and null. Our group did not have to use the latter two classes, so the dataset was modified only to have one class using the workspace function that RoboFlow provides. Figure 4.78 visualises the labelling process.

Figure 4.78: Labelling an image with a person [10]

The dataset had to be exported in a certain format that was also acceptable by the MediaPipe library. The documentation showed that the Pascal VOC (Visual Object Classes) format and COCO format were accepted [112]. The dataset was exported in the Pascal VOC format. Figure 4.79a shows the directory structure for the annotations(with the bounding boxes and labels) and images and figure 4.79b shows the format of the .xml file [113].

(a) Pascal VOC **(b)** The XML file format

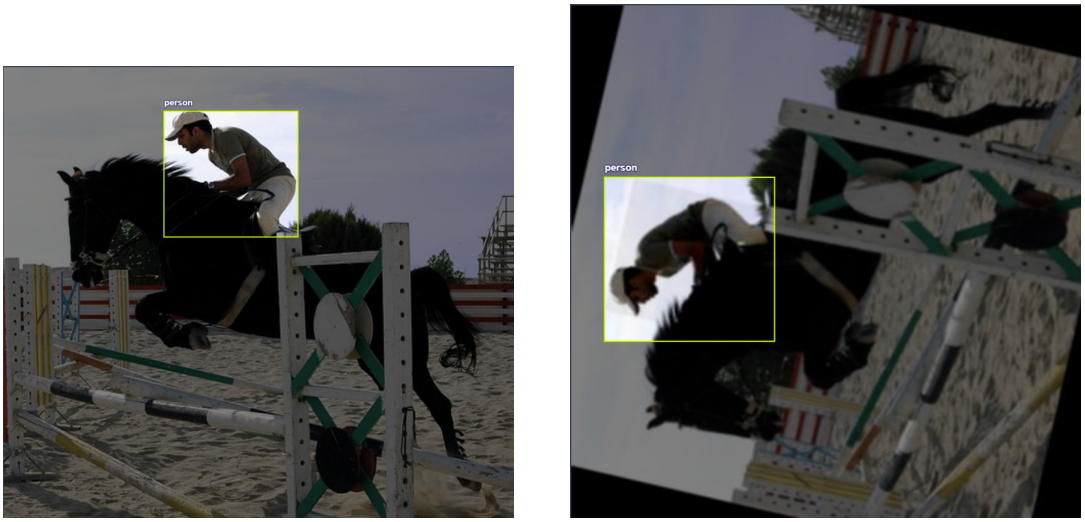
Figure 4.79: The Pascal VOC directory structure and XML file format

4.5.14 Dataset augmentation **AD** | *OM*

Data augmentation is an important step when preparing the dataset. It is the process of creating different variations from the existing dataset to give the ML model diversity in the dataset. It artificially increases the amount of data by making small changes to the data. This can help a model improve the accuracy of its predictions and decrease the possibility of overfitting [114]. Therefore, it was decided to augment the data using the interface that RoboFlow provides. Listed below are the techniques that were used.

- **Flipping**: Flipping the image horizontally and vertically. This is one of the earliest and most widely used methods of data augmentation [115].
- **Rotating**: Randomly rotating the image.
- **Blur**: Adding blur to an image. One might think that only clear images should be in the dataset, but adding imperfect images is also necessary as it will make the model more resilient to the harsh realities they will encounter in real-world situations [115].
- **Bounding box brightness**: Brightening the bounding box region of the image up to a certain degree.

Figure 4.80 shows the results of data augmentation on an image. Figure 4.80b has been rotated 77° degrees counter-clockwise, blurred 3.75px, and has 15 % brightness around the region of the bounding box.



(a) Image without any augmentation **(b)** Image with augmentation

Figure 4.80: Difference without and with data augmentation [10]

4.5.15 Training the model **AD** | *OM*

The training process was followed according to the documentation for MediaPipe [112]. The activity diagram in figure 4.81 shows the training process. The T4 GPU offered by Google Colab was used to train the model which decreased the training time significantly. The data was downloaded directly using RoboFlow's integration to Python and Google Colab, and loaded into the model.
The dataset was to be divided into two sections, training and validation data. **Training** data is used to train the model, where it learns to recognize patterns and make predictions based on the dataset it gets as input. 80 % of the data were used for training the model. **Validation** data is used to fine-tune model parameters and select the best model after training. It can also be used iteratively to adjust the model and its parameters to avoid overfitting to the training data. 20 % of the data were used for validating the model [6, p. 666].
While training, a model cannot process all the images at once. Therefore, the training dataset is divided into smaller, manageable chunks called batches. One entire passing of data while training a model is called an **epoch**. Models are usually trained for multiple epochs. This allows it to repeatedly "see" the same data so that it can adjust its parameters each time to hopefully improve its predictions.
MobileNet V2 was chosen as the base model. The hyperparameters such as the number of epochs was chosen to be **100**, batch size **8** and learning rate as **0.1**. The model was exported as a float32 model, which is ideal for maintaining the balance between model performance and precision. The float32 format provides high precision for the model's calculations as it saves the data in a single-precision floating-point format, which is a standard computer number format [116]. The export method also includes metadata in the model, which eliminates the need for an external label file for the classes. The results of the training are discussed in section 7.2.1. Figure 4.81 shows an activity diagram describing the training process.

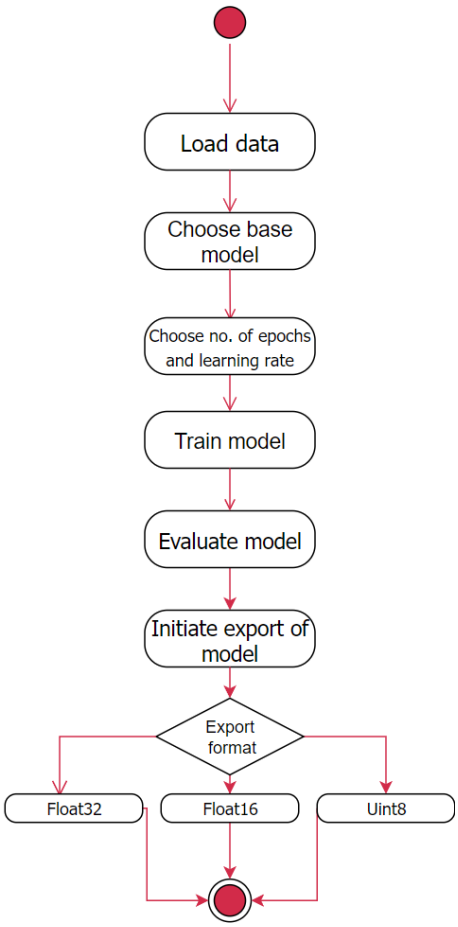


Figure 4.81: Training process

4.5.16 Displaying object information

OM, AD | *AEH*

One of the requirements was to display information related to the recognized objects. After discussing ideas with Kongsberg Maritime and considering the ability to recognize objects, we decided to use a database to store object information and retrieve data from there. However, since using a database was not a requirement, the explanation, installation, and implementation of this are detailed in 4.6.14.
The results of the recognition and information display are explained in the results section, 7.3.

4.6 Additional work **OM** | *AD*

This section describes functionalities and work implemented beyond the set requirements by Kongsberg Maritime. Although not being requirements, these features are relevant to the thesis. This is due to the thesis being research-oriented and aiming to offer insights into new technologies for Kongsberg Maritime. Prior to implementation, the additional features mentioned in this section were discussed with supervisors and Kongsberg Maritime.

4.6.1 Getting depth data **OM** | *AD*

Astra Pro Plus is the camera we are using. In addition to taking RGB pictures it has an Infrared (IR) projector and an IR camera. This makes it possible to get depth information from the camera.

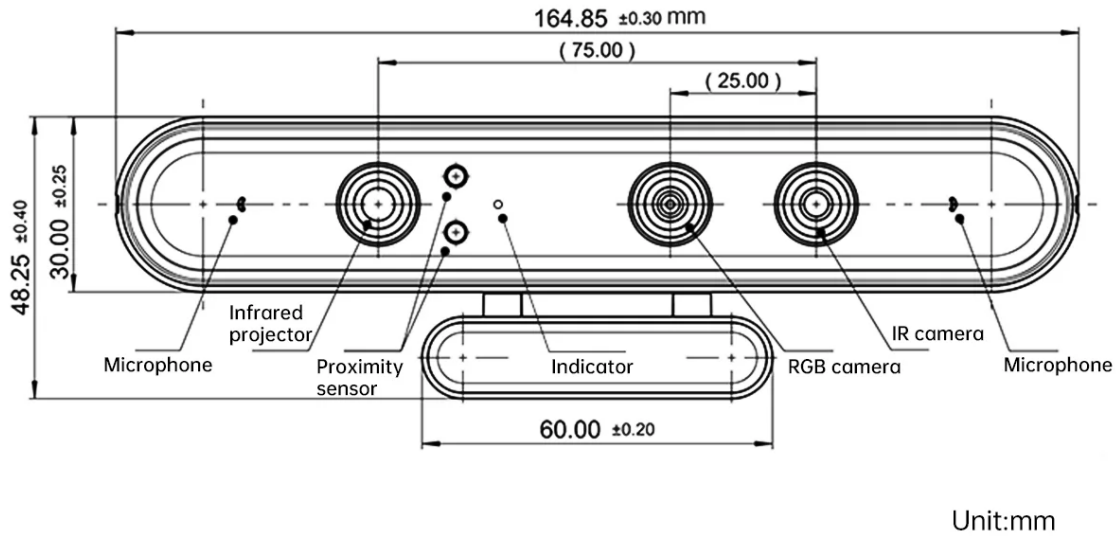


Figure 4.82: Astra Pro Plus Diagram [11]

Why get depth data? The concept of integrating depth data into VR space was introduced and implemented. This decision came from the belief that incorporating depth data into VR could provide a better understanding of the robot's environment, including the relative dimensions and distances of objects within it. To the group's best knowledge, this integration has not been done before.

Astra driver The Astra camera is made by Orbbec and they have made an open-source ROS wrapper which is available on GitHub [117]. However, the code is made for ROS 1, not ROS 2. After some searching, it was found that GitHub user Javier Monroy had made a fork which indicated to be working for ROS 2 [118]. Due to running the ROS environment in Docker, all of the depending packages used by this driver also needed to be installed in the Docker image. These packages were not explicitly specified anywhere but were found by running into errors when building and going through the source code finding them on index.ros.org (see O.16). Given the substantial size of the driver, mainly written in C/C++, the majority of the robot's codebase is not in the Python language.

Getting raw data When launching the ROS node for the Astra camera, there are multiple options that can be specified, including the enabling of the depth camera. If the

depth camera was enabled when launched, it was found that the ROS node would publish *PointCloud2* messages to the "/camera/depth/points" topic. The messages published to this topic is subscribed to by the **Depth data node**, which processes the messages and lastly sends them to the **VR linker node**.

Understanding the data The *PointCloud2* message interface is apart of the "sensor_msgs" in ROS 2 and "holds a collection of N-dimensional points, which may contain additional information such as normals, intensity, etc. [...] The point cloud data may be organized 2d (image-like) or 1d (unordered). Point clouds organized as 2d images may be produced by camera depth sensors such as stereo or time-of-flight" [119]. Below is a shortened version of one *PointCloud2* message received from the **Astra node**.

```
PointCloud2(
    header=Header(
        stamp=Time(
            sec=1713167964,
            nanosec=751190626
        ),
        frame_id='camera_depth_optical_frame'
    ),
    height=480,
    width=640,
    fields=[
        PointField(name='x', offset=0, datatype=7, count=1),
        PointField(name='y', offset=4, datatype=7, count=1),
        PointField(name='z', offset=8, datatype=7, count=1)
    ],
    is_bigendian=False,
    point_step=16,
    row_step=10240,
    data=[
        0,
        0,
        192,
        127,
        ...
    ],
    is_dense=False
)
```

The `data` field is a one-dimensional array which consists of `uint8` values and these values are the actual depth data. With the resolution specified, one will get **4,915,200 values**. The values range from 0 to 255, as this is the range an unsigned byte or 8 bits can hold. The `fields` field tells the user what the `data` means. In this case `data` consists of three values; x, y and z. Each of these `PointFields` being of the datatype 7 which correlates to a `FLOAT32` type [120]. `offset` tells the user the offset from the start of the point struct to start reading that value.
`point_step` indicates the length of each point as bytes in `data`, which in this case is **16 bytes**. As each value in `data` is a byte and x starts on index 0 (`offset=0`) it has to end on index 3, as this is in total 4 bytes (32 bits), which is the size of the `FLOAT32` type. y and z are also 4 bytes in length, due to them having the same type as x. The 3 coordinates

which are 4 bytes each adds up to a total of 12 bytes. The last 4 bytes of the `point_step` (16 bytes in total) therefore needs to be ignored.

$$total_pixels := height \cdot width \Rightarrow 480 \cdot 640 = 307,200 \tag{4.34}$$

$$total_points := \frac{data_values}{point_step} \Rightarrow \frac{4,915,200}{16} = 307,200 \tag{4.35}$$

$$total_pixels = total_points \tag{4.36}$$

`is_bigendian` tells how the `data` is arranged. In this case, the values in the data array are arranged as little-endian.

address	0	1	2	3
value	0	0	192	127

Table 4.3: Big-endian

address	0	1	2	3
value	127	192	0	0

Table 4.4: Little-endian

By converting each of these decimal values to 8 bit binary data, and then into a FLOAT32 we get Not a Number (NaN) for little-endian. However, if `is_bigendian` was true, the FLOAT32 value would be $6.9055e^{-41}$ which in this case would be wrong, as the value of this x coordinate could not be represented.

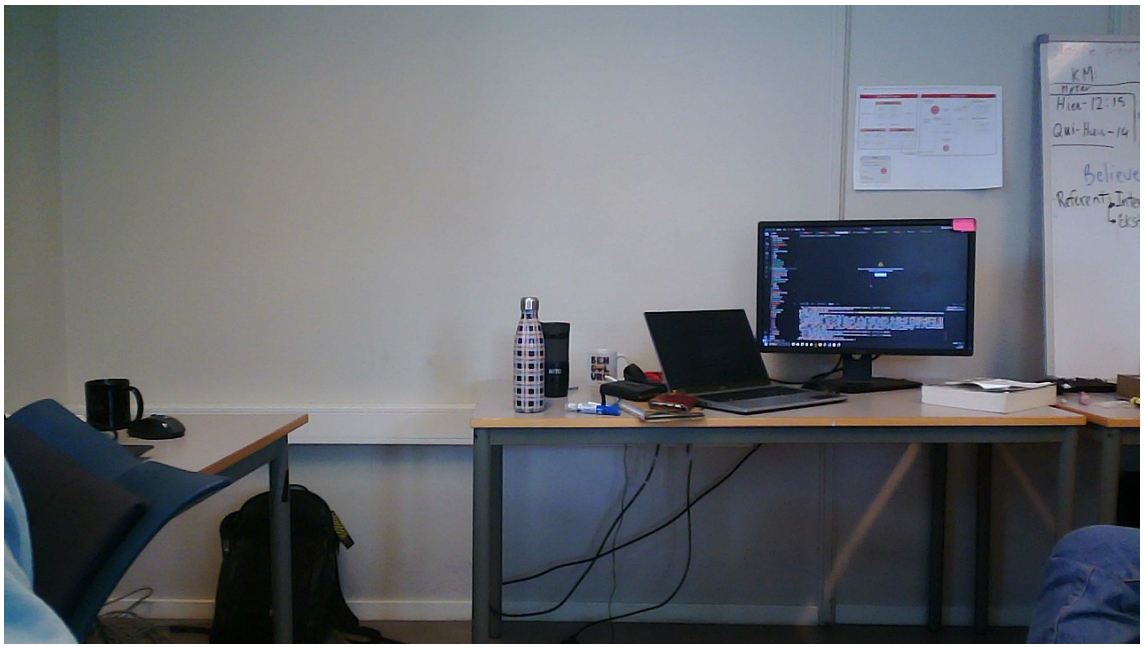


Figure 4.83: Astra Pro Plus RGB picture (1280x720)

Plotting *Plotly* was used for the plotting, as it performed better for bigger data than *Matplotlib* did. This plotting was only done for testing purposes and to determine if the data was sufficient or not. The VR application does not use *Plotly* nor *Matplotlib* for plotting the actual points in VR space. Only 255,888 out of the total 307,200 points were plotted as 51,312 of them were NaN.

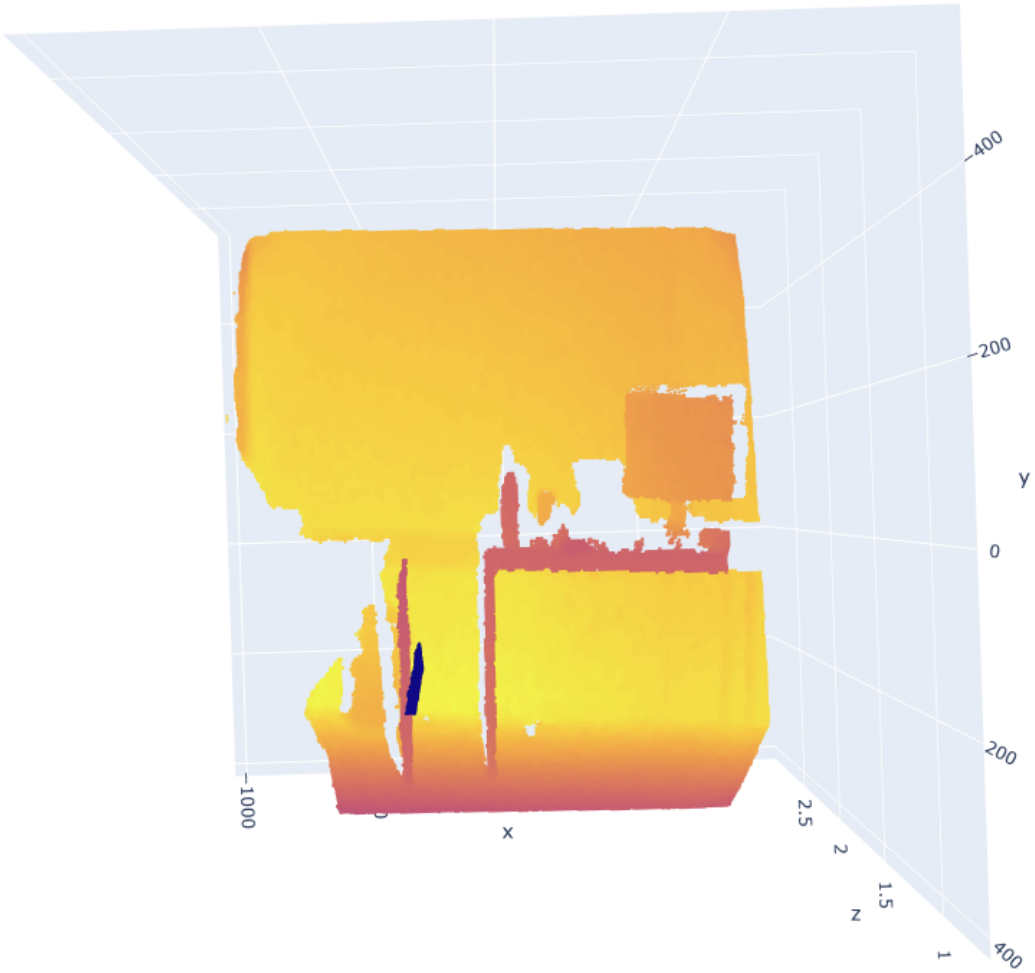


Figure 4.84: Depth data plot of 255,888 points (640x480)

This is the depth data for the image 4.82, and is the most detailed plot we can get at the 640x480 resolution. It is possible to increase the resolution, but this will make the image bigger and lead to more points, which means even more data. Due to there being no requirement for resolution, the dimensions were kept as is.

Issues transferring Size of the raw *PointCloud2* data is ∼19MB at the (640x480) resolution, which leads to issues when transferring this data to the VR headset over the TCP socket connection (see 9.2.7). Therefore multiple steps were taken in order to decrease the size of the data enough to be able to transmit the depth data often, while still being detailed enough to make sense of what the data represents. *Brotli* were chosen as the data compression algorithm. "Brotli is a generic-purpose lossless compression algorithm that compresses data using a combination of a modern variant of the LZ77 algorithm, Huffman coding and 2nd order context modelling, with a compression ratio comparable to the best currently available general-purpose compression methods" [121]. How *Brotli* was used is mentioned later in this section.

Picking random points Plotting all 255,888 points are not needed when the wanted result is to be able to make up what the plot represents. The coordinates retrieved from

the *PointCloud2* data were found to be random. By simply looping through every 75th valid point the data can be reduced from 255,888 valid points to only 3,412 points.

```
>>> len(range(0, 255_888, 75))
3412
```

At first, a step of 100 was used, but this could be increased to 75 as more data could fit inside the socket message without affecting the max size for the receiving client end.

Decreasing the resolution Each coordinate consists of three float values being x, y and z, with a high resolution. This resolution was deemed unnecessary when the points are decreased substantially, losing important information. Therefore, the coordinates are simplified by multiplying each coordinate value with ten and converting this to an integer. This does not only make the size of the data smaller, as integers can be represented using fewer bits than floats, but also significantly increases the compression-rate, due to the Huffman-encoding.

```
-1.2974001169204712 -> -12
-1.2933393716812134 -> -12
-1.2981855869293213 -> -12
-1.3099732398986816 -> -13
```

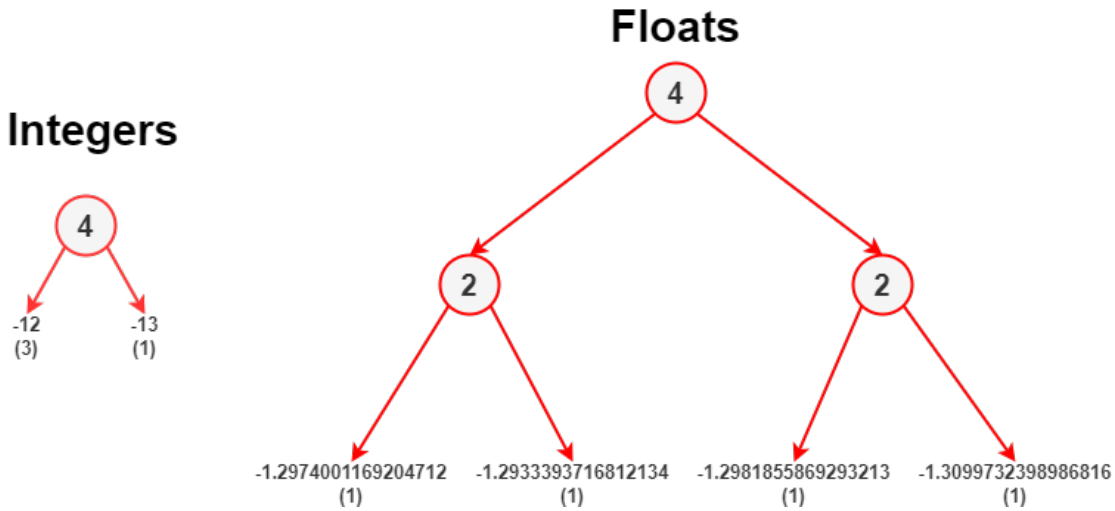


Figure 4.85: Huffman encoding difference

With the values being represented as integers the data will be more similar than with exact float numbers. As Huffman-encoding works the best for data which is repetitive the compression is greater than for floats.

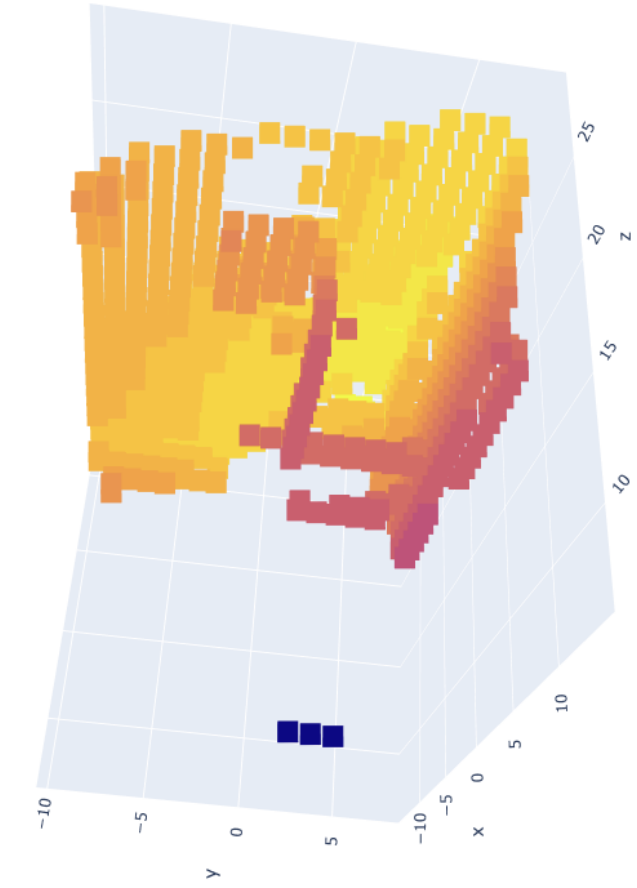


Figure 4.87: Plot seen from the right-hand side

Figure 4.88: Compressed depth data plot of 3,412 points

Figure 4.86: Plot seen from front

Structuring the JSON The dynamic nature of JSON allows for various methods of structuring the data. Any chosen structure affects both readability and size of the message. The points gets structured as a JSON file like shown below. The first list under the `data` key are x, the second are y and the last are z values.

```
{
    "type": "depth",
    "time": 1274382748327432,
    "data": [
        [-12, -10, ...],
        [-9, -10, ...],
        [23, 26, ...]
    ]
}
```

Structuring it this way saves bytes, compared to for instance:

```
{
    "type": "depth",
    "time": 1274382748327432,
    "data": [
        {"x": -12, "y": -9, "z": 23},
        {"x": -10, "y": -10, "z": 26},
        ...
    ]
}
```

Therefore, the first example is the one which is used.

Brotli compression When the JSON message is structured, *Brotli* is used to compress the data. Compressing the data before sending makes using the TCP socket easier as there is no need to split up the packets, it can be sent in one go with all of the data included. The JSON shown above is ∼38.2KB (excluding newlines and indents), with *Brotli* compression all of this data gets down to ∼2.0KB or a ∼94.7% decrease. Due to *Brotli* being lossless no data is lost, but some milliseconds are used when compressing and decompressing the contents. Before sending the compressed data over the TCP socket, "bro" is prepended to the bytes string to work as a header to indicate the bytes data is compressed with *Brotli*.

```
bro\x1b\'\x95D\x141\x00\x9c\x07\xb6\x8dl\xbb1\x86\x83\x81\xc6W\xde
\x8c\x91xn\xb3\xe8\xec\xf5q\xab\x81\xa9/\xff\xee\x00\x154\xe5\xdd
\xdc`lnZ\x03\xc4\xffw\xe6\'...
```

When the bytes are received it has to be checked if the data has been compressed or not. This way a JSON decode error can be avoided, since the JSON decoder cannot decode *Brotli* compressed data directly without decompressing the data first. Checking if the byte string is compressed can be done by checking the first three characters of the byte string, which was prepended before sending the message. This check is important to do within the VR application. Below is an example in Python which is used in "robot-test-client" (see 7.6).

```
data = b"bro\x1b\'\x95D\x141\x00\x9c\x07..."

if data[:3] == b"bro":
    # data is compressed
else:
    # data might be uncompressed JSON
```

4.6.2 Showing depth data in the VR Space **SO** | *OM*

Data Processing Workflow: The process of visualizing depth data in the VR application begins when the data is transmitted from the robot. As illustrated in Figure 4.89, the depth data goes through several transformation steps before visualization. Initially, the data, which includes a *Brotli* header labeled *'bro'*, is received by the network manager. This header is removed, and the message is decompressed to convert the data into a custom **DepthData** datatype suitable for further processing within the VR environment.

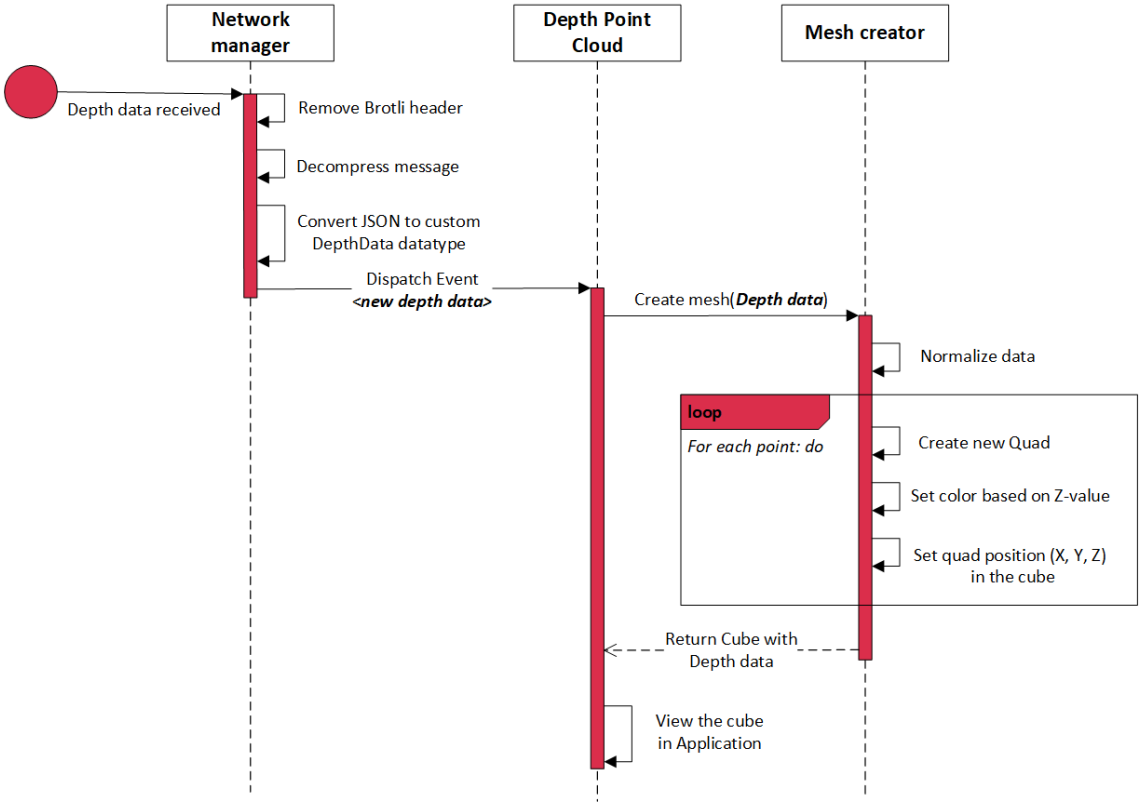
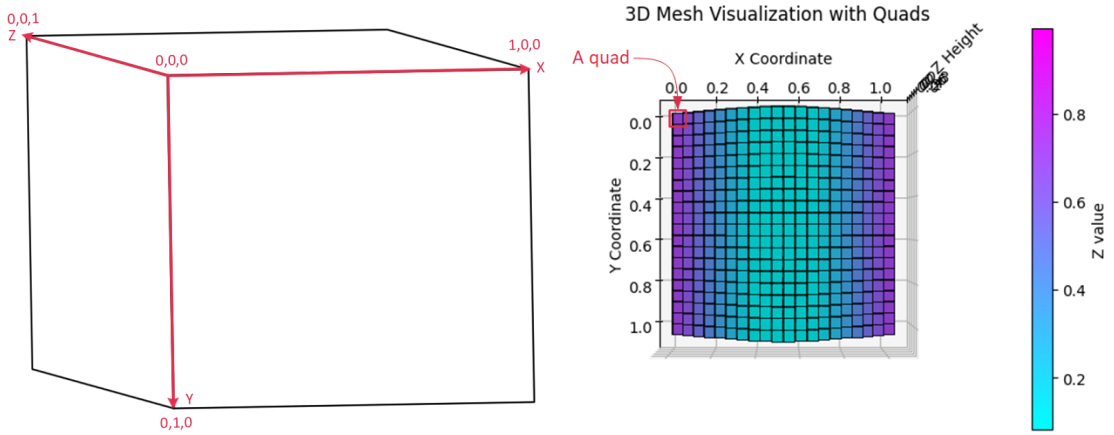


Figure 4.89: Sequence of activities between involved classes for mapping depth data

Mesh Creation and Normalization Following decompression, the *Depth Point Cloud* class receives the transformed data. It collaborates with the *Mesh Creator* class to process these data points and generate a mesh for visualization. Given that the raw depth data from the robot encompasses actual (x, y, z) values within a 640x480 resolution and a depth range of 0-30, these values are normalized between 0 and 1 to ensure consistency in the visualization scale.

Visualization and Color Mapping Each normalized data point is represented as a rectangular quad within a virtual cube. The placement of each quad corresponds to its normalized (x, y, z) coordinates. The color of each quad varies from dark blue (representing a value of 0) to purple (representing a value of 1), based on the z-value, which indicates

depth. Figures 4.90a and 4.90b showcase the normalized cube used for visualization and the distribution of quads within the cube, respectively.



(a) Normalized cube used for depth data visualization

(b) Quad distribution within the normalized cube

Figure 4.90: Conceptual illustrations of depth data visualization

Example of Visualization An example of visualizing depth data captured by the robot within the VR application is presented in Figure 4.91. The results are suboptimal due to the time constraints of the project, and further improvements are recommended for future work.

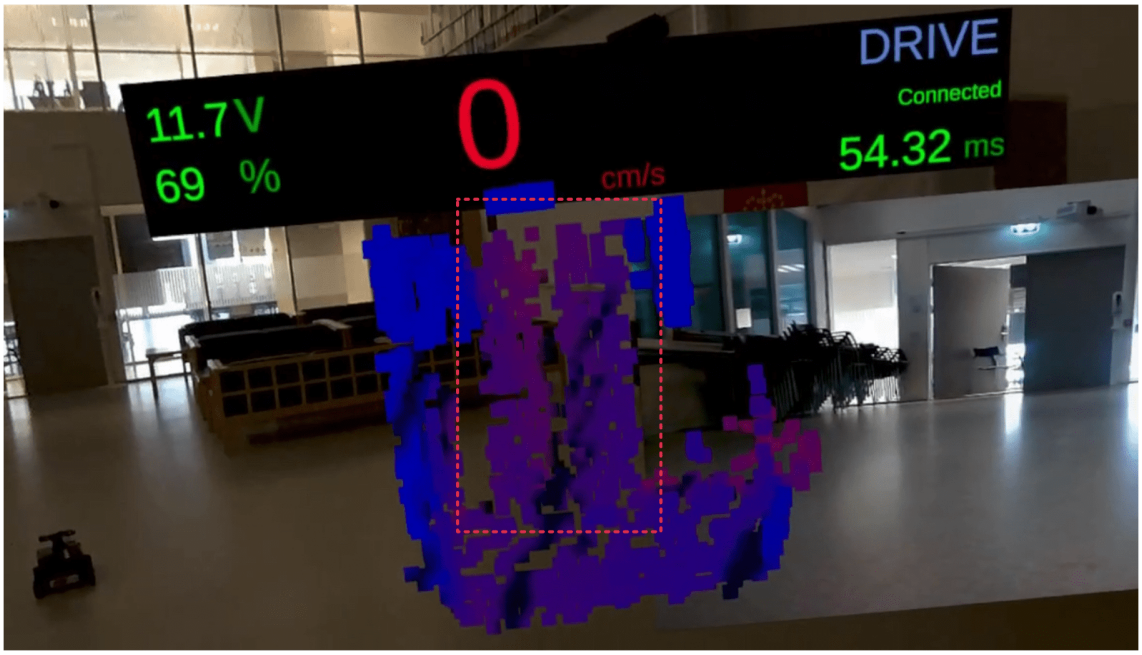


Figure 4.91: Real-time depth data visualization in the VR application showing a slightly opened door.

4.6.3 Digital twin of the robot arm

SO, OM | *AEH*

The digital twin offers a real-time visualization of the physical robot's position within the VR application. This virtual representation mirrors the actual movements and status of the physical robot. More details on this feature can be seen in the video attached in the

folder: **Digital twin**. The digital twin was developed from scratch by our team, starting with a 3D model created by our mechanical engineering team members (see 6.10). This model was then exported to the VR application, where functionalities to control each joint angle were implemented. The digital twin is displayed in Figure 4.92.

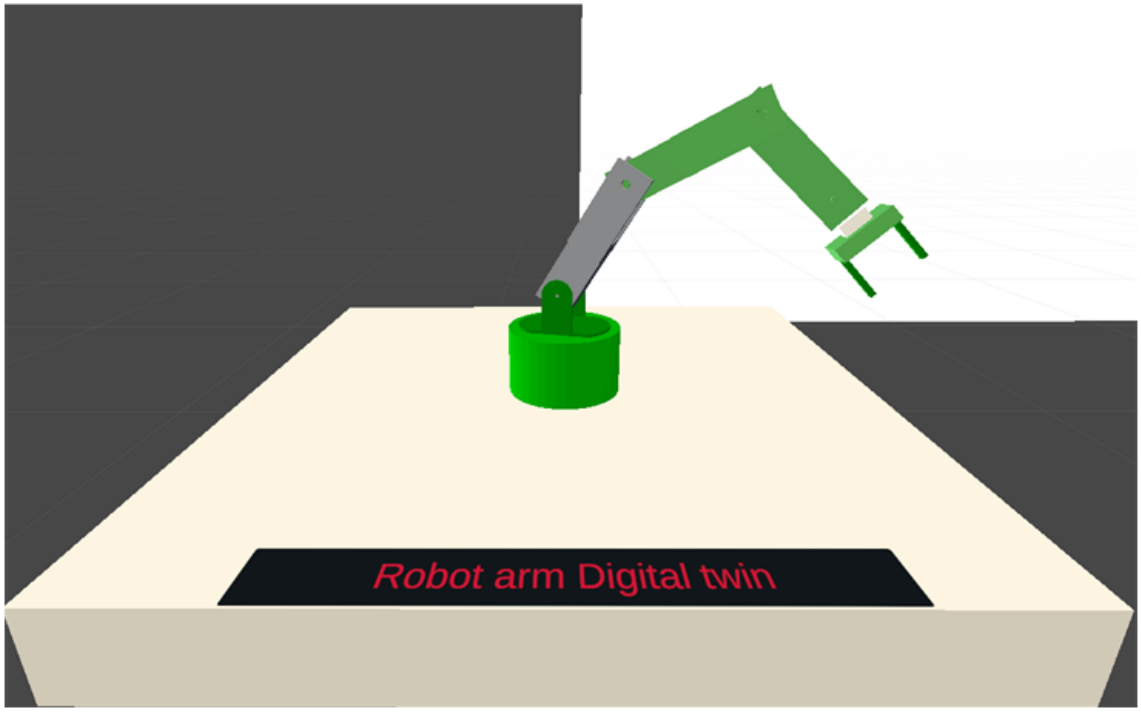


Figure 4.92: Visualization of the Digital Twin in the VR Application

The interfaces for all joint angles are designed to replicate those of the actual robot. For example, setting a 30-degree angle at a specific rotation point on the physical robot will result in an identical appearance at the corresponding rotation point in the digital twin. Figure 4.93 demonstrates the alignment of these rotation points and pinch mechanisms, mirroring the actual physical structure of the robot.

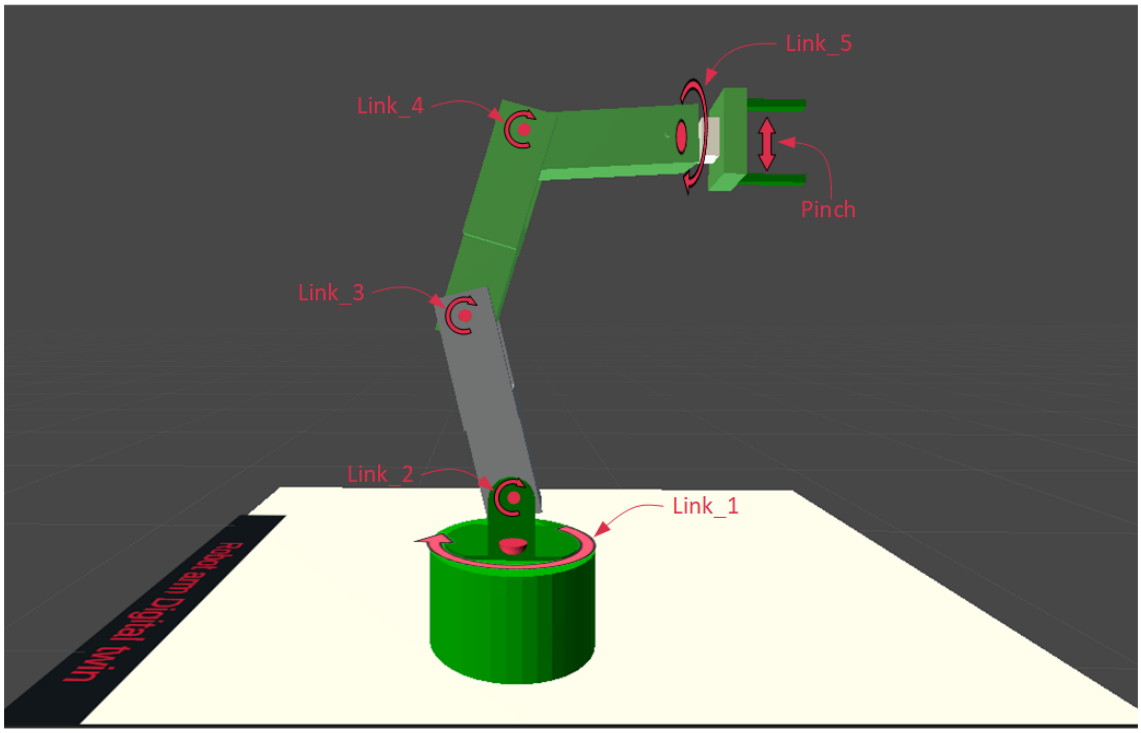


Figure 4.93: Digital twin rotation points and pinch

Figure 4.94 displays both the digital twin and the actual robot arm within the VR application, showcasing the replication of the robot's physical movements in the digital environment.

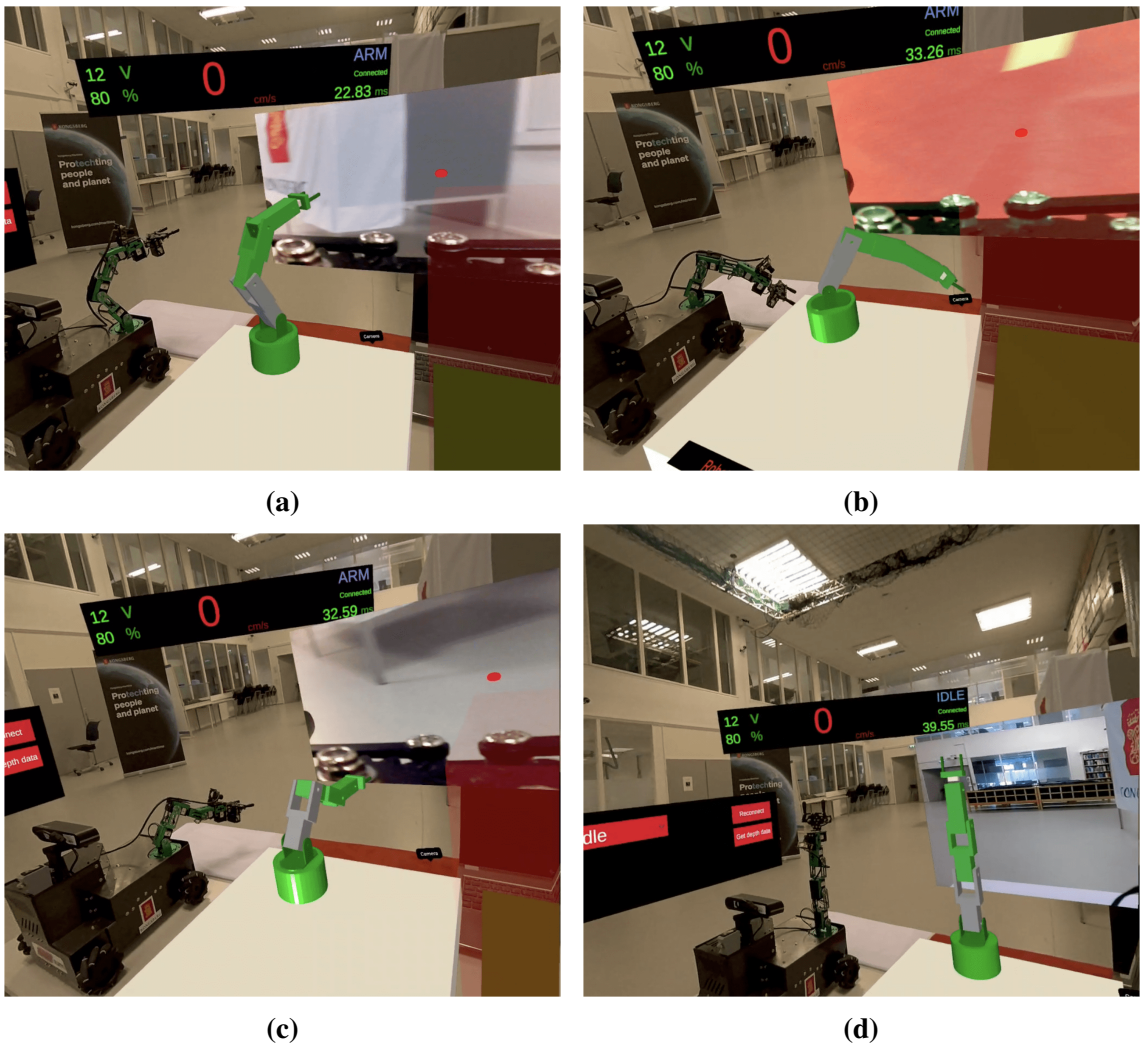


Figure 4.94: Examples of the digital twin alongside the real robot arm

Figure 4.95 illustrates the workflow between the robot operator and the digital twin system. The red path highlights the data flow from the operator to the physical robot, triggered by the operator's hand movements. These movements are captured and processed by the VR application to generate robot arm data 6.10, which is subsequently transmitted to the robot's ROS system. This system adjusts the angles on the robot's arm servos. On the other hand, the blue path illustrates the feedback loop, where the actual angles of the robot arm are captured and sent back to the VR application. This ensures a continuous update of the digital twin within the VR application, providing the operator with an accurate and dynamic visualization of the robot's current status.

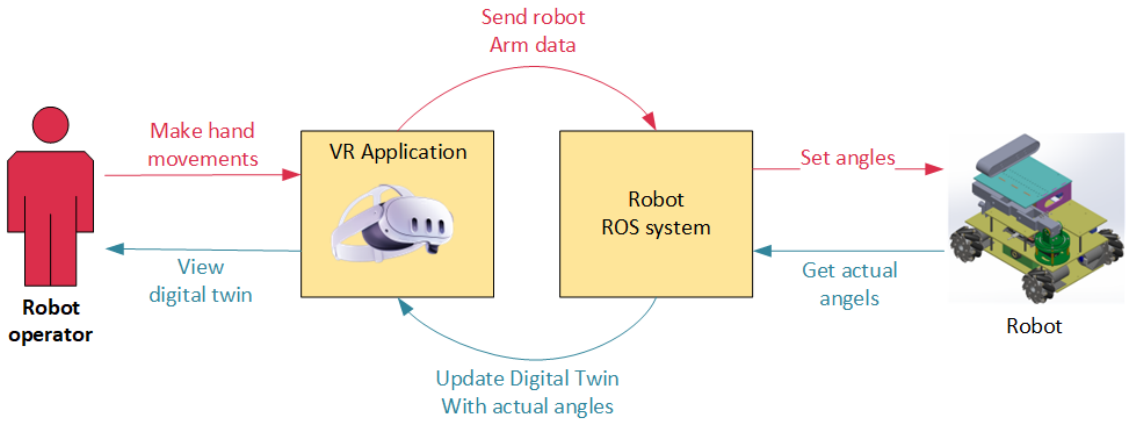


Figure 4.95: Workflow of the Digital Twin Data Visualization

However, for the digital twin to work correctly it needs to act on real data.

Getting the data A `dict` was implemented inside the *Rosmaster* code, which gets updated every time an angle is set. Each key is the servo id and the value is the degrees as an integer. The key is the servo id because when setting an angle it has to be an ID and not a name.

```
class Rosmaster:
    def __init__(self, *args):
        self.servos = {
            "1": 170,
            "2": 90,
            "3": 80,
            "4": 70,
            "5": 60,
            "6": 50,
        }

    ...
```

This data is only available inside the **Controller node** and has to be sent to the **VR linker node** to be able to send this data to the VR application. The data is formatted to be more explicit for what each of the keys mean, this way the VR application does not need to know what each key value corresponds to (e.g. six being pinch). For how these get mapped see O.12. Using the names, a ROS message is created and sent with the following interface.

```
uint8 rotation
uint8 shoulder
uint8 elbow
uint8 tilt
uint8 wrist
uint8 pinch
```

The **VR linker node** receives this message, but the socket client (VR application) expects a JSON, so this has to be formatted. There were not found any already implemented

function which does this, so a dynamic function for getting ROS message contents as a dictionary was implemented. The code for this is shown below.

```
class AnyMessage:
    pass

def get_content_any_msg_type(msg) -> dict:
    message = AnyMessage()

    for slot in msg.__slots__:
        key = slot[1:]  # first char is _
        value = getattr(msg, key)
        setattr(message, key, value)

    return message.__dict__
```

Figure 4.96: Dynamic content gathering of ROS 2 messages

This code creates an empty class instance (object) which gets filled by looping through each slot (key) of the ROS message. Using the name of this slot, as well as getting the value of this slot (value), a key-value pair is created within the class using attributes. These pairs can be gathered using the built-in `__dict__` class method, which as the name implies returns a dictionary. By adding the message type to the dictionary, the message below gets sent to the VR headset.

```
{
    "type": "arm_angles",
    "rotation": 170,
    "shoulder": 90,
    "elbow": 80,
    "tilt": 70,
    "wrist": 60,
    "pinch": 50,
}
```

The JSON data gets sent over the TCP socket at a rate of 10 Hz. This frequency was found to be often enough for the digital twin to be responsive, while not being excessive.

Updating the Digital Twin The digital twin is managed by the *Digital Twin Controller*, which subscribes to the Network Manager event *New arm angles*. It updates the digital twin whenever new data is received from the robot. The data distribution process is illustrated in Figure 4.97.

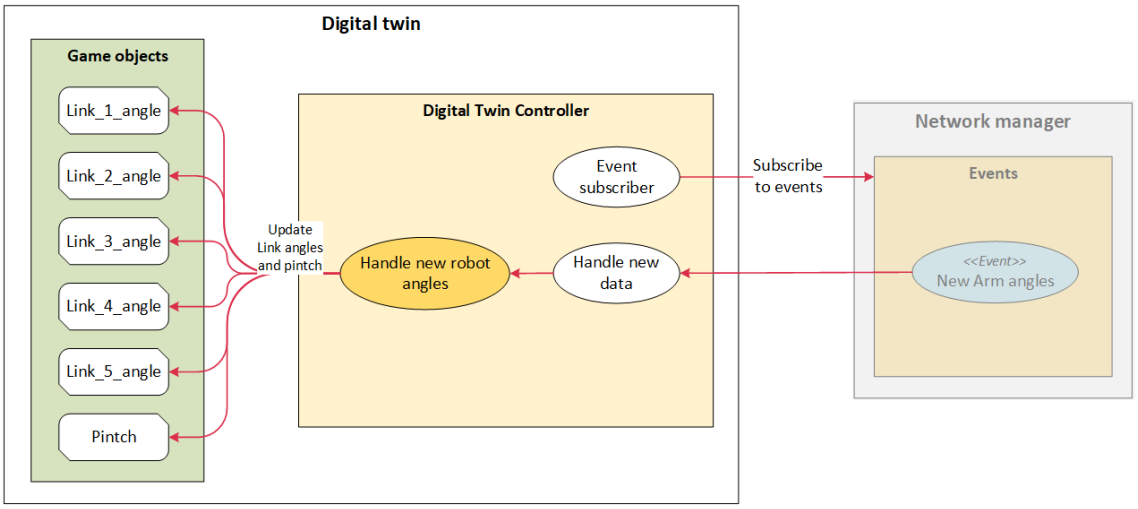


Figure 4.97: Data Distribution by the Digital Twin Controller

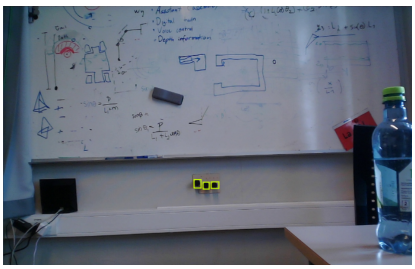
4.6.4 Detection of bolts using a self-made dataset

AD | *OM*

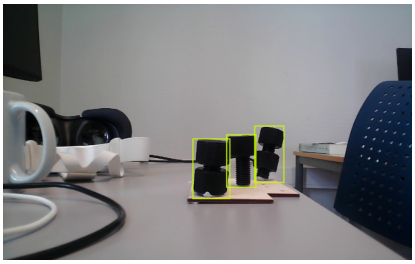
Automating bolt fastening and inspection can be a valuable addition to a potential offshore robot. Automating these tasks can enhance safety by reducing the need for human workers to perform repetitive operations. It can also improve efficiency and precision.

Detection: To show the fastening as a PoC, we decided to first detect the bolt. We found a 3D model for three sets of nuts and bolts on Thingiverse which was 3D-printed by one of our mechanical engineers [122].

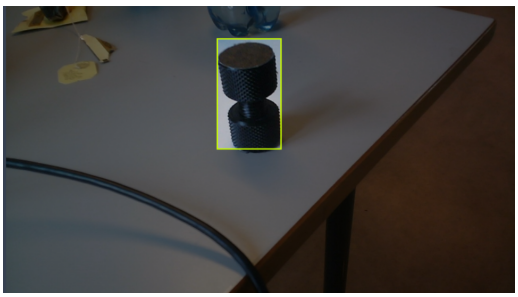
Our group utilized Transfer Learning with MediaPipe, which was the same process that was used to detect people for the object detection part of the project (see subsection 4.5). There was, however, an extra major task, namely to create the dataset. This was done with the help of the OpenCV library in Python. This involved capturing images of the bolts from various viewpoints, including top-down, side-views, and a frontal perspective. Additionally, images were captured at three distinct distances from the camera to account for potential variations in working range. We had 168 images in total, and we labelled the images using RoboFlow. The images can be found in the attachments. The dataset can also be found in the RoboFlow universe (robowflow/bolt-dataset).



(a) Far distance Image

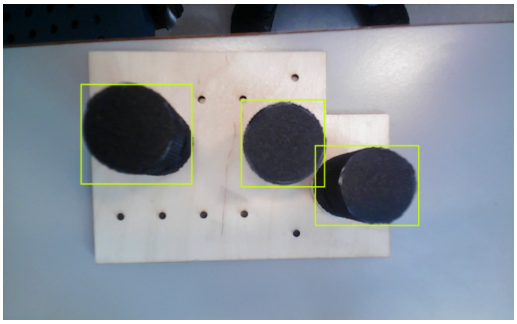


(b) Mid distance image

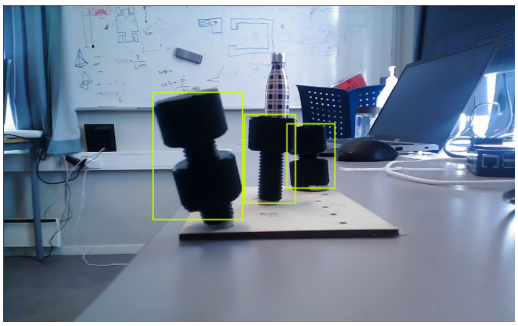


(c) Close range image

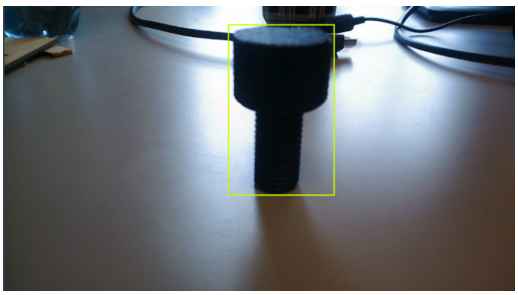
Figure 4.98: The different distance perspective



(a) Top perspective



(b) Side perspective



(c) Front perspective

Figure 4.99: Top - side - front perspective

All the images were resized to 640x640 in height and width, with the centre crop. We applied data augmentation to the images. We used saturation, which changes the intensity of the image randomly from -15% to 15%. We also added brightness ranging from -15% to 15%. This created a total of 404 images. 88% of the images (354 images) were used for training and 12% were used for validation. The images were exported in the Pascal VOC format.
During the training process in Google Colab, the hyperparameters from 4.5.15 were not changed. The results of the training are discussed in section 7.

4.6.5 Automated fastening of bolts

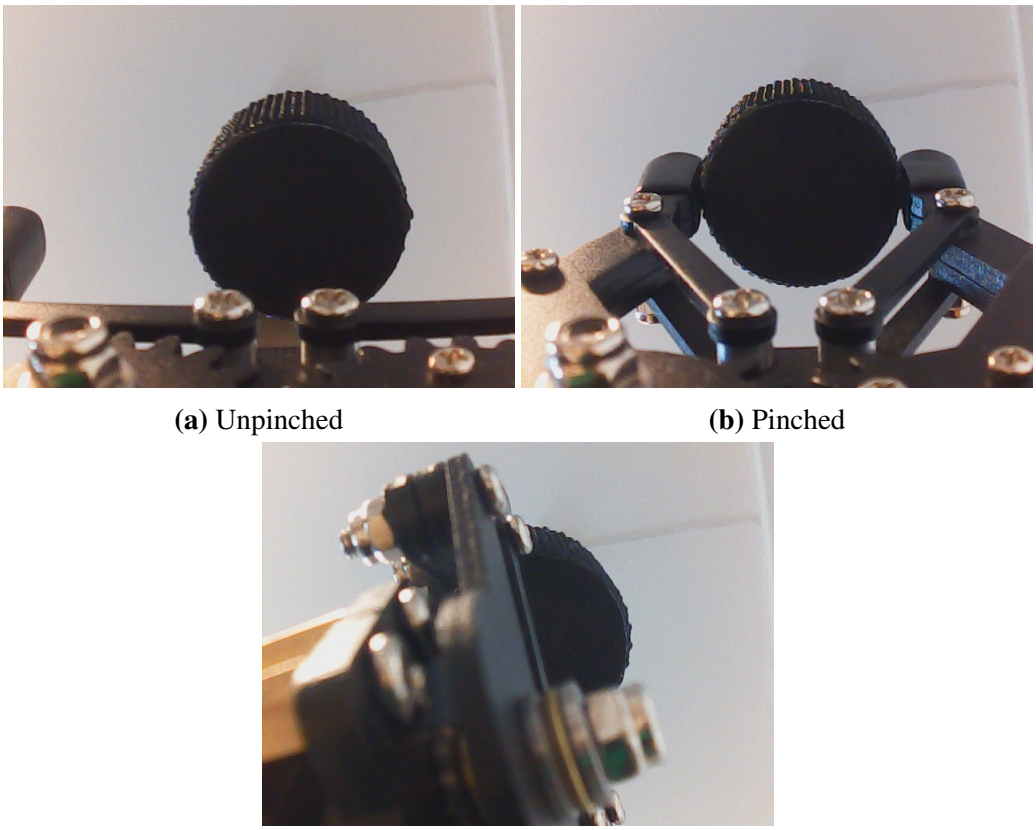
OM | *AD*

The robot can "screw" and "unscrew" bolts using the VR voice interface, as detailed in 4.6.8. When the operator says "screw," the JSON message below gets sent.

```
{
    "screw": "right"
}
```

In this context, "right" signifies clockwise rotation, and "left" denotes counterclockwise. This convention was established after implementing the interface. Upon receiving this message, the robot verifies that it is in "arm mode". If not, the message is discarded and not acted upon. However, if in the correct mode, the robot disregards any external arm commands to ensure the arm remains stationary during the operation.

The operation consists of timely pinching and unpinching while rotating around the bolt. This process is hardcoded to repeat three times before completing the operation.



(a) Unpinched **(b)** Pinched

(c) Rotated

Figure 4.100: Arm camera perspective of fastening without bolt detection

4.6.6 Reversing camera

OM | *AD*

Due to the robotic arm folding and being placed under the drive camera when the robot switches to drive mode (see 4.3.7), the idea of using the arm camera as a reversing camera came up. This feature was implemented, and an example image is shown below.

Figure 4.101: Reversing camera

Although the visibility is limited, it provides a better view than the drive camera when reversing. Since the camera is positioned upside down, the video feed is rotated 180 degrees to ensure it is oriented correctly, making it easier for the operator to control. This adjustment makes the arm camera multifunctional, as it is also used when controlling the arm (see 4.40).

4.6.7 Backup logging

OM, SO | *AD*

After feedback from our second presentation, the idea of external logging for incidents or data loss prevention was suggested. This was implemented by using the existing connection with the VR headset as a backup, which can be thought of as a cloud.

As explained in section 4.3.9, the system logs data to a local `.log` file. Before saving, it sends a backup log message via the TCP socket to the VR headset. An example of this message, using the dynamic dictionary getting (see 4.96), is shown below.

```
{
    "type": "log",
    "topic": "robot_data",
    "message": "accelerometer=geometry_msgs.msg.Vector3(x=0.023...-",
    "time": 1715801305009
}
```

The log time and data are sent to the VR application. By checking the `type` key, the VR application can distinguish between `robot_data`, `ping`, and similar messages.

Logging in the VR application The VR application now includes capabilities to both display the logs to the user on a virtual screen within the application and store all the logs in the headset for later retrieval. This ensures access to previous data for troubleshooting and analysis when necessary.

4.6.8 Integration of voice commands in the VR application **SO, AD** | *OM*

Our external supervisor had expressed enthusiasm for the potential impact of incorporating voice commands. Voice command enhances the experience of human-robot interaction. Using voice commands through the VR headset to send commands to the robot was then requested as an additional requirement. The voice command functionality we implemented provides a hands-free alternative to manual inputs, allowing the operator to issue commands without physical interactions. We have implemented a range of voice commands that enable operators to easily switch between the robot's operational modes and execute specific tasks. The following commands are supported by the system:

- **Idle:** Puts the robot and VR application in a standby state where no actions are performed.
- **Drive:** Changes the robot and VR application to drive mode.
- **Arm:** Switches the robot and VR application to arm control mode.
- **Emergency:** Sets the robot and the VR application to emergency mode.
- **Screw:** Informs the robot to start automated screwing.
- **Unscrew:** Informs the robot to start automated unscrewing.

Each command triggers a specific function or mode, requiring the operator only to say the commands out loud for the VR headset to pick up, similar to how we use voice commands on our cell phones. This functionality makes the application more flexible. For example, the operator can pinch and hold an object with the robot's hand and change to drive mode with a voice command to hold and move the object to another place with the robot.

The voice commands were implemented with the help of Wit.ai, a natural language processing (NLP) platform. Wit.ai allows us to translate spoken commands into actionable instructions for the VR application [12]. With Wit.ai, we created a new application tailored to our specific use case. The next step we did was to define intents, entities and utterances.

Utterances: To train the application to recognize our desired voice commands and extract the relevant details within them, we provided a set of diverse sample example phrases, *i.e.* what the VR user might say to enable an action. For example, if the user intended to screw a bolt, then their utterance would be the phrase, screw. From the commands list mentioned above, we entered all the possible utterances a user might say.

Intent: Intents are used to understand the overall meaning of the phrase uttered. Intents represent the goal of the user or the action they wish to perform. For example, if the user says "Activate emergency mode", they intend to change the mode of the robot. Therefore, we defined the intent to "change mode".

Entities: Entities are the specific details or parameters within the utterances. To recognize the intent, we need to focus the application's attention on looking for a specific word(s). So if we look back at the example of a user saying "Activate emergency mode", for the sake of simplicity the application does not need to pay any attention to the words *activate*, and *mode*. The most important word is *emergency*, as it means that the user intends to change the mode to emergency mode. Therefore, the word is further labelled as an entity.

All the modes were labelled as entities, along with the words “screw” and “unscrew”. Figures 4.102 and 4.103 show the wit.ai application and the different entities used.

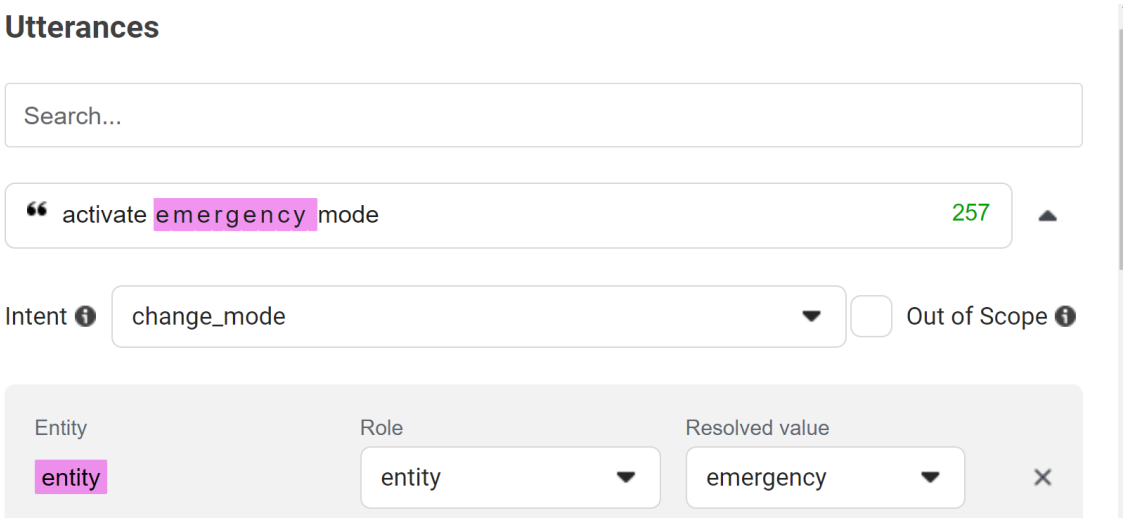
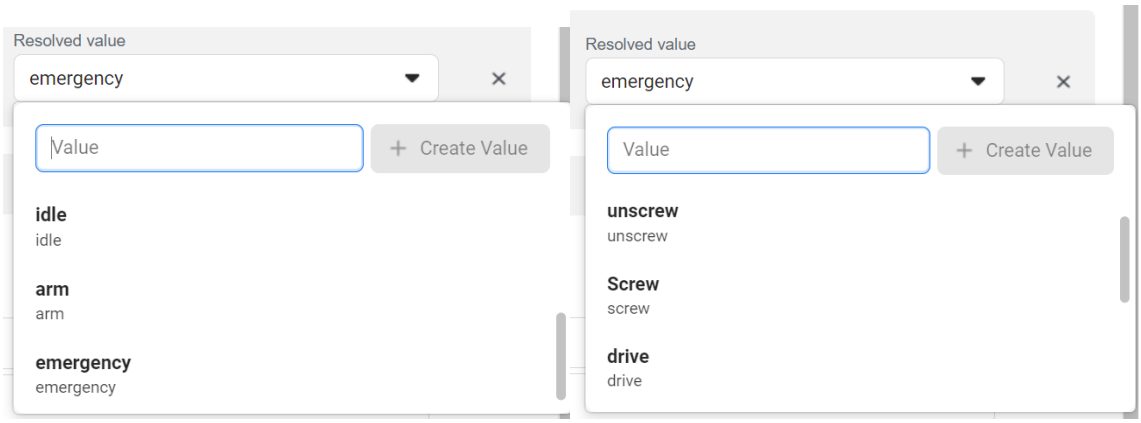


Figure 4.102: Wit.ai application interface



(a) Dropdown box menu **(b)** Dropdown box menu

Figure 4.103: The different entities available [12]

After defining the utterances, intents and entities, we trained and validated the application. Wit.ai’s ML algorithms utilize this data to learn the underlying patterns and relationships between the utterances, intents, and entities. The VR application integrates the wit.ai application, utilizing Unity’s Voice Software development kit (SDK) to capture the VR headset user’s voice. This technology provides essential functionalities for capturing vocal inputs. Once captured, the voice is processed through the wit.ai application, which is configured to recognize our predefined entities; idle, arm, drive, emergency, screw, and unscrew (commands). Whenever one of these commands is recognized, the VR application triggers an event in the *Voice Controller*. This controller is tasked with handling all voice commands, utilizing the application’s interfaces to change modes and send commands to the robot for execution. For a demonstration of this functionality, see the video on folder: Digital twin—ensure your volume is on!

4.6.9 Arm safety limits **OM** | *AEH*

As mentioned in section 4.4.6, one important consideration is whether a point is illegal. A point is considered illegal if it is likely to cause a collision with the front of the robot.

These potential collisions were identified during testing and movement of the robot. Certain angles would result in colliding with the robot's front, which could possibly cause damage. Since the group intends for visitors at the USNexpo stand to try out the robot themselves, it was important to minimize these collisions to reduce the risk of damaging components.

Illegal points are determined using a defined box. The dimensions of this box are as follows:

$$-26 \leq x \leq 0 \cap 10 \geq y \cap -2 > z \tag{4.37}$$

The values are in centimeters and were measured using a ruler by manually moving the arm around and observing. If a point falls within this box, it is considered illegal, and the robotic arm will not move there. This check is performed before any inverse kinematics calculations are made. If an illegal point is detected, the process is prematurely terminated, and the robot will wait for the next hand coordinate.

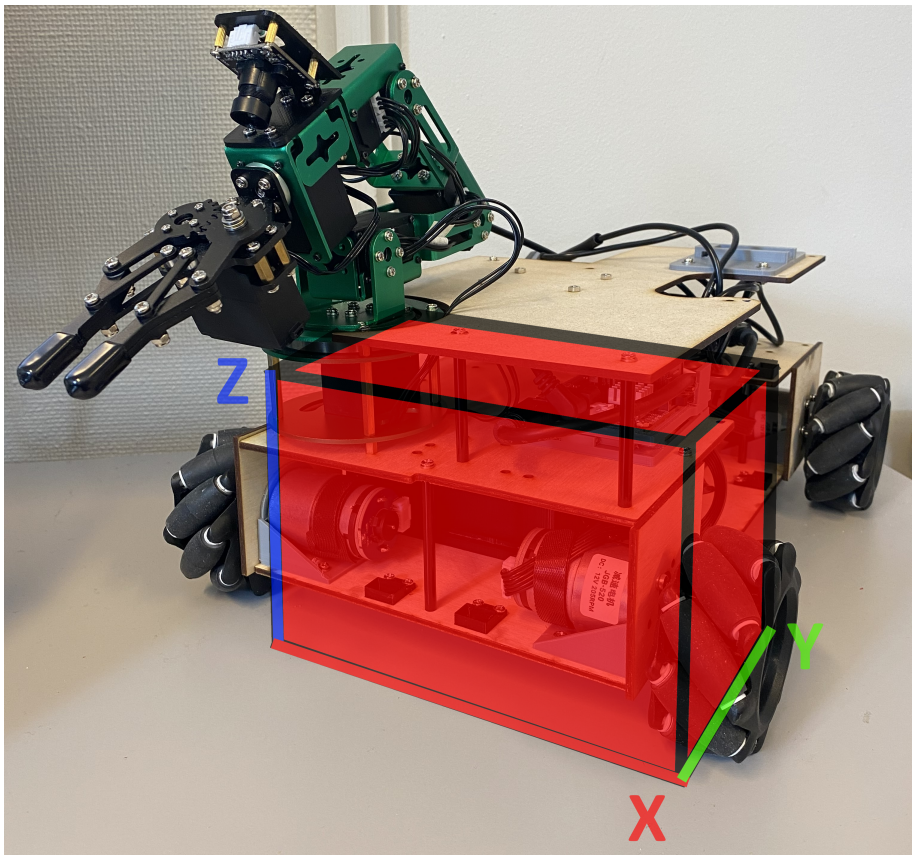


Figure 4.104: Illustration of box containing illegal points

4.6.10 Optimizing the video stream **OM** | *AD*

The current solution for the video stream, as referenced in 4.3.4, utilizes a WebSocket stream that continuously emits images. Initially, an alternative solution was implemented using a Hypertext Transfer Protocol (HTTP) server with *Flask*, as described in O.15. However, this was later changed due to concerns regarding overhead, latency, and suspicion of causing a memory leak in the VR application.

Comparatively, the WebSocket solution appeared to perform better than the *Flask* version, and the memory leak is assumed to have been resolved. Nonetheless, further video

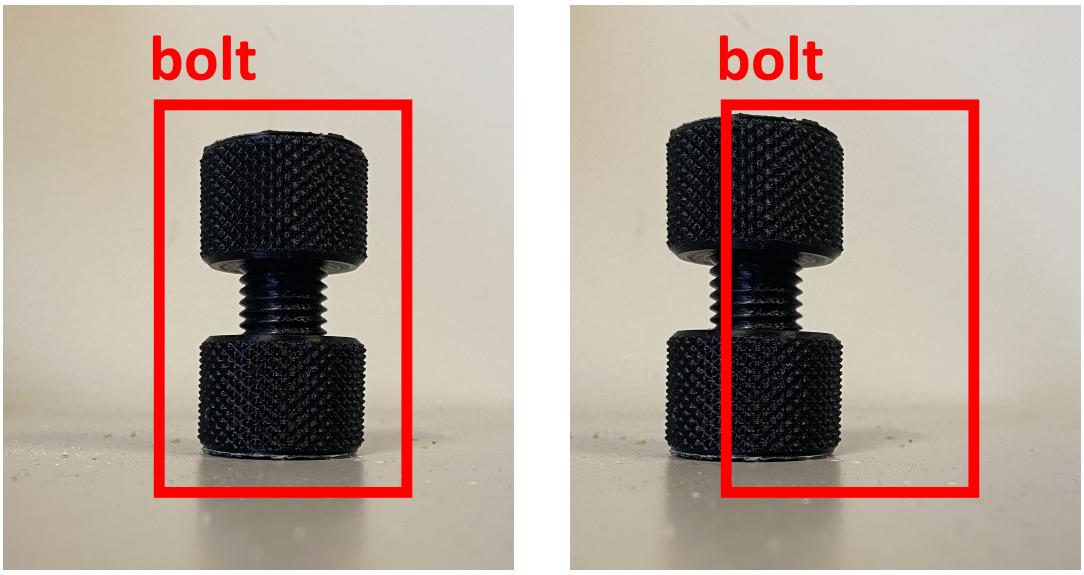
stream optimisation could be achieved by transitioning to a User Datagram Protocol (UDP) stream or a similar protocol. This would minimize the TCP overhead inherent in the WebSocket solution, resulting in reduced latency for the operator, which might be crucial in a production environment.

It is worth noting that, due to the nature of UDP, the delivery of datagrams is not guaranteed, but occasional frame loss would likely have minimal impact. Despite the potential benefits of optimizing the stream further, this was not deemed necessary, as the WebSocket implementation proved sufficient for the PoC, and there were no requirements for a faster stream than what was already functional.

4.6.11 Caching of object detection boxes **OM** | *AD*

Due to performance concerns, a "caching" mechanism for object detection results was developed and tested but did not make it into the final product. The concept involved running a separate thread that would continuously analyze the latest available picture and save the result in a variable. These results would then be applied to the final picture before sending it to the VR operator. The aim was to potentially accelerate the process by running the analysis as frequently as possible, assuming it would be slower than the VR application's request rate for pictures from the *Flask* endpoint (see O.15), thereby increasing the FPS.

It is worth mentioning that caching would have the side effect of making the detection boxes lag one or more frames behind, as illustrated in the images below.



(a) Camera has not moved **(b)** Camera has moved

Figure 4.105: Caching of detection results

The detection box could also be misplaced if the camera remained stationary but the object moved. The rationale behind this decision was to prioritize FPS for the operator, even if it meant slightly inaccurate detection boxes. If precision was crucial, the operator could stabilize the camera, and the detection box would update shortly afterwards.

Despite these considerations, the caching mechanism was not integrated into the final product. There was uncertainty regarding whether it would improve FPS, and the object

detection implementation underwent frequent model and implementation changes, which made the caching implementation not carry over. The code for this is available on GitHub [123].

4.6.12 Displaying latency of the robot

OM, SO | *AD*

Determining the latency every second a "ping" TCP socket message is sent from the VR application. This is a JSON message which includes the seconds since starting up the application, which increments over time.

```
{
    "type": "ping",
    "ping": 1593187.52443142
}
```

When the robot receives this message, it is sent back to the VR application without making any modifications to the original message. When it is received by the VR headset again, the time difference is calculated by taking the current time since the application was started and subtracting the received time to determine the delay between sending and receiving. The ping is an indication for the operator of how responsive the connection and system are.

```
private void ProcessPingData(OVRSimpleJSON.JSONNode jsonData)
{
    float latency = Mathf.Round(
        (
            Time.realtimeSinceStartup - jsonData["ping"].AsFloat
        ) * 100000
    ) / 100;
    ...
}
```

The latency is originally in seconds, converted to milliseconds and multiplied by 100 to not display more than two decimals of precision, which is often how latency is displayed. Latency display in the VR application is explained in 4.2.7.

```
pickle@sulfur:~$ ping hydroplant.no
PING hydroplant.no (158.36.70.44) 56(84) bytes of data.
64 bytes from 158.36.70.44 (158.36.70.44): icmp_seq=1 ttl=61 time=8.95 ms
64 bytes from 158.36.70.44 (158.36.70.44): icmp_seq=2 ttl=61 time=5.62 ms
64 bytes from 158.36.70.44 (158.36.70.44): icmp_seq=3 ttl=61 time=5.63 ms
64 bytes from 158.36.70.44: icmp_seq=4 ttl=61 time=6.18 ms

--- hydroplant.no ping statistics ---
^C4 packets transmitted, 4 received, 0% packet loss, time 3092ms
rtt min/avg/max/mdev = 5.615/6.593/8.950/1.379 ms
```

Figure 4.106: Ping command showing latency in milliseconds

4.6.13 Internal logging

SO | *OM*

Although internal logging was not part of the initial requirements, we quickly realized its necessity during the early stages of testing the VR application. Debugging and testing were exhausting because debug logs were printed on the development computer, requiring

developers to remove the VR headset to view the logs. To streamline the development process and enhance efficiency, we implemented an internal logging monitor within the VR application. This internal logging monitor allows developers to see useful debug information directly in the VR environment, reducing interruptions and accelerating development. The internal logging monitor is shown in Figure 4.107.

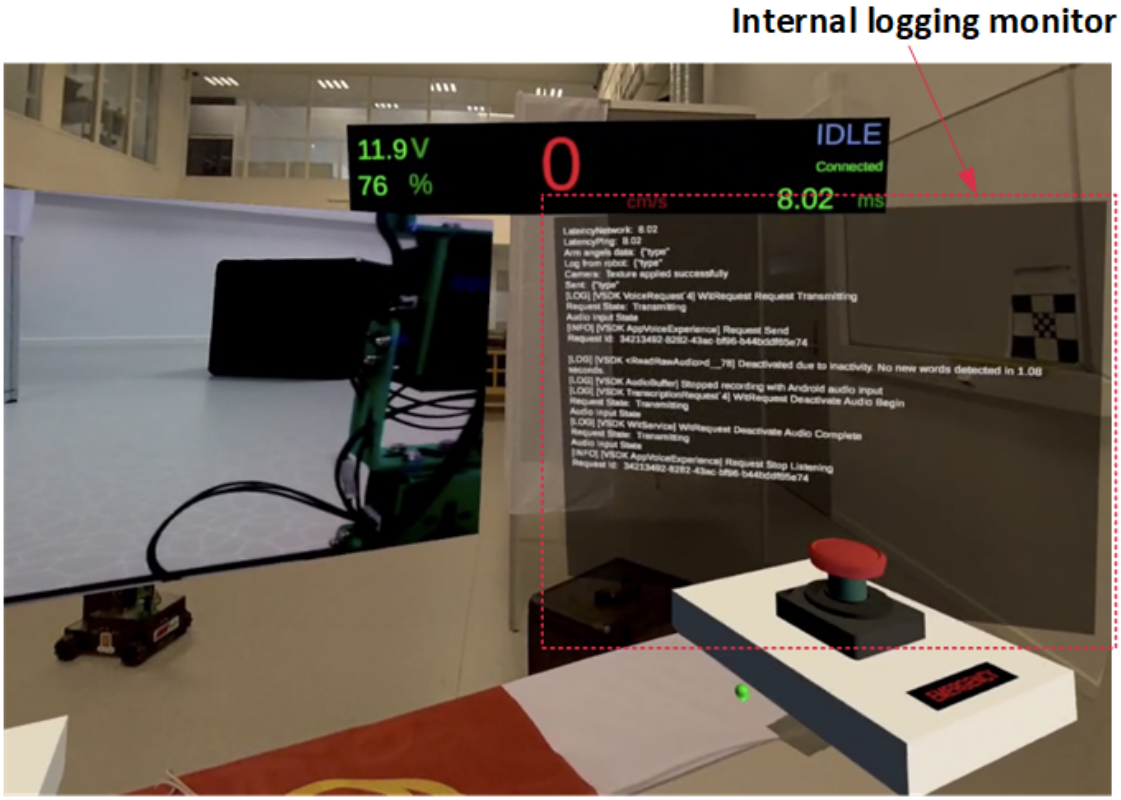


Figure 4.107: Internal logging monitor inside VR-application

4.6.14 Database integration with object detection

AD, OM | *AEH*

Displaying information with Databases One of the requirements in our project was to display information related to recognized objects. It was decided that a database that stored information about the targeted objects/classes would be created. The reason for opting a database was that databases provide a very structured way to organize information [124]. Storing data within the code is also difficult to manage especially considering the scale of the project. Databases are separate entities that can store data, therefore updating and retrieving information becomes easier [125].

There are many types of databases, but our group chose to approach a NoSQL (non-structured query language) or non-relational database, which allows unstructured and semi-structured to be stored in a database. Normally, relational databases are more popular and the software engineering students in the group had experience with using them but NoSQL databases were opted as they excel in managing unstructured and semi-structured data [124]. This flexibility is particularly beneficial for our use case, where detected objects exhibit diverse data types and features. Relational databases are not as suitable for handling the variability and non-uniformity of our data. With NoSQL, the data can be stored as it is detected, without needing to conform to a strict schema that simplifies data management.

MongoDB database program A Database Management System (DBMS) software program usually controls a database. It acts as an interface between the database and the user using it. It allows users to insert, retrieve, update data and much more [125]. MongoDB is a NoSQL DBMS that was chosen for our project because it offered a lot of features that greatly benefited us. Firstly, it stores data in flexible JSON-like documents, so that data can be added or changed easily without transforming. It also has a straightforward query language and a lot of community support [126].

Creating the MongoDB database Since the robot is already running as a Docker container, it was logical to install the database as a Docker container. The MongoDB database operates within its own Docker container, separate from the robot container, to maintain clear separation between the two. Connecting to the database does not require starting the robot container. Specific commands for this setup are provided in O.8.

Example data was created and imported into the database using MongoDB Compass [127]. This example data is shown below.

```
[
  {
    "class": "bolt",
    "addtional_data": {
      "length": 4
    },
    "color": "black"
  },
  {
    "class": "person",
    "additional_data": {
      "age": 24,
      "job": "Software Engineer"
    }
  }
]
```

Figure 4.108: Example JSON data used

Fetching data for detected objects When the object detection system identifies bolts or humans, the corresponding data is fetched from the database. Although this data could have been hardcoded, using a database demonstrates the ability to dynamically gather data based on detected objects. The database can be updated to include unique information such as installation time, manufacturer, previous issues, and dimensions, which can be valuable for remote operators.

Using the Python package *pymongo*, it is straightforward to establish a connection to the database and retrieve data.

```
from pymongo import MongoClient

client = MongoClient("127.0.0.1", 27017)
items = client["object_detection"]["items"]
```

Here, `127.0.0.1` is the `localhost` address of the system, and the default MongoDB port is `27017`. The database is named `object_detection` and the data is stored in the `items` collection. To fetch data, you can call:

```
items.find_one({"class": "bolt"})
```

This will return:

```
{
  "class": "bolt",
  "addtional_data": {
    "length": 4
  },
  "color": "black"
}
```

The retrieved data is then displayed for the operator as explained in the results section 7.3.

5 Electrical

5.1 Electrical components

5.1.1 Yahboom Battery **AEH** | *HB*

The main battery used to power the robot is a custom-made lithium battery pack delivered by Yahboom as part of the ROSMASTER X3 PLUS. The battery has a capacity of 9600 mAh at 12 volts, a rated capacity of 6 amperes, and a maximum discharge current of 10 amperes. It comes with a female T-type discharge port and 4017 charge port. The battery pack BMS also protects against over-current, over-charge, over-discharge, short-circuit and power-off. [74]

5.1.2 Custom built battery **HB** | *AEH*

The process of creating the custom battery was started after the project group received a new requirement from the client. To create the battery, the project group were handed some components that could be used to create the battery. The components were 21 battery cells from old laptop batteries, a bunch of short wires, six battery cell holders with some wires attached from previous projects, and some single-cell BMSs of the type 03962A.

To make this battery, several iterations were designed. The first iteration was a battery without the BMSs, and there was started to design a charger for the battery cells as well. Then the project group found out that it would be smart to use the BMSs and the first iteration was scraped. For the second iteration, some research had to be done because of the BMSs. This was because the BMSs were meant only for one cell each, and connecting them in series and parallel seemed like the only option at the start. But after researching a bit, it was found that connecting these in series would not be possible. The solution had to be that the cells themselves were connected in series and parallel and at the same time the BMSs were connected to each cell. Then a switch would be connected in between this to function as a “mode changer” to switch between charging mode and use mode. The switch was designed to look like a fork, that slides in and out to switch the different modes. The problem with this design was that the battery would have short-circuited every time the fork was taken out or put in. One solution that was designed to try to fix this was that the fork had to be inserted all the way, and then slide to the side to make contact, but this idea was scrapped.

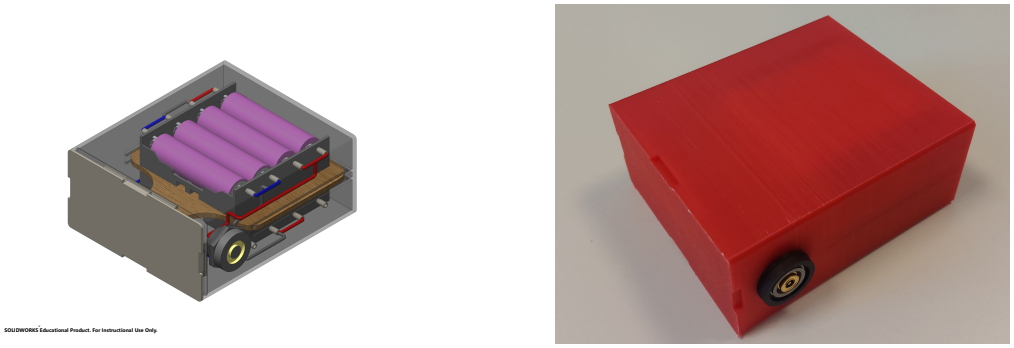
The next iteration of the battery had the same electrical circuit as the one with the fork switch, but the fork switch was swapped out with proper switches. The switches were of the category Double Pole Singe Throw (DPST). This meant that three separate switches were needed to be able to disconnect the battery cells from each other. The reason the battery cells had to be disconnected from each other is that the BMSs is not able to monitor the battery while connected in series, with more batteries.

Now the battery was built and tested. The first test was simply plugging the battery cells into the cell holder, and checking if there were any short-circuits, this test was a success. The next test was a simple voltage test, where the voltage was measured. This test was carried out to see if the battery gave the amount of voltage it was supposed to give. This test was also a success. The last test was the charging test. The battery was connected to

a charger, and the charger was plugged into an electrical outlet from the wall. Unfortunately, this test failed due to a short circuit that fried a component on one of the BMSs.

As the last iteration had failed, the project group had two options. One of them was to make a new, different circuit or make a battery without any BMS. As the deadline of the project was closing in, it was decided to go for a battery without the BMSs and that the cells either must be swapped out or recharged in a separate battery cell charger when the battery is flat.

To read more about the battery design and building process, see appendix J (Battery).



(a) Final battery design **(b)** Finished battery

Figure 5.1: Final battery design complete

5.1.3 DC motors **AEH** | *HB*

The DC motors used to drive the robot are the 520 DC gear motor with 1:56 gear reduction ratio. The max speed provided by these motors is 0.45 m/s and the max load capacity is 7-10 kg. [128]

5.1.4 Servos **AEH** | *HB*

To operate the robotic arm we use the YB-P15M 15 kg- and 6 kg- torque servos from Yahboom. The servos are connected in serial and communicate by single-bus with a baud rate of 115200. The servos have their own ID for communication which can be changed in the software provided by Yahboom [129].

5.1.5 USB 3.0 HUB expansion board **AEH** | *HB*

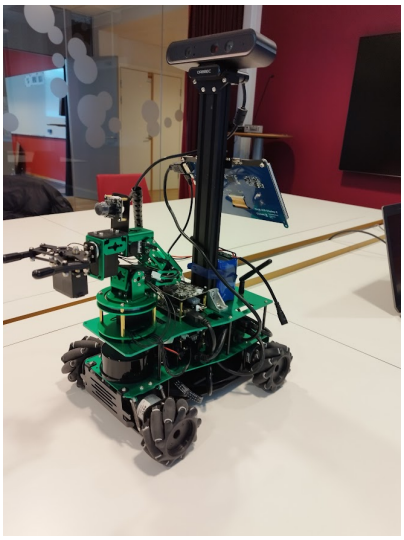
The USB 3.0 HUB expansion board is connected to the ROS robot control board and acts as a link between the expansion board (which does not have USB 2.0/3.0 type A connectors) and the RPi [130].

5.1.6 ROS robot expansion board **AEH** | *HB*

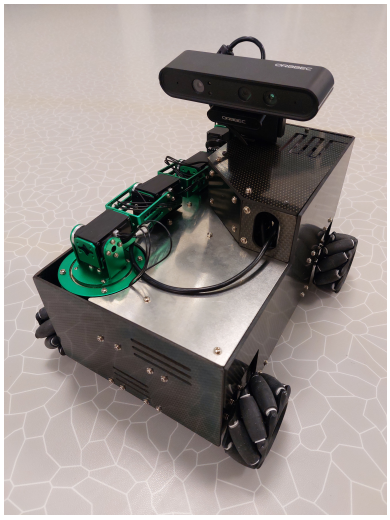
The ROS robot control board works as the link in the system by being connected to the Raspberry Pi, battery pack, servos, USB 3.0 HUB, and motors.[131]

6 Mechanical hardware development

This chapter contains the development process for the robot design from the equipment we were given at the start of the project to the final robot. At the end of the chapter is a section dedicated to the production of the different parts of the system.



(a) Start



(b) End

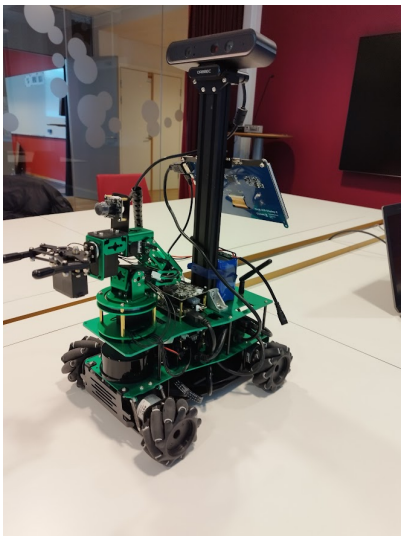
Figure 6.1: The robot at the start and end of the project

6.1 Given equipment

AEH | *HB*

In addition to the budget, we got the hardware from two previous bachelor projects which we could use the parts from. The first robot we got was the K-Spider[132] which had a lot of servos and some old electronics. The other robot we got was the ROSMASTER X3 PLUS from the K-AIoT[133] project. The ROSMASTER had a lot of different electronic components where most of it was usable but not necessary within the scope of the project. An example of this is the Lidar.

(a) The K-spider



(b) The ROSMASTER X3 PLUS from K-AIoT

Figure 6.2: Equipment from previous projects

6.2 Robot design requirements **AEH** | *HB*

The design of the robot is affected by the requirements defined in Appendix C. In addition to the requirements defined by the project, we have some internal requirements that will be defined in this chapter.

Camera position: The camera must be positioned such that the camera feed is not obstructed by the robotic arm and has visibility for steering.

Low centre of gravity: The heaviest components should be placed at the bottom of the robot to create a low centre of gravity. A low centre of gravity will give better traction, balance, and counterweight when lifting an object.

Space between components: To make sure that the components can work at the highest capacity possible without overheating, the components need enough room for air to flow throughout the case.

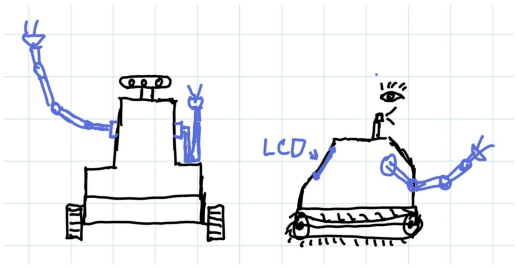
Space for two Raspberry Pi: The robot needs to have space for two Raspberry Pi in case the tasks have to be divided from one over to two, communicating via Ethernet.

Cable management: Enough space is needed inside the robot to line the cables.

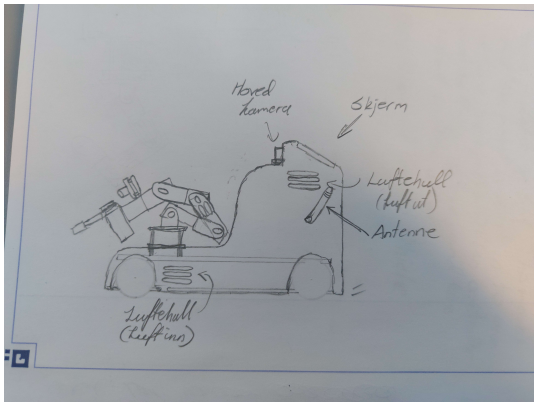
Padding/shock absorber: Padding and shock absorbers is needed to protect both the robot and the environment from collision damage.

6.3 Robot design ideas **AEH** | *HB*

We discussed many ideas both internally and with the first external supervisor regarding the design and capabilities of the robot. Figure 6.3a shows a robot using two arms, we concluded that the system needs to control one arm first before researching the possibility of adding more. This was more relevant as a possible later bachelor project. Before losing our electrical engineer 9.1.1 we also discussed the possibility of having a robot arm that can change between different hands and change the battery itself. To be able to change its battery we would need an internal power supply to power the system while the battery gets changed or two battery packs installed at the same time that it could change between while swapping the other. The consensus we got was a robot using a single arm to perform simple actions that may simulate offshore operations like turning a bolt, flipping a switch, or picking up something. The robot should not be too big so that it can be used to showcase the concept in small meeting rooms or offices. It should also be able to manoeuvre with a small turn radius ideally rotating around its own axis for this reason. The robot should have an outer shell that both protects the electronics but also looks good for spectators when showcasing the concept. With the robot structure at the beginning of the project, you have to dismantle most of the robot to access and change electronic components that are stacked on top of each other. Therefore a modular design where the electronic components can be interacted with individually is needed both during development and for checking the system during demonstrations.



(a) Design idea: robot with two arms



(b) Design idea: arm front, tech back

Figure 6.3: Some of the design ideas

6.4 Robot iteration one

AEH | *HB*

For the first iteration of our robot, we wanted to get a better understanding of the components we acquired and which were relevant. We removed a lot of the electronics such as the lidar, voice module, Jetson NANO, Raspberry Pi screens, LED strip and extra USB expansion board. Instead, we added the Raspberry Pi 5 8 GB to control the robot. The remaining components did not need the whole structure of the robot so we removed the top floor as well as the large 20 mm x 40 mm profile that held the Astra Pro camera and the screen. We made a bracket T.1.1 to fasten the robot arm at the front of the car, a cover T.1.2 to protect the electronics during testing, and a new support for the camera T.1.3.

6.4.1 Robot iteration one, assembly

AEH | *HB*

After modifying the robot as discussed in the previous section and painting the plywood black for the second presentation, it looked like the picture 6.4. The results from testing the first iteration can be found here: 7.8.1.

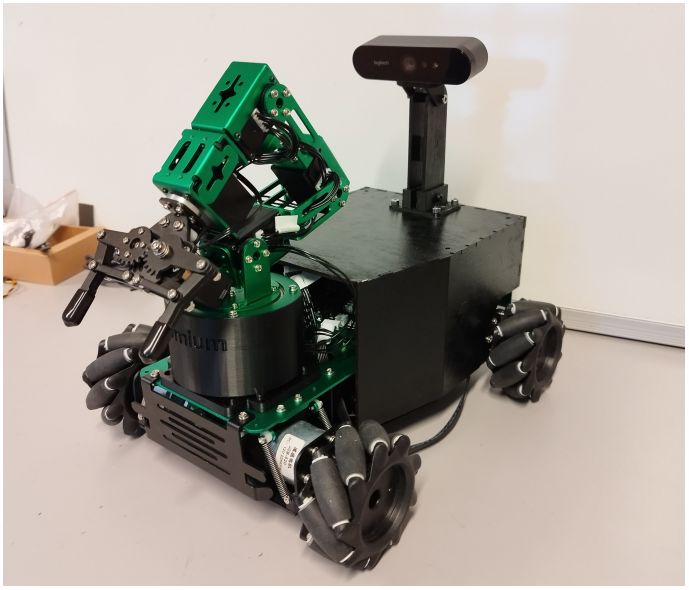
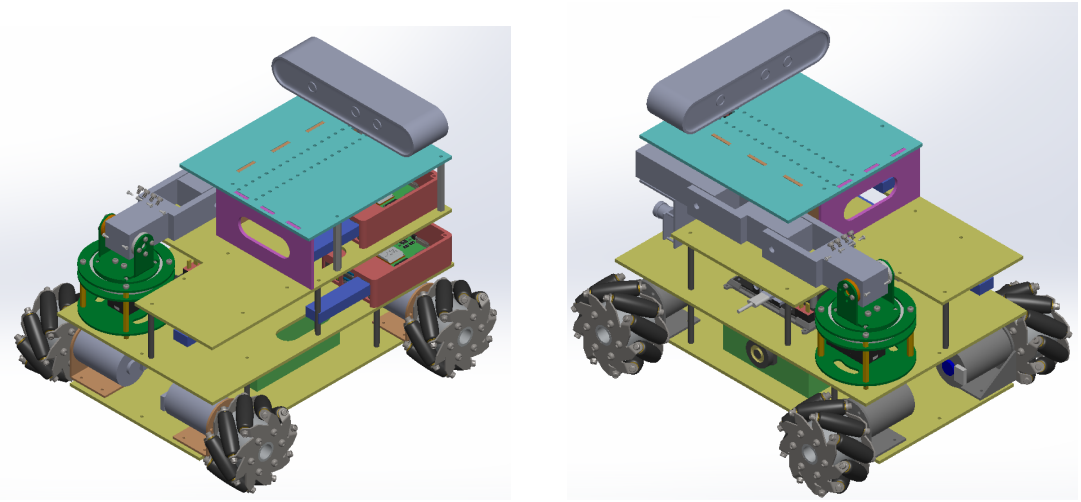


Figure 6.4: Robot at 02.03.2024

6.5 Robot iteration two

AEH | *HB*

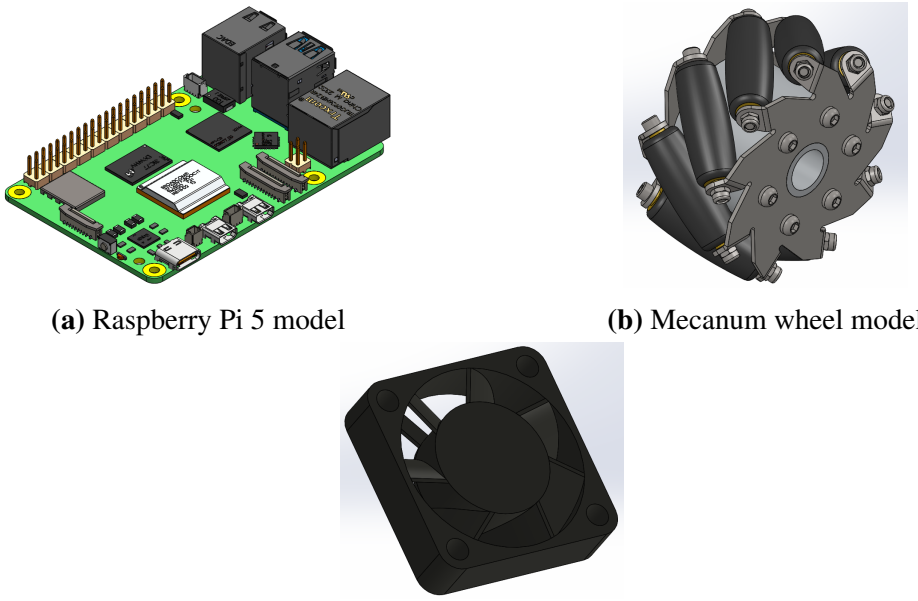
The second robot design was made in *SOLIDWORKS*. In this new design, the arm is placed on the front right of the robot. This placement has multiple purposes; firstly, placing the arm on the right side makes it more intuitive since the operator controls the arm with his or her right arm; secondly, when the arm is moved backwards it gives the operator better visibility when driving the car. Storing the arm backwards also makes it possible for us to use the arm camera as a reversing camera. The robot design assembly is comprised of models made by us, models made following technical drawings, and models downloaded from *GrabCad* and *Yahboom*.



(a) Isometric view **(b)** Opposite isometric view

Figure 6.5: Iteration two SOLIDWORKS model

The Raspberry Pi 5 [134], Raspberry Pi 5 fan [135], and mecanum wheels [136] used in the assembly were downloaded from *GrabCad*.



(a) Raspberry Pi 5 model **(b)** Mecanum wheel model

(c) Raspberry Pi 5 fan model

Figure 6.6: Models downloaded from *GrabCad*

The arm model was acquired from a different robot sold by *Yahboom*, we downloaded the model from a Google Drive [137] linked to the website and modified it to fit our model.

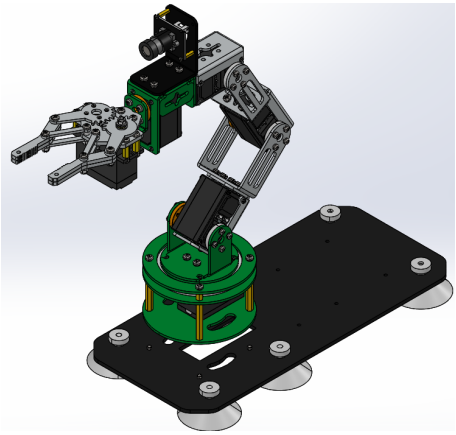
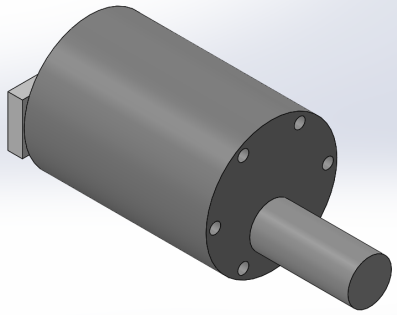
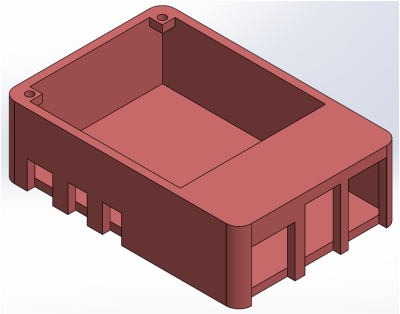


Figure 6.7: Robot arm assembly from *Yahboom*

The 520 DC motors and Raspberry Pi 5 case were made by measuring the physical parts and recreating them in *SOLIDWORKS*.



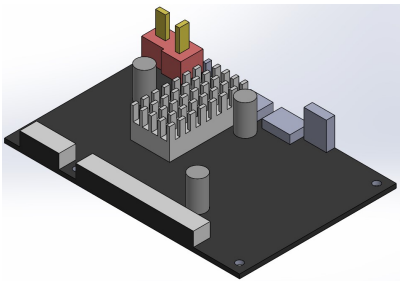
(a) 520 DC motor simple model



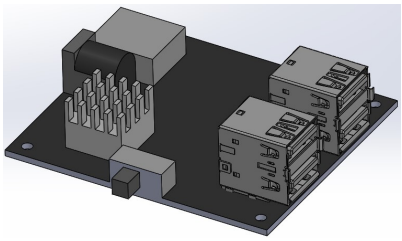
(b) Raspberry Pi 5 simple case model

Figure 6.8: Simple models motor and case

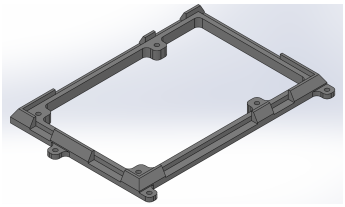
The motor expansion board and USB hub expansion board were designed individually as one part following the technical drawings acquired from the *Yahboom* website in addition to physical measurements. These models were used to design the brackets around as well as visual aid in designing the robot. The mass properties in *SOLIDWORKS* are measured by the volume and material weight of the model. For the parts with unknown material, the weight was found by scale and the mass properties in the program were overwritten to match the real weight.



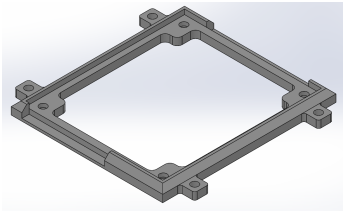
(a) Motor expansion board model



(b) USB hub model



(c) Motor expansion board bracket



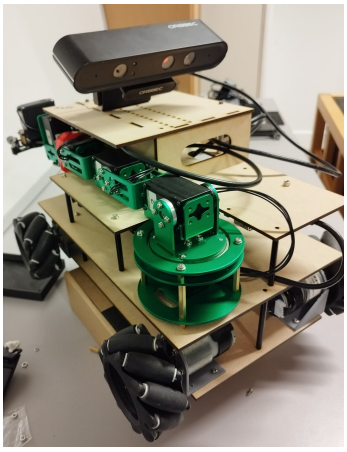
(d) USB hub bracket

Figure 6.9: Expansion board models and brackets

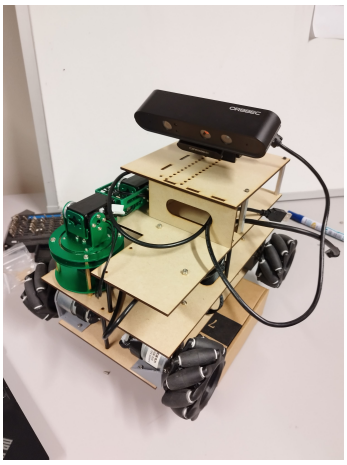
6.5.1 Iteration two assembly

AEH | *HB*

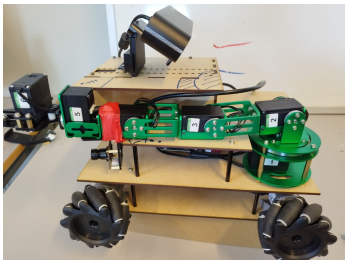
To test the second design the sheets were cut out of plywood and the brackets for the motor expansion board, USB hub and motors were 3D-printed from Polylactic acid (PLA). For more details regarding the testing, see 7.8.2



(a) Front left view



(b) Front right view



(c) Right side view

Figure 6.10: Iteration two assembled

6.6 Robot iteration three

AEH | *HB*

In this iteration of the robot, we made a friction-fit bracket for the Raspberry Pi case. We designed a bracket for the power switch and designed walls and brackets 7.15. Most importantly we integrated the new battery drawer assembly 6.9.

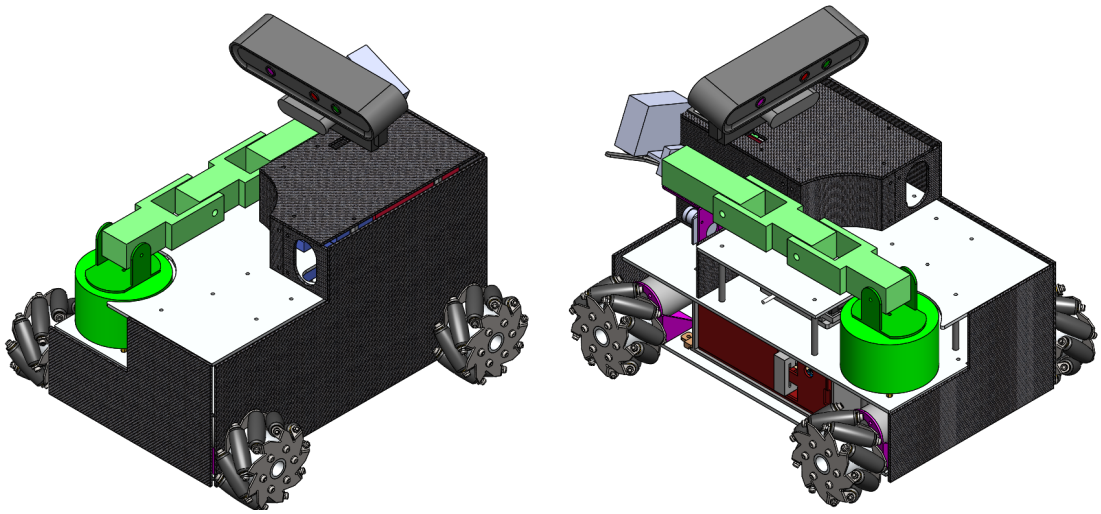
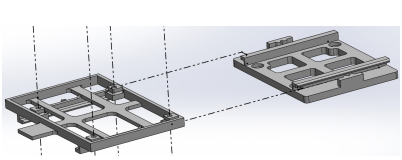


Figure 6.11: Robot iteration three design

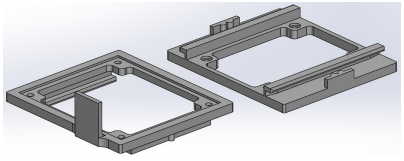
6.6.1 Robot iteration three redesigns

AEH | *HB*

The overhang over the arm was meant to protect the camera when changing operational modes in case a fault in the system caused the arm to go straight up instead of following the assigned path. This has been removed to make space for the arm. We decided that it's better to bump into the camera than have the servos try to push against resistance and possibly burn out.
We made a circular shape on the roof to make space for the arm to rotate when partially upright. The expansion board brackets were changed to a snap lock rail design for easier access. For easier access to the components we also removed part of the edge on the middle and top floor to make an entryway for the cables.
The robot arm 6.7 we got from yahboom gives us a lot of error messages in the 3D assembly and can also not be manipulated. Therefore we designed a simplified version of the robotic arm 6.13 that we could use in the main assembly as a reference and export to use as a digital twin 6.10.



(a) New expansion board bracket



(b) New USB hub bracket

Figure 6.12: New expansion board brackets

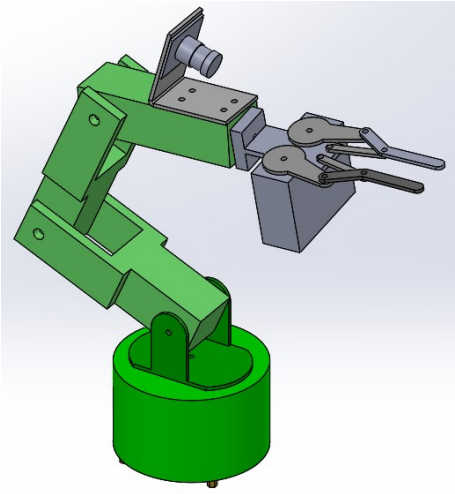


Figure 6.13: Simple robot arm model

6.6.2 Robot iteration three assembly

AEH | *HB*

The assembly of iteration three also used plywood sheets and 3d printed parts. The testing can be found here: 7.8.3.

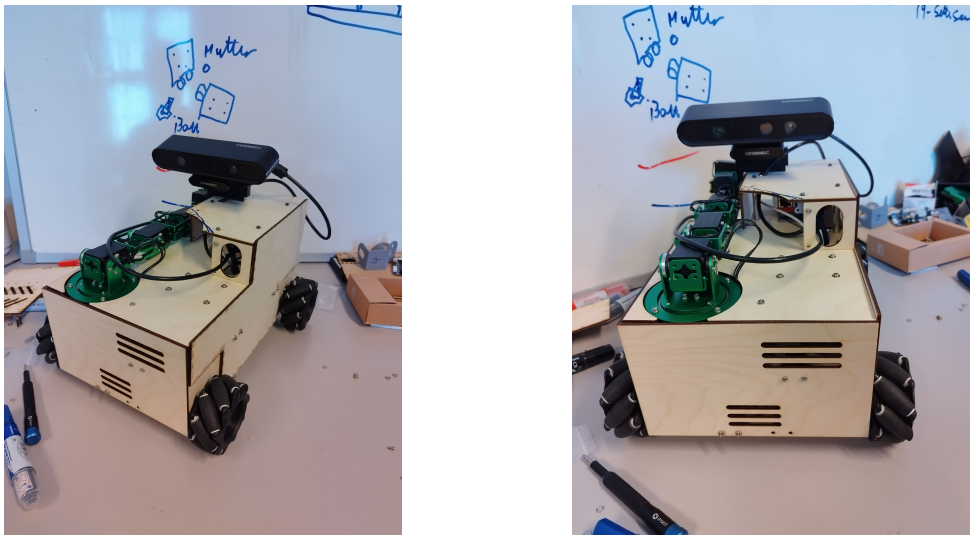


Figure 6.14: Robot iteration three assembly

6.7 Robot iteration four

AEH | *HB*

In this iteration, we added hinges and a magnet bracket to the right wall.

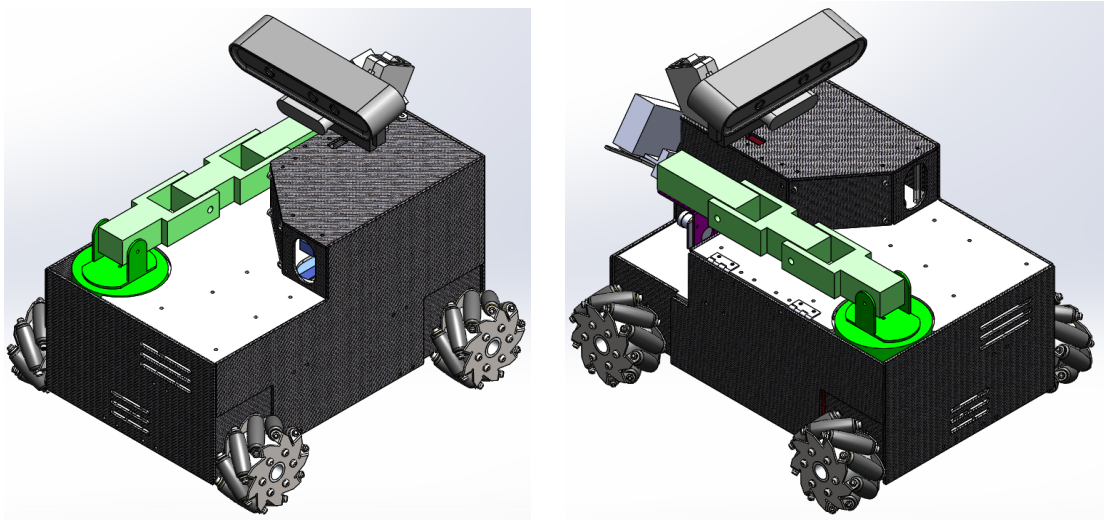


Figure 6.15: Robot iteration four design

6.7.1 Robot iteration four redesigns

AEH | *HB*

The front of the roof had to be redesigned again to be a straight cut instead of a half circle because the arm collided with it 7.8.3. The pathway for the cables also got adjusted larger.

6.7.2 Robot iteration four assembly

AEH | *HB*

The testing of iteration four can be read here: 7.8.4.

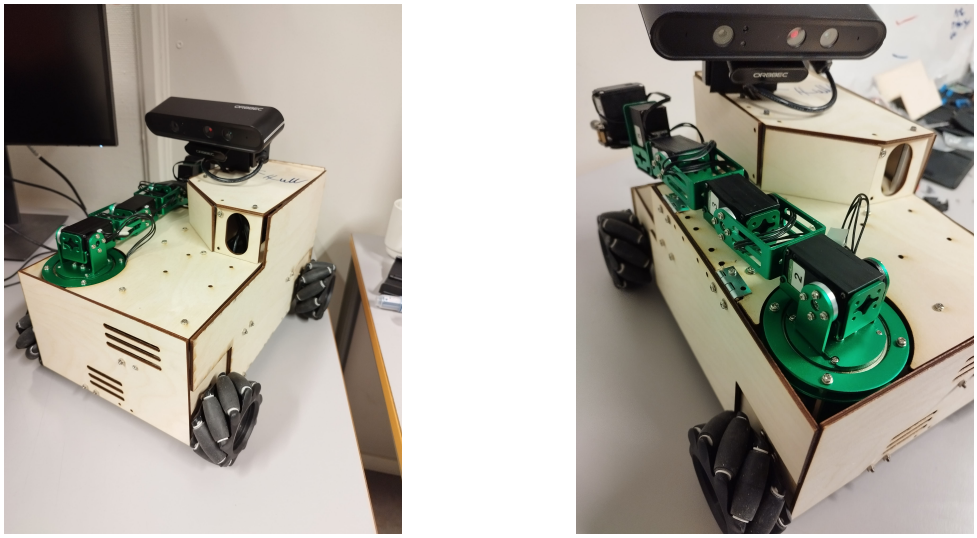


Figure 6.16: Robot iteration four assembly

6.8 Iteration five design

AEH | *HB*

In the fifth iteration of the robot design, we added a vent at the top of the roof for the Raspberry Pi fan to blow out of. For the front and rear of the car, we added the crash pads with the Kromium name extruded on them, these are meant to be 3D printed from a rubber-like material and act as a layer between the robot and whatever it may collide with.

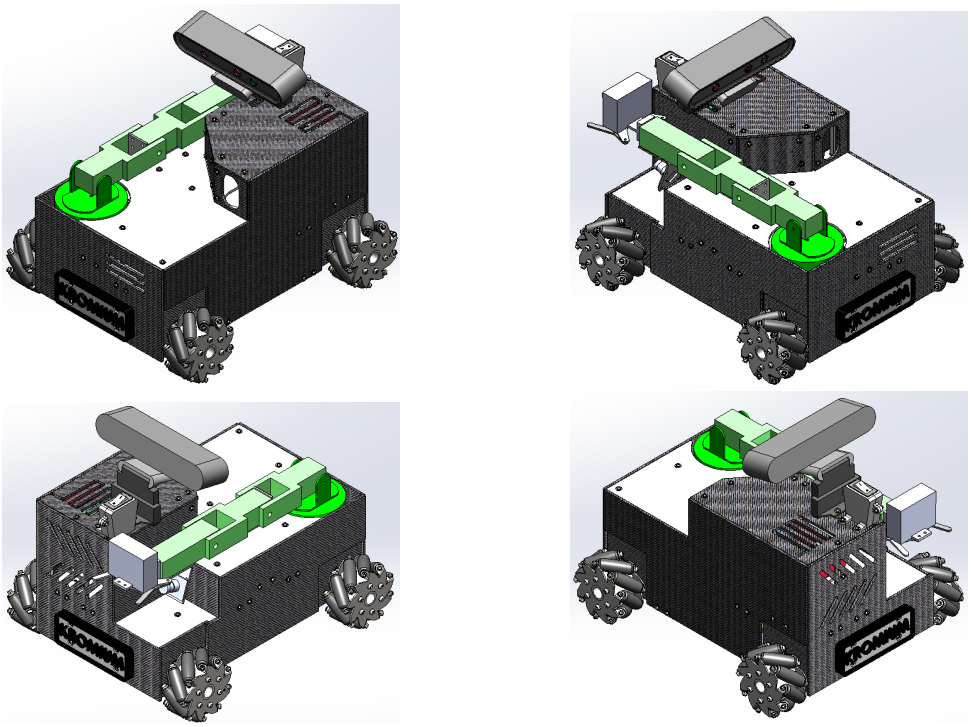


Figure 6.17: Robot iteration five design

6.8.1 Robot iteration five redesigns

AEH | *HB*

As a result of the iteration four tests 7.8.4 we had to adjust the pathway for the wires and make it larger. The Raspberry Pi bracket T.67 was changed from sliding backwards to sideways. The power switch bracket T.66 was changed to point upwards and the standoffs in the middle of the cable pathways were moved to the outer edges.

6.8.2 Robot iteration five assembly

AEH | *HB*

The fifth iteration of the robot is built using aluminium sheets for the floors, carbon fibre sheets for the outer shell and 3D-printed components for brackets. Information regarding the production of these parts can be found in chapter 6.11. All the parts in the system and how to assemble the robot can be found in chapter T.3.

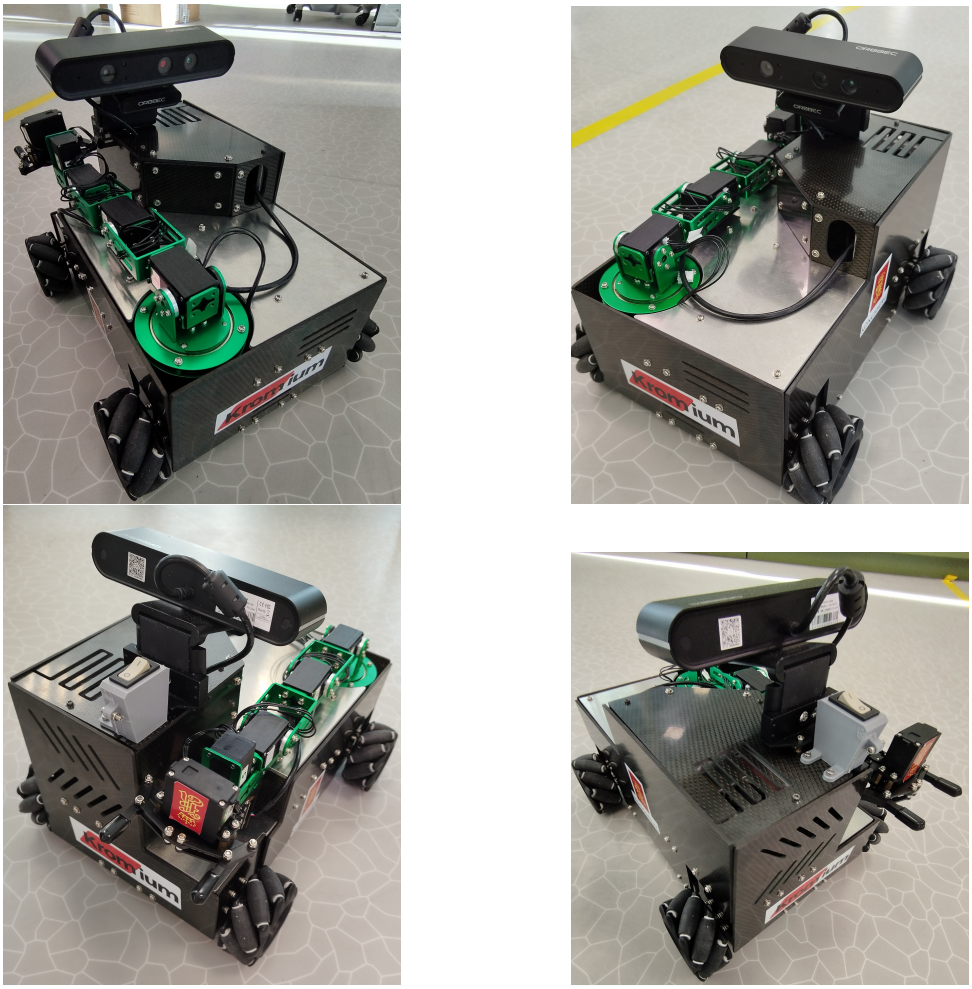


Figure 6.18: Robot iteration five assembly

6.9 Battery design

HB | *AEH*

The battery design had several iterations before the final design. The reasons for this are new decisions being made during the development process, and some improvements had to be implemented.

6.9.1 Battery iteration one

HB | *AEH*

The first design of the battery was a 3S2P battery that did not have BMS, a charger was also designed. This was so that the BMSs could be used. The reason the BMSs were not added to the battery was that the space required from the battery would be too wide to fit in the robot. Because the battery had no BMSs it would not be able to charge them. Therefore, the cells had to be taken out of the battery and charged in a separate charger, before they could be inserted into the battery, and then the battery could be used again. Another reason was that the BMSs had not been properly researched at the time, and how they worked was not clear.

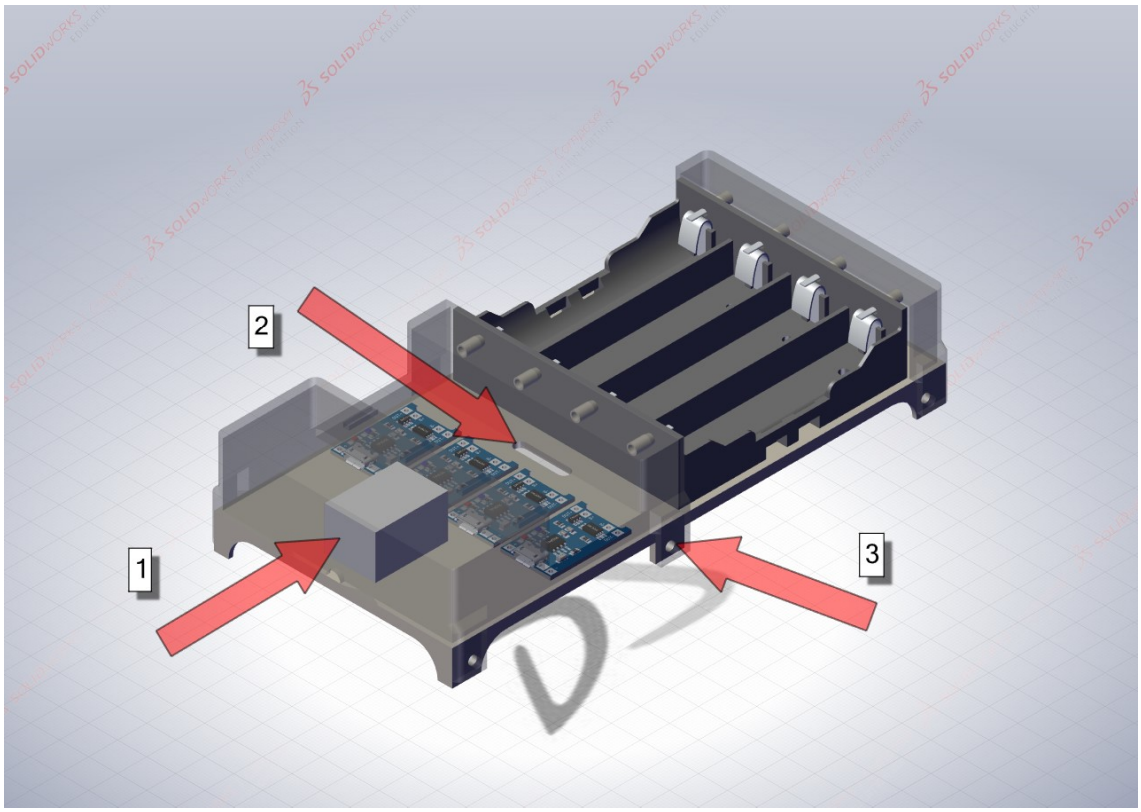


Figure 6.19: A part of the first iteration. Arrow 1 points towards a block representing the attachment between the charger wires and wires leading to the BMSs. Arrow 2 points towards a hole where the wires from the end will appear. Arrow 3 points towards one of the attachment holes for the cover

6.9.2 Battery iteration two

HB | *AEH*

In the second iteration of the battery, BMSs were added. The reason for this was so that the battery could be charged without the need for removal of the cells. However, due to the way the BMSs were designed, they could not connect the outputs together in a series. This would lead to at least one BMS being fried. Because of this, the BMSs can not use their output connectors. The batteries themselves have to be connected directly to each other and the BMSs must be connected to each cell at the same time. Because of this, the battery needs a switch which can disconnect the cells from each other so the BMSs can monitor the cells, so they wo not overcharge. However, with this solution, the cells cannot be monitored while the battery is in use because the cells are connected to each other. Luckily the expansion board on the robot has a built-in alarm when the voltage is below 9.6 volts[84, p. 1].

The second iteration of the battery has a BMS for each cell in the battery. This makes the connections inside the battery a bit more complicated than in the first iteration. This is because the BMSs can not be connected in series because they are single-cell BMSs. This means they will not be able to monitor the designated battery cell when the battery is in use and will therefore not have a purpose when the battery is in use.
When charging, the BMSs will be disconnected in such a way that they will only be connected to the designated battery cell. This way the battery can safely be charged and overcharging is avoided. The only disadvantage of this battery is the battery cells are not protected from over-discharging. Luckily the expansion board on the robot has an inbuilt alarm if the battery voltage drops below 9.5 volts.

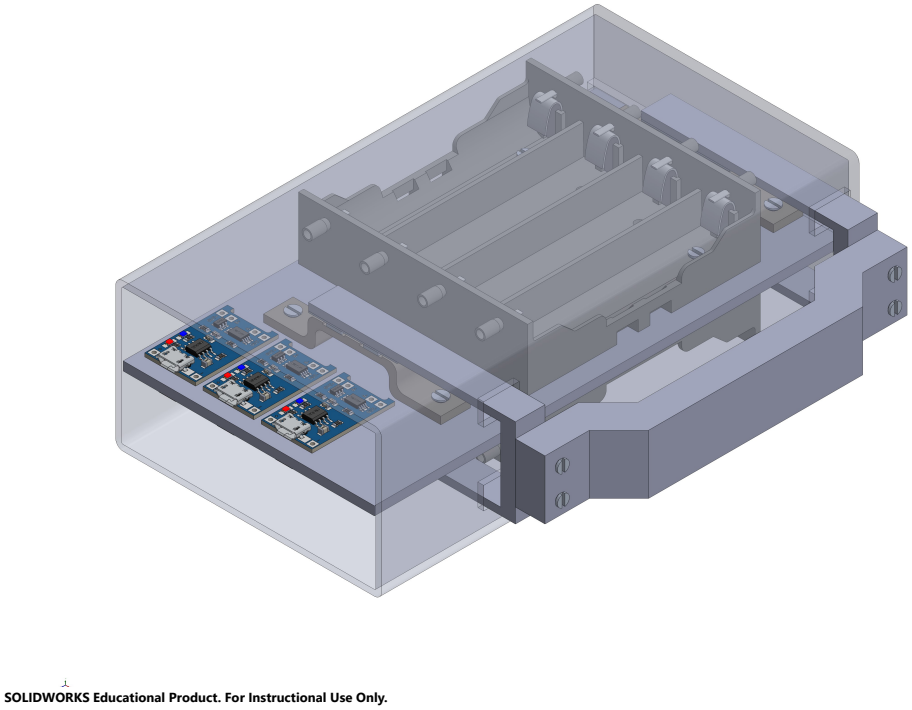
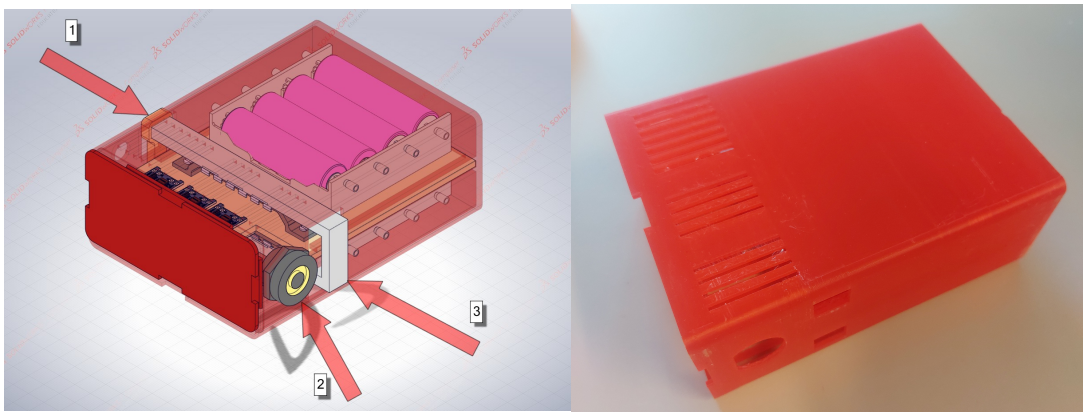


Figure 6.20: The second design iteration of the battery

6.9.3 Battery iteration three

HB | *AEH*

In the third design iteration of the battery, the handle was removed from the switch mechanism, and the two switches were reduced to one switch. This was made possible by making the connector plate bracket hold four connector plates each instead of two. One of the reasons for this was that the battery would not fit within the robot car. Another thing that was done to save space was to rotate the battery cell holders 90 degrees. This way the battery could be even shorter than by just removing one of the switches.



(a) Battery design iteration three. **(b)** Battery box from iteration 3

Figure 6.21: In figure a: Arrow 1 points towards the holder for the mechanical switch. Arrow 2 points towards the magnetic connector. Arrow 3 points towards the mechanical switch

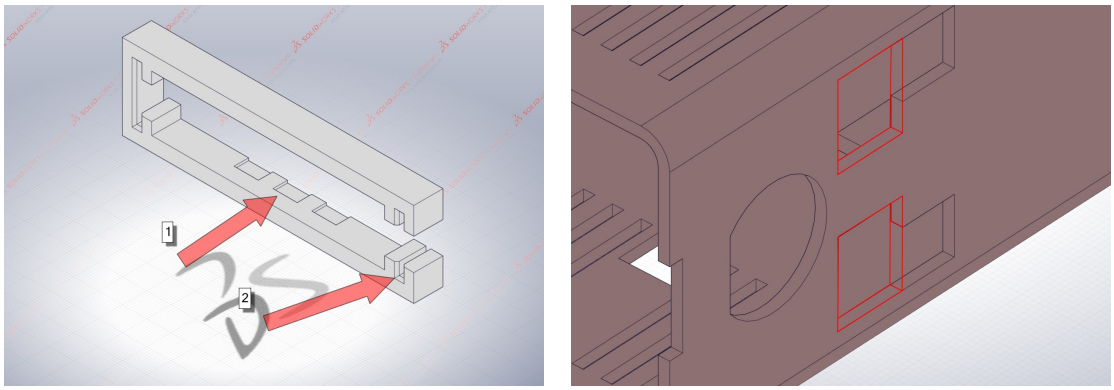
6.9.4 Battery iteration four

HB | *AEH*

In the fourth iteration, the mechanical switch received an improvement. The reason for this was that the switch was sliding over the connectors in the battery. This would have led to a short circuit every time the switch would be taken in or out.

To prevent a short circuit when inserting or removing the switch, the opening for the switch was made larger and there was added some taps to the switch. The larger hole in the battery box made it possible to slide the switch in and through the whole box, then slide the switch into position without the connectors on the switch touching other connector plates than they are supposed to. The taps on the switch make it possible to attach the switch to the battery box, so the switch will not move while in use and disconnect.

A side effect of switching to proper switches rather than the makeshift switch was that the connector bracket that was between the BMSs and the battery cell holders could be removed. This led to that the length of the battery could be reduced further. In total, the battery with the BMSs had reduced its length from 161.5 mm from the second design iteration to 145mm for this design iteration.



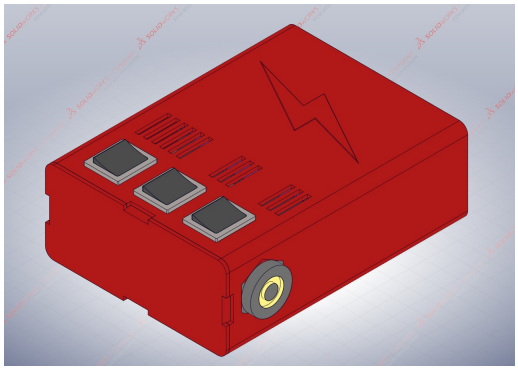
(a) The mechanical switch, or fork switch **(b)** Marked area is entry hole for the switch

Figure 6.22: Arrow 1: area for connector plates. Arrow 2: wedging mechanism

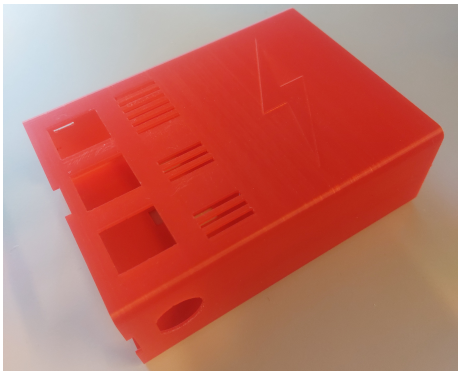
6.9.5 Battery iteration five

HB | *AEH*

The fifth iteration was swapping out the large fork-like switch, to three separate switches of the type DPST. This type of switch has two separate inbuilt switches that are controlled by a single trigger. This change meant that the battery box had to get a redesign as well. The holes for the previous switch had to be removed, and new holes for the new switches had to be made. Because the new switches were placed on the top of the battery, the battery had to be made slimmer. This was because the height of the space for the battery was limited to 57 mm, and the battery height exceeded this with the new switches. A side effect of changing to proper switches rather than the makeshift switch was that the connector bracket that was between the BMSs and the battery cell holders could be removed. This led to that the length of the battery could be reduced further. In total, the battery with the BMSs had reduced its length from 161.5 mm from the second design iteration to 145 mm for this design iteration.



(a) Moddel of iteration five

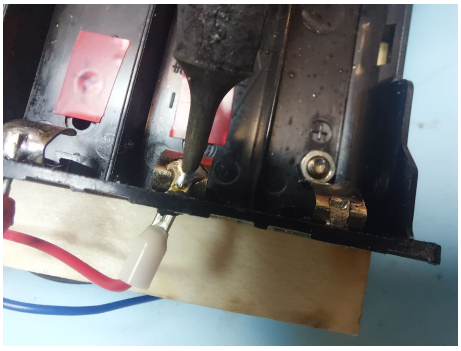


(b) Battery iteration five cover

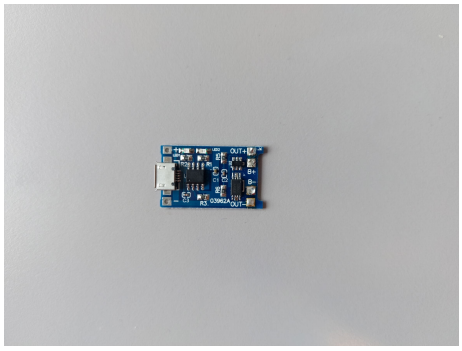
Figure 6.23: Battery iteration five

The production of the battery started once the designing process was complete. The first part made was the separator plate for the battery cell holders, this was to attach the battery cell holders. This part was laser-cut from a 3.5 mm thick plywood plate. When the battery cell holders were attached, the BMSs were attached using double-sided tape. Due to the reduced space, three BMSs were attached to each side. Then the wires were soldered to the components and arranged in proper order. Then the cable lugs were attached to the wires, and then the cable lugs were attached to the switches. While these operations were done, the battery box and the lid for the battery box were being 3D printed. This ended up being scrapped due to a failed test. See 7.9.2 for the reason this battery was scrapped.

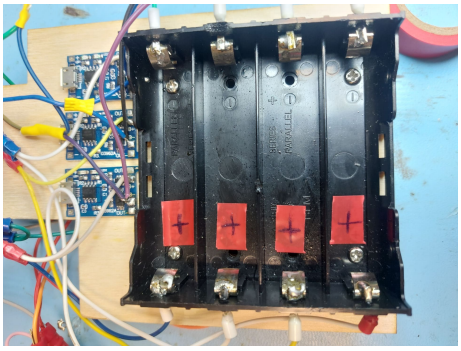
(a) Attaching cable lugs



(b) Soldering wire to cell holder



(c) BMS without wires



(d) Marking of polarity

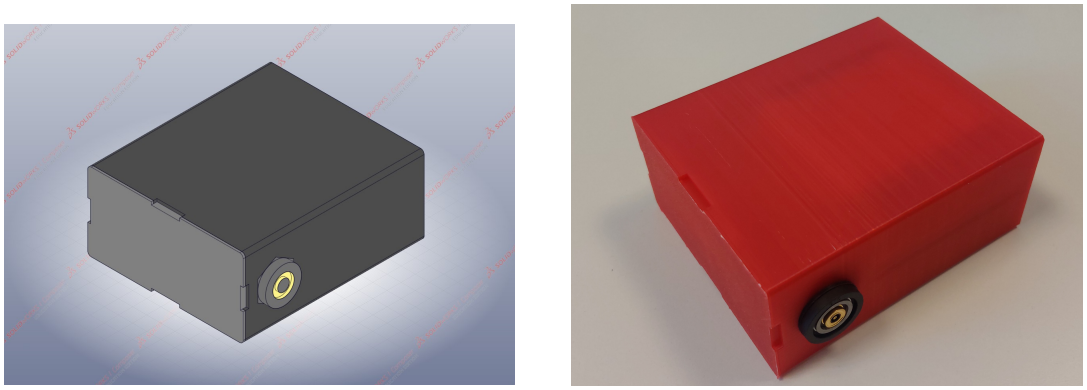
Figure 6.24: Producing battery iteration five

6.9.6 Battery iteration six

HB | *AEH*

The sixth iteration was almost a complete redesign of the battery. This was because of a failed test of the previous iteration. The BMSs were removed, which meant much simpler

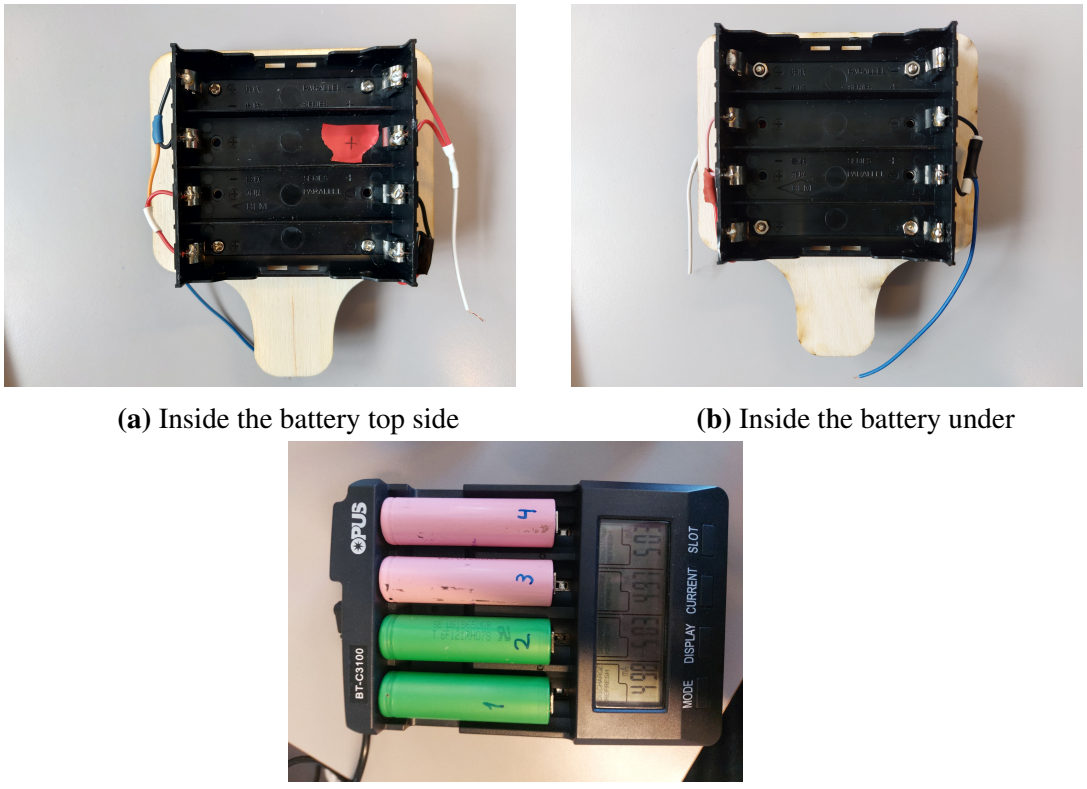
wiring and a smaller battery. At the same time, the battery box and the separator plate got a new design. The fifth iteration ended up being almost the same as iteration one. The main difference was that the first design had a 3S2P design and this battery has a 2P3S design.



(a) SOLIDWORKS model of final design **(b)** Final battery

Figure 6.25: Final battery design

The production of the fifth design iteration was just like the previous design started right after the design was finished. Because the BMSs and switches were removed from the previous design, the battery could be made smaller. Therefore it was decided to make a smaller battery box. The separator plate was redesigned to fit the new design of the battery box. The production steps were similar to the production of the previous battery iteration, except for the cable lugs, BMSs, and switches. The battery box was 3D printed while the other steps were completed.



(a) Inside the battery top side **(b)** Inside the battery under

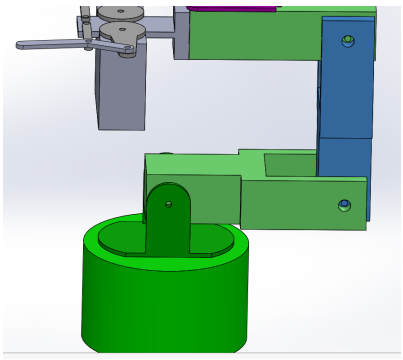
(c) How the battery is charged

Figure 6.26: Inside of final battery

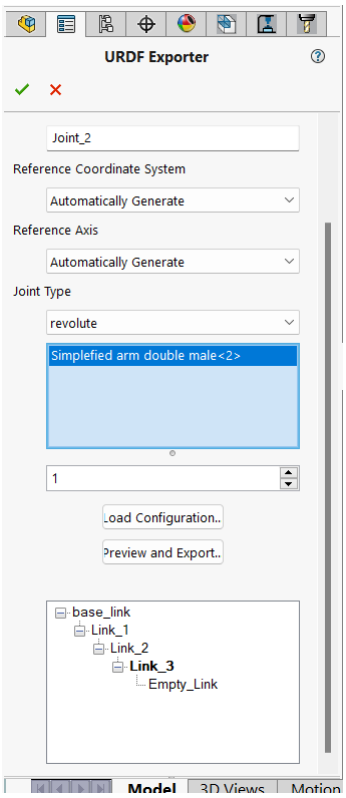
6.10 Exporting the model for the digital twin

AEH | *HB*

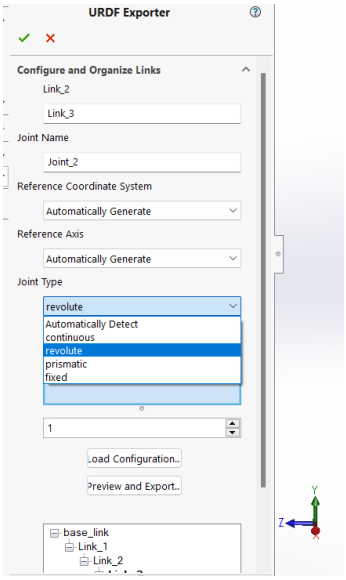
The 3d model we downloaded from the *Yahboom* website was exported as a single rigid part. We could therefore not change any of the components or move the arm. To develop and implement a digital twin we needed a Unified Robotics Description Format (URDF) of the robotic arm. We made a simple model of the robotic arm that has the correct dimensions between the joint and the correct weight but not all the details.
After making the simple robotic arm the file needed to be exported as a URDF file. To do this we used the *solidworks* plugin URDF exporter [138].



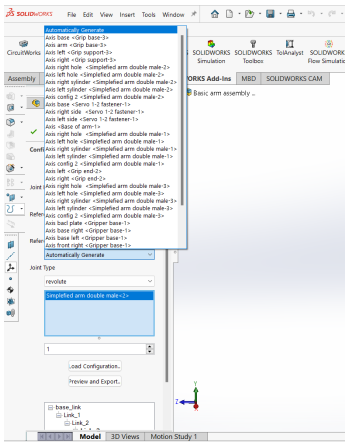
(a) Selected link on robot arm



(b) Selected link in menu



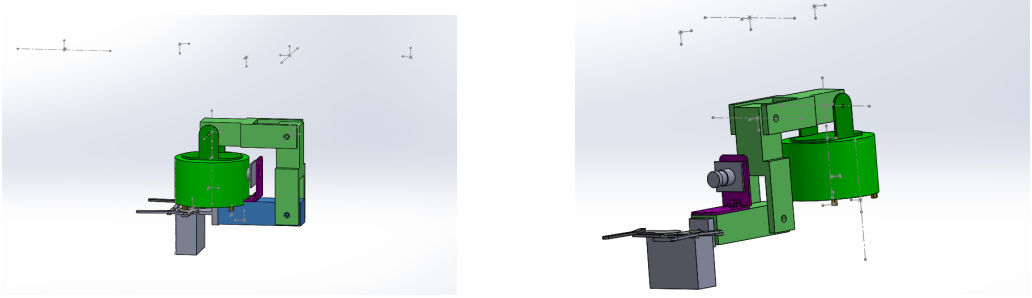
(c) Selecting joint type



(d) Selecting reference geometry for links

Figure 6.27: Configure links and joints

Using the plugin menu and starting from the base of the robot arm, each link and joint needed to be defined. Our robot arm uses revolute joints and we used the auto generate for the coordinate system and reference axis.
When the links and joints were defined we could go to the next step where the reference axis and coordinate systems are automatically generated. This step was a success but the final step was to export and this did not work as it should. The pictures below are the result of the export where the model would move away from its coordinate systems and reference axis as well as redefine parts of the system from revolute to fixed.
The file got exported anyway to see how it would look and if it would work when exported



(a) Basic robot arm URDF export **(b)** Basic robot arm URDF export

Figure 6.28: Robot arm moved during export

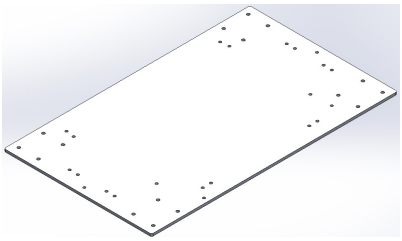
6.11 Production **AEH** | *HB*

This chapter will discuss the production method of different components in the system.

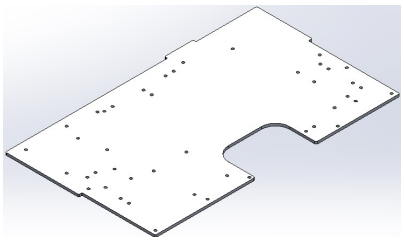
6.11.1 Robot car sheet floors **AEH** | *HB*

The bottom, middle and top floors are aluminium sheet metal 5052. We ordered the parts from "Vannskjæresenteret AS" which had aluminium 5052, 5754 and 6082 sheets in stock. The 6082 aluminium under 5 mm thickness is not normally stocked in Norway but they had some we could buy, but this would be quite expensive. Between aluminium 5052 and 5754 the 5052 has a higher hardness and stiffness. We therefore chose to order the parts in 3 mm thick aluminium 5052.

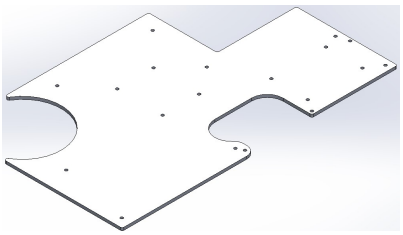
"Vannskjæresenteret AS" uses a water jet to cut the sheet metal parts. This method is both quick, accurate and easy to prepare. The water jet machine functions as a laser cutter and uses a 2D DXF file. The exported parts sent for production were a 1:1 scale, 2D drawing with no markings and annotations. This is because the machine cuts whatever lines or text on the drawing.



(a) Bottom floor sheet



(b) Middle floor sheet



(c) Top floor sheet

Figure 6.29: Aluminium sheet parts

6.11.2 Robot car cover **AEH** | *HB*

The cover for the robot car is made of Hexply 8552 [139], which is a pre-preg carbon fibre.
The plan was to produce a seven-layer thick 500 mm x 700 mm sheet. This would make it about 2,5 mm thick and is a good amount of layers to keep it as light as possible while minimizing the risk of warping. However when discussing with the lab supervisor concerning production; Computer numerical control (CNC) machining of parts; and post-processing, he informed us that he would be away in the coming weeks and would not be able to help with the CNC machining. Lucky for us he was going to help another bachelor group the next day with their machining and he had an extra sheet comprised of nine layers and a 2,9 mm thickness we could use. The sheet was leftover from a teaching lesson and he could machine our parts as well, as long as we had the machining assembly ready by the next day and we could help him with all the preparations, observing and cleaning afterwards.

Machining assembly refers to an assembly in *solidworks* which has to be prepared before utilizing Computer aided manufacturing (CAM). In this assembly, there needs to be a coordinate system where the origin is in the bottom left with the x-axis as the short end with 350 mm and the y-axis as the long end with 500 mm length the z-axis has to point straight up and all the parts that were to be machined needed to be spaced on the same plane with a clearing of preferably 12 mm. The work area on the CNC machine was not big enough for the whole 500 mm x 700 mm sheet so it needed to be cut in half. Using a angle grinder we cut the sheets in two so that they were roughly 500 mm x 350 mm. We also went over all the parts in the assembly and changed the thickness to 4 mm and hole diameters to 3,175 mm, this made it easier for the machine and software to machine the holes since we used a 3,175 drill bit and the extra thickness ensured that the machine cuts all the way through.

(a) CNC assembly solidworks **(b)** CNC assembly cut

Figure 6.30: CNC assembly in solidworks and after cut

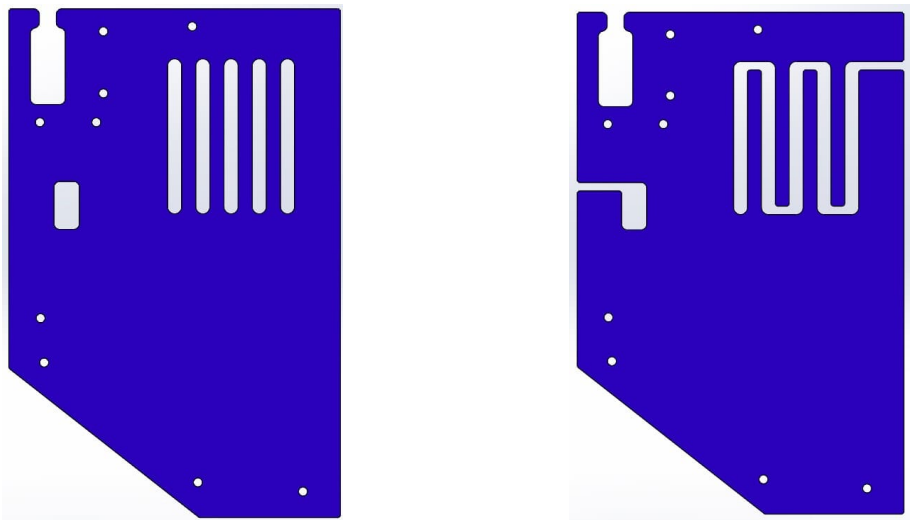
In the first session, we managed to cut out most of our parts but not all of them. So it was decided that if we prepared everything he could come back for a short period a week later to fix the CAM pathways. When this day came we, unfortunately, found out that the machine was not working correctly and had to be fixed. Troubleshooting could take a day or two, which had to wait, so we needed to find another solution.

We spoke to the other team that needed the rest of their parts and they could check if Kongsberg Defence & aerospace could cut the parts for all of us using a water jet. They learned that the qualified personnel were away and would not be back for a couple of weeks.

Since our team already were in dialogue with "Vannskjæresenteret AS" about our aluminium sheet production we asked if they could cut our carbon fibre parts if we provided the sheets.

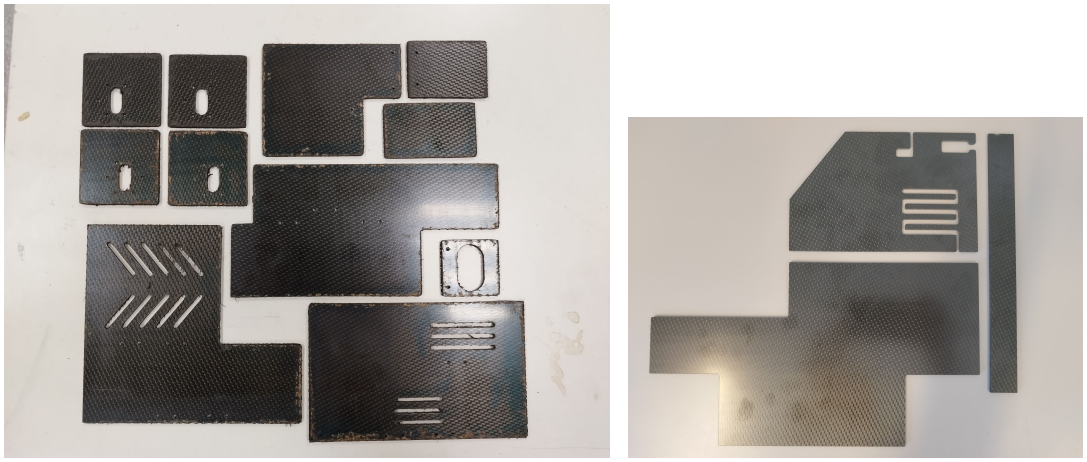
They agreed to this, however, they informed us that holes in the middle of the parts were problematic because the chance of delamination was so high. We agreed that they would cut the outline of the parts and we would drill the holes ourselves. This also meant that we had to make a pathway for the water jet if we wanted something cut in the middle of

the part.



(a) Roof before adjustment **(b)** Roof after adjustment

Figure 6.31: Before and after adjusting the roof for water jet production



(a) The CNC machined parts **(b)** The water jet parts

Figure 6.32: Cover parts after production

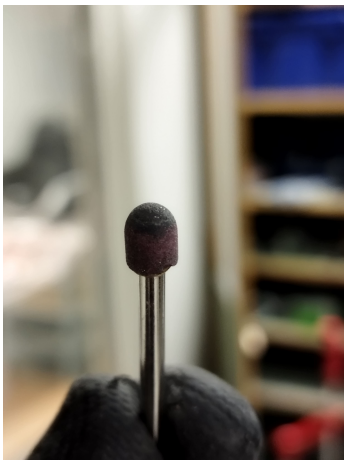
6.11.3 Post processing **AEH** | *HB*

The feed rate during machining was set to low and as a result, we got burn marks and heat damage to the carbon fibre parts.

Figure 6.33: Burn marks on edge of front cover

The post-processing of the carbon fibre parts consisted of sanding away the burn marks using a kitchen swamp. The swamp was abrasive enough to remove the burnt epoxy without damaging the part itself. To remove the sharp edges we sanded them using 400 and 2000 grit paper. For the holes, we used a grinding head to smooth the edges.

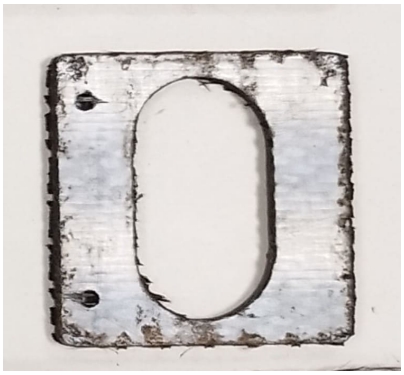
(a) Swamp



(b) Grinding head

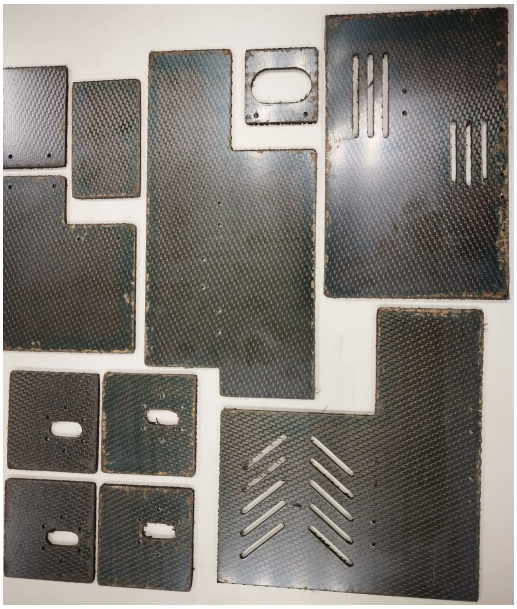
Figure 6.34: Post processing tools

After cleaning up the parts we could see a large improvement. We can still see the blue-green colour in the matrix of the parts which is a result of the heat damage. But the burns on the surface were removed and the edges were no longer sharp.

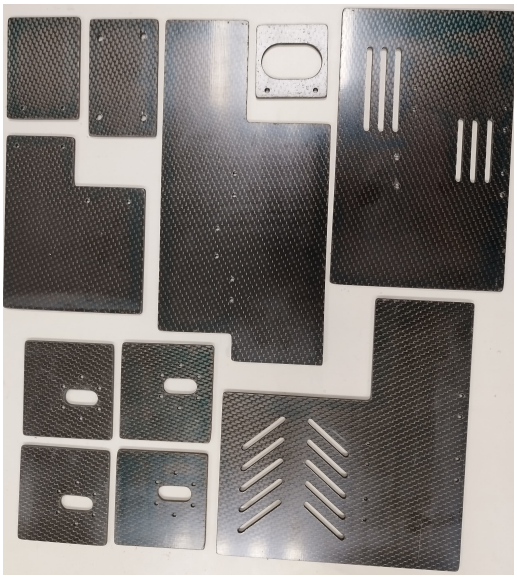


(a) Open wall before

(b) Open wall after



(c) Overview before



(d) Overview after

Figure 6.35: Before and after clean up

The parts that were cut using the water jet had smooth edges and did not need sanding but the holes had to be drilled. We secured the parts to a wooden slab so that they would not move. During the drilling of the 17 holes, we broke three drill bits because the high hardness of the material quickly wore down the bits. First, we drilled the holes using a 2 mm bit and then with a 4 mm bit, this was because the smaller bits grabbed the surface better. After drilling we smoothed the hole edges using the grinding head.

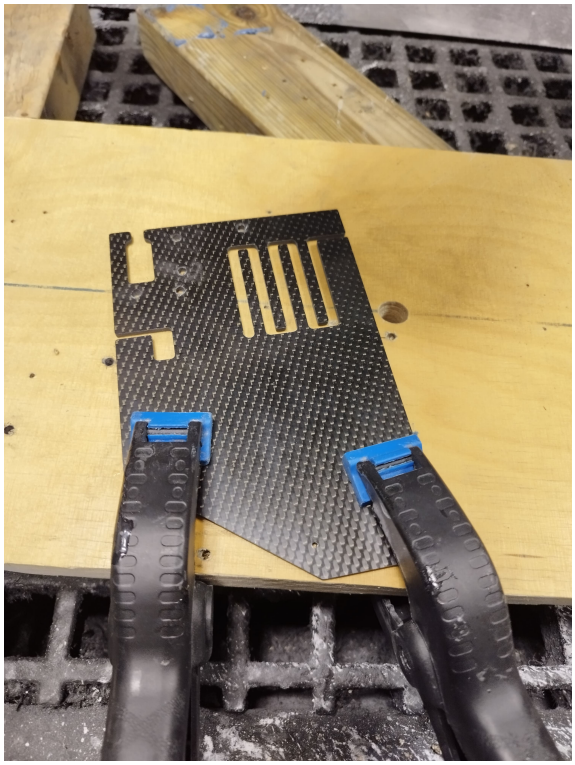


Figure 6.36: Drilling the holes for the roof

6.11.4 3D printed components

Some of the components used in our system were 3D printed using an Ender 3-V2 printer. These are the parts marked in section T.3 with the material as Clas Ohlson PLA.

7 Testing & Results

In this section, we have discussed testing of the project implementation and development. This includes using unit testing, deployment of object detection models, battery test, integration and user survey testing. Consequently, we have written about the results we received.

7.1 Integration testing **SO** | *OM*

Integration testing was conducted throughout the VR application and extended to the robot system. This testing ensured that functionalities involving components from the VR application, robot system, AI, and the physical robot were working together seamlessly. Detailed documentation of these tests can be found in our test template in Appendix E.

7.2 Object detection model performance **AD** | *OM*

This section will evaluate the performance of the object detection models developed in the previous sections. The group will start by discussing the evaluation metrics used to assess the models' performance, and then present the results of our experiments.

7.2.1 People detection **AD** | *OM*

Average Precision (AP) and Average Recall (AR) are two metrics that are used to analyse an object detection model's performance. Average Precision indicates the accuracy of the positive predictions. High precision means that the model returned more relevant than irrelevant results [140]. Equation 7.1 shows the precision calculation for one class A, if there is more than one class, then the statistical mean is taken.

$$Precision_A = \frac{True\ Positive_A}{True\ Positive_A + False\ Positive_A}[140] \tag{7.1}$$

Average Recall indicates the ability of the model to find all the positive/relevant cases. High recall means that the model returned most of the relevant results [140].

$$Recall_A = \frac{True\ Positive_A}{True\ Positive_A + False\ Negative_A} \tag{7.2}$$

Figure 7.1 represents the AP and AR of the people detection model across different **Intersection over Union (IoU)** thresholds and object size categories (small, medium, large). The IoU threshold determines how much a predicted bounding box must overlap with the ground truth bounding box to be considered a true positive. So, the 0.50:0.95 all shows that AP and AR are calculated across all sizes and IoU thresholds ranging from 0.5 to 0.95 [141].
The AP for all categories is high, ranging from 0.7 and higher. This means the model performs well across the data across all sizes and performs exceptionally well for small and medium objects, which means the dataset was well distributed.
The AR is also high, ranging from 0.7 and higher for all thresholds. It is also very high for small and medium objects, which again proves that the dataset was very well distributed.

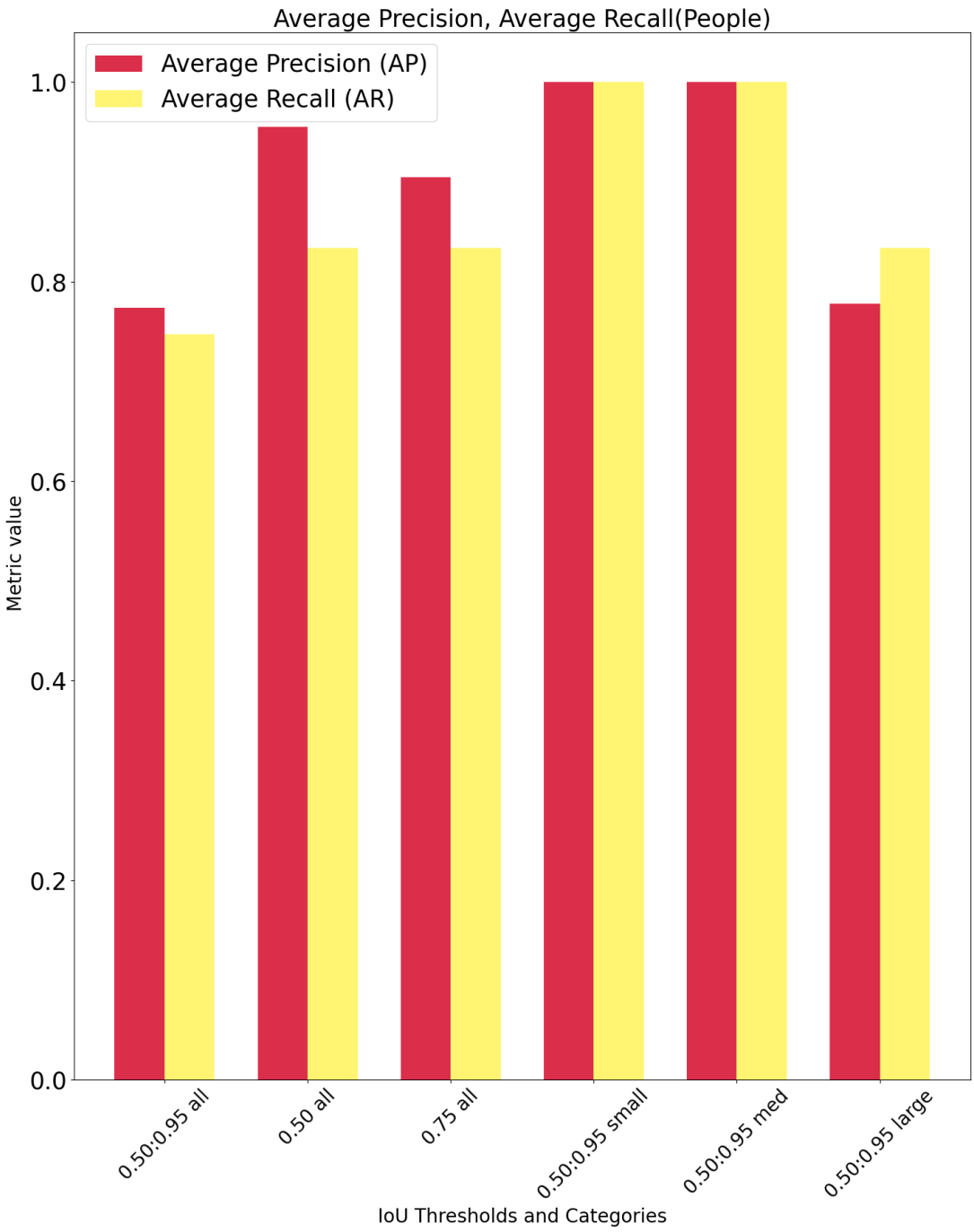
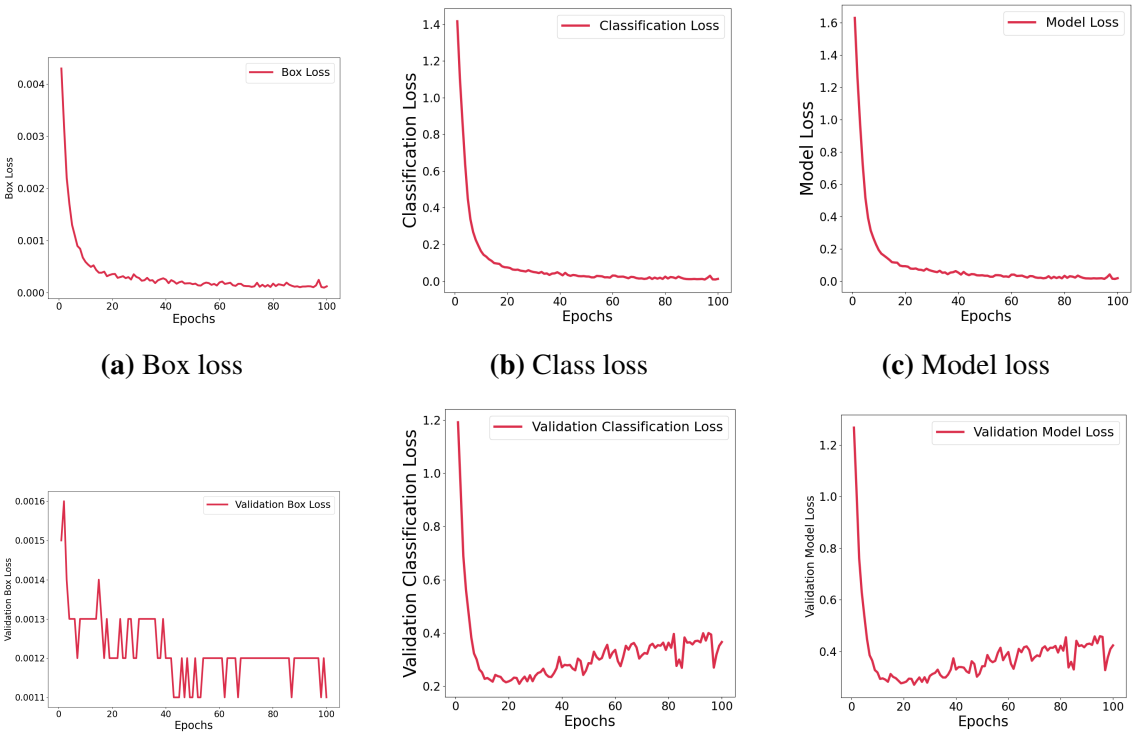


Figure 7.1: Precision and Recall of people detection model

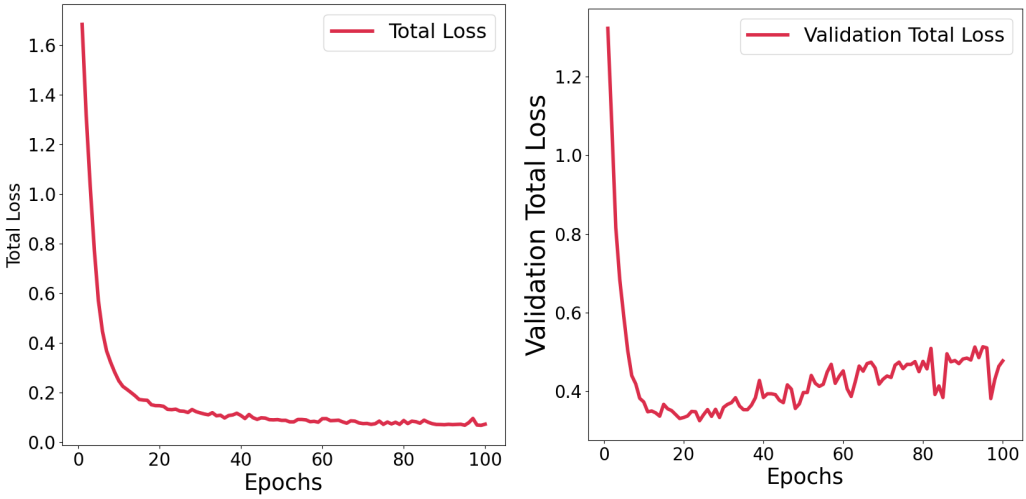


(a) Box loss **(b)** Class loss **(c)** Model loss

(d) Validation box loss **(e)** Validation class loss **(f)** Validation model loss

Figure 7.2: Loss functions

The above graphs represent the different loss functions of the model.
Box loss: Box loss is better known as bounding box loss function (L1 loss mentioned in 4.5.10). It measures how accurately the model localizes the object [95]. The curve starts with a low box loss value but still decreases quickly for 20 epochs. It then slowly decreases, coming closer to 0 with some minor fluctuations.
Class loss: This is the classification loss, which measures how accurately the model classifies the object (focal loss function explained in 4.5.10). The class loss in 7.2 starts with a high value, which suggests that the initial predictions were inaccurate, but it also decreases quickly and approaches 0. This suggests that the model is approaching its best possible performance.
Model loss: Model loss is the error calculated for a single training example [142]. This also moves the same way as the class loss, which suggests that the model is becoming increasingly accurate in predicting the locations and sizes of bounding boxes.
Validation box, class and model loss: These terms mean the same as above but only for the validation dataset, which is unseen data for the model. The validation box loss initially decreases rapidly, then fluctuates with an overall downward trend, suggesting that the model is learning and improving its predictions over time, but not as smoothly as in the initial stages.
The validation class loss and model losses develop similarly because the loss value drops quickly for 20 epochs, but after the initial drop, the losses fluctuate while showing a gradual decrease over time. These fluctuations could be due to the model finding it challenging to learn specific patterns in the validation set. The plateauing suggests the model may be reaching its limit in improving on this specific dataset.



(a) Total loss **(b)** Validation total loss

Figure 7.3: Total losses

Figure 7.3 represents the total loss of the training data and validation data. It is a weighted sum of all the individual losses (from 7.2) calculated during one epoch. The total loss decreases rapidly then after epoch 20 it continues to decrease but very slowly and plateauing, reaching its peak performance. The validation total loss also decreases like the total loss but has an upward trend which may be due to overfitting.

7.2.2 Bolt detection

AD | *OM*

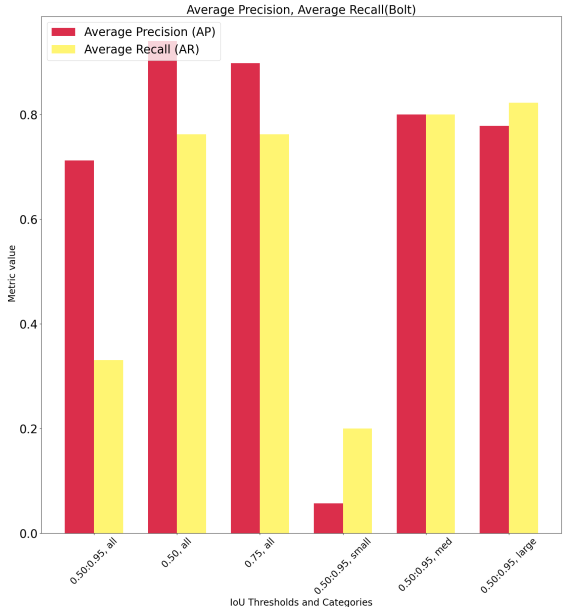
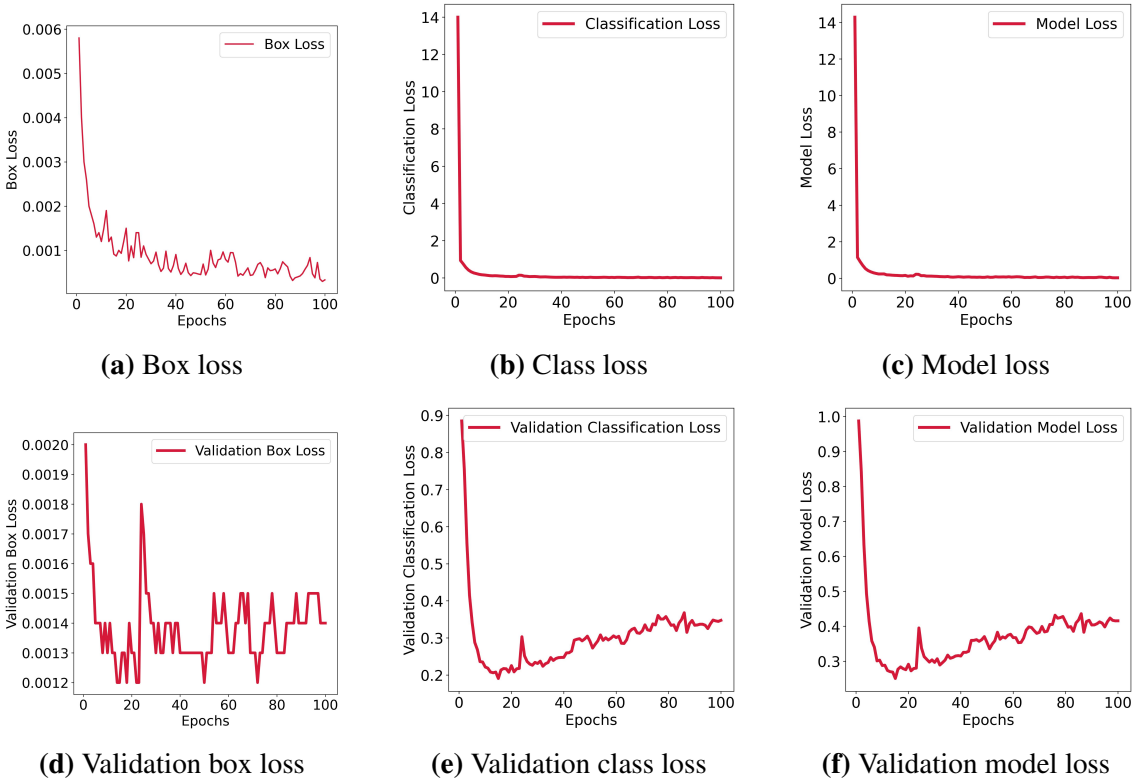


Figure 7.4: AP and AR (Bolt)

Figure 7.4 represents the AP and AR for the object detection model for bolt detection. The AP for the 'all objects' category is high, ranging from 0.7 to 0.9. This means the model performs well across the data without considering the object size. It also detects objects in the medium and large size category. However, the AP significantly drops for small objects, which means it has difficulty recognizing smaller objects.

The AR is weak for IoU thresholds 0.5-0.95 and small objects, otherwise it is over 0.7, which is high. This suggests that the dataset was not varied enough.

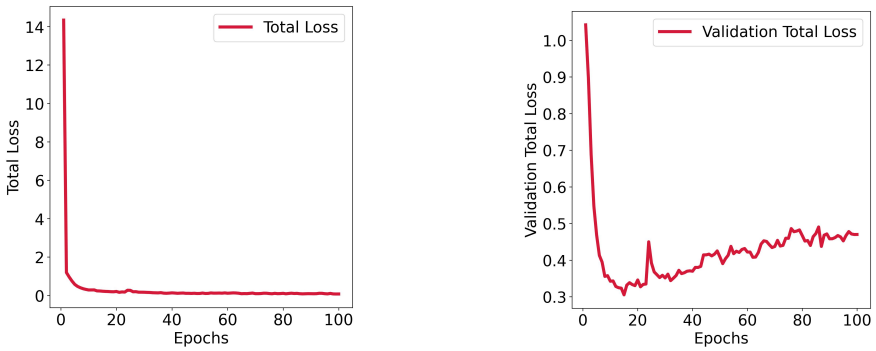


(a) Box loss **(b)** Class loss **(c)** Model loss

(d) Validation box loss **(e)** Validation class loss **(f)** Validation model loss

Figure 7.5: Loss functions

Figure 7.5 represents the many loss values of the model.
Box loss: We can observe that the loss value is already minute when it starts learning, and it gradually decreases with minor fluctuations.
Class loss and model loss: The class loss starts off being a very high value but drops significantly. It holds itself stable after that. The model loss has the same change as the class loss.
Validation box loss: This graph shows a decreasing trend initially, indicating that the model is better at predicting bounding boxes. However, after around 40 epochs, the loss fluctuates and plateaus, suggesting the model's ability to improve localization has stagnated. This can be because of the limited amount of data.
Validation class and model Loss: Initially, the loss functions decrease rapidly, showing that the model is learning to classify objects better. However, after about 20 epochs, the losses start increasing and fluctuating. This could indicate overfitting. Again, this could be because of the small validation data.



(a) Total loss **(b)** Validation total loss

Figure 7.6: Total losses

The graph 7.6a representing total loss shows that the model learns rapidly initially, but plateaus around epoch 20, which suggests that the model approached its optimal performance on training data. The graph 7.6b shows that the total loss in validation data also decreases at the start, but increases after around epoch 20-30. This suggests overfitting, thereby struggling to generalize to new examples.

7.2.3 Analyzing the results

AD | *OM*

Analyzing the model performance, it is reasonable to say that overall, the people detection performed very well across various IoU thresholds and excelled with small and medium-sized objects. It also had high APand AR values. The loss also consistently decreased and stabilized. However, there were some fluctuations in the box loss values. The bolt detection model had high APvalues except for small objects. The ARwas also low for IoU thresholds 0.5-0.95. The validation loss showed fluctuations and an increase after a point.
Some ways of increasing the people detection model performance could have been to have more varied and diverse data using more data augmentation which could have decreased the fluctuations [114]. The hyperparameters like batch size, learning rate and number of epochs could also be changed to increase performance [28].
The bolt detection model performance could have increased if the dataset had been bigger, and more varied. To avoid overfitting, more data augmentation could have been applied. Another technique called early stopping could also have been used, where we stop the training of the model once the validation loss begins decreasing and stabilising [114].

7.3 Running inference

AD | *OM*

Inference is the process that follows the training of a ML model. Here, the model is in action [143] and we can test the model and observe how it performs. Our project was required to detect objects from a live stream sent by the camera. A live stream is essentially a series of frames/images moving quickly. Therefore, inference has to run on every frame.

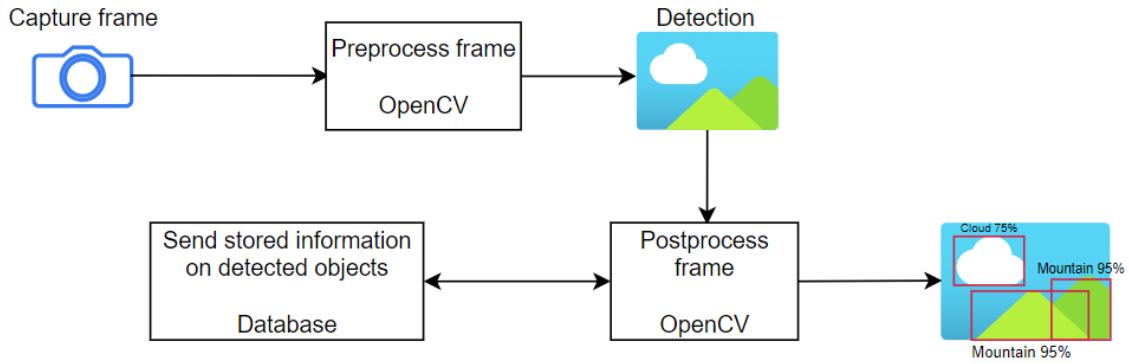
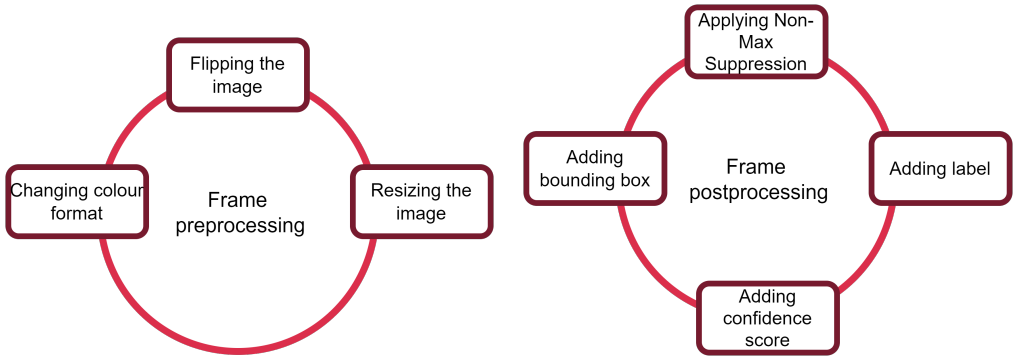


Figure 7.7: Simple inference process

The frames are captured by the OpenCV library. Each frame needs to be pre-processed before detection and post-processed after to achieve the best results. Figure 7.8 shows the steps of pre and post-processing.



(a) Preprocessing **(b)** Postprocessing

Figure 7.8: Pre- and post-processing steps

7.3.1 Pre-processing

AD | *OM*

Pre-processing is all the steps taken to ensure that the raw frame data is converted into a suitable format that the detection model understands. Here is a list of pre-processing steps:

- **Colour format** : OpenCV captures a frame with a Blue-Green-Red (BGR) format. MediaPipe models use the RGB format; therefore, the frame needs to be converted. Luckily, OpenCV itself provides a method to do this.
- **Resizing**: A model is trained on images with a specific input size. Therefore it is a requirement that the input frame it gets while performing the detection has to be resized. MediaPipe does the resizing for us.
- **Flip**: The flipped frame is a data augmentation technique used to make the data more varied as the object is now in a different place. OpenCV has a method to flip an image as well.

7.3.2 Detection and post-processing

AD | *OM*

The MediaPipe library offers a package called *vision* which creates a detector object using the options that the user creates. The user has the following choices:

- The TFLite model path
- The maximum amount of objects to detect in one frame
- The confidence threshold
- The input data mode: a) An image, b) A video, c) Live stream
- A method that saves the results

Post-processing: Post-processing is where the raw detection results are refined and interpreted. Here are the steps of post-processing:

- **Non-Maximum Supression (NMS)**: This process eliminates redundant bounding boxes that might overlap around the same object, keeping only the most confident prediction. MediaPipe does this for us, as it has a calculator made for NMS [144].
- **Labelling**: Extracting the class index from the detector object, finding the value from the metadata, and visualising it with OpenCV.
- **Bounding Box Visualization**: Extracting the bounding boxes from the detector object, and drawing them using OpenCV.
- **Confidence scores:** Extracting the confidence score from the detector object, converting the probability score to percentage and displaying it on the frame using OpenCV.

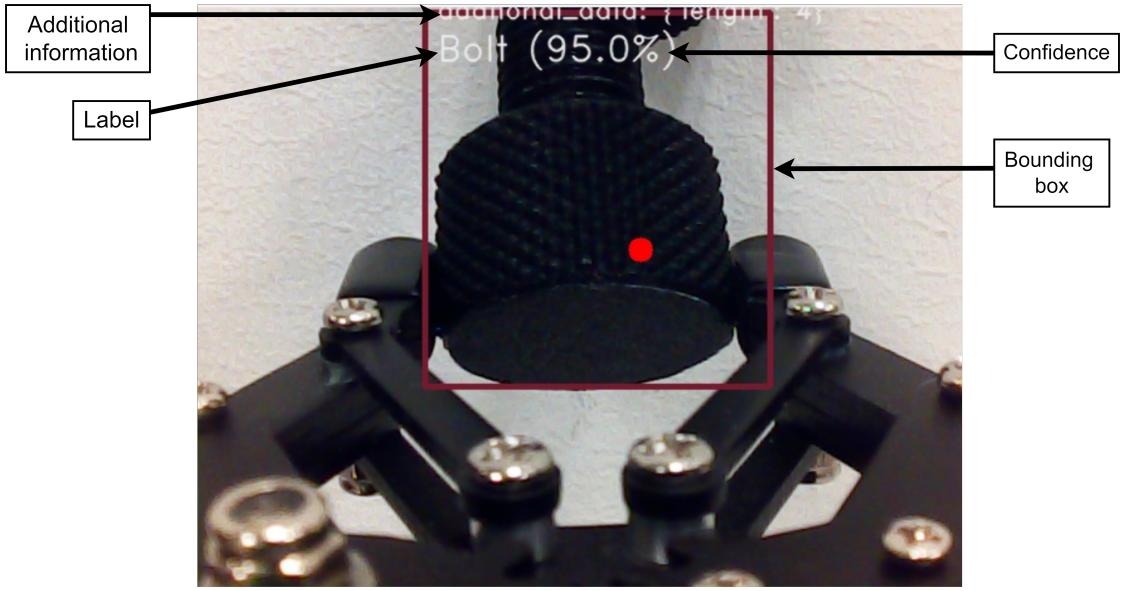


Figure 7.9: The annotated frame after post-processing

Figure 7.10: Person

Figure 7.11: Object detection recognition and displaying of information

Figure 7.9 shows a frame captured while testing the whole project. It can be observed that NMS has been applied as we see only one bounding box, we also see the label, confidence score and additional information from the database. We can also see the result in 7.10, even though the data goes out of the frame.

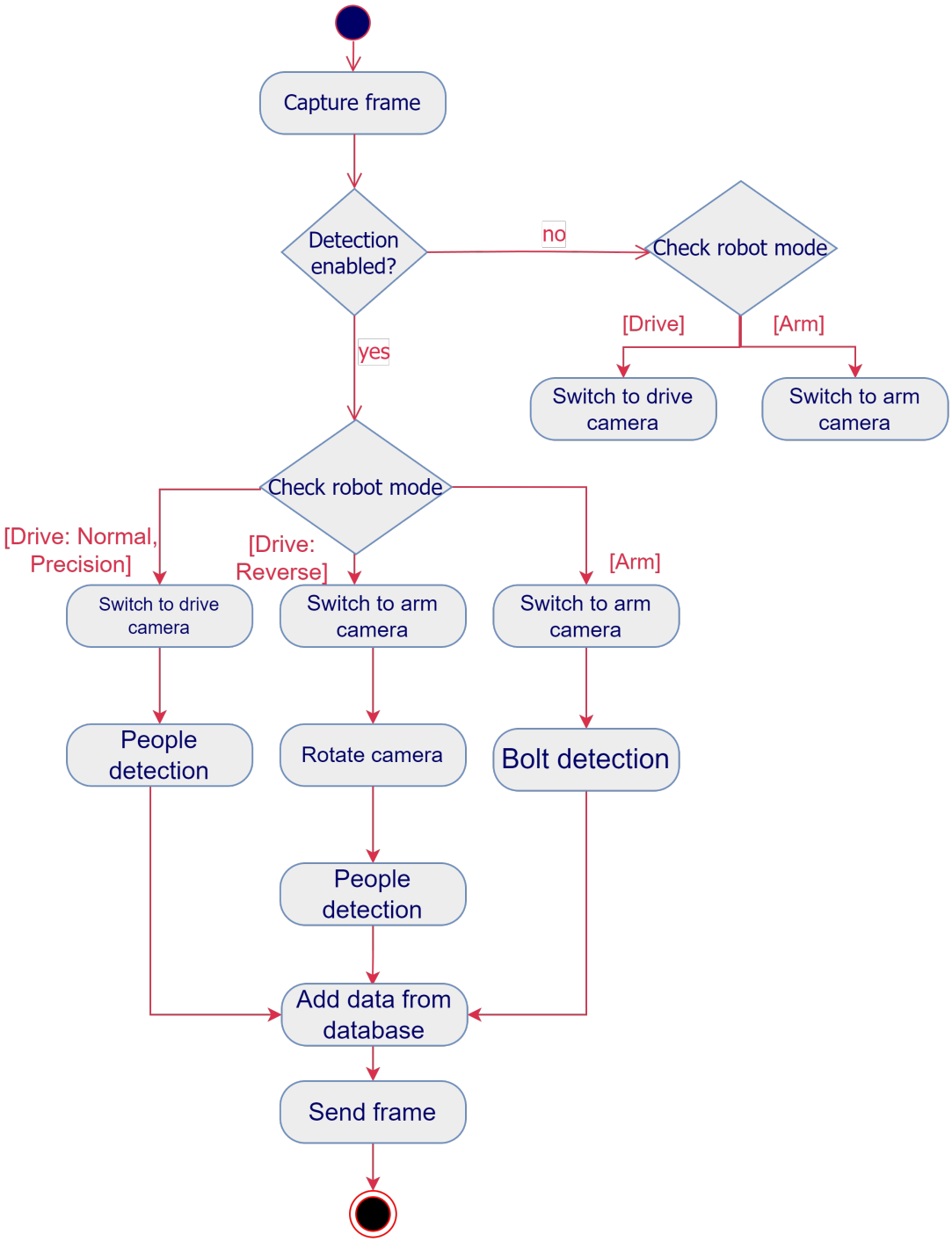


Figure 7.12: Full activity diagram of testing the model

The activity diagram in 7.12 shows the process of capturing and processing a frame in the totality of the system. After capturing a frame, the system checks if detection is enabled. If so, it determines the robot's mode (Drive or Arm) and switches to the corresponding camera. Based on the mode, it performs either people detection (in Drive mode) or bolt detection (in Arm mode). Finally, it adds relevant data from a database and sends the processed frame.

7.4 Production mode **OM** | *AD*

Due to the amount of testing, a "production mode" functionality was added. This mode determines whether the expansion board will be utilized and is simply represented as a boolean value specified in a JSON file.

```
{
    "production": false
}
```

The rationale behind using a JSON file instead of a variable within the Python code or elsewhere lies in the principle discussed in 4.3.1. By altering the contents of the JSON file, there is no need to rebuild the code. This approach allows for the execution of the same code while enabling the switching of production mode on or off, thereby saving time without requiring code rebuilding.

When production mode is disabled, the serial connection with the expansion board is not established. This allows for the testing of functionality without the need to have the expansion board connected, thereby mitigating the risk of damaging any physical components on the robot. This aspect is crucial for conducting unit tests on portions of the code without encountering exceptions.

7.5 Unit tests **OM** | *AD*

Unit tests have significantly sped up the system development process. It has facilitated the testing of functionalities and concepts without the need to set up a physical test environment each time. A physical test involves preparing the robot, connecting devices, establishing a connection with the VR application, and so forth. While each setup may only require a few minutes in total, the cumulative time spent adds up throughout development, resulting in slowdowns. In our system, this kind of testing requires two people at minimum. Unit testing has ensured that the code-base segments are correct without necessitating this setup. All of Kromium's unit tests can be found on GitHub [145].

7.5.1 Writing an unit test **OM** | *AD*

Every ROS package in the codebase includes a `test/` folder where corresponding unit tests for that package are written if found applicable. The default Python package *unittest* was used for writing and running these unit tests.

Every test file in this directory should start with "test", as should the classes and methods within these files. If they do not, the tests will not be executed. The test class inherits from the `TestCase` class, making commonly used methods like `assertEqual(a, b)` and `assertTrue(c)` available. These methods are used to verify that variables and values are as expected.

```
from unittest import TestCase

class TestSomething(TestCase):
    def test_equal(self):
        self.assertGreater(4, 3)
        self.assertEqual(4, 3)
```

```
    def my_test(self):
        self.assertAlmostEqual(2.9999, 3.0000, places=2)

    def test_true(self):
        self.assertFalse(True)
```

To run the tests, the command `python3 -m unittest` is used. The output of the tests is as follows:

```
FF
======================================================================
FAIL: test_equal (test_something.TestSomething.test_equal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\oscar\kromium\test\test_something.py", line 7, in tes
    self.assertEqual(4, 3)
AssertionError: 4 != 3

======================================================================
FAIL: test_true (test_something.TestSomething.test_true)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\oscar\kromium\test\test_something.py", line 13, in te
    self.assertFalse(True)
AssertionError: True is not false

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=2)
```

Here we can see that all tests failed, along with the reasons for their failure. If a unit test fails, it indicates that either the code being tested is incorrect or the expected value in the unit test is wrong. In this example, only static values were used, but typically one would import functionality from other files and compare it with the expected result. The static values could be replaced with function calls and class attributes. When changing the implementation of the code, the output might be intended to remain the same, or it may change, as the previous code written was wrong. Therefore, unit tests need to be updated accordingly. If the output was not supposed to change but did, the code within the function is likely incorrect.

Typically, a unit test function would contain multiple assertions. If the unit test output shows `OK`, the functionality likely being tested works correctly, provided that the tests are written to represent possible inputs and states accurately. Due to the file structure when working with ROS packages (see 4.3.1), there might be issues with paths, as discussed in O.14.

7.5.2 Ignoring messages in wrong mode **OM** | *AD*

When starting the robot, it initially enters idle mode, which is verified through testing.

```
from master.master_node import MasterNode
from unittest import TestCase

class TestModes(TestCase):
    def test_master_mode(self):
        master = MasterNode()

        self.assertEqual(master.mode, Mode.IDLE)

        master.handle_unsafe_vr_arm(None)

        master.mode = Mode.ARM
        master.handle_unsafe_vr_drive(None)

        self.assertRaises(
            AttributeError,
            master.handle_unsafe_vr_arm,
            None
        )
```

In this test, `None` is passed because the function requires a ROS message argument, which is not accessible.

Since the master node operates in unit test mode (`UNITTEST=True`, see O.14), the Python interpreter will raise an `AttributeError` when the master node is in the correct mode, which is the expected behaviour. When the master node is in the wrong mode, nothing happens when a message is received, which is also the expected behaviour.

If the interpreter were to raise an `AttributeError` when the master node is in the wrong mode, it would indicate there was an issue, as the message would be processed even though in the wrong mode. This could lead to dangerous situations, such as the operator being able to drive the robot while it is in emergency mode.

7.5.3 Battery percentage verification **OM** | *AD*

Before physically testing the battery percentage estimation, unit tests were conducted. Below is the test that was used:

```
from controller.robot import Robot
from unittest import TestCase

class TestRobot(TestCase):
    @classmethod
    def setUpClass(cls):
        cls.robot = Robot(production=False)

    def test_battery_percentage(self):
        self.assertEqual(self.robot.estimate_percentage(12.6), 100)
        self.assertEqual(self.robot.estimate_percentage(9.6), 0)
        self.assertEqual(self.robot.estimate_percentage(11.15), 51)
```

These tests were successful, as shown in the output below:

```
----------------------------------------------------------------------
Ran 14 tests in 0.001s

OK
```

The battery percentage is defined as 0% at 9.6V, which is when the robot stops and beeps to indicate low voltage [84]. However, if it was assumed that 0% is at 9.7V instead while keeping the implementation the same, the output would look like this:

```
======================================================================
FAIL: test_battery_percentage (test.test_robot.TestRobot)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/robot/src/controller/test/test_robot.py", line 15, in test_ba
    self.assertEqual(self.robot.estimate_percentage(9.7), 0)
AssertionError: 3 != 0

----------------------------------------------------------------------
Ran 14 tests in 0.003s

FAILED (failures=1)
```

Since the actual calculation returns 3% at 9.7V, this test fails because the implementation still considers 9.6V as 0%. In this case, the test is incorrect, not the code.

7.6 robot-test-client

OM | *AD*

A smaller program was developed to test communication with the robot, simulating a basic VR socket client. This tool was made to manually send and receive data, eliminating the need to launch the VR application and reducing reliance on the presence of the VR headset and other team members when testing. The test client possesses the capability to receive various data types from the robot, including depth data compressed using *Brotli*. Upon reception, the test client decompresses the data for processing.

The test client was frequently used during the testing of depth data and the enhancements of predefined arm movements. As testing could be conducted without the VR headset, this fine-tuning occurred in parallel with the implementation of voice commands, which naturally required the use of the headset. Consequently, this test client enabled parallel development of features, reducing the development time.

Furthermore, the test client was instrumental in validating the compression of depth data. *Plotly* was configured to generate plots each time the test client received depth information, allowing for the evaluation of live plotting before integration into the VR application. If the depth data could not be compressed enough before transmission, developing a plotting algorithm on the receiving end would have resulted in inefficient use of time. This was proved not to be the case, and the implementation on the VR application followed shortly thereafter.

7.7 Simulations **AEH** | *HB*

We have utilized SolidWorks simulation to run these simulations. To get as accurate a reading as possible we performed a tensile test[146] of the PLA we were going to use and made our custom material in SolidWorks based on these values.

7.7.1 Testing of 520 DC motor brackets **AEH** | *HB*

The brackets (T.45) used for the 520 DC motors should be able to withstand the weight of the robot car. The brackets are made of PLA and since the car weighs 5.4 kg, each bracket should hold at least 1.35 kg each. We have done a static simulation of the motor bracket and found that they are more than strong enough. The report from the simulation[147] states that the part has a factor of safety of 79 or higher. This does not account for faults in layer adhesion or other defects, but the brackets should still be more than enough.

7.7.2 Arm camera bracket **HB** | *AD*

The original arm camera bracket that came with the robot could not adjust the angle of the camera. Because the camera was mounted in a position that did not show the grippers on the arm, it would be very hard to use the arm and gripper. Therefore the arm camera bracket was replaced with a newly designed arm camera bracket. There were two iterations of the arm camera bracket. The first design could adjust the angle of the bend on the bracket. In the second design iteration, the bracket had a fixed angle that was at a specific angle.

First iteration, arm camera bracket
The first arm camera bracket consisted of four parts. The base plate that attached the bracket to the arm, the plate that holds the camera, a screw that holds the two plates together, and finally a nut that tightens the screw so the angle of the bracket will be consistent.

During use, the nut gradually came looser and the angle started to change during use. This was because of the vibrations created by the wheels, and the tension in the camera cable constantly changed due to the arm movements. It was decided to ditch the design when the bracket broke off. This was because the camera cable got pulled so hard that it broke the attachment mechanism between the two plates. The reason the cable got pulled so hard was because the robot changed mode, from arm mode to drive mode. When the arm moved to the drive mode position, the camera cable hooked itself to the robot, and therefore ripped off the arm camera bracket.

Second iteration, arm camera bracket
The second arm camera bracket design consisted of one part which has a fixed angle. In addition to the bracket, a cable holder was made to prevent the arm camera cable from attaching itself to the robot and possibly breaking anything. To find the optimal angle for the camera bracket, a picture was taken to measure the angle. See O.10 for more.

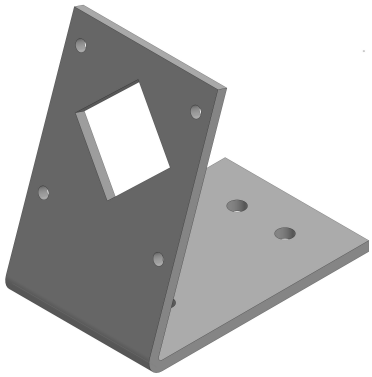


Figure 7.13: The fixed angle bracket. The hole is square to remove the need for support when 3D printing the part.

The cable attachment is attached to one of the arm brackets a couple of steps behind the camera. To make it possible to attach and detach the camera cable without damaging the cable, the attachment point has rounded-off edges on the inside and the outside of the cable holder.

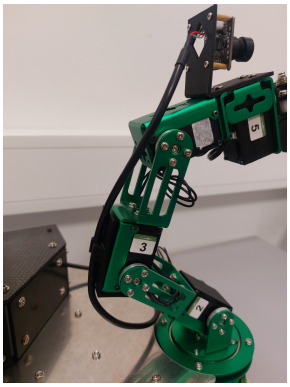


Figure 7.14: This way the cable for the arm camera does not hook itself to the robot.

A static simulation of the arm camera bracket was made, this was to simulate the camera cable attaching itself to the robot and therefore dragging in the camera bracket. The force here was sat to 2.5N dragging backwards. The calculations gave a safety factor of approximately 1.5. This sounds quite low, but it is better if the bracket breaks instead of the cable ripping apart. This is because it is much easier to replace the 3D-printed bracket than the camera cable. For the simulation results, see [148].

7.7.3 Wall attachment bracket assembly **HB** | *AD*

The "Wall attachment bracket assembly" is the assembly of the wall attachment brackets. Each end has a similar shape, so they fit each other. Parts of the walls are also angled, so the pieces fit easier with each other. Because these pieces are load-bearing, a static simulation has been made to make sure that the brackets hold the weight of the walls. The load was put to 5N, this is more than what each of the walls weighs. Still, some extra forces are added when attaching the walls to the robot. Still, the factor of safety was calculated to be just above 7.5, which is more than enough. Still, the brackets did not have a redesign because of their small size and they were 3D printed. Because the parts are 3D printed the quality varies from part to part. The factors playing a role here are; the adhesion between the printing layers vary from print to print and printer to printer, and the dimensional tolerances. The dimensional tolerances play a role because the part is

first produced in one program, then transferred to another program, in the end, the printer also has its limits to the resolution. Because of this, the part loses accuracy and because it is already so small, it would not be smart to reduce its size. For the simulation results, see [149].

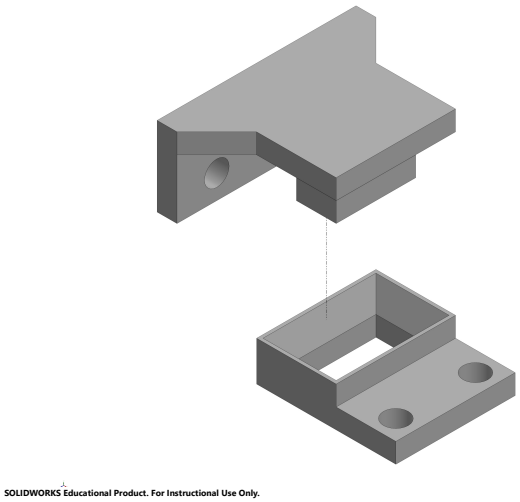


Figure 7.15: This is how the pieces fit together.

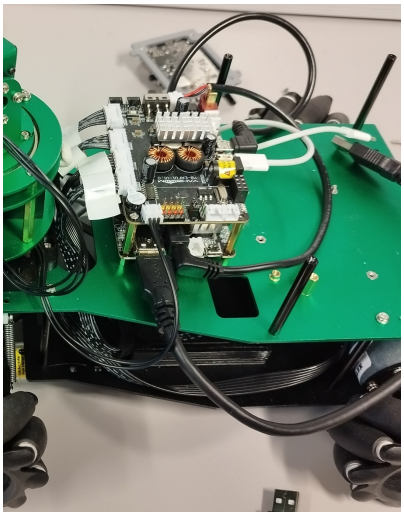
7.8 Visual physical testing **AEH** | *OM*

Utilizing Rapid Prototyping (RP) we have made a lot of iterations of the physical robot as discussed in chapter 6.5. This was a quick and easy way to find mistakes, modify and correct them. Next, we will go through the different iterations of the robot and what needs to be fixed for each one. The areas marked by blue on the pictures are areas to remove.

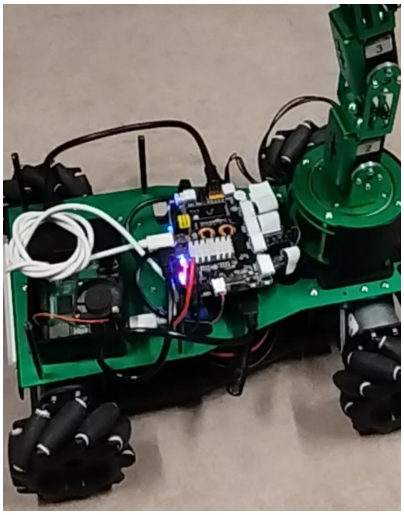
7.8.1 Robot iteration one testing **AEH** | *HB*

In this iteration, we learned that the USB cable heads take up a lot of space and this needs to be accounted for. We had to remove parts of the cover because the cables did not fit. We also learned that the robot arm needs to be placed in relation to the camera such that the arm does not block the view.

(a) Cover with section removed



(b) Electronics view right



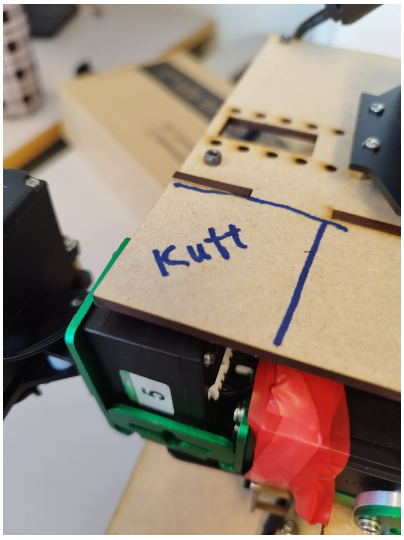
(c) Electronics view right

Figure 7.16: Iteration one testing

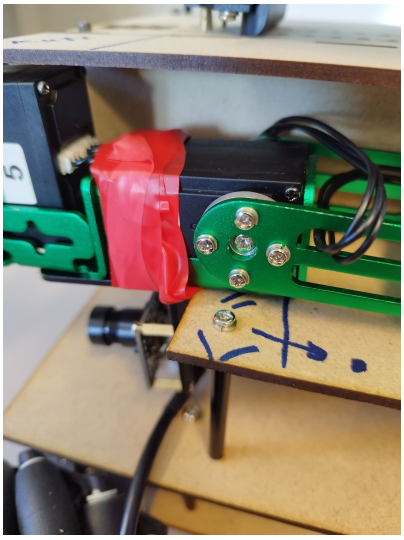
7.8.2 Robot iteration two testing

AEH | *HB*

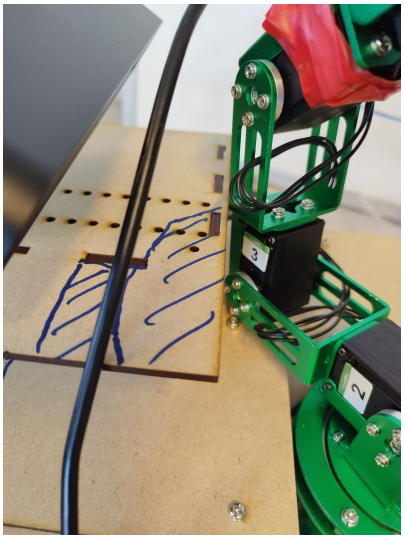
The overhang 7.17a from the roof over the robot arm collides with the end servo on the arm. The section 7.17b removed to make space for the robot arm camera needs to be larger so that the camera does not collide and the cable does not get caught on it. When the robot arm is being controlled the roof 7.17c is in the way of the arm when bent backwards. In this iteration the pathways 7.17d meant for the cables were closed off, thus the components needed to be unplugged before removal, in the next iteration there needs to be room for removing the cable without disconnecting.



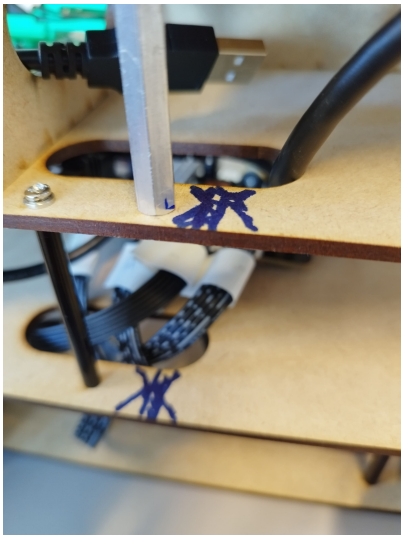
(a) Back of the roof overhang



(b) Top floor space for camera



(c) Front of roof



(d) Cable pathways

Figure 7.17: Iteration two sections to remove

The holes on the roof shown in 7.18 were a way for us to test different placements for the astra pro camera. The result of testing these was that the camera should be placed as far back as possible.

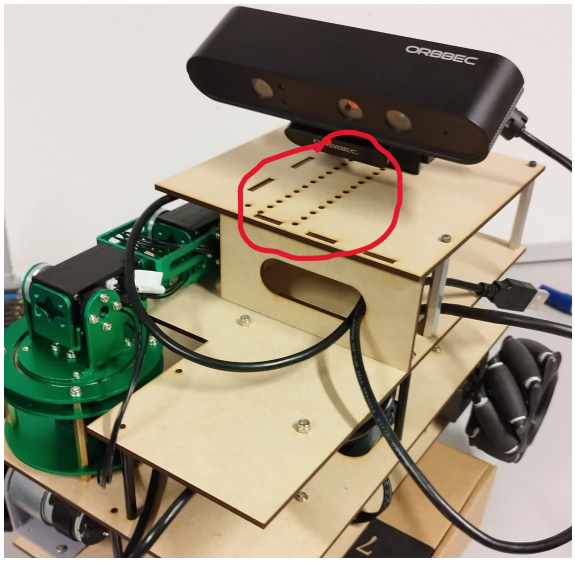
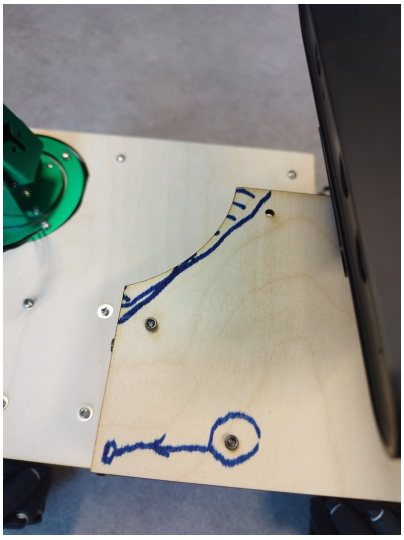


Figure 7.18: Holes to find the best camera position.

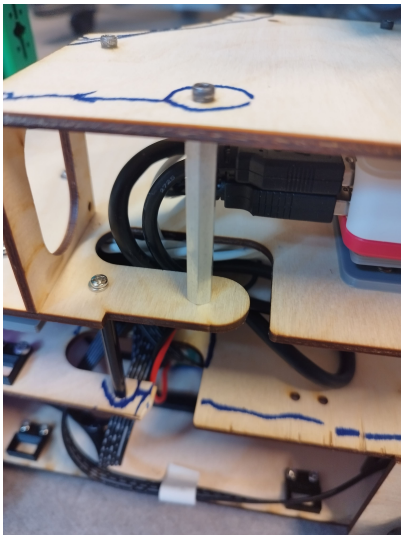
7.8.3 Robot iteration three testing

AEH | *HB*

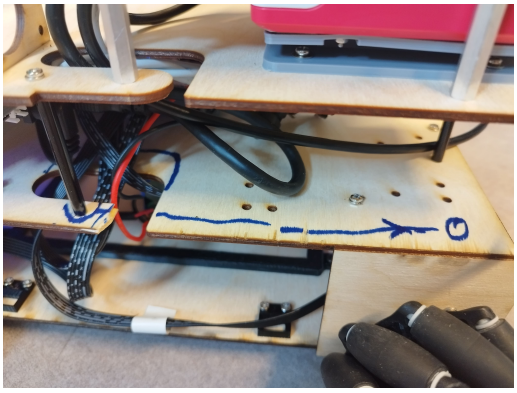
The redesigned front section of the roof 7.19a had to be redesigned again and the corners on the circular cut had to be removed since these still obstructed the path of the arm. The standoffs in the middle on the middle 7.19c and top floor 7.19b need to be moved since they currently obstruct the path for components and cables when these are changed. The new entryway for the cables is too thin and the cables do not pass through smoothly and therefore need to be made larger.



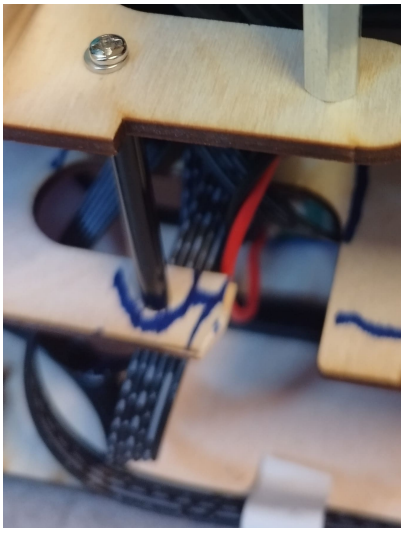
(a) Front of the roof



(b) Top floor standoff



(c) Middle floor cable pathway standoff



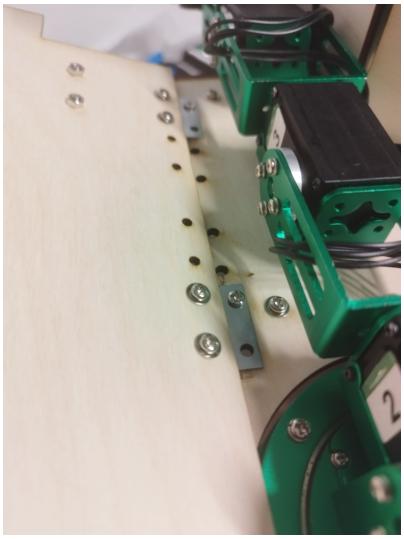
(d) Middle floor cable pathway entry

Figure 7.19: Iteration three sections to fix

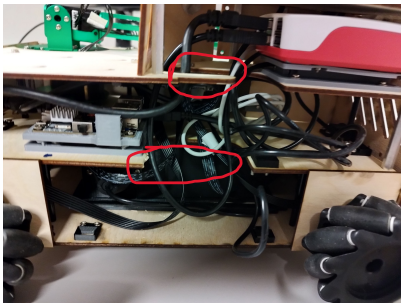
7.8.4 Robot iteration four testing

AEH | *HB*

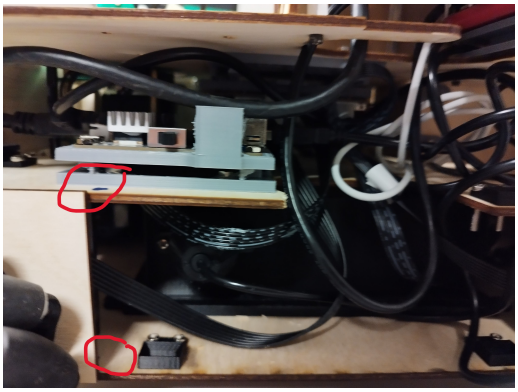
In this iteration, we wanted to test different ways to fasten the cover walls. The female to male brackets T.43 work but we wanted to test hinges and magnets as possible solutions. The hinges 7.20a worked poorly in that they restricted access to the battery drawer. The magnets however worked great. The space for the cables and the pathway 7.20b is still too small and needs to be removed and made bigger. Since we moved the motor expansion board up and the USB hub down the standoff under the USB hub which connects the bottom and middle floors needs to be moved closer to the edge 7.20c.



(a) Hinges



(b) Cable pathway and entry



(c) Standoff under USB hub

Figure 7.20: Iteration four sections to fix

7.8.5 Robot iteration five test

AEH | *OM*

In the fifth and final iteration, we wanted to measure the robot's weight. To get the weight we used a case for the robot and a hook weight. We emptied the case, put it on the scale to zero it, and put the robot in the case and got the measurement. The final weight for the complete system is 5.4 kg.

Figure 7.21: Weight of the robot

7.9 Battery testing

7.9.1 Test of battery cells

HB | *AD*

Introduction
To make sure that the battery cells are still alive and have enough capacity to be used, they are tested. To make sure that the capacity of the battery cells is good enough to be used, they are tested using the OPTUS BT-C3100 V2.2, which is a battery cell tester with other functions as well. If the battery cell has too little capacity left or they are dead, there would be no use for them.

Equipment used for testing of battery cells The equipment that is being used for the testing is a multimeter and a battery tester. The multimeter is a FLUKE 8010A Digital Multimeter and the battery tester is an OPUS BT-C3100 V2.2.

Fluke 8010A Digital Multimeter is a portable bench-type digital multimeter that USN got to their possession in 1992. Even though this is an old multimeter, it is reliable. It can measure voltage, current and resistance in both AC and DC. For the creation of the battery, it is only needed to use the DC voltage function. The values measured are displayed on a small $3\frac{1}{2}$ -digit LCD.

The *OPUS BT-C3100 V2.2* is capable of charging, discharging, restoring and testing different types of battery cells. The capacity is four battery cells at a time, and it can have different modes for each one at the same time. The different sections in the battery cell tester consist of a positive and a negative pole. The positive pole is at the far end of the tester, while the negative side is a spring-loaded metal plate that makes sure that the battery cell fits nicely and firmly in the tester.

The test sequence consists of first charging the battery cells up to 4.20 V. Then a discharging sequence is initiated, and the battery cells are discharged to 2.80 V. During this discharging sequence the battery capacity is measured, and displayed in mAh when the sequence is finished. The final step is charging the battery cells up to 4.20 V, so the battery cells do not die. All of these steps happen automatically when the device is set to test mode.

The formula for measuring the capacity of a cell can be written like this:

$$mAh = mA \cdot h \tag{7.3}$$

Where mA is the discharging current used during the testing, and h is the time, in hours, it takes to discharge the battery.

Testing of battery cells
The testing process is divided into two tests. The first test is to check the voltage of the batteries using a multimeter. The second test is to check the capacity that's left of the battery cells, measured in mAh. This test is carried out using a battery cell tester. To keep track of each battery cell, they are numbered from "1" and up and then logged into a table. This is to prevent mixing them if there are some of them are dead and to separate the cells with the least capacity and most capacity.

The reason the voltage of the battery cells was checked is because if the cells are below 2.80 volts, the cells "die". A battery cell that has died is a battery cell that has lost all of its voltage and it is not possible to recharge the cell. Luckily only one of the given battery cells was dead. This gives 20 battery cells that could be tested for their capacity. The battery capacity test was a tedious process. The test consists of placing the cells in the battery cell tester and then switching the battery cell tester into testing mode. The way this testing process works is that the cell is charged all the way up to 4.20 volts. Then a discharging-sequence is initiated and the cells are discharged all the way down to 2.80 volts. Then the test is finished. To prevent the cells from dying, the battery cell tester automatically starts to charge the cells again after the test is finished. Because there was 20 battery cells that had to be tested, and the capacity test took at least six hours for the cells with the least capacity. The total time for testing the cells was over two and a half days to complete because the cells had to be watched when charged and tested.
As the cells were done with the capacity test, the result was logged into a table and ranged from 1 to 20, where 1 was the cell with the most capacity and 20 was the cell with the least amount of capacity.

Results of cell testing
The results from the battery check were quite promising. Only 1 out of 21 battery cells were dead. This meant that 20 battery cells were possible candidates for the battery pack. The results from the battery testing can be viewed in table 7.1.

Battery cell test				
Cell №:	Voltage:	Tolerance:	Capacity:	Ranking:
1	3,84V	+/- 0,0038V	1805mAh	20
2	3,84V	+/- 0,0038V	1853mAh	19
3	4,09V	+/- 0,0041V	1984mAh	14
4	4,1V	+/- 0,0041V	2031mAh	7
5	4,1V	+/- 0,0041V	1984mAh	14
6	4,1V	+/- 0,0041V	2031mAh	7
7	4,1V	+/- 0,0041V	1996mAh	13
8	4,13V	+/- 0,0041V	2060mAh	4
9	4,09V	+/- 0,0041V	2023mAh	10
10	4,09V	+/- 0,0041V	2047mAh	5
11	0,01V	+/- 0,V	*	*
12	4,07V	+/- 0,0041V	2091mAh	1
13	3,88V	+/- 0,0039V	2038mAh	6
14	4,07V	+/- 0,0041V	1874mAh	18
15	3,96V	+/- 0,004V	2086mAh	2
16	4,07V	+/- 0,0041V	2082mAh	3
17	4,07V	+/- 0,0041V	2028mAh	9
18	3,96V	+/- 0,004V	2005mAh	12
19	3,81V	+/- 0,0038V	1979mAh	16
20	3,82V	+/- 0,0038V	1940mAh	17
21	3,88V	+/- 0,0039V	2019mAh	11

*The tolerance for DC-Voltage measurements is 0.1% at all ranges in FLUKE 8010A Digital Multimeter
**The voltage listed is the voltage measured before the capacity test

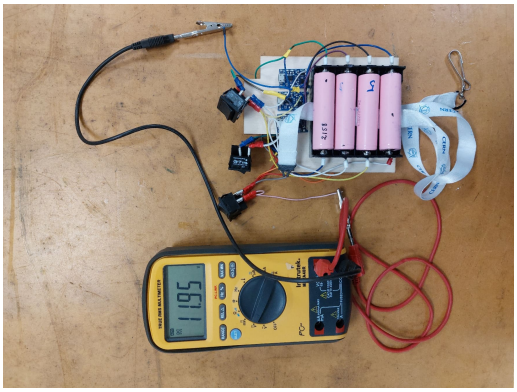
Table 7.1: Battery cell test results. Cell number 11 was dead.

7.9.2 Test of battery iterations

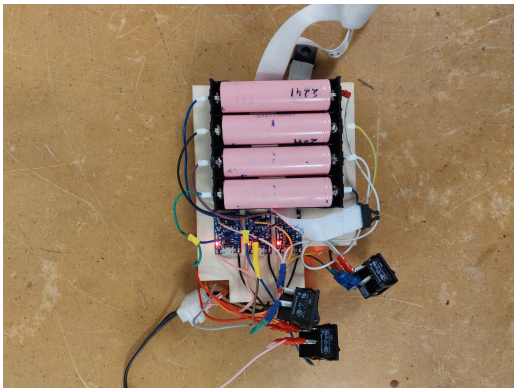
HB | *AEH*

Two types of batteries were tested, one with BMS and one without BMS. The reason that two different batteries were built and tested was that one of them failed a test it had to pass. Therefore, some changes had to be made to the battery.

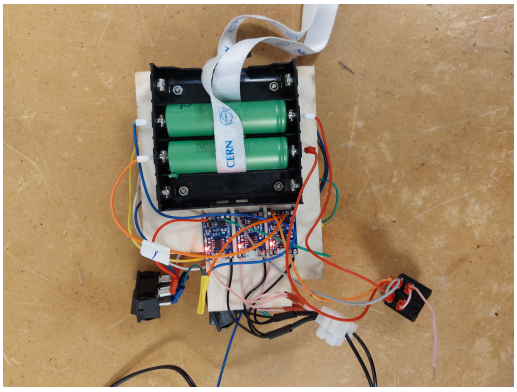
Test of battery iteration five The testing of the battery with BMS consisted of two stages. The first test was attaching the battery cells before the charging wires were attached. The negative and the positive wires were attached to the multimeter, then the battery was put into “use mode”.
The second test was charging the battery. When the charging wires were attached, the battery cells were inserted and the battery was put into charge mode. The charger was plugged into the power outlet from the wall and the charging module LEDs lit up. All seemed fine up until 30 seconds in when a burning smell occurred. The charger was disconnected, but the component did not stop frying. So all of the cells were ripped out of the charger to prevent further damage. This incident is one of the reasons the white band is placed below the cells.



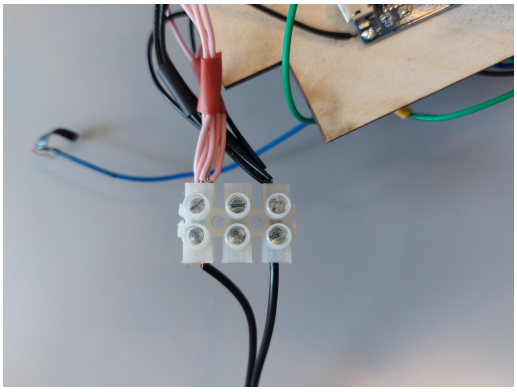
(a) Voltage test of the battery with the BMSs.



(b) Battery charging test top



(c) Battery charging test under



(d) Battery connection to charger during test

Figure 7.22: Test of battery with BMS. The red light on the BMSs indicates that the cells are charging.

Test of battery iteration six The battery without any BMS had only one power test, this was a voltage test where the output voltage was measured. This test was done in the same way as the battery with the BMSs. Inserting the cells, then measuring the output voltage. This test was a success, just like with the other battery.

The next test was to check that all of the produced parts fitted each other and the other parts that were provided to the project. To go through with this test, all of the parts were assembled and the battery was put together. Then check the output voltage again, but this time on the magnetic connector. The battery passed this test also.

The next test is checking if the battery fits in the battery drawer and that the pins align so that the magnetic connectors attach themselves. This test was carried out by inserting the battery, and then turning the robot on. This test was also a success.

The next and final test is to check the duration of the battery, one of the requirements is that the robot should be operational for at least 20 minutes before the battery must be changed. To test this, the battery cells were fully charged and inserted into the battery, and the battery was inserted into the robot. Then the robot was driven for approximately 30 minutes, when the robot suddenly died.

It turned out that the output voltage from the battery was 0 volts when measuring the voltage with a multimeter. The battery was then dismantled and the voltage was measured directly at each end of the battery cell holders. This was to locate the problem. Either the battery cells and the wires connecting them, or the wiring from the battery cell holders

to the magnetic connector. Luckily the battery cells still had a great amount of charge, 12.05 volts. This meant that the problem had to be the wiring from the cells to the magnetic connector. The wiring was checked to make sure that there was still contact between the connection surface on the magnetic connector and the wires that are attached to this. There was contact between these. The battery was then assembled again and the voltage was measured again at the magnetic connector, still measured at 0 volts. The next thing that was checked was the wiring going into the VAGO clips. It turned out that the isolation around the negative wire going from the battery cells had not been properly stripped, and the wires inside had been partially cut. These wires had fallen apart from each other. This led to a poor connection in the VAGO connector, therefore, the electricity could not flow through and the battery had zero output voltage. To fix the problem, the wire was re-stripped and the battery was yet again assembled. Now the battery worked as normal and the battery voltage was measured at 12.05 volts.

The battery cells were then swapped out with the cells from "Battery Cluster 2" and the robot was driven for another 30 minutes. Then the remaining voltage of the battery was measured. The voltage was measured at 11.98 volts. This means that the battery with "Battery cluster 2" also meets the requirement for 20 minutes of operational time. With the remaining amount of voltage in the battery and the time the battery was used, we can calculate the battery's operational time (see D).
The estimated operational duration of each battery cluster. See Q for the complete calculation of the estimated battery duration.

Battery cluster 1		
Test time:	30min	,50h
Voltage start:	12,60v	
Voltage empty:	9,60v	
Voltage end:	12,05v	
Voltage difference start empty:	3,00v	
Voltage difference start end:	,55v	
Drop per hour:	1,10v/h	
Estimated duration:	2,73h	
	2h	44min

Battery cluster 2		
Test time:	30min	,50h
Voltage start:	12,60v	
Voltage empty:	9,60v	
Voltage end:	11,98v	
Voltage difference start empty:	3,00v	
Voltage difference start end:	,62v	
Drop per hour:	1,24v/h	
Estimated duration:	2,42h	
	2h	25min

Table 7.2: Operational time based on 30 min operation, and calculations

Discussion
The reason that the battery with the BMS did not have the last test was that it failed on a functional test which it was required to pass to be able to be used. It was therefore no point in testing the parts to see if they fit.

The reason for the failed test can be several. The suspected reasons are; the design of the BMS not being compatible with the way the battery was designed, or bad soldering. The reason there might be the BMSs not being compatible is that the BMSs are designed to monitor only one cell at a time. This may lead to problems when connecting several cells. When researching the BMS type, it was discovered that the BMSs could not be connected in series. This would lead to overloading the BMSs. This is the reason the battery cells were directly connected, rather than through the BMSs. The reason it might be bad soldering is that the group participant that did the soldering was not very experienced with this. This led to a couple of bad connections, and probably applying too much heat when

soldering the wires to the BMSs. If too much heat is applied to the BMS while soldering, some of the components on the BMS can get destroyed and some connections get broken. The connections might loosen between the different components and attach themselves to a place where they are not supposed to have contact.

7.10 User survey testing **SO** | *AD*

We conducted full system testing with 11 students from software, mechanical, and electrical engineering disciplines at the university to evaluate our application for controlling the robot remotely through the Meta Quest 3 headset. The goal was to determine how easily individuals who had not previously seen or used the application could drive the robot, manipulate its arm, and switch modes using voice commands and the visual menu.

The results varied among the participants. Students with prior experience using VR headsets found the application intuitive and easy to use after a brief introduction. In contrast, students who had never used a VR headset needed some time to acclimate and initially found the process more challenging. However, after some practice, these students also found the application intuitive and user-friendly.

8 Future Work

This section describes functionality which were sub-par and could have been made differently, or features that could be useful additions but were ultimately not done or did not make it into the final product. The reasons for this could be time constraints or it was decided the current solution worked well enough for a PoC.

8.1 Software **OM** | *AD*

This section focuses on potential software enhancements or improvements that could be implemented in the future.

8.1.1 3D mapping using depth information **OM** | *AEH*

To accurately map the 3D space, knowledge of the system's time and position is essential. While the accelerometer, positional, and velocity data can be obtained from the expansion board, interpreting this data by observing it and moving the robot proved challenging. Integrating a Global Positioning System (GPS) or similar positional sensor to the robot, or tracking movement relative to camera frames, could provide the necessary information to determine the robot's position and trajectory.

By storing depth data relative to time with each movement, it becomes feasible to construct a 3D environment of depth data, which can then be transmitted to the VR headset. To manage the data volume, it would be practical to only transmit partial data or changes at a time. As the robot moves in its surroundings, it gradually maps the environment.

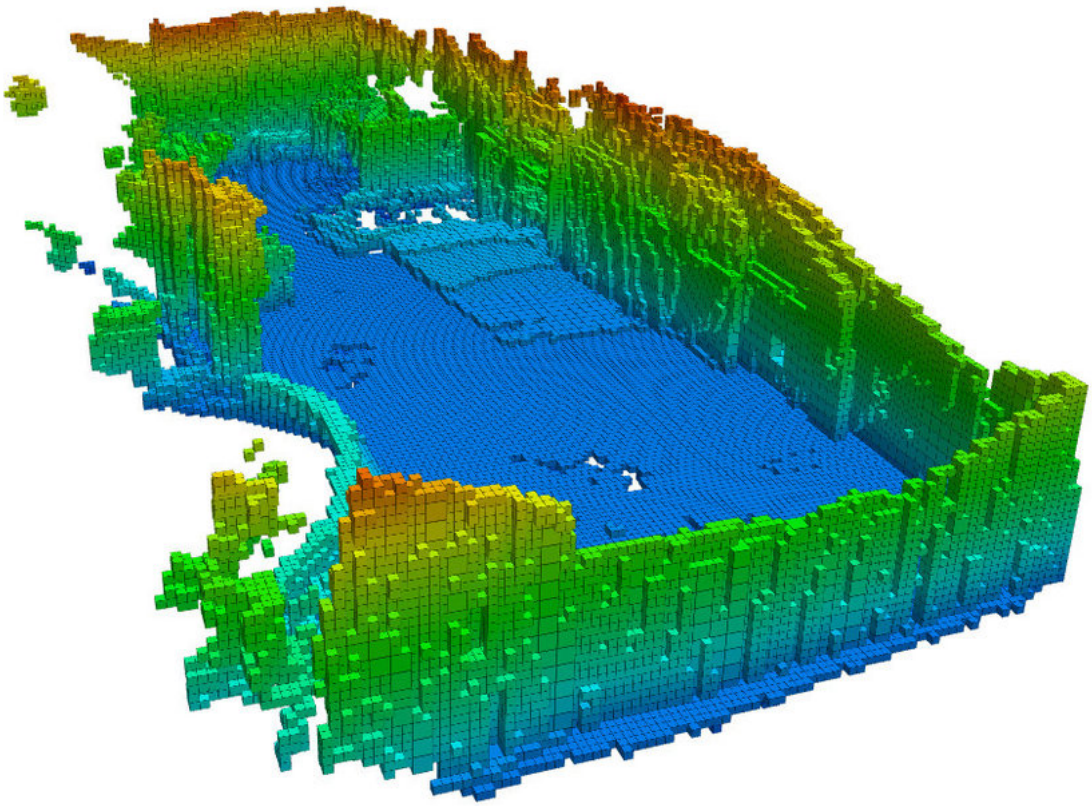


Figure 8.1: 3D mapped environment with OctoMap [13]

This approach offers several benefits for the operator, who could be virtually immersed in the 3D environment without physically being there. By looking around or even moving within the VR space, the operator gains insights into the environment's layout, distances between objects, and potential hazards to avoid. Moreover, this technology could aid navigation in low visibility conditions, such as at night, in poor lighting or weather, when the RGB data is subpar.

8.1.2 Autonomous movement, object avoidance and operation **OM** | *AD*

If an unmanned oil rig were assumed to be nearly static, a 3D map of the surroundings could be generated once and stored. The 3D environment could enable autonomous navigation, allowing the robot to manoeuvre around fixed obstacles like walls or pipes with less or no reliance on a RGB camera. This autonomy could, as earlier mentioned, extend to operation in diverse weather and lighting conditions, assuming the 3D mapping has been stored and mapped under optimal conditions.

Frequently traversed paths could be pre-defined and automated, streamlining routine movements. By integrating predefined arm movements, including rotations, entire operations could be autonomously executed. An example could be turning a valve every 24 hours, moving it back to the home station or charging port every time it is done.

However, real-world conditions introduce unpredictable factors, and having a dynamic aspect to predefined instructions is often essential. For instance, the robot would have to avoid unexpected obstacles like occasional workers or birds and adjust its position to counteract external forces like strong winds by using a Proportional-Integral-Derivative (PID) controller or like.

8.1.3 Avoid arm collisions with MoveIt **OM** | *AEH*

In its current state, the robot can avoid most collisions with itself, such as the arm crashing into the front of the robot. This is implemented by hard-coding a range of coordinates which are deemed illegal. By using a manipulator framework such as MoveIt it is possible to have collision avoidance in a live dynamic environment based on sensor inputs and other data. This could be beneficial in a live setting on an oil rig, as external objects such as pipes or valves can be avoided and not collide. This avoidance works also for the actual path the arm is moving and is being calculated live, so if a new object suddenly appears the arm could move around it on the go [150]. In addition, by supplying an URDF file MoveIt can know whether or not an arm movement would collide with the arm itself [151]. Due to most of the servos on our robotic arm being limited to ± 90 degrees of movement, it is harder for it to collide with itself. However, if the servos could move further this would be an issue, which could be solved with MoveIt.

8.1.4 Simultaneous driving and arm manipulation **SO** | *AEH*

While the group has tested both driving and arms control independently, the necessity of simultaneous driving and arm manipulation has been identified for certain situations. Specifically, when the operator moves the robot to a destination and begins an operation such as picking up an object and placing it elsewhere, it is crucial to be able to drive the robot while also controlling the arm. This is especially important when the arm does not reach the desired location. The ability to drive while controlling the arm would enhance the robot's functionality.

However, this feature was not developed within the project due to limited time and the risk of damaging the robot or its arm. Further work is needed to implement and test this capability.

8.1.5 Increase performance with Ubuntu OS **OM** | *AD*

When embarking on the thesis project, there was no Ubuntu LTS release compatible with the RPi 5. This, coupled with concerns about potential camera compatibility issues, led us to opt for RPiOS 64-bit with ROS 2 running within a Docker environment. This configuration offered a fusion of advantages, including Tier 3 ROS 2 support [59], camera compatibility and an OS deemed to be more stable.

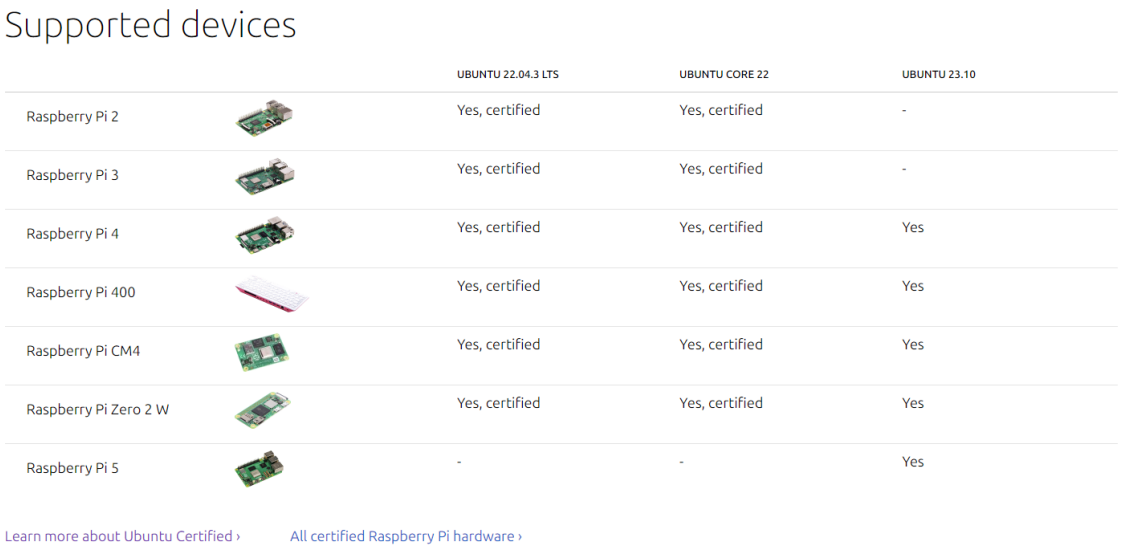


Figure 8.2: Ubuntu support as of 9th of February 2024 [14]

However, during the thesis, LTS support was introduced for Ubuntu 24.04 on the RPi 5 [14]. The group believes that the cameras will remain functional on this OS, and ROS 2 can run natively. This shift potentially reduces the overhead associated with Docker and might lead to slight performance improvements.

	UBUNTU 24.04 LTS	UBUNTU CORE 22
Raspberry Pi 5	Yes, certified	-
Raspberry Pi Zero 2 W	Yes	Yes, certified
Raspberry Pi CM4	Yes, certified	Yes, certified
Raspberry Pi 400	Yes, certified	Yes, certified
Raspberry Pi 4	Yes, certified	Yes, certified
Raspberry Pi 3	-	Yes, certified

Figure 8.3: Ubuntu support as of 6th of May 2024 [14]

Nevertheless, it is important to note that running the robot system within a Docker container simplifies deployment on other systems. This is facilitated by bundling all necessary packages within the Docker image, ensuring that the code operates within a standardized container environment, thereby mitigating compatibility concerns on alternative platforms. Since Docker also functions on Ubuntu, evaluating performance disparities between configurations should be feasible without requiring significant modifications.

8.1.6 Offload heavy computation

OM | *AD*

If improved performance is desired for tasks such as object detection or heavier calculations, it is feasible to delegate these tasks to a separate station equipped with more powerful hardware. Specifically, in the context of object detection, this approach is entirely viable because raw images can be transmitted rapidly for analysis elsewhere, and the VR operator can subsequently view the results of this analysis. This strategy has the potential to accelerate the analysis process, optimize battery usage, and reduce costs, as acquiring a powerful edge device is typically more costly than obtaining a larger stationary computer.

8.1.7 Tracking of object detection boxes

AD | *OM*

The current implementation of the object detection system successfully identifies and localizes objects in each frame. Still, a valuable enhancement for future improvements would be the integration of object-tracking capabilities. Object tracking is a component of computer vision that enables machines to track and follow objects in motion [152]. This enhancement would offer several advantages:

- **Improved Accuracy**: By essentially "fixing" the bounding box across frames, tracking can help reduce false positives and improve the overall accuracy of object detection, especially in challenging scenarios like changes in lighting.
- **Enhanced Functionality**: Tracking opens up possibilities for new features, such as analyzing object movement patterns, predicting trajectories, generating alerts based on object behaviour and much more. This could be particularly valuable for safety applications, where tracking the movement of personnel or equipment can potentially help prevent accidents.

The pipeline we used for implementing object detection, MediaPipe, supports integration with object tracking algorithms such as DeepSORT (Simple Online and Real-time Tracking with a Deep Association Metric) [153]. In addition, the OpenCV library also provides tracking support. The system would transition from a purely detection-based approach to a more comprehensive and dynamic perception system by incorporating object tracking. This could offer great potential for safety enhancements and advanced analytics on an oil platform.

8.1.8 3D object recognition with graph CNNs **AD** | *OM*

Extracting depth data and mapping the 3D space is a useful technology, explained in 4.6.1. The use of 3D mapping is increasing in daily life such as in VR and remote sensing mapping, which require processing and analysis of the collected 3D data [154].
In current times, 3D data has been studied using deep neural networks. Different types of CNNs can be used on 3D data, such as graph CNNs, which can operate directly on graph-structured data [155]. This makes them well-suited for processing 3D shapes, which can be represented as meshes, a type of graph where the vertices are the points in the shape and the edges are the connections between them [154]. VR applications can also use meshes to visualise the 3D data 9.2.7.
By accurately classifying 3D objects within the environment, the system could enable more intelligent interactions and realistic simulations.

8.2 Mechanical

8.2.1 Increase lifting capacity of the arm **OM** | *AD*

As of now, the robotic arm is limited to carrying small and lightweight objects, such as plastic bolts. It was earlier considered that redesigning the arm or acquiring stronger servos to address this limitation. However, these considerations were quickly shut down by the software discipline due to limited knowledge about inverse kinematics at that stage of the project. Moreover, the group anticipated using MoveIt, which required a URDF file already available for the arm provided by Yahboom. Uncertainty surrounded the potential difficulties and implications of changing dimensions or servo limitations.

In hindsight, with the knowledge acquired later in the project, it became evident that minimal software changes would have been necessary. This could have led to an even more intriguing design process for the mechanical discipline if the arm were to be redesigned.

Nevertheless, if no additional time were to be spent on inverse kinematics, the arm would have needed to be designed fairly similarly. Given the arm's current design, inverse kinematics is relatively straightforward compared to a more complex design. The servos would have to move in the same plane as the current version, albeit with increased strength and possibly flexibility.

However, using other servos could necessitate changes to the code for driving the servos, potentially requiring time to develop a servo driver. Additionally, the robot would likely require new electronics to power these servos, which in turn might have necessitated a larger battery.

8.3 Electrical

8.3.1 Upgrade hardware **OM** | *AD*

Some of the challenges encountered (see 9) could be mitigated by investing in more expensive hardware, which could enhance operations. A more powerful edge device or increased bandwidth, as well as a dedicated depth-sensing device, could provide better depth data results. Additionally investing in more reliable electronics would be beneficial (see 9.2.5).

A more powerful VR headset could also be advantageous. Although the Meta Quest 3 offers many features at a relatively low cost, it may not be sufficient for more demanding tasks. Constantly running passthrough to stay connected with the physical environment around the operator while rendering a virtual environment with menus and buttons, and listening for data, is a demanding task. Upgrading to a more capable VR headset could improve performance and accuracy.

8.3.2 Battery **HB** | *OM*

The battery has several things that could be improved, one of these things is the switches used in the battery. The switches are on/off switches, but instead switches with three poles could have been used to properly disconnect the BMSs when the battery is put into use mode.
The battery that was made was not ideal, because it ended up with not having a BMS. Because of this, it would not be possible to charge the battery itself. However, the cells have to be taken out and put in a battery cell charger or the cells have to be swapped out. This should not be necessary, and would not be if there had been bought a 2P3S BMS. This way, all of the cells could be monitored during use, and the battery could have been charged without removing the cells. This way, the switches could have been ditched and there would not be a need for a charging mode and a using mode, that would have been sorted out by the BMS.

9 Challenges

Kromium faced a lot of technical challenges and non-technical challenges during the course of the project. The section will delve into the key challenges encountered. It outlines the primary obstacles faced and highlights areas that require attention to ensure continuous improvement and efficiency. Understanding these challenges has been crucial for developing effective strategies to mitigate potential issues and enhance overall performance.

9.1 Non-technical challenges

9.1.1 Electrical engineer could not continue **AEH** | *OM*

When we formed our group back in August 2023 we were six members. The group stayed the same through discussions with the project client till the start of January 2024 when we lost our electrical engineer and expert in LaTeX. He could not continue the project, unfortunately.

As a result, more time had to be devoted to understanding LaTeXand developing our report outline at the start of the project. The requirements from Kongsberg Maritime also had to be re-discussed. This led to the removal of a requirement to design our circuit board. We thought that this would affect the budget as we thought we needed to buy battery packs, but as we built our own battery, it affected the budget minorly.

9.1.2 Previous report being exempt from public **AEH** | *OM*

Documentation of the last bachelor project using the ROSMASTER X3 PLUS was not available. We wanted to know more about the equipment we had available from Kongsberg Maritime so we found the report from K-Spider (2016) but could not find the report from K-AIoT (2023). The report was not available in the USN archive of previous projects or on the web page for the group. After asking around we found out that the project was not available and had to ask our client Kongsberg Maritime to access it.

9.1.3 Carbon CNC mill broken **HB** | *AD*

The walls on the robot are made from carbon fibre sheets. The sheets were made in advance by someone else, which meant the only thing that needed to be done to the plates was to fabricate them in a milling machine. Because the lab engineer responsible for the composite lab had limited time to help the project group, the milling process had to be divided into two sessions, a week apart from each other. However, before the second session with machining could begin, it was discovered that the milling machine was broken and impossible to use. This meant that there had to be found another way to produce the parts.

The possible options were: to drop the last carbon fibre parts, to use an angle grinder on the carbon plate to cut the parts out, or to find a company that could cut the parts. It was decided to go for the last option. “Vannskjæresenteret AS” was already producing some parts for the robot. After communicating with them, they said it was okay for them to produce the parts from the carbon plates with their water jet. However, some changes had to be made to the technical drawing to prevent delamination. Overall, it took one and a half weeks extra to get the final carbon parts for the robot.

9.1.4 External supervisor changing job during the project **OM** | *AD*

Qui-Huu Le-Viet informed the group early on during the thesis that he would be changing jobs. He made efforts to schedule this transition after the thesis had been delivered to avoid interfering with our work. However, due to unforeseen circumstances, Qui-Huu had to transition to his new job earlier than initially anticipated. This led the group to receive a new external supervisor. Qui-Huu was replaced by Merethe Gotaas, whom the group promptly informed and brought up to speed. Merethe was present during our initial discussions about undertaking Kongsberg Maritime's bachelor thesis and during our first presentation. The group would like to thank Qui-Huu, Merethe, and Kongsberg Maritime for making this transition as seamless as possible, given the circumstances.

9.2 Technical challenges

9.2.1 Corrupting one of our microSD cards **OM** | *AD*

Installing and setting up Ubuntu 23.10 on the RPi 5 was successful and got booted into the desktop as normal. However, after a few minutes, the RPi did not respond to any mouse or keyboard interaction. Measuring the heat of the CPU with our fingertips, we noticed it was getting quite hot. With a suspicion that the RPi was overheating and limiting its performance, the power source was unplugged to possibly prevent it from getting damaged.

After waiting some time, we powered the RPi up again. Instead of getting booted into Ubuntu, *BusyBox* was instead noticed. Trying to exit this prompt, using the "exit" command, "Kernel panic!" could be observed in the terminal. Having little knowledge about the error itself, we tried to reformat the microSD card to reinstall Ubuntu.

Trying to reformat the microSD card using Raspberry Pi Imager [57], the card claimed to be "write protected". It was made sure that the SD card adapter was open and not locked. After searching around for a solution *diskpart* on Windows was said to be able to disable this write protection. None of the commands failed, but the microSD card remained write-protected.

9.2.2 Frying an expansion board diode **OM** | *AD*

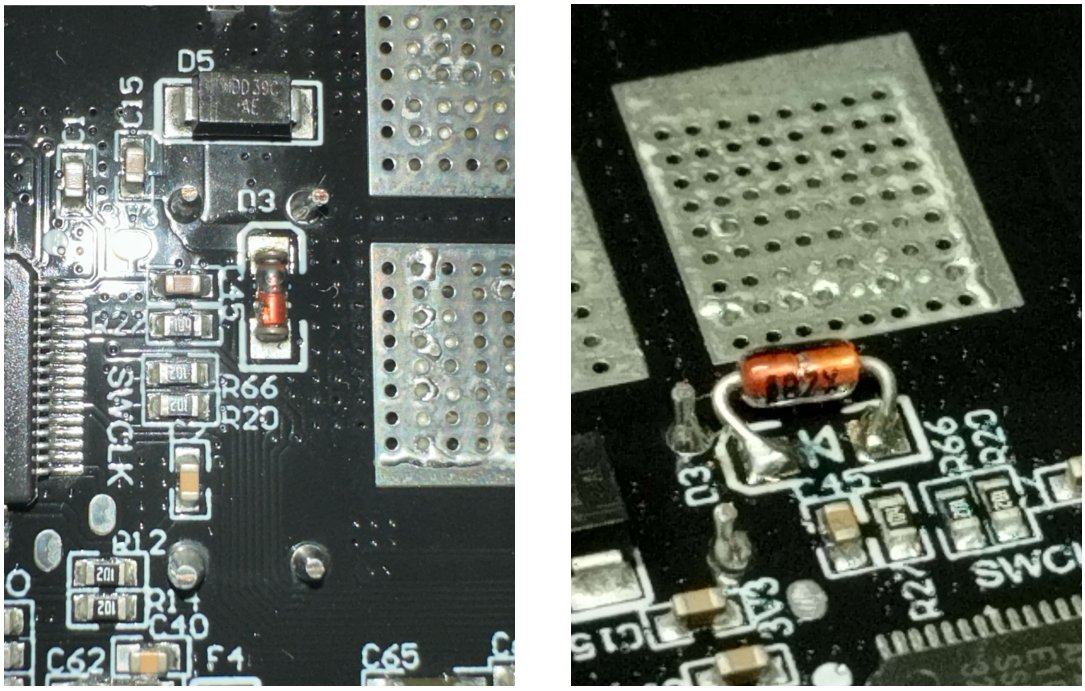
Expansion board refers to the same as in 4.3.5. On the 1st of March 2024, while testing some predefined arm movements the battery plug came loose and disconnected from the expansion board. The battery plug was purposely not fully connected, as the plug sits tight and is tough to disconnect if completely connected. Connecting the battery loosely makes it more practical if unexpectedly have to disconnect it.

When the expansion board cannot drive power from the main power source, it will try to drive power from the USB ports instead, where in our case the RPi was connected, which again was connected to the wall outlet. The RPi cannot provide the expected 12V of power, but instead only 5V. This is not enough power for the expansion board and the buzzer will start to beep, indicating that not enough power is being provided.

Without turning the board off, we reconnected the 12V battery again. After which a fuzzing sound could be heard and smoke was observed. A visual inspection on the back side of the board was made, which concluded that the D3 mounted Zener diode ZMM5V1 got fried (see I.2). This diode is a safety feature and works as a fuse that protects the board.

The circuit is still closed and can work normally without this connection, but is not recommended. The group decided that testing without this safety was not applicable, and efforts would instead go to get the diode replaced before our presentation at Kongsberg Maritime (06.03.2024) and the second presentation (08.03.2024).

The group would like to thank Ole Eirik Solberg Seljordslia, who took time out of his weekend to help us get this diode replaced. We did not have the skill set nor the tools to do this ourselves, as our group had no electrical engineer. This incident led us to buy an additional expansion board, to act as backup. We also changed to always have the RPi powered through the expansion board and not an external power source. This slowed down the testing process as the RPi changes power source and has to boot up again, but increased safety.



(a) The damaged diode

(b) The replaced diode

Figure 9.1: Both pictures show the back side of the expansion board.

9.2.3 Problems with downloading tflite-model-maker

AD | *OM*

When creating a custom-dataset transfer learning model (see Appendix M), our group encountered significant compatibility and dependency challenges while integrating the package *tflite-model-maker*. Initially, the package was identified to be compatible with Python 3.11. However, upon attempting installation within our primary development environment on the Windows OS, it became apparent that a downgrade to an earlier Python version was necessary due to compatibility issues. But even this did not bring success as there was still the incompatibility of a critical dependency package called Scann [156], which lacked support for Windows. Subsequent efforts to resolve this issue by downgrading the Python version within a cloud-based service called Google Colab also did not yield positive results. The solution was eventually found by transitioning the development environment to the WSL on Ubuntu 22.04 and installing Python 3.8 there, which enabled the installation of *tflite-model-maker*. The encountered difficulties resulted in unforeseen

delays and long waiting times as downloading different versions took time.

9.2.4 Difficulties in using the PyTorch library **AD** | *OM*

Pytorch for edge devices: Deploying deep learning models on edge devices presents unique challenges, particularly when using libraries initially designed with server-grade resources in mind. Our project encountered obstacles while exploring PyTorch. One of the most pressing issues was the lack of comprehensive documentation for **PyTorch Mobile** which is the PyTorch library for deploying models on edge devices [157].

Despite PyTorch's robust features and active community, PyTorch Mobile's documentation remains scant. The documentation referred only to iOS and Android development and not RPi [157]. This lack of detailed support hindered our ability to efficiently troubleshoot and fix errors, leading to prolonged development times. We then found the You Look Only Once (YOLO) algorithm, written below.

You Look Only Once algorithm (YOLO): In object detection, YOLO is considered a standard and powerful detection algorithm because it detects objects with great precision and speed [158]. The latest version of YOLO uses the PyTorch library to implement object detection. The basic principle of YOLO is that it applies a single CNN to the full image. It divides the image into regions and predicts bounding boxes and probabilities for every region. It then uses complex post-processing methods and uses NMS to keep only the bounding boxes and probabilities with the highest confidence scores while also managing the IoU [158]. It was therefore why we considered using the algorithm for our project. However, one of the downsides of the algorithm is that it needs a lot of computational power. The official documentation recommends running YOLO with a GPU [159]. The RPi 5 has a GPU called Broadcom VideoCore VII GPU [160], but neither TensorFlow nor PyTorch has support to use this GPU.
So, we decided to export the YOLO model to a TFLite model as it is recommended for edge devices such as the RPi. However, due to the nature of the YOLO model, it was challenging to deploy it in the same way one would deploy a normal TFLite model made from the TensorFlow library particularly because of the overlapping boxes and probabilities. Figure 9.2 shows a frame where the custom YOLO model was applied without NMS. The absence of NMS led to numerous duplicate detections. We attempted to create our

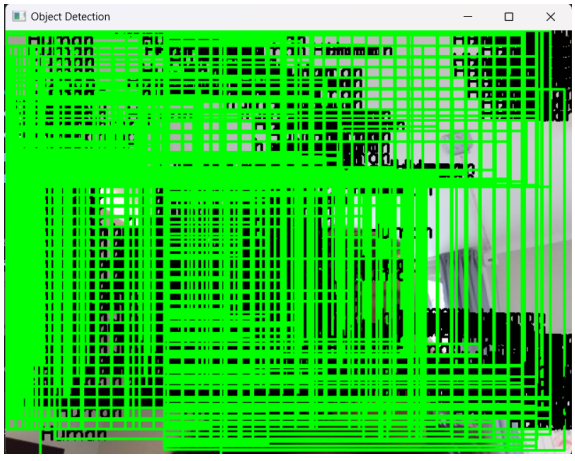


Figure 9.2: YOLO real-time detection without NMS

own method, but the documentation was not easy to understand and community solutions

were scarce. The NMS method used in the documentation [161], along with another implementation we found, severely affected the speed of the inference, reducing it to 4-5 FPS [162]. It was like there was no difference between using the original YOLO model with CPU support and using the TFLite model. This problem also resulted in significant delays in our project timeline.
We wished we had access to more powerful hardware, such as an edge device with GPU support, which could have potentially handled the computational demands of the model more effectively. Ultimately, our group abandoned the YOLO approach and opted to use MediaPipe, as mentioned earlier in our implementation. However, this change in strategy set us back several days as we had to reconfigure our setup, but it was necessary to move forward under the constraints we faced.

9.2.5 Robot randomly going unresponsive **OM** | *AD*

Shortly after the initial interaction with the expansion board, it became evident that the *Rosmaster* code would randomly throw an exception, rendering it unresponsive to new commands. Despite debugging efforts, the root cause of this exception remains unknown, with no obvious pattern linking command execution to exception occurrence. The exception is also inconsistent as sometimes the exception does not surface throughout an entire day of testing, and other times it might occur immediately upon the first command sent. Checks are done to ensure that only valid types and values are transmitted.

In the event of this exception, commands sent to the expansion board will not be executed. The origin of this issue remains uncertain, whether it stems from a flaw in the *Rosmaster* code, a firmware issue with the expansion board, or a hardware-related anomaly. The reported errors, namely `set_servo_error`, `set_motor_error`, `get_servo_angle_error` and alike, persist as recurring issues. Additionally, no reliable solution has been identified. Sometimes restarting the robot code is enough, other times a complete reboot of all electrical components is needed.

9.2.6 Issues with MoveIt **OM** | *AD*

A significant amount of time was dedicated to reviewing *MoveIt* documentation and experimenting with different solutions. Ultimately, it was concluded that implementing arm movements independently would be more time-efficient.

Initially, there was uncertainty regarding the compatibility of *MoveIt* with the arm64 architecture. Yahboom had a solution for *MoveIt* 1, but this version relies on ROS 1, which differs and is incompatible with ROS 2 unless a ROS 1 bridge or similar workaround is utilized [163].

Attempts to install *MoveIt* using the binary `sudo apt install ros-humble-moveit` resulted in the RPi running out of space on the SD card overnight. This led to the purchase of two new SD cards, followed by the installation of the OS, the addition of SSH keys for Git handling, and the reinstallation of Docker. While *MoveIt* offers Docker images, these are only available for the amd64 architecture, not arm64, which the RPi operates on. After encountering some difficulties, *MoveIt* was eventually installed.

One of the significant features of *MoveIt* is its ability to simulate and provide a Graphical user interface (GUI) for configuration and verification purposes. However, due to the complexity of *MoveIt*'s functionalities, understanding its setup was challenging. It was

unclear how to access *RViz* and other GUI applications within the Docker container when the RPi was accessed remotely via SSH. Resources on setting up were difficult to understand and configuring *MoveIt* was expected to be time-consuming. Additionally, there were doubts about whether *MoveIt* would meet the project's requirements, particularly in defining custom movements such as assembling a bolt.

Ultimately, it was concluded that developing a custom solution for controlling the arm would be more time-effective. *MoveIt* was deemed overly complex and unnecessary for the PoC, as real-time obstacle avoidance and highly precise movements were not essential. This decision allowed for learning about inverse kinematics, flexibility in adding features as needed and achieving sufficient precision for the project's objectives.

9.2.7 Trouble transferring ∼300,000 points of depth data **OM** | *AD*

Due to the large size of the depth data, it was not possible to send all of it in one message without any compromises. This section describes different solutions that were tried or considered but ultimately not used in the final product. The current implementation is discussed in 4.6.1.

Splitting packages The raw depth data, totalling approximately 19MB, was initially split into smaller packets to be sent more frequently. However, splitting the data into 1024-bit (1KB) packets resulted in around 19,000 packets, which proved impractical. The VR headset struggled to receive all this data and would crash shortly after receiving the first packets. Additionally, since the socket was a TCP connection, transferring the data took significant time. Another issue was determining when a new frame was taken or resetting the scene in the VR application. This could be addressed by checking if the same x, y coordinate was already plotted, indicating a new frame. These challenges led us to consider other options for sending the data.

Creating object files Creating 3D meshes or objects from point clouds was considered a viable option. These objects would be `.obj` files that Unity can display directly. All objects were tested and created using the Open3D Python package [164]. However, to make these meshes small enough, they needed to be simplified. Using Poisson surface reconstruction and limiting the number of triangles helped with this. Running these processes on the RPi took significant time, as doing so on a laptop took more than 15 seconds. The size of these meshes was reduced to as low as 332KB, and could potentially be decreased further, but still too large to be sent comfortably as one message. Open3D's support for arm64 was limited, as the RPi could produce a `.obj` file, but not consistently. This led the group to search for another solution. An example mesh created using depth data is shown below, with the code for the plot provided in O.13.

Figure 9.3: Mesh seen from front

Figure 9.4: Mesh seen from the right-hand side

Figure 9.5: 3D mesh of depth data with highlighted objects

Broadcasting the points Another solution that was discussed but never implemented was broadcasting the points using UDP or a similar protocol. This approach would lead to some points being lost, but with so many points, missing a few would not significantly impact the overall plotting. However, as with splitting the messages, this method would result in a huge number of messages that the VR application would need to process. Additionally, the issue of determining when a new frame was received persisted.

9.2.8 Frying a BMS

HB | *AEH*

During the second test of the battery with BMS, the battery was charging, which it was supposed to do. After approximately half a minute a burning smell occurred, quickly followed by smoke coming from one of the Integrated circuit (IC) on one of the BMSs. The charger was quickly disconnected but frying of the IC continued. The cells had to be quickly disconnected before anything caught fire. Luckily, the ribbon placed under the battery cells worked as intended and loosened the cells from the battery cell holders.

The reason the ribbon is placed between the battery cells and battery cell holders is to make the removal of the battery cells easier. If the ribbon is not inserted there needs to be used something to pry the cells out of the holders. This can possibly damage the battery cells and make the cell useless.

Because there was only a short amount of time left before the deadline of the project and the reason why the BMS got fried was unknown, it was decided to make a battery without BMS. The reasons the BMS got fried are either that the schematics for the battery were made wrong or that there was added too much heat when soldering the wires to the BMSs. If too much heat was added to the BMS the components in the BMS can get damaged or short-circuited.

Because it was decided to make a new design iteration of the battery, the battery needed a redesign and its schematics had to be updated. The new design ended up being smaller than the previous design, and the wiring was a lot simpler than the battery with BMS.

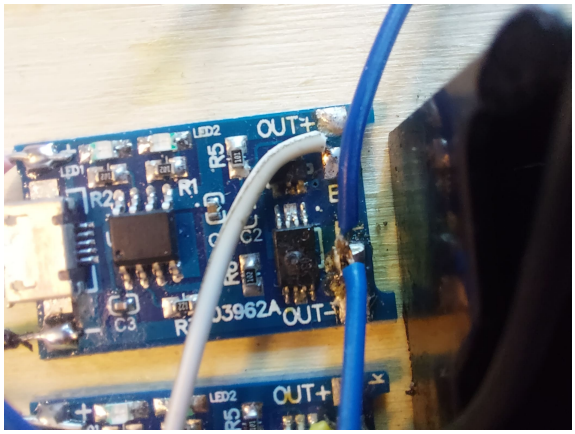


Figure 9.6: The BMS with the fried IC

10 Conclusion

The project aimed to develop a PoC prototype that leverages VR, AI, and robotics to facilitate remote operations on offshore platforms. By integrating these cutting-edge technologies, the goal was to enhance operational safety, efficiency, and cost-effectiveness while reducing the need for personnel to be physically present in hazardous offshore environments.

Throughout the project, we explored the use of VR for immersive interaction, AI for object detection, and robotics for executing precise tasks. The combination of these technologies demonstrated significant potential in replicating human operations remotely. Key achievements include successful integration of VR with the robot for real-time interaction, allowing for driving the robot with hand movements and manipulating the robot arm, successfully extracting depth data, AI detection of humans and bolts, digital twin replication of the robot arm inside the VR application, voice control for robot commands, and extensive physical design of the robot.

Despite the successes, the project faced several challenges, including technical limitations in hardware and extensive troubleshooting due to random hardware failures. Ensuring real-time information exchange between the VR application, robot system, and AI modules was also a significant challenge. These issues were addressed through iterative testing and refinement, as documented in our integration testing section.

The project has laid a strong foundation for future work in this field. Future enhancements could include improving the robot's autonomous capabilities, enhancing the VR interface for better user experience, and integrating more advanced AI algorithms for complex decision-making. Additionally, exploring the scalability of this solution for larger and more complex offshore operations could further validate its practical applicability.

In conclusion, the successful development and demonstration of the PoC prototype underscore the potential of combining VR, AI, and robotics to transform offshore operations. This project not only highlights the feasibility of such an integration but also paves the way for future innovations that can significantly improve safety and efficiency in the offshore industry.

11 References

[1] Getting started with hand tracking on meta quest headsets | meta store. [Online]. Available: https://www.meta.com/en-gb/help/quest/articles/headsets-and-accessories/controllers-and-hand-tracking/hand-tracking/

[2] Raspberry pi 5 - 8 GB • RaspberryPi.dk. [Online]. Available: https://raspberrypi.dk/no/produkt/raspberry-pi-5-8-gb-2/

[3] RoboteQ. Driving mecanum wheels omnidirectional robots. [Online]. Available: https://www.roboteq.com/applications/all-blogs/5-driving-mecanum-wheels-omnidirectional-robots

[4] O. Robotics. Understanding nodes — ROS 2 documentation: Humble documentation. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

[5] Amazon. Amazon.com: Yahboom robot arm kit 6dof for raspberry pi 4b AI programmable electronic DIY robot hand building with camera for adults ROS open source (nano-DOFBOT without nano) : Toys & games. [Online]. Available: https://www.amazon.com/Yahboom-Jetson-Nano-Identity-Programming%EF%BC%88DOFBOT/dp/B08T6N36YR?th=1

[6] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, Inc. [Online]. Available: https://dl.ebooksworld.ir/books/Artificial.Intelligence.A.Modern.Approach.4th.Edition.Peter.Norvig.%20Stuart.Russell.Pearson.9780134610993.EBooksWorld.ir.pdf

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," version: 4. [Online]. Available: http://arxiv.org/pdf/1801.04381

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," version: 2. [Online]. Available: http://arxiv.org/pdf/1708.02002

[9] C. Lugaresi, J. Tang, and C. McClanahan, "MediaPipe: A framework for building perception pipelines," p. 9. [Online]. Available: https://arxiv.org/pdf/1906.08172

[10] YoloDataset. Person dataset dataset overview. [Online]. Available: https://universe.roboflow.com/yolodataset/person-dataset-mvbk4

[11] Astra pro plus realsense depth camera support 3d mapping navigation fo. [Online]. Available: https://www.hiwonder.com/products/astra-pro

[12] wit. Wit.ai. [Online]. Available: https://wit.ai/

[13] C. Wang, L. Meng, S. She, I. M. Mitchell, T. Li, F. Tung, W. Weiwei, M. Q.-H. Meng, and C. de Silva. Autonomous mobile robot navigation in uneven and unstructured indoor environments. [Online]. Available: https://www.researchgate.net/publication/321811453_Autonomous_mobile_robot_navigation_in_uneven_and_unstructured_indoor_environments

[14] Ubuntu. Install ubuntu on a raspberry pi. [Online]. Available: https://ubuntu.com/download/raspberry-pi

[15] H. Technology, "18650 lithium battery charger with protection." [Online]. Available: https://www.handsontec.com/dataspecs/module/18650-Lithium%20charger.pdf

[16] H. , L. , and R. , "Magnetic 5a 2 contacts rotationg 360 degrees panel sloder," technical drawing. [Online]. Available: https://docs.rs-online.com/5cef/0900766b8166fb65.pdf

[17] Kromium. ar-robotics/robot: Kromium's code which controls the robot and communicates with the VR headset. [Online]. Available: https://github.com/ar-robotics/robot

[18] CodeTabs. Count LOC online. [Online]. Available: https://codetabs.com/count-loc/count-loc-online.html

[19] U. Technologies. Unity - manual: Scenes. [Online]. Available: https://docs.unity3d.com/Manual/CreatingScenes.html

[20] ——. Unity - manual: GameObjects. [Online]. Available: https://docs.unity3d.com/Manual/GameObjects.html

[21] ——. Unity - manual: Prefabs. [Online]. Available: https://docs.unity3d.com/Manual/Prefabs.html

[22] Kromium, "ar-robotics/transfer-learning-training," original-date: 2024-02-28T09:32:55Z. [Online]. Available: https://github.com/ar-robotics/transfer-learning-training

[23] ——, "ar-robotics/obj-detection-pi," original-date: 2024-02-23T09:07:18Z. [Online]. Available: https://github.com/ar-robotics/Obj-detection-pi

[24] TronicsBench. The TP4056: Lithium ion/polymer battery charger IC. [Online]. Available: https://www.best-microcontroller-projects.com/tp4056-page2.html

[25] Closed-source software (proprietary software). [Online]. Available: https://encyclopedia.kaspersky.com/glossary/closed-source/

[26] Programiz. Python docstrings (with examples). [Online]. Available: https://www.programiz.com/python-programming/docstrings

[27] F. L. a. I. Lunden. Microsoft has acquired GitHub for $7.5b in stock. [Online]. Available: https://techcrunch.com/2018/06/04/microsoft-has-acquired-github-for-7-5b-in-microsoft-stock/

[28] A. Takyar. Hyperparameter tuning: Optimizing ML models for excellence. [Online]. Available: https://www.leewayhertz.com/hyperparameter-tuning/

[29] Jetson nano. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano

[30] Cambridge, *Cambridge Dictionary*. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/

[31] IBM. What is a machine learning pipeline? | IBM. [Online]. Available: https://www.ibm.com/topics/machine-learning-pipeline

[32] R. P. Ltd-os. Operating system images. [Online]. Available: https://www.raspberrypi.com/software/operating-systems/

[33] Webflow. Wrapper – definition | webflow glossary. [Online]. Available: https://university.webflow.com/glossary/wrapper,https://webflow.com/glossary/wrapper

[34] Home - hydroplant. [Online]. Available: https://hydroplant.no/

[35] R. Sørensen, H. Moholth, H. M. Moholth, A. Moholth, and N. H. L. Håre, "Hivemind," accepted: 2023-11-01T12:42:20Z Publication Title: 518. [Online]. Available: https://openarchive.usn.no/usn-xmlui/handle/11250/3099988

[36] M. Smith-Solbakken and E. A. Dahle. Alexander kielland-ulykken. Store norske leksikon. [Online]. Available: https://snl.no/Alexander_Kielland-ulykken

[37] R. Ferguson and K. McMenemy, *A Hitchhiker's Guide to Virtual Reality*. [Online]. Available: https://learning.oreilly.com/library/view/a-hitchhikers-guide/9781568814773/

[38] B. Marr. The important difference between virtual reality, augmented reality and mixed reality. Section: Enterprise & Cloud. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2019/07/19/the-important-difference-between-virtual-reality-augmented-reality-and-mixed-reality/

[39] C. Linn, S. Bender, J. Prosser, K. Schmitt, and D. Werth, "Virtual remote inspection — a new concept for virtual reality enhanced real-time maintenance," in *2017 23rd International Conference on Virtual System & Multimedia (VSMM)*, pp. 1–6, ISSN: 2474-1485. [Online]. Available: https://ieeexplore.ieee.org/document/8346304

[40] E. Charniak and D. V. McDermott, *Introduction to artificial intelligence*. Addison-Wesley.

[41] S. Srivastava. Top AI trends in 2023: Unveiling use cases across industries. [Online]. Available: https://appinventiv.com/blog/ai-trends/

[42] M. Tatum. What is machine perception? [Online]. Available: https://www.easytechjunkie.com/what-is-machine-perception.htm

[43] I. . What is computer vision? | IBM. [Online]. Available: https://www.ibm.com/topics/computer-vision

[44] I. IBM. What is machine learning? | IBM. [Online]. Available: https://www.ibm.com/topics/machine-learning

[45] S. Daley, "Robotics." [Online]. Available: https://builtin.com/robotics

[46] R. R. Murphy, *Introduction to AI robotics*, 2nd ed., ser. Intelligent Robotics and Autonomus Agents Series. MIT Press, e-book. [Online]. Available: https://ebookcentral.proquest.com/lib/ucsn-ebooks/reader.action?docID=6340541

[47] KongsbergM. About kongsberg maritime. [Online]. Available: https://www.kongsberg.com/maritime/about-us/

[48] KongsbergG, "Kongsberg gruppen årsrapport." [Online]. Available: https://www.kongsberg.com/globalassets/corporate/investor-relations/annual-report-2022/kog-rapport-2022-no.pdf

[49] J. M. Safi, "Module 12 system integration & qualification v23."

[50] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review." [Online]. Available: http://arxiv.org/pdf/1807.05511

[51] S. Srichitra and S. Sreeja, "Implementation of ROS-based mobile robots with few shot object detection using TensorFlow API," in *Soft Computing: Theories and Applications*, R. Kumar, C. W. Ahn, T. K. Sharma, O. P. Verma, and A. Agarwal, Eds. Springer Nature, pp. 457–468.

[52] A. Farasin, F. Peciarolo, M. Grangetto, E. Gianaria, and P. Garza, "Real-time object detection and tracking in mixed reality using microsoft HoloLens," pp. 165–172. [Online]. Available: https://www.scitepress.org/Link.aspx?doi=10.5220/0008877901650172

[53] The mad inventor – perfect escape oslo. [Online]. Available: https://perfectescape.no/oslo/en/spill/den-gale-oppfinneren/

[54] Lodge free website template | free CSS templates | free CSS. [Online]. Available: https://www.free-css.com/free-css-templates/page283/lodge

[55] R. P. . Ltd. Buy a raspberry pi 5. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[56] Ubuntu release cycle. [Online]. Available: https://ubuntu.com/about/release-cycle

[57] R. P. Ltd. Raspberry pi OS. [Online]. Available: https://www.raspberrypi.com/software/

[58] Kromium. Kromium's GitHub organization. [Online]. Available: https://github.com/ar-robotics

[59] ROS 2 on raspberry pi — ROS 2 documentation: Humble documentation. [Online]. Available: https://docs.ros.org/en/humble/How-To-Guides/Installing-on-Raspberry-Pi.html

[60] ROS 2 humble hawksbill release! [Online]. Available: https://www.openrobotics.org/blog/2022/5/24/ros-2-humble-hawksbill-release

[61] Amazon. What is docker? | AWS. [Online]. Available: https://aws.amazon.com/docker/

[62] Install docker engine on debian. [Online]. Available: https://docs.docker.com/engine/install/debian/

[63] U. Technologies. Learn how to code in c# for beginners | unity learn. [Online]. Available: https://unity.com/how-to/learning-c-sharp-unity-beginners

[64] Google. Google colab. [Online]. Available: https://research.google.com/colaboratory/faq.html

[65] Unity. Unity hub – manage editor versions and collaborate with other creators | unity. [Online]. Available: https://unity.com/unity-hub

[66] styleguide. [Online]. Available: https://google.github.io/styleguide/pyguide.html

[67] sphinx.ext.autodoc – include documentation from docstrings — sphinx documentation. [Online]. Available: https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html

[68] sphinx.ext.napoleon – support for NumPy and google style docstrings — sphinx documentation. [Online]. Available: https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html

[69] P. Gedam, "pradyunsg/furo," original-date: 2020-09-01T21:08:17Z. [Online]. Available: https://github.com/pradyunsg/furo

[70] GitHub. About workflows. [Online]. Available: https://docs.github.com/en/actions/using-workflows/about-workflows

[71] ——. GitHub copilot · your AI pair programmer. [Online]. Available: https://github.com/features/copilot

[72] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*, 1st ed. Addison-Wesley Professional. Part of the Addison-Wesley Signature Series (Cohn) series. [Online]. Available: https://www.informit.com/store/essential-scrum-a-practical-guide-to-the-most-popular-9780137043293

[73] N. H. ASA and DNV, "Methology for rapid risk ranking of h2 refuelling station concepts," p. 9. [Online]. Available: http://www.eihp.org/public/documents/RRR%20methodology_final_SEP2002.pdf

[74] rosmasterx3. ROSMASTER x3 PLUS ROS robot for jetson NANO 4gb/xavier NX/orin NX/orin NANO/RPi 4b. [Online]. Available: https://category.yahboom.net/products/rosmaster-x3-plus

[75] B. Wu. 2.2. degrees of freedom of a robot – modern robotics. [Online]. Available: https://modernrobotics.northwestern.edu/nu-gm-book-resource/2-2-degrees-of-freedom-of-a-robot/

[76] U. Technologies. Unity - manual: VR development in unity. [Online]. Available: https://docs.unity3d.com/2022.3/Documentation/Manual/VROverview.html

[77] o. developer. Set up development environment and headset | oculus developers. [Online]. Available: https://developer.oculus.com/documentation/unity/unity-env-device-setup/

[78] Docker. What is a container? | docker. [Online]. Available: https://www.docker.com/resources/what-container/

[79] Unity-Technologies, "Unity-technologies/ROS-TCP-connector," original-date: 2020-09-22T20:40:02Z. [Online]. Available: https://github.com/Unity-Technologies/ROS-TCP-Connector

[80] json.org. Introducing JSON. [Online]. Available: https://www.json.org/json-en.html

[81] A. Augustin. websockets. [Online]. Available: https://websockets.readthedocs.io/en/stable/index.html

[82] Orbbec. Astra series. [Online]. Available: https://www.orbbec.com/products/structured-light-camera/astra-series/

[83] 3. install rosmaster driver library. [Online]. Available: http://www.yahboom.net/public/upload/upload-html/1689913026/3.%20Install%20Rosmaster%20driver%20library.html

[84] YahboomTechnology. Precautions for battery. [Online]. Available: https://github.com/YahboomTechnology/ROSMASTERX3/blob/main/01.About%20ROSMASTER%20X3/2.%20Precautions%20for%20battery/Precautions%20for%20battery.pdf

[85] A. Stevens, "Forward kinematics," book Title: Modeling, Motion Planning, and Control of Manipulators and Mobile Robots. [Online]. Available: https://opentextbooks.clemson.edu/wangrobotics/chapter/forward-kinematics/

[86] A. Lunia, "Inverse kinematics," book Title: Modeling, Motion Planning, and Control of Manipulators and Mobile Robots. [Online]. Available: https://opentextbooks.clemson.edu/wangrobotics/chapter/inverse-kinematics/

[87] MoveIt motion planning platform. [Online]. Available: https://picknik.ai/moveit/

[88] moveit2. MoveIt 2 documentation — MoveIt documentation: Rolling documentation. [Online]. Available: https://moveit.picknik.ai/main/index.html

[89] ros-planning/moveit2 at humble. [Online]. Available: https://github.com/ros-planning/moveit2

[90] A. Kieu. 3-dof-planar/InverseKinematics.py at master · aakieu/3-dof-planar. [Online]. Available: https://github.com/aakieu/3-dof-planar/blob/master/InverseKinematics.py

[91] KhanAcademy. Laws of sines and cosines review (article). [Online]. Available: https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:trig/x9e81a4f98389efdf:solving-general-triangles/a/laws-of-sines-and-cosines-review

[92] Kromium. arm_kinematics.py at main · ar-robotics/robot. [Online]. Available: https://github.com/ar-robotics/robot/blob/main/robot/src/controller/controller/arm_kinematics.py

[93] S. Berge, R. Eckholdt, M. Leander, S. Løver, J. Søbstad, and M. Sørensen, "An indirect method for predicting bending moments with machine learning," p. 216. [Online]. Available: https://hdl.handle.net/11250/3030483

[94] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," p. 10. [Online]. Available: https://hal.science/hal-04206682/file/Lecun2015.pdf

[95] R. Alake. Loss functions in machine learning explained. [Online]. Available: https://www.datacamp.com/tutorial/loss-function-in-machine-learning

[96] R. Ratan. Modern computer vision GPT, PyTorch, keras, OpenCV4 in 2024! [Online]. Available: https://www.udemy.com/course/modern-computer-vision/

[97] R. Alake. A data scientist's guide to gradient descent and backpropagation algorithms | NVIDIA technical blog. [Online]. Available: https://developer.nvidia.com/blog/a-data-scientists-guide-to-gradient-descent-and-backpropagation-algorithms/

[98] Softmax function. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[99] cocodataset. COCO - common objects in context. [Online]. Available: https://cocodataset.org/#home

[100] QuinnRadich. What is a machine learning model? [Online]. Available: https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model

[101] Netron. [Online]. Available: https://netron.app/

[102] goozit. How to train your object detection model using tensorflow | GOOZIT tutorial. [Online]. Available: https://www.goozit.com/tutorial/ObjectDetection

[103] J. Nelson. How to label image data for computer vision models. [Online]. Available: https://blog.roboflow.com/tips-for-how-to-label-images/

[104] AllegroAdmin1. The battle of speed vs. accuracy: Single-shot vs two-shot detection meta-architecture. [Online]. Available: https://clear.ml/blog/the-battle-of-speed-accuracy-single-shot-vs-two-shot-detection

[105] Ida. How object detectors learn. [Online]. Available: https://ambolt.io/en/how-object-detectors-learn/

[106] Github. mediapipe-loss_functions. [Online]. Available: https://github.com/google/mediapipe/blob/master/mediapipe/model_maker/python/core/utils/loss_functions.py

[107] F. Alvi. PyTorch vs TensorFlow in 2024: A comparative guide of AI frameworks. [Online]. Available: https://opencv.org/blog/pytorch-vs-tensorflow/

[108] TensorFlow lite | ML for mobile and edge devices. [Online]. Available: https://www.tensorflow.org/lite

[109] tflitemodelmaker. TensorFlow lite model maker. [Online]. Available: https://www.tensorflow.org/lite/models/modify/model_maker

[110] github. TFlite model maker installation issue with python 3.10 in colab. · issue #62942 · tensorflow/tensorflow. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/62942

[111] mediapipe. MediaPipe. [Online]. Available: https://developers.google.com/mediapipe

[112] ——. Object detection model customization guide | MediaPipe | google for developers. [Online]. Available: https://developers.google.com/mediapipe/solutions/customization/object_detector

[113] p. roboflow. Pascal VOC XML annotation format. [Online]. Available: https://roboflow.com/formats/pascal-voc-xml

[114] AWS. What is overfitting? - overfitting in machine learning explained - AWS. [Online]. Available: https://aws.amazon.com/what-is/overfitting/

[115] Roboflow. Create augmented images | roboflow docs. [Online]. Available: https://docs.roboflow.com/datasets/image-augmentation

[116] java. Primitive data types (the java™ tutorials > learning the java language > language basics). [Online]. Available: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

[117] Orbbec, "orbbec/ros_astra_camera," original-date: 2016-05-12T05:50:25Z. [Online]. Available: https://github.com/orbbec/ros_astra_camera

[118] J. Monroy. JGMonroy/ros2_astra_camera: ROS2 wrapper for astra camera. [Online]. Available: https://github.com/JGMonroy/ros2_astra_camera

[119] ros. sensor_msgs/PointCloud2 documentation. [Online]. Available: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud2.html

[120] ——. sensor_msgs/PointField documentation. [Online]. Available: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointField.html

[121] Google, "google/brotli," original-date: 2014-10-09T14:35:14Z. [Online]. Available: https://github.com/google/brotli

[122] Thingiverse.com. Knurling bolt and nut by akira3dp0. [Online]. Available: https://www.thingiverse.com/thing:1460364

[123] Kromium. ar-robotics/obj-detection-pi at offish. [Online]. Available: https://github.com/ar-robotics/Obj-detection-pi/tree/offish

[124] oracle. What is a database? [Online]. Available: https://www.oracle.com/database/what-is-database/

[125] I. Hosting. Database management systems: Organizing and accessing data. [Online]. Available: https://www.inmotionhosting.com/blog/what-is-a-database-management-system/

[126] mongodb. NoSQL vs SQL databases. [Online]. Available: https://www.mongodb.com/nosql-explained/nosql-vs-sql

[127] MongoDB. MongoDB compass | MongoDB. [Online]. Available: https://www.mongodb.com/products/tools/compass

[128] yb. 520 DC gear motor with encoder 205rpm 333rpm 550rpm. [Online]. Available: https://category.yahboom.net/products/md520

[129] yahboomarm. Yahboom 15kg serial bus smart servo and driver debugging board for robotic arm. [Online]. Available: https://category.yahboom.net/products/15kg-serial-bus-servo

[130] USB3.0 HUB expansion board 1 to 4 support 5a current 9-24v power for raspberry pi jetson. [Online]. Available: https://category.yahboom.net/products/usb-hub

[131] Yahboom. [Online]. Available: http://www.yahboom.net/study/rosmaster-x3-plus

[132] A. Tokle Poverud, M. Shah Pasand, T. G. Finnerud, H. Kåsastul, P. Knutson Sæther, and A. Senkaya, "K-spider, designdokument," p. 66. [Online]. Available: https://openarchive.usn.no/usn-xmlui/handle/11250/2396944

[133] K-AIoT. [Online]. Available: https://itfag.usn.no/grupper/D02-23/index.html

[134] M. Zelek. Raspberry pi 5 | 3d CAD model library | GrabCAD. [Online]. Available: https://grabcad.com/library/raspberry-pi-5-2/details?folder_id=13858786

[135] K. Piekutowski. LD3007ms FAN | 3d CAD model library | GrabCAD. [Online]. Available: https://grabcad.com/library/ld3007ms-fan-1

[136] N. Naranjo. Yahboom ros master x3 | 3d CAD model library | GrabCAD. [Online]. Available: https://grabcad.com/library/yahboom-ros-master-x3-1

[137] Yahboom. Robot arm 3d_model_file - google drive. [Online]. Available: https://drive.google.com/drive/folders/1rfVteXm0RxCnI9E3B7XQJjLv7kHEIvXD

[138] sw_urdf_exporter - ROS wiki. [Online]. Available: http://wiki.ros.org/sw_urdf_exporter

[139] HexTow laminate properties in HexPly® 8552 | hexcel. [Online]. Available: https://www.hexcel.com/Products/Resources/1664/hextow-laminate-properties-in-hexply-8552

[140] E. A. Team. Accuracy, precision, and recall in multi-class classification. [Online]. Available: https://www.evidentlyai.com/classification-metrics/multi-class-metrics

[141] D. Shah. Intersection over union (IoU): Definition, calculation, code. [Online]. Available: https://www.v7labs.com/blog/intersection-over-union-guide,https://www.v7labs.com/blog/intersection-over-union-guide

[142] Google. Descending into ML: Training and loss | machine learning. [Online]. Available: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[143] Cloudfare. AI inference vs. training: What is AI inference? | cloudflare. [Online]. Available: https://www.cloudflare.com/learning/ai/inference-vs-training/

[144] Google. Mediapipe-non_max_suppression_calculator. [Online]. Available: https://github.com/google/mediapipe/blob/master/mediapipe/calculators/util/non_max_suppression_calculator.cc

[145] Kromium. Code search results. [Online]. Available: https://github.com/search?q=repo%3Aar-robotics%2Frobot+path%3A*.py+path%3A*test%2F*&type=code

[146] A. E. Haugjord and H. Bertelsen, "Tensile test report of polylactic acid (PLA)."

[147] A. E. Haugjord, "520 DC motor bracket weight simulattion," p. 11.

[148] H. Bertelsen, "Simulation of arm cam holder fixed angle."

[149] ——, "Simulation of wall attachment bracket assy."

[150] MoveIt. Hybrid planning — MoveIt documentation: Rolling documentation. [Online]. Available: https://moveit.picknik.ai/main/doc/concepts/hybrid_planning/hybrid_planning.html

[151] ——. URDF and SRDF — MoveIt documentation: Rolling documentation. [Online]. Available: https://moveit.picknik.ai/main/doc/examples/urdf_srdf/urdf_srdf_tutorial.html#urdf

[152] encord. Object tracking definition | encord. [Online]. Available: https://encord.com/glossary/object-tracking-definition/

[153] Deci. Object tracking with DeepSORT and YOLO-NAS. [Online]. Available: https://deci.ai/blog/object-tracking-with-deepsort-and-yolo-nas-practitioners-guide/

[154] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition." [Online]. Available: http://arxiv.org/abs/1505.00880

[155] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," vol. 6, no. 1, p. 11. [Online]. Available: https://doi.org/10.1186/s40649-019-0069-y

[156] G. Inc. scann: Scalable approximate nearest neighbor search library. [Online]. Available: https://github.com/google-research/google-research/tree/master/scann

[157] PyTorch. Home. [Online]. Available: https://pytorch.org/mobile/home/

[158] YOLO: Real-time object detection. [Online]. Available: https://pjreddie.com/darknet/yolo/

[159] Ultralytics. Frequently asked questions (FAQ). [Online]. Available: https://docs.ultralytics.com/help/FAQ

[160] R. P. Ltd. Raspberry pi for home. [Online]. Available: https://www.raspberrypi.com/for-home/

[161] Ultralytics. yolov5/utils/general.py. [Online]. Available: https://github.com/ultralytics/yolov5/blob/21f8f94d1169bd28c5d59fe6ccc23b13ddb997e2/utils/general.py

[162] K. Jakhar, "karanjakhar/yolov5-export-to-cpu," original-date: 2021-07-07T04:41:02Z. [Online]. Available: https://github.com/karanjakhar/yolov5-export-to-cpu

[163] ros2, "ros2/ros1_bridge," original-date: 2015-06-17T18:36:47Z. [Online]. Available: https://github.com/ros2/ros1_bridge

[164] isl org, "isl-org/open3d," original-date: 2016-12-02T16:40:38Z. [Online]. Available: https://github.com/isl-org/Open3D

[165] Tensorflow. tf.lite.interpreter | TensorFlow v2.15.0.post1. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/lite/Interpreter

[166] O. Robotics. Quality of service settings — ROS 2 documentation: Humble documentation. [Online]. Available: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Quality-of-Service-Settings.html

[167] pypa. pip documentation v24.0. [Online]. Available: https://pip.pypa.io/en/stable/

[168] Yahboom. ROS robot expansion board. [Online]. Available: http://www.yahboom.net/study/ROS-Driver-Board

[169] A. Rosebrock. OpenCV getting and setting pixels. [Online]. Available: https://pyimagesearch.com/2021/01/20/opencv-getting-and-setting-pixels/

[170] Reddit. install ros2 on debian 12 and ubuntu 23. Post URL: www.reddit.com/r/ROS/comments/14axhdt/install_ros2_on_debian_12_and_ubuntu_23/. [Online]. Available: www.reddit.com/r/ROS/comments/14axhdt/install_ros2_on_debian_12_and_ubuntu_23/kaed05m/

[171] Set up hand tracking | oculus developers. [Online]. Available: https://developer.oculus.com/documentation/unity/unity-handtracking/

A User stories

ID	US-1
Title:	Robot movement
User story:	**As an** operator **I want** control the robot (movement drive mode) **So that** I can inspect the environment and perform operations
Acceptance criteria:	**Given** that the operator is wearing a VR headset and Kromium software is running **When** the operator makes hand gestures **Then** the robot will act accordingly

ID	US-2
Title:	Control change
User story:	**As an** operator **I want** to change the robot's mode **So that** I can perform an operation
Acceptance criteria:	**Given** that the camera and controls are connected to steering the robot **When** changing operational control **Then** I can see the camera feed from the arm and control the arm

ID	US-3
Title:	Camera feed
User story:	**As an** operator **I want** to see the camera feed **So that** I can drive and control the robot
Acceptance criteria:	**Given** the cameras are connected and the robot is turned on **When** I wear the VR headset **Then** I see the camera feed

ID	US-4
Title:	Control the robot's arm
User story:	**As an** operator **I want** to control the robot arm **So that** I can perform “tasks with the arm”
Acceptance criteria:	**Given** that the system is in “arm mode” **When** I make signature hand movements **Then** the robot arm performs the desired action

ID	US-5
Title:	Robot's status
User story:	**As an** operator **I want** to see the robot's status **So that** I'm updated about the robot's status such as battery percentage, connection something
Acceptance criteria:	**Given** the robot is in operation **When** I see the live feed **Then** I can estimate time for battery change and maintenance

ID	US-6
Title:	Spectator view
User story:	**As a** spectator **I want** to see the live video feed from the operator's perspective **So that** I can follow along what is happening
Acceptance criteria:	**Given** that there is an external monitor connected (that the VR headset is connected to external monitor) **When** there is a live feed sent to the VR headset (I press "stream" from the Meta Quest app and choosing the monitor device) **Then** the monitor displays the same live feed

ID	US-7
Title:	Battery change
User story:	**As a** technician **I want** a seamless and quick battery replacement process **So that** I can minimize downtime and ensure the robot is always powered
Acceptance criteria:	**Given** robot's low power status and the availability of a charged battery **When** I press the "get out battery" button and swap the batteries **Then** the robot should have full power and can resume its operation

ID	US-8
Title:	Recognize objects
User story:	**As an** operator **I want** the robot to recognize objects **So that** I have more information
Acceptance criteria:	**Given** the robot is on **When** it recognizes an object **Then** I should see information about the object

ID	US-9
Title:	Emergency stop
User story:	**As an** operator **I want** the robot to have an emergency stop feature **So that** I can prevent an accident
Acceptance criteria:	**Given** the robot is on **When** I interact with the emergency stop interface **Then** the robot should stand still and ignore incoming commands

B Use cases

ID	UC-1.1
Use case	The operator wants to move the robot in a given direction
Description	If the operator signals the robot to move using their hands which gets picked up by the VR headset, the robot should move in the direction which were intended.
Actor	Operator, robot

ID	UC-2.1
Use case	The operator wants to change the robot's mode
Description	There should be an interface for the operator to change the robot’s mode (idle, drive, arm, emergency). When interacted with, the robot should change mode.
Actor	Operator, robot

ID	UC-3.1
Use case	The operator wants to change the robot’s camera feed independent of mode
Description	The operator should be able to change the robot’s camera view and see this, no matter what mode the robot is currently in. This could be used in order to get a better perspective of an operation or object.
Actor	Operator, robot

ID	UC-4.1
Use case	The operator wants the robot arm to imitate their hand gestures picked up by the VR headset
Description	When the operator moves their hand, the VR headset should pick this up and the robotic arm on the robot should be able to perform a movement which has been translated from the operator’s movement.

ID	UC-4.2
Use case	The operator wants to be able to pick up an object with the robot's arm using hand imitation
Description	If the operator imitates picking up an imaginary object with their hands, the robotic arm on the robot should be able to mimic this movement to pick up a physical object which fits the constraints.
Actor	Operator, robot

ID	UC-4.3
Use case	The operator wants to be able to rotate an object using hand imitation
Description	If the operator imitates rotating an imaginary object with their hands, the robotic arm on the robot should be able to mimic this movement to rotate a physical object which fits the constraints.
Actor	Operator, robot

ID	UC-4.4
Use case	The operator wants to be able to press an external button using hand imitation
Description	If the operator imitates pressing an imaginary button with their hands, the robotic arm on the robot should be able to mimic this movement to press a physical button which fits the constraints.
Actor	Operator, robot

ID	UC-5.1
Use case	The operator wants to see the live status of the robot
Description	The operator should be able to view the live status of the robot when wearing the VR headset.
Actor	Operator, robot

ID	UC-6.1
Use case	The spectator wants to follow along the live feed from the VR headset
Description	A spectator which does not wear the VR headset and watches an external monitor should be able to follow along and see the same live feed which the operator wearing the VR headset sees.
Actor	Spectator

ID	UC-7.1
Use case	The technician wants to easily change batteries
Description	If a technician wants to change the batteries of the robot, it should be a quick and easy process.
Actor	Technician, robot

ID	UC-7.2
Use case	The operator wants the robot to change its own battery
Description	When the operator interacts with some interface in the VR headset, the robot should be able to use its own robotic arm to change its own battery without external interaction.
Actor	Operator, robot

ID	UC-7.3
Use case	The operator wants to manually change the batteries using the robotic arm
Description	The operator should be able to remotely control the robotic arm such that the robot can change its own battery without external interaction.
Actor	Operator, robot

ID	UC-8.1
Use case	The operator wants the robot to recognize objects
Description	If the robot recognizes a physical object using its cameras, the operator should be able to see this recognition.
Actor	Operator, robot

ID	UC-9.1
Use case	The operator wants to stop the entire operation in case of an emergency
Description	There should be an interface for triggering an emergency stop. If emergency stop is triggered, the robot should enter emergency mode. In emergency mode the robot should physically not move and ignore almost all commands. This mode requires an exit-emergency-stop-command to go out of emergency mode to operate normally again.
Actor	Operator, robot

C System requirement specification

ID	User story	ID	Use case	ID	Requirement	Priority	Technical performance parameters	Test method	Verification	Test ID	Status
US-1	Robot movement	**UC-1.1**	The operator wants to move the robot in a given direction	**R-1.1.1**	The VR headset should capture the operator's hand gesture.	**A**	Constraints: - Cannot control the arm while its driving	Make hand gesture for the VR and wait for confirmation	A sign should appear in the VR headset and show that a hand gesture has been captured.	**T-1**	**Success**
				R-1.1.2	The Kromium software must be able to convert the hand gesture to a robot command.	**A**	Requirement driving commands: - Left - Right - Forward - Backward	Compare the hand gesture made by the operator and the command result	A sign should appear in the VR headset showing the command result which matches the hand gesture made by the operator	**T-2**	**Success**
				R-1.1.3	The converted command should be sent to the robot.	**A**	Not applicable	The command is printed out by the VR (sender) and the robot (receiver)	The printed command from both sides should be equal	**T-1, T-2**	**Success**
				R-1.1.4	The robot should act on the sent command.	**A**	Operating area: - Be able to navigate within a minimum of one square meter Latency: - Maximum 2 seconds latency from the operator does a movement till the robot arm responds	Send a specific command to the robot and observe the action	The action from the robot matches the intended command	**T-2, T-3**	**Success**
US-2	Control change	**UC-2.1**	The operator wants to change the robot's mode	**R-2.1.1**	The software should provide an interface for switching modes	**A**	Constraints: - Cannot drive while its controlling the arm - Modes: - Driving - Arm - Idle	Change mode in the VR application through the menu or voice command	Verify that the robot indicates change to the corresponding mode	**T-3, T-4, T5**	**Success**
				R-2.1.2	The software should change the camera view according to the given mode automatically	**A**	Not applicable	Switch mode	Verify that the camera view now matches the current mode	**T-5**	**Success**
				R-2.1.3	The VR headset should change mode for interpreting hand movements	**A**	Using either or both: - Hand interactable menu - Voice command	Switch mode to arm or drive mode and make hand movements	The hand movements is converted to drive and arm commands	**T-6**	**Success**
US-3	Camera feed	**UC-3.1**	The operator wants to change the robot's camera feed independent of mode	**R-3.1.1**	The software should provide an interface to change camera feed	**C**	Not applicable	Switch camera feed by changing mode from arm mode to drive mode	Visually verify that the camera feed is changed correctly	**Not viable**	**Not vaiable**
US-4	Control the robot's arm	**UC-4.1**	The operator wants the robot arm to imitate their hand gestures picked up by the VR headset	**R-4.1.1**	The Kromium software should convert hand movement into positional data .	**A**	Constraints: - Positional data in form of the cartesian coordinate system	Move hands around within the VR's hand detection boundaries and print out the positional data	Verify that the printed data corresponds to the positional data datatype	**T-7**	**Success**
				R-4.1.2	The robot arm should move according to the given positional data.	**A**	Latency: - Maximum 2 seconds latency from the operator does a movement till the robot arm responds	Send some predefined positional data examples to the robot	Verify that the robotic arm moves in the expected predefined position	**T-7, T-8**	**Success**

		UC-4.2	The operator wants to be able to pick up an object with the robot's arm using hand imitation	**R-4.2.1**	The robot arm should have a pinching mechanism.	**A**	Constraints: - Maximum object width: 5 cm - Minimum object width 1 cm - Maximum object weight: 200g	Visual inspection and send a pinch command while holding an object with fits the constraints	Verify that the robotic arm can hold and move the object and does not break or lose the object	**T-14**	**Success**
		UC-4.3	The operator wants to be able to rotate an object using hand imitation	**R-4.3.1**	The robot arm should have a wrist tool enabling the robot to rotate an object.	**C**	Constraints: - Maximum object width: 5 cm - Minimum object width 1 cm	Visual inspection and send a rotate command for the arm to rotate an object which fits the constraints	Verify that the robotic arm can rotate the object and does not break	**T-14**	**Success**
		UC-4.4	The operator wants to be able to press an external button using hand imitation	**R-4.4.1**	The robot arm should be designed such that it can press a button.	**B**	Not applicable	Visual inspection and imitate pressing a button	Verify that the robotic the button gets pressed	**Not viable**	**Not vaiable**
US-5	Robot's status	**UC-5.1**	The operator wants to see the live status of the robot	**R-5.1.1**	The software should display the battery level of the robot.	**B**	Constraints: - Format: percentage	Visual inspection of the VR's GUI	Verify that there is a battery level indicator	**T-9**	**Success**
				R-5.1.2	The software should display the speed of the robot.	**C**	Constraints: - Format: percentage of maximum speed	Visual inspection of the VR's GUI	Verify that there is a speed indicator	**T-9**	**Success**
				R-5.1.3	The software should display connection information between the robot and Kromium's software system.	**B**	Constraints: - Latency in milliseconds - Connection in boolean	Visual inspection of the VR's GUI	Verify that there is a connection status indicator	**T-9**	**Success**
				R-5.1.4	The software should display the mode the robot is in.	**A**	Modes: - Idle - Drive - Arm - Emergency	Visual inspection of the VR's GUI	Verify that there is a mode indicator and it shows the correct mode	**T-9**	**Success**
US-6	Spectator view	**UC-6.1**	The spectator wants to follow along the live feed from the VR headset	**R-6.1.1**	The system should display a live feed from the operator's perspective on an external monitor.	**A**	Constraints: - The monitor should be connected to the network. - perspective: - Camera view from the robot - Operators pass-through - All the GUI elements: - statuses	Connect an external monitor, and have two developers. One of the developers should observe the monitor, while the other wears the VR headset. They should verbally explain what they see at the current time.	Verify that the visual elements on the VR and external monitor match	**T-15**	**Success**
US-7	Battery change	**UC-7.1**	The technician wants to easily change batteries	**R-7.1.1**	The robot's design should allow for easy access to the battery interface.	**A**	Constraints: - Easy: - less than 5 operations - Total time less than 2 minutes	A developer manually changes the battery	Verify that the process matches the constraints	**T-16**	**Failed**
		UC-7.2	The operator wants the robot to change its own battery	**R-7.2.1**	The robot should have access to a power source while changing batteries.	**C**	Constraints: - the arm must be able to carry the battery - affordable batteries (update this) - access to charged batteries	Observe the autonomous process of changing the battery	Verify that the robot does not shut down while changing its own battery	**Not viable**	**Not viable**
				R-7.2.2	The robot should have a software implementation for changing the batteries autonomously.	**C**	Constraints: - When the robot's batter goes below 10%.	Observe the autonomous process of changing the battery	Visually verify that the new battery is correctly inserted and observe that the battery level indicator updates	**Not viable**	**Not viable**

		UC-7.3	The operator wants to manually change the batteries using the robotic arm	**R-7.3.1**	The operator should be able to change the battery using the robotic arm with hand imitation.	**C**	Not applicable	Observe the process of changing the battery	Visually verify that the new battery is correctly inserted and observe that the battery level indicator updates	**Not viable**	**Not viable**
US-8	Recognize objects	**UC-8.1**	The operator wants the robot to recognize objects	**R-8.1.1**	The robot should have functionality for visually recognizing objects.	**A**	Constraints: - Predefined object classes - Accuracy over 60% - Using machine learning	Placing an object in front of the robot's camera	Verify that there is a border around the object and the description corresponds to the object	**T-10**	**Success**
				R-8.1.2	The robot should have functionality for displaying data related to recognized objects.	**A**	Constraints: - Label - boundary box - Object-specific information - Accuracy	Placing an object in front of the robot's camera	Verify that the data displayed matches the object's data	**T-13**	**Success**
US-9	Emergency stop	**UC-9.1**	The operator wants to stop the entire operation locally and remotely in case of an emergency	**R-9.1.1**	The robot should have functionality for stopping immediately.	**A**	Constraints: - independent of setting	Activate the emergency stop and send commands to the robot	Visually verify that the robot stops and does not respond to commands afterwards	**T-2, T-11**	**Success**
				R-9.1.2	The robot should have an interface for remote emergency activation.	**A**	Constraints: - Requires less than two operations	Activate the emergency stop through the VR headset and send commands to the robot	Visually verify that the robot stops and does not respond to commands afterwards	**T-11**	**Success**
				R-9.1.3	The robot should have an interface for physical emergency activation.	**A**	Constraints: - Requires less than two operations	Physically activate the emergency stop and send commands to the robot	Visually verify that the robot stops and does not respond to commands afterwards	**Not viable**	**Not viable**
				R-9.1.4	The robot should log important information.	**A**	Important information: - Commands received and sent - Current mode the robot is in	Log data from all the relevant components	Verify that the log data is saved to the file it was indented to	**T-12**	**Success**

D Global requirements

Global requirements

Description	Requirement	Requirement ID	Priority	Performance
Robot operational time	The robot should have at least 20 minutes of uptime.	R-G.1	A	-
Component access	The robot should be modular and interchangeable. We can change one component of the robot without it affect the rest of the system.	R-G.2	A	-
Software architecture	Modular design for adding new functionality on top	R-G.3	B	have to make minimal adjustments to current code in order to expand with additional functionality
Physical appearance	The robot should have a physical design such that it looks presentable for demonstrations.	R-G.4	A	-
Budget	The project has been given ~25,000 NOK budget.	R – G.5	*Not applicable*	-

E Software testing documentation

Test ID	T-1
Requirements tested	**R-1.1.1, R-1.1.3**
Description	Testing hand gesture performed by the operator and sent to ROS master node.
Steps	1. Both applications running(ROS master and VR application) 2. System mode has to be in arm mode 3. The operator makes a hand gesture
Verification	1. The application indicates hand gesture captured 2. ROS master prints out the result Validate that the ROS master node prints data
Performed by	SO, OM
Related components	Hand gesture controller (VR), ROS VR-linker node (Robot), ROS master node (Robot)
Status	Success

Test ID	T-2
Requirements tested	**R-1.1.2, R-1.1.3, R-1.1.4, R-1.1.9**
Description	Converting hand gestures to a robot command. **Requirement driving commands:** 1. Left 2. Right 3. Forward 4. Backward
Steps	1. Both applications running(ROS master and VR application) 2. System mode in drive mode 3. The operator makes a hand gesture within the robot control boundary signalizing all four driving commands
Verification	1. The ROS master receives a JSON command 2. Validate that the JSON data is related to the hand gesture 3. ROS master node converts the data into a direction 4. The hand motion matches the printed command from ROS master node
Performed by	SO, OM
Related components	Hand gesture controller (VR), ROS VR-linker node (Robot), ROS master node (Robot)
Status	Success

Test ID	T-3
Requirements tested	**R-1.1.4, R-2.1.1**
Description	Sending specific commands to the robot and observe that the robot does accordingly
Steps	1. Both applications running(ROS master and VR application) 2. Set the robot and VR application to drive mode 3. Send following commands 1. LEFT, RIGHT, FORWARD, STOP
Verification	1. Wait for the robot to indicate it is in drive mode 2. Verify that the robot acts correctly to the sent command
Performed by	SO, OM
Related components	Hand gesture controller (VR), ROS VR-linker node (Robot), ROS master node (Robot), ROS controller node (Robot, Expansion board)
Status	Success

Test ID	T-4
Requirements tested	**R-2.1.1**

Description	Switch the robot modes from the VR application.
	• **Modes:**
	  ○ Drive
	  ○ Arm
	  ○ Idle
	  ○ Emergency

Steps	1. Both applications running(ROS master and VR application) 2. Click on the mode in the VR menu
Verification	1. Verify that the robot beeps (signal for drive mode) and prints out the correct mode
Performed by	SO, OM
Related components	Hand gesture controller (VR), ROS VR-linker node (Robot), ROS master node (Robot)
Status	Success

Test ID	T-5
Requirements tested	**R-2.1.2, R-2.1.1**
Description	Switch to arm or drive mode and make sure the camera feed in the VR changes.
Steps	1. Both applications running(ROS master and VR application) 2. If the robot is in drive mode, change to arm mode, if in arm mode change to drive mode
Verification	1. Verify that the robot beeps 2. Verify that the VR feed changes to the correct camera
Performed by	None
Related components	**VR** • Hand gesture controller **Robot** • ROS VR-linker node • ROS master node • ROS camera node
Status	Success

Test ID	T-6
Requirements tested	**R-2.1.3**
Description	Switch mode to drive or arm mode and make hand movements to control the robot.
Steps	1. Change mode to drive mode 2. Make hand movements for controlling the robot 3. Change mode to arm mode 4. Make hand movements for controlling the robot
Verification	1. Verify that the robot drives correctly in drive mode 2. Verify that the arm moves correctly in arm mode
Performed by	SO, OM
Related components	**VR** • Hand gesture controller **Robot** • ROS VR-linker node • ROS master node • ROS controller node
Status	Success

Test ID	T-7
Requirements tested	**R-4.1.1, R-4.1.2**
Description	Converting hand movements to positional data to control the robot's arm.
Steps	1. Switch mode to arm mode 2. Make hand movements in in the "arm boundary" inside the VR application
Verification	1. Verify that the positional data visually 2. Visually verify that the robot arm moves according to the hand movements
Performed by	SO, OM
Related components	**VR** • Hand gesture controller **Robot** • ROS VR-linker node • ROS master node • ROS controller node
Status	Success
Comment	Tested following: 1. Pinching mechanism - Complete 2. Tilting the "wrist" - Complete 3. Tilting the "shoulder" - Complete 4. Rotating the arm - Complete

Test ID	T-8
Requirements tested	**R-4.1.2**
Description	Testing the latency from an operator performs a movement until the robot arm responds. Test **T-7** has to be complete before starting this test. **Required performance(Latency):** • Maximum 2 seconds
Steps	1. Follow the steps in **T-7** 2. Manually take the time when the operator makes a hand movement 3. Stop the time when the robot responds
Verification	1. Verfy that the time measured is below 2 seconds
Performed by	SO, OM
Related components	**VR** • Hand gesture controller **Robot** • ROS VR-linker node • ROS master node • ROS controller node
Status	Success

Test ID	T-9
Requirements tested	**R-5.1.1, R-5.1.2, R-5.1.3, R-5.1.4**
Description	Testing and verification to display robot status 1. Mode 2. Speed 3. Battery level (percentage) 4. Latency
Steps	1. Change mode to an arbitrary mode
Verification	1. Verify that the VR headset shows the correct mode after changing the mode
Performed by	SO, OM
Related components	**VR** • Mode controller **Robot** • ROS VR-linker node • ROS master node • ROS controller node
Status	Success

,

Test ID	T-10
Requirements tested	**R-8.1.1**
Description	Testing of machine learning algorithms to recognize objects. **Constraints:** • Predefined object classes • Accuracy over 60% • Using machine learning
Steps	1. Make sure you have a camera connected to the computer 2. Start the object detection application 3. Place a predefined object in front of the camera 4. Observe the result displayed on the screen
Verification	1. Verify that the algorithm has a bounding box around the object 2. Verify that the label of the recognized object is correct 3. Verify that the accuracy is according to the minimum requirement
Performed by	AD
Related components	Object detection
Status	Success

Comment	Tested the following
	
	• Custom model with MediaPipe
	• Objects:
	  ○ Mentioned in the COCO dataset (Ref: Github/object-detection)
	  ○ Custom dataset of object class “People” and “bolt”
	• Success rate
	  ○ Custom: over 70%

Test ID	T-11
Requirements tested	**R.9.1.1, R.9.1.2**
Description	Testing of the emergency mode activation from the VR application.
Steps	1. Switch mode to emergency 2. Make other hand movements or change mode to another mode
Verification	1. Verify that the reboot stops immediately 2. Verify that the robot does not respond to any new commands after the emergency mode is activated
Performed by	SO, OM
Related components	**VR** • Mode controller • Emergency controller **Robot** • ROS VR-linker node • ROS master node
Status	Success

Test ID	T-12
Requirements tested	**R-9.1.4**
Description	Testing of log functionality in the system.
Steps	1. Both applications running 2. Open the log file
Verification	1. Verify that the logdata is written to [TIME].log file. 2. Verify that the [TIME] corresponds the UNIX integer time datatype. 1. For example: 170951600.log where the time(170951600) is the time when you started the application 3. Verify that the data logged is written to the file
Performed by	OM
Related components	**Robot** • ROS VR-linker node • ROS logger node
Status	Success

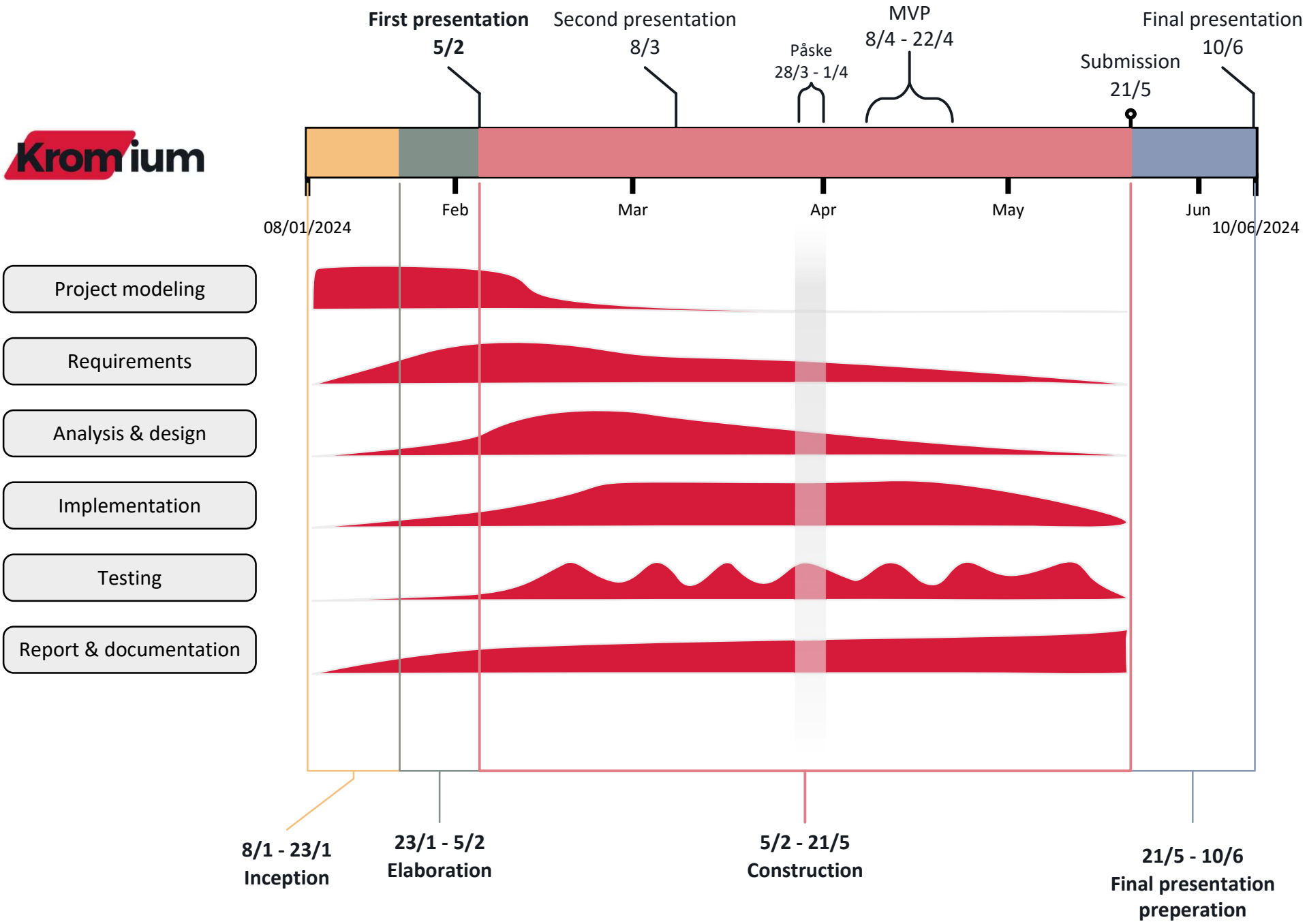
Test ID	T-13
Requirements tested	**R-8.1.2**
Description	Testing the robot's functionality for displaying data related to recognized objects. **Constraints:** 1. Label 2. Boundary box 3. Object-specific information 4. Accuracy
Steps	1. Make sure you have a camera connected to the computer 2. Make sure the database is running 3. Start the object detection application 4. Place a predefined object in front of the camera 5. Observe the result displayed on the screen
Verification	1. Verify that the frame shows the label of the object recognized 2. Verify that the frame draws a boundary box around the object 3. Verify that the data(object-specific information displayed matches the object's data 4. Verify that the frame shows the accuracy of the prediction, in percentage
Performed by	AD
Related components	Object detection Camera
Status	Success
Test ID	T-14
Requirements tested	**R-4.2.1, R-4.3.1**

Description	Testing rotating an object
Steps	1. Switch to arm mode 2. Move the arm and pinch an object 3. Say “screw” to the VR headset
Verification	1. Verify that the VR application activating the screw mechanism 2. Observe and verify that the arm pinches and rotate the object
Performed by	SO, OM
Related components	Voice command, robot arm
Status	Success

Test ID	T-15
Requirements tested	**R-6.1.1**
Description	Displaying live video to an external monitor
Steps	1. Connect the VR application 2. Make sure the VR headset is connected to the internet 3. On the external screen open oculus casting 4. Connect the headset
Verification	1. Observe and verify the stream on the external monitor is live
Performed by	SO, OM
Related components	VR application
Status	Success

Test ID	T-16
Requirements tested	**R-7.1.1**
Description	Changing battery manually Constraints: • Easy: • less than 5 operations • Total time less than 2 minutes
Steps	• Turn off the robot • Change battery
Verification	1. verify that the process and compare to the constraints
Performed by	SO, OM
Related components	Robot, Battery
Status	Success for total time less than 2 minutes, failed more than 5 operations

F Estimated project timeline


Kromium
First presentation
5/2
Second presentation
8/3
Påske
28/3 - 1/4
MVP
8/4 - 22/4
Submission
21/5
Final presentation
10/6
08/01/2024
Feb
Mar
Apr
May
Jun
10/06/2024
Project modeling
Requirements
Analysis & design
Implementation
Testing
Report & documentation
8/1 - 23/1
Inception
23/1 - 5/2
Elaboration
5/2 - 21/5
Construction
21/5 - 10/6
Final presentation
preperation

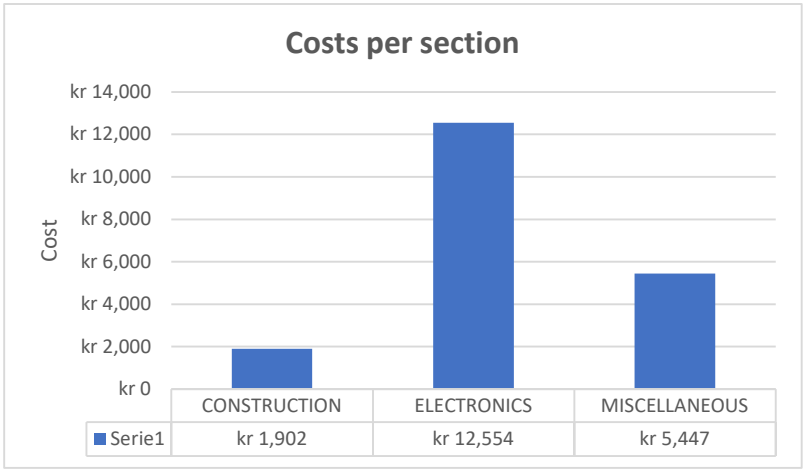
G Budget

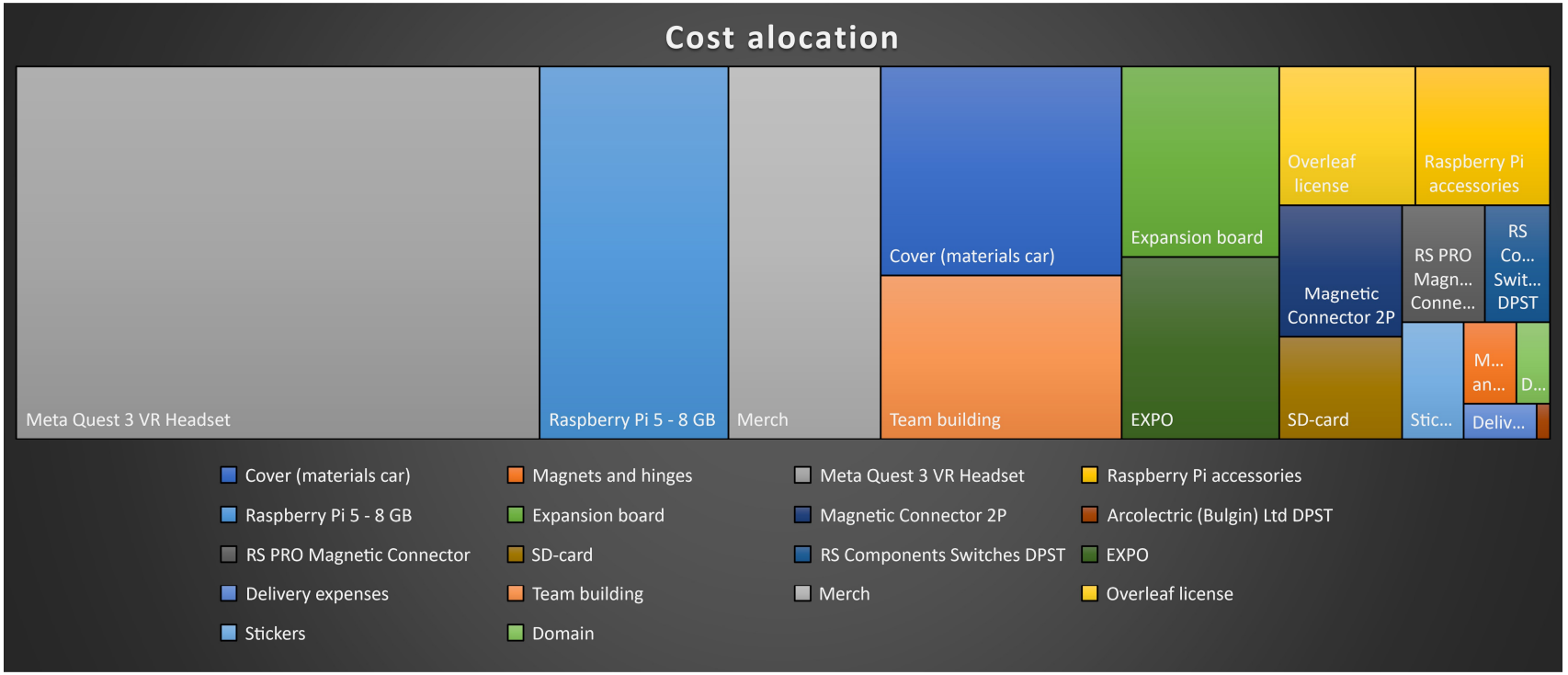
Component	Description	Supplier	URL	Price (NOK)	Amount	Cost	Status
CONSTRUCTION							
Cover (materials car)				1,752	1	1,752	Delivered
Magnets and hinges		Clas Ohlson		150	1	150	Delivered
SUM						**kr 1,902**	
ELECTRONICS							
Meta Quest 3 VR Headset		Elkjøp		6,795	1	6,795	Delivered
Raspberry Pi accessories	Micro SD Card, Power supply, case	Raspberrydk		652	1	652	Delivered
Raspberry Pi 5 - 8 GB	Raspberry Pi 5 - 8 GB	Raspberrydk		1,224	2	2,448	Delivered
Expansion board	Control board			1,042.38	1	1,042	Delivered
Magnetic Connector 2P	5A - Panel mount	RS online Norway	https://no.rs-onli	281.05	2	562	Delivered
Arcolectric (Bulgin) Ltd DPST	On-Off Rocker Switch Panel Mount	RS online Norway	https://no.rs-onli	16.09	1	16	Delivered
RS PRO Magnetic Connector		RS online Norway	https://no.rs-onli	336.74	1	337	Delivered
SD-card				219.00	2	438	
RS Components Switches DPST	On-off rocker switch	RS Components AS		263.90	1	264	Delivered
SUM						**kr 12,554**	
MISCELLANEOUS							
EXPO				900	1	1,000	Budgeted
Delivery expenses	Customs			90	1	90	Delivered
Team building				1,378	1	1,378	Delivered
Merch				395	5	1,975	Delivered
Overleaf license				659	1	659	Delivered
Stickers	Kromium & KM stickers			250	1	250	Delivered
Domain	Kromium & KM stickers			95	1	95	
SUM						**kr 5,447**	

UPDATED: 20 May 2024

Summary	
All sections	**Cost**
CONSTRUCTION	kr 1,902
ELECTRONICS	kr 12,554
MISCELLANEOUS	kr 5,447
Sum TOTAL	**kr 19,903**
Balance left	**kr 5,097**

Budget
NOK 25,000





H Rapid risk ranking

Consequence severity		Rev.3	Updated: 2024.03.13			
Level	**Description**	**Definition**				
		Developers	**Environment**	**Kongsberg Maritime**	**Material values**	**People**
1	Catastrophic	Loss of multiple group members. Loss of all work.	The robot burns to the ground. (Toxic gasses & CO2)	Loss of all documentation and parts.	Total loss of robot and headset. Project room burns down.	Fatalities. Permanent disability.
2	Severe	Loss of one group member. Loss of most work. Section of robot burn.	Large part of the robot burns. (Toxic gasses & CO2)	Loss of most documentation and parts.	Loss of main part of robot or headset. Considerable structural damage.	Prolonged hospital treatment.
3	Major	Loss of section of work. Multiple group members gets reduced work capacity. Multiple electronics gets fried.	Section of robot burn. (Toxic gasses & CO2)	Loss of some documentation and parts.	Minor structural damage. Section of robot burn.	Medical treatment.
4	Moderate	The group looses up to one half day of work. One group member gets reduced work capacity. One component gets fried.	Multiple electronics burns. (Toxic gasses & CO2)	Loss of some documentation or parts.	Larger scratch or dent. Multiple electronics gets fried.	Injury.
5	Minor	Reduced component functionality. Loss of some work.	Multiple electronics gets fried. (CO2)	Loss of minor documentation or parts.	One component gets fried.	Minor injury.
6	Minimal	Annoyance. Disturbance.	Gas release from fried component. (CO2)	Loss of minimal documentation or parts.	Scratch or dent.	Annoyance. Disturbance.

Figure H.1: Consequence severity

Probability levels	Rev.2	Updated:2024.02.07
Level	**Description**	**Definitions**
A	IMPROBABLE	Possible, but highly unlikely.
B	REMOTE	Unlikely to occur during project.
C	OCCASIONAL	May occur several times during development or demonstration.
D	PROBABLY	Will occur several times during development or demonstration.
E	FREQUENT	Will occur frequently during development or demonstration.

Figure H.2: Probability levels

Risk matrix		Rev.4	Updated:2024.02.07			
		PROBABILITY (per month)				
		A Improbable	B Remote	C Occasional	D Probable	E Frequent
SEVERITY	1 (Catastrophic)	H	H	H	H	H
	2 (Severe)	H	H	H	H	H
	3 (Major)	M	M	H	H	H
	4 (Moderate)	L	L	M	M	H
	5 (Minor)	L	L	L	M	H
	6 (Minimal)	L	L	L	L	M

Figure H.3: Risk Matrix

Risk levels	Rev.1	Updated:2024.01.24
Level	**Level name**	**Description**
H	High	High risk, not acceptable. Further analysis should be performed to give a better estimate of the risk. If this analysis still shows unacceptable or medium risk redesign or other changes should be introduced to reduce the criticality.
M	Medium	The risk may be acceptable but redesign or other changes should be considered if reasonably practical. Further analysis should be performed to give a better estimate of the risk. When assessing the need of remedial actions, the number of events of this risk level should be taken in to consideration.
L	Low	The risk is low and further risk reducing measures are not necessary.

Figure H.4: Risk levels

Hazardous event allocated to stakeholders	Rev.2	Updated: 2024.03.13		
Hazardous events				
Developers	**Environment**	**Kongsberg Maritime (KM)**	**Material values (MV)**	**People**
Robot arm hits car	Battery catches fire	Battery catches fire	Robot arm hits car	Battery catches fire
Battery catches fire	Fire	Break in/burglary	Battery catches fire	Crash between robot and bystander
Break in/burglary	*	Drops headset	Break in/burglary	Drops headset
Component delivery time	*	Fire	Crash between robot and bystander	Electric shock
Crash between robot and bystander	*	Loss of developers/group participants	Crash between robot and wall/ object	Fire
Crash between robot and wall/ object	*	Loss of progress	Drops headset	Motion sickness
Bad connection headset	*	Meta quest software problems	Fire	*
Bad connection robot	*	No/low battery power	Fried circuit	*
Drops headset	*	Out of budget	Short circuit	*
Electric shock	*	Drops headset	Static shock	*
Fire	*	Theft	Theft	*
Fried circuit	*	*	*	*
Loose electrical connectors	*	*	*	*
Loss of developers/group participants	*	*	*	*
Loss of progress	*	*	*	*
No connection to robot during demonstration	*	*	*	*
Meta quest software problems	*	*	*	*
Motion sickness	*	*	*	*
No/low battery power	*	*	*	*
Out of budget	*	*	*	*
Production time	*	*	*	*
Short circuit	*	*	*	*
Theft	*	*	*	*

Figure H.5: Hazardous events allocated to stakeholders

Possible hazards and description		Rev.2	Updated: 2024.03.13
ID.	**Hazard**	**Description**	
1	Robot am hits car	The robots arm could hit camera, wire, etc. when operated due to narrow view	
2	Battery catches fire	The battery powering the robot could catch fire	
3	Break in/burglary	The project room containing all components could get broken into.	
4	Component delivery time	When ordering components or material for production	
5	Crash between robot and bystander	The robot could hit someone during testing and demonstration or someone could hit the robot if they don't pay attention	
6	Crash between robot and wall/ object	The robot could hit the environment around it due to bad or slow connection, or due to lack of control from operator.	
7	Bad connection headset	Headset doesn't get camera feed or doesn't pick up operator movements	
8	Bad connection robot	Robot car or arm does not respond	
9	Drops headset	It's possible to drop the headset due to bumping into it, during handover or when placing it back after use.	
10	Electric shock	Bad isolation in the robot could shock someone during modifications and building.	
11	Fire	The robot catches fire externally or from internal component. This will effect the robot and the building around it.	
12	Fried circuit	One or more of the circuits on the robot gets destroyed from overloading.	
13	Loose electrical connectors	Bad connection between ports because of poorly connected cable.	
14	Loss of developers/group participants.	People could fall off the project	
15	Loss of work progress	Documentation, code, technical drawings, models, etc.	
16	No connection to robot during demonstration	We can't get a stable connection between the headset and robot.	
17	Meta quest software problems	Problems can arise between our own program and the internal software of the VR headset.	
18	Motion sickness	Operator can get motion sickness from headset	
19	No/low battery power	Forgot to charge batteries	
20	Out of budget	The team could run out of money	
21	Production time	It's a possibility that production time exceeds necessary completion date.	
22	Short circuit	We could damage electronics by giving of a static shock.	
23	Static shock	From equipment malfunction	
24	Theft	Someone could steal our equipment.	

Figure H.6: Possible hazards and description

Risk main form - Part 1/2		Rev. 3	Updated: 2024.02.07												
			Inherent consequence						Inherent Risk						
Area/ID	Hazard	Cause	Developers	Environment	K. Maritime	M. Value	People	Inherent probability	Developers	Environment	K. Maritime	M. Value	People	Comment
1	Robot arm hits car	No restriction to movement and limited view	4	-	-	4	-	E	H	-	-	H	-	
2	Battery catches fire	Wrong connection or external damage. Charges to long. Direct sunlight.	2	2	2	2	3	D	H	H	H	H	H	Batteries from previous projects, after reading the user manuals
3	Break in/burglary	Someone breaks into the project room and steals components	2	-	3	1	-	B	H	-	H	H	-	This will also affect USN
4	Component delivery time	We order to late or the shipment gets delayed	3	-	-	-	-	D	H	-	M	-	-	
5	Crash between robot and bystander	Either the robot drives into a person or someone walks into the robot	4	-	-	3	4	D	M	-	-	H	M	
6	Crash between robot and wall/ object	The robot hits some inanimate object during testing or demonstration	5	-	-	4	-	E	H	-	-	H	-	
7	Bad connection to headset	Damaged or loose connectors	4	-	-	-	-	C	M	-	-	-	-	
8	Bad connection robot	Damaged or loose connectors, low power	4	-	-	-	-	C	M	-	-	-	-	
9	Drops headset	Headset gets dropped when changing operators	3	-	4	2	6	C	H	-	M	H	L	
10	Electric shock	Components have open electric sources that gives of shock	4	-	-	-	4	C	M	-	-	-	M	
11	Fire	Components starts a fire. Material catches fire from ovens in project room.	2	1	2	1	2	B	H	H	H	H	H	This will also affect USN
12	Fried circuit	Wrong power output, wrong connector	4	-	-	5	-	D	M	-	-	M	-	
13	Loose electrical connectors	Something nicks a wire, blunt force or vibrations loosens connection	4	-	-	-	-	E	H	-	-	-	-	
14	Loss of developers /group participants	Injury, fatality, kicked out for cheating, unable to meet criteria for bachelor	2	-	3	-	-	B	H	-	M	-	-	
15	Loss of work progress	Software crash (SolidWorks, IDEs), overleaf gets corrupted	1	-	2	-	-	D	H	-	H	-	-	
16	No connection to robot during demonstration	Router disfunction	4	-	-	-	-	C	M	-	-	-	-	
17	Meta quest software problems	The internal software of the meta quest could produce unforeseen problems with our own developed program.	3	-	3	-	-	C	H	-	H	-	-	
18	Motion sickness	The operator could get motion sickness from wearing the headset.	6	-	-	-	6	D	L	-	-	-	L	
19	No/low battery power	We can forget to charge batteries before demonstration. Problems in wiring could drain the batteries to early .	4	-	4	-	-	C	M	-	M	-	-	
20	Out of budget	We can run out of money	4	-	4	-	-	C	M	-	M	-	-	
21	Production time	If we are to late when starting production of parts, there is a possibility of not finishing in time	3	-	-	-	-	C	H	-	-	-	-	
22	Short circuit	Wrong wiring could short circuit electronics	3	-	-	4	-	C	H	-	-	M	-	
23	Static shock	We can damage electronics through static shock if we don't ground ourself	-	-	-	4	6	C	-	-	-	M	L	
24	Theft	Someone could run up and steal equipment during demonstration.	2	-	3	1	-	B	H	-	M	H	-	

Figure H.7: RRR-main form 1/2

Risk main form - Part 2/2		Rev. 3	Updated: 2024.02.07													
				Residual consequence						Residual Risk						
Area/ID	Hazard	Cause	Mitigating measures	Developers	Environment	K. Maritime	M. Value	People	Residual probability	Developers	Environment	K. Maritime	M. Value	People	Comment
1	Robot arm hits car	No restriction to movement and limited view	Set up virtual movement restrictions, limiters or padding.	6	-	-	6	-	C	L	-	-	L	-	
2	Battery catches fire	Wrong connection or external damage. Charges to long. Direct sunlight.	Don't charge to max capacity. Shade from sunlight. Don't leave the battery when charging. Store the battery in a non-flammable environment.	4	4	5	5	6	A	L	L	L	L	L	Batteries from previous projects, after reading the user manuals
3	Break in/burglary	Someone breaks into the project room and steals components	Lock doors and windows when leaving the room.	2	-	3	1	-	A	H	-	M	H	-	This will also affect USN
4	Component delivery time	We order to late or the shipment gets delayed	Order components with a large time buffer.	3	-	6	-	-	B	M	-	L	-	-	
5	Crash between robot and bystander	Either the robot drives into a person or someone walks into the robot	Install emergency stop, kill switch, keep area clear when operating robot. Place the robot in protected area when not in use. Add padding to robot shell.	5	-	-	6	6	C	L	-	-	L	L	
6	Crash between robot and wall/ object	The robot hits some inanimate object during testing or demonstration	Drive slow and set up boundary. Add padding to robot shell.	6	-	-	5	-	D	L	-	-	M	-	
7	Bad connection to headset	Damaged or loose connectors	Double check connections before use. Handle with care.	4	-	-	-	-	B	L	-	-	-	-	
8	Bad connection robot	Damaged or loose connectors, low power	Double check connectors before use. Use proper fastening when building robot. Handle with care.	4	-	-	-	-	B	L	-	-	-	-	
9	Drops headset	Headset gets dropped when changing operators	Security band fastened to operator and headset.	6	-	4	3	6	B	L	-	L	M	L	
10	Electric shock	Components have open electric sources that gives of shock	Protect cables and electronics when building robot. Turn of all power when checking or modifying robot.	4	-	-	-	4	B	L	-	-	-	L	
11	Fire	Components starts a fire. Material catches fire from ovens in project room.	Keep work stations clean. Build with non flammable materials. Find out where the closest fire extinguisher is located.	3	3	3	3	3	A	M	M	M	M	M	This will also affect USN
12	Fried circuit	Wrong power output, wrong connector	Make proper power diagrams and electrical preparation. Check all connections before powering on.	4	-	-	5	-	B	L	-	-	L	-	
13	Loose electrical connectors	Something nicks a wire, blunt force or vibrations loosens connection	Organize cables. Fasten wires to components properly.	4	-	-	-	-	C	M	-	-	-	-	
14	Loss of developers /group participants	Injury, fatality, kicked out for cheating, unable to meet criteria for bachelor	Be careful. Always write down sources and citations immediately when writing so you don't forget it. Take breaks to avoid burnout. Work together to have a open and good work environment.	4	-	3	-	-	A	L	-	M	-	-	
15	Loss of work progress	Software crash (SolidWorks, IDEs), overleaf gets corrupted	Download security copies daily or weekly. Save models and code hourly.	4	-	4	-	-	C	M	-	M	-	-	
16	No connection to robot during demonstration.	Router disfunction	Have a backup router.	5	-	-	-	-	C	L	-	-	-	-	
17	Meta quest software problems	The internal software of the meta quest could produce unforeseen problems with our own developed program.	Research and testing.	3	-	3	-	-	B	M	-	M	-	-	
18	Motion sickness	The operator could get motion sickness from wearing the headset.	Frequent brakes and slow speed for new users. First time users should be seated.	6	-	-	-	6	C	L	-	-	-	L	
19	No/low battery power	We can forget to charge batteries before demonstration. Problems in wiring could drain the batteries to early .	Double check charge early on presentation days.	4	-	4	-	-	B	L	-	L	-	-	
20	Out of budget	We can run out of money	Don't waste money on unnecessary equipment. Use what we have to the best extent. Prioritise purchases.	4	-	4	-	-	B	L	-	L	-	-	
21	Production time	If we are to late when starting production of parts, there is a possibility of not finishing in time	Be prepared with backup solutions. When 3D-printing do so with a buffer of at least 3x estimated printing time.	3	-	-	-	-	B	M	-	-	-	-	
22	Short circuit	Wrong wiring could short circuit electronics	Double check wiring before turning on power.	3	-	-	4	-	B	M	-	-	L	-	
23	Static shock	We can damage electronics through static shock if we don't ground ourself	Ground yourself while working. Don't wear clothing that cause static build up.		-	-	4	6	B		-	-	L	L	
24	Theft	Someone could run up and steal equipment during demonstration.	Don't leave equipment unsupervised.	6	-	6	4	-	A	L	-	L	L	-	

Figure H.8: RRR-main form 2/2

Consequence severity	
Revision no. & date	Changes
Rev.2, 2024.01.30	Added catastrophic level. Adjusted the consequences for each stakeholder
Rev.3, 2024.02.07	Fixed some double meaning words.
Rev.3, 2024.03.13	Specified moderate, developers

Risk Matrix	
Revision no. & date	Changes
Rev.2, 2024.01.24	Adjusted the probabilities to nearest tenth
Rev.3, 2024.01.30	Added catastrophic and readjusted after .
Rev.4, 2024.02.07	Adjusted matrix to new consequence and probability

Risk levels	
Revision no. & date	Changes

Probability levels	
Revision no. & date	Changes
Rev.2,2024.02.07	Removed the percentage of frequency

Hazardous events; descriptions	
Revision no. & date	Changes
Rev.2, 2024.02.09	Changed the names of some hazards.
Rev.2, 2024.03.13	Enlarged the writing

Hazardous events; allocated	
Revision no. & date	Changes
Rev.2, 2024.01.31	Updated the hazards
Rev.2, 2024.03.13	Enlarged the writing, made boxes and filled the empty boxes with " * "

Figure H.9: RRR-revision overview

I Schematic diagrams

I.1 Expansion board schematics

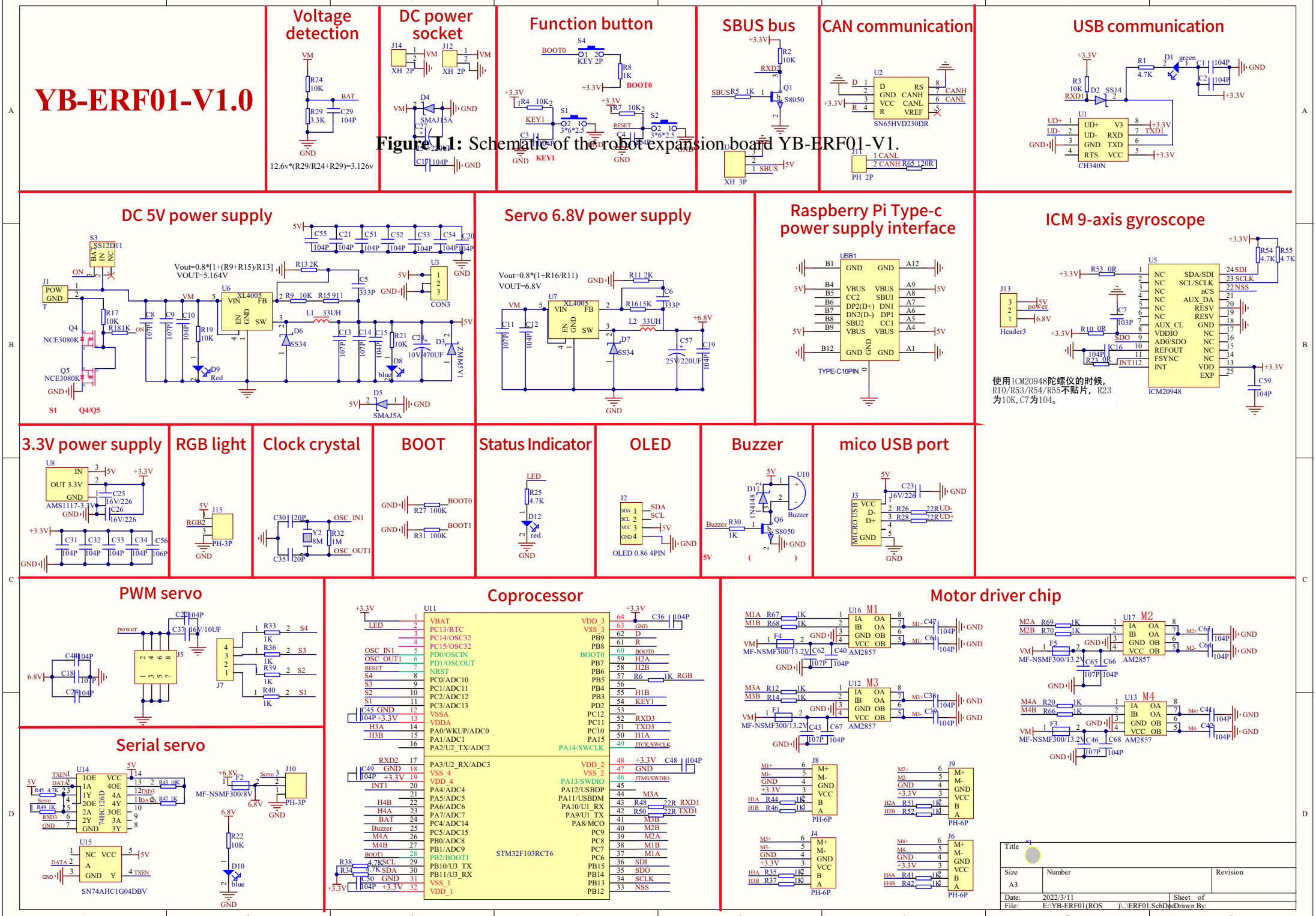


Figure 1.1: Schematic of the robot expansion board YB-ERF01-V1.

Figure I.1: Schematic of the robot expansion board YB-ERF01-V1.

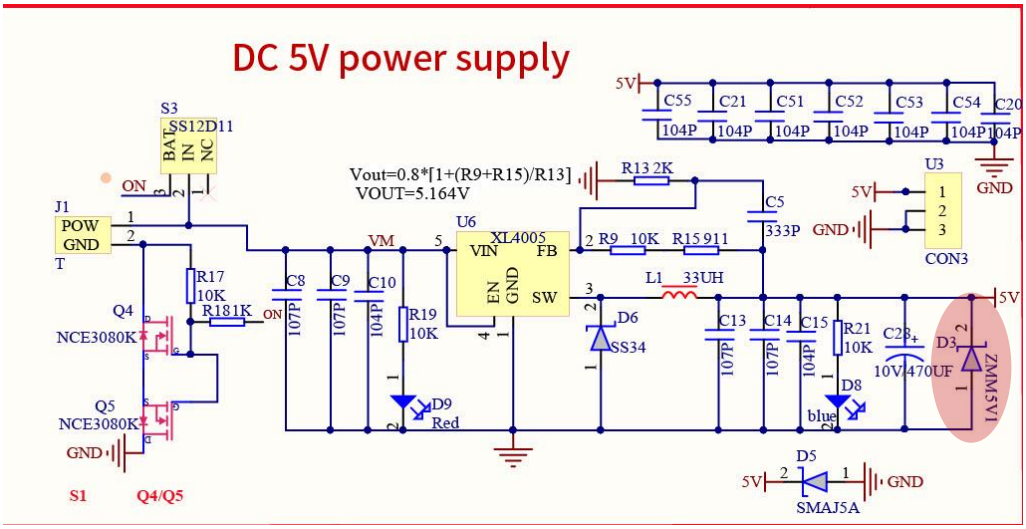


Figure I.2: Crop out from YB-ERF01-V1. D3 diode is observable in the bottom right corner.

J Battery

J.1 Introduction **HB** | *AEH*

The group was given a new requirement after the second presentation on March 8. The client wants the project group to build a battery from some old battery cells. This is because the robot should be operating for as long as possible, with as little downtime as possible. The group were handed a bag of old battery cells that came from old computer batteries. Some of these battery cells are Lithium Polymer (LiPo) cells but it is unknown which, the rest of the batteries are Lithium-ion (Li-ion) batteries. Some other components the group received were single-cell BMS', battery cell holders that hold four 18650 battery cells, and a bunch of short wires.

J.2 Design

J.2.1 General design **HB** | *AEH*

The battery which is being made is a 12-volt battery. To make this with 3.7-volt cells, it is needed to connect these in a series of three. Because the cell capacity is quite low compared to the battery that came along with the robot, there must be added more cells to the battery. Because the desired voltage is already reached and the only thing that needs to be increased is the capacity, the rest of the cells are added in parallel. This gives the option of two possible configurations. The first configuration is a 3S2P configuration, while the second one is a 2P3S configuration.
In the 3S2P battery setup, the cells are organised such that there are two series of three, connected in parallel. In the 2P3S battery setup, two cells are connected in parallel. Then three of these sets are connected in a series of three. Theoretically, there are no differences between the two configurations. Both the capacity and the voltage will be the same. However, the way they are connected makes a huge difference when it comes destruction or removal of one of the cells in the battery. In the 3S2P configuration, the battery loses half of its capacity when one cell is destroyed or removed. In the 2P3S configuration, the battery can operate at a slightly reduced capacity if one of the cells is damaged or removed. This is because the cells need to be in place and working, for the electrons to flow freely through the battery. As seen in the provided schematics, J.1 and J.2, it is visible how the battery capacity gets reduced with the different types of configurations.

Series connection involves connecting the negative terminal of one cell to the positive terminal of another cell, this way the output voltage increases. Parallel connection is connecting the positive terminals together and the negative terminals together, this way the capacity increases while obtaining the voltage.

In a 2P3S battery package, both connection methods are combined. There are three parallel pairs connected in series. To achieve 12 volts from 3.7-volt cells, they must be connected in a series of three. In this case, the three pairs of parallel are connected in series.

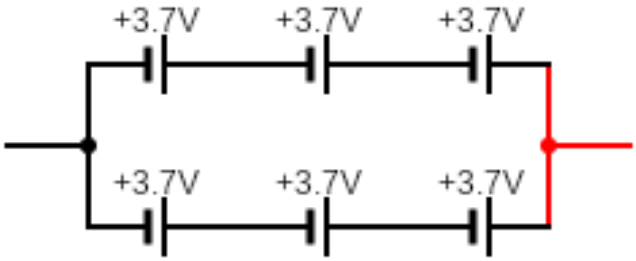


Figure J.1: This is how a 3S2P battery is connected

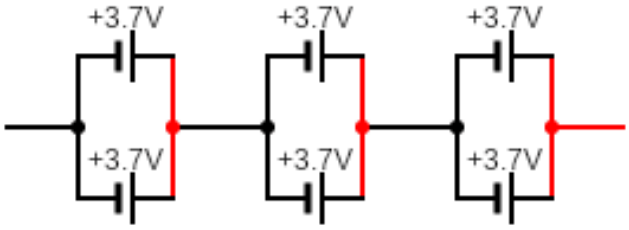


Figure J.2: This is how a 2P3S battery is connected

J.2.2 Components

HB | *AEH*

The components the project group received were 21 18650 battery cells, 6 battery cell holders with a capacity of 4 cells per holder, 12 charging/discharging modules (also known as BMS) of the type 03962A, and a bunch of short wires. Below is a quick description of the BMS type.

"5V microUSB 1A 18650 Lithium Battery Charging Board Charger Module with Protection. This versatile charger module is based on TP4056 IC, a constant-current/constant-voltage linear charger chip for single-cell lithium-ion batteries. This charger module can be powered by USB or a wall adapter. The features include a current monitor, under voltage lockout, automatic recharge and two status LEDs to indicate charge termination and the presence of an input voltage."[15]

J.2.3 Research

HB | *AEH*

Before the designing process could begin, the functions and capabilities of the BMS had to be known. The reason for this was that the BMS is a single-cell BMS, which means that it is designed to only monitor one battery cell at a time. If the BMS gets input from any other cells, it might not be able to detect if the cell is fully charged or completely discharged, and in that way, the battery cells can be destroyed. The reason the cells get destroyed if they are discharged completely is that they will no longer be able to obtain any charge and thereby not able to recharge. If the cells get overcharged, they will overheat and catch fire or even explode. This is because they receive more energy than they are designed to hold and therefore they become unstable.

Functional Diagram:

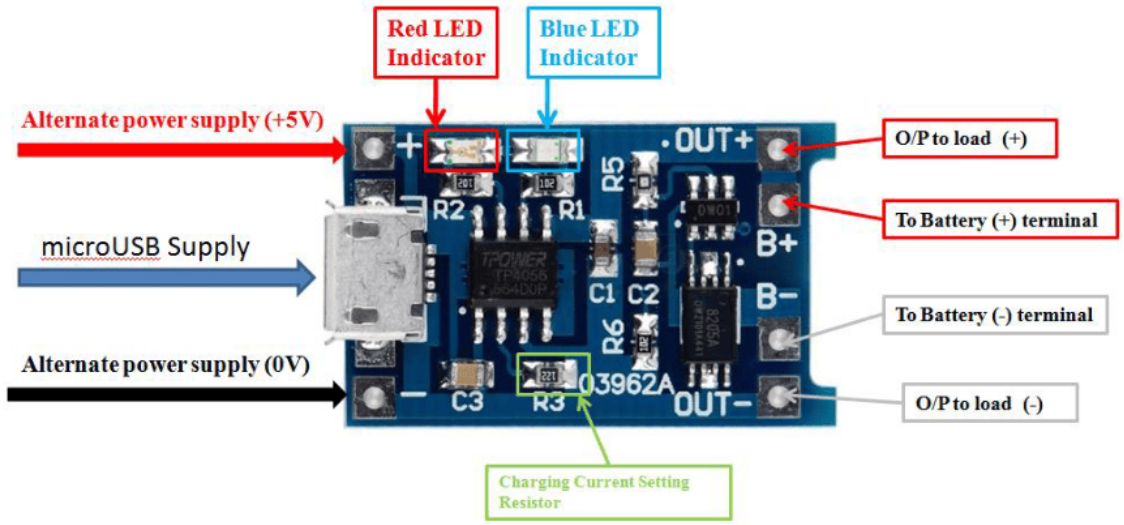


Figure J.3: Functional diagram of the charging module [15]

There are three IC 's on the module:

- straight under the LED indicators **(TP4056)**
- to the left of the connector to the positive battery cell terminal **(DW01A)**
- to the left of the connector to the negative battery cell terminal **(FS8205A)**

TP4056
The TP4056 is an IC designed to charge a single battery cell.

DW01A
The DW01A is an IC designed to protect a single Li-ion battery cell from overcharging and over-discharging (or undercharging). If the battery cells overcharge they could potentially explode, and if they over-discharge the battery cells will die.

FS8205A
The FS8205A is a PMOSFET used for controlling the charging and discharging sequence of the battery. It turns on and off depending on the polarity of the current. If the current is negative the PMOSFET turns on and the battery starts charging. If the current is positive, the PMOSFET turns off and the battery stops charging.

The BMS is also programmable, by changing the resistor located at R3 at the PCB. In figure fig: Charging module, R3 is marked as “Charging Current Setting Resistor”. This resistor can be swapped out so it will fit the battery cell capacity. This is not necessary in this case, because the battery cells have a greater capacity than the current charging current setting.

RGRPOG (k)	I_{BAT} (mA)
10	130
5	250
4	300
3	400
2	580
1.66	690
1.5	780
1.33	900
1.2	1000

Table J.1: The resistance (in kilo Ω) for the different charging currents [24]

As seen in table J.2, the capacity of each cell surpasses the 1000 mA charging current which is the standard swappable resistor on the BMS.

J.2.4 Design

HB | *AEH*

The design process of the battery went through several iterations. This was because some new decisions were made during the designing process, and some improvements had to be implemented.

- **Iteration one**
  The first design of the battery was a 3S2P battery that didn't have BMS, a charger was also designed. This was so that the BMSs could be used. The reason the BMSs were not added to the battery was that the space required from the battery would be too wide to fit in the robot. Because the battery had no BMSs it would not be able to charge them. Therefore, the cells had to be taken out of the battery and charged in a separate charger, before they could be inserted into the battery, and then the battery could be used again. Another reason was that the BMSs hadn't been properly researched at the time, and how they worked was not clear.

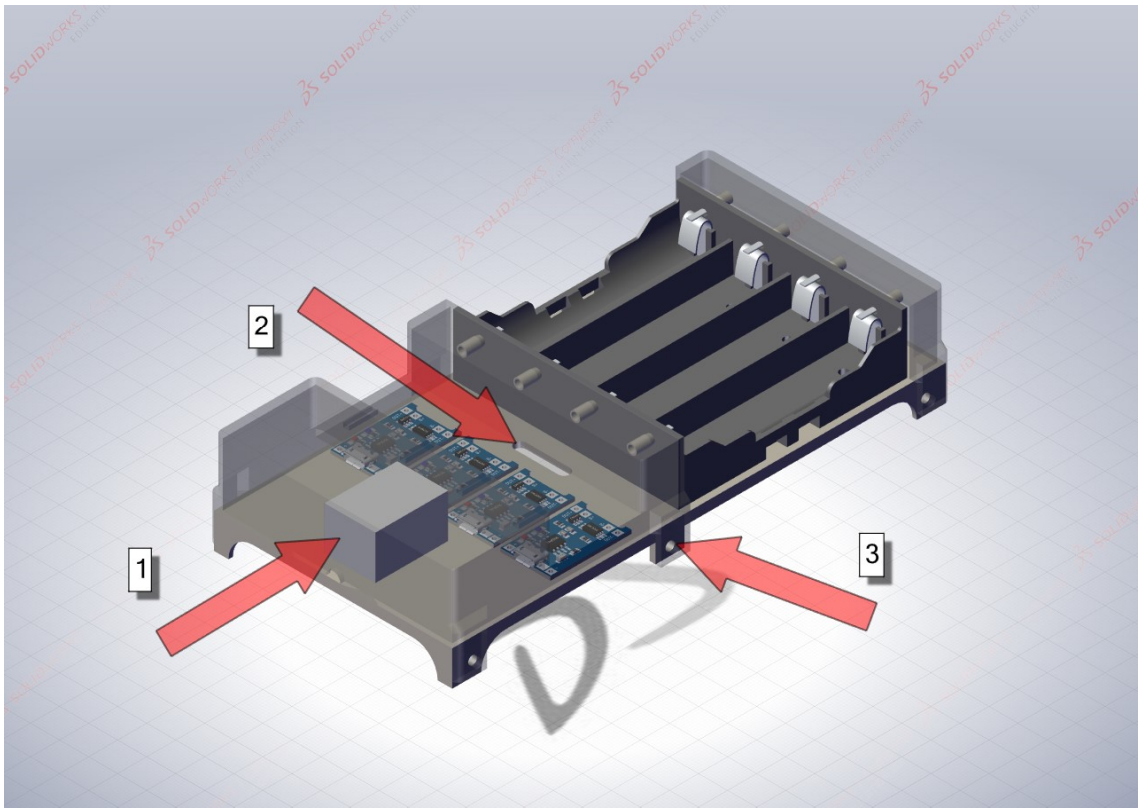


Figure J.4: A part of the first iteration. Arrow 1 points towards a block representing the attachment between the charger wires and wires leading to the BMSs. Arrow 2 points towards a hole where the wires from the end will appear. Arrow 3 points towards one of the attachment holes for the cover

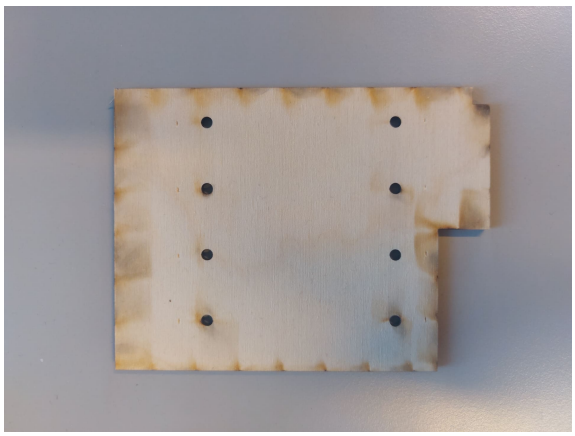


Figure J.5: The mounting plate for the battery cell holders for the first battery design

- **Iteration two**
  In the second iteration of the battery, BMSs were added. The reason for this was so that the battery could be charged without the need for removal of the cells. However, due to the way the BMSs were designed, they could not connect the outputs together in a series. This would lead to at least one BMS being fried. Because of this, the BMSs can not use their output connectors. The batteries themselves have to be connected directly to each other and the BMSs must be connected to each cell at the same time. Because of this, the battery needs a switch which can disconnect the cells from each other so the BMSs can surveil the cells, so they won't overcharge.

However, with this solution, the cells can't be surveilled while the battery is in use because the cells are connected to each other. Luckily the expansion board on the robot has a built-in alarm when the voltage is at 9.5 volts or below.

The second iteration of the battery has a BMS for each cell in the battery. This makes the connections inside the battery a bit more complicated than in the first iteration. This is because the BMSs can't be connected in series. After all, they are single-cell BMSs. This means they will not be able to surveil the designated battery cell when the battery is in use and will therefore not have a purpose when the battery is in use.

When charging, the BMSs will be disconnected in such a way that they will only be connected to the designated battery cell. This way the battery can safely be charged and overcharging is avoided. The only disadvantage of this battery is the battery cells are not protected from over-discharging. Luckily the expansion board on the robot has an inbuilt alarm if the battery voltage drops below 9.6 volts [84].

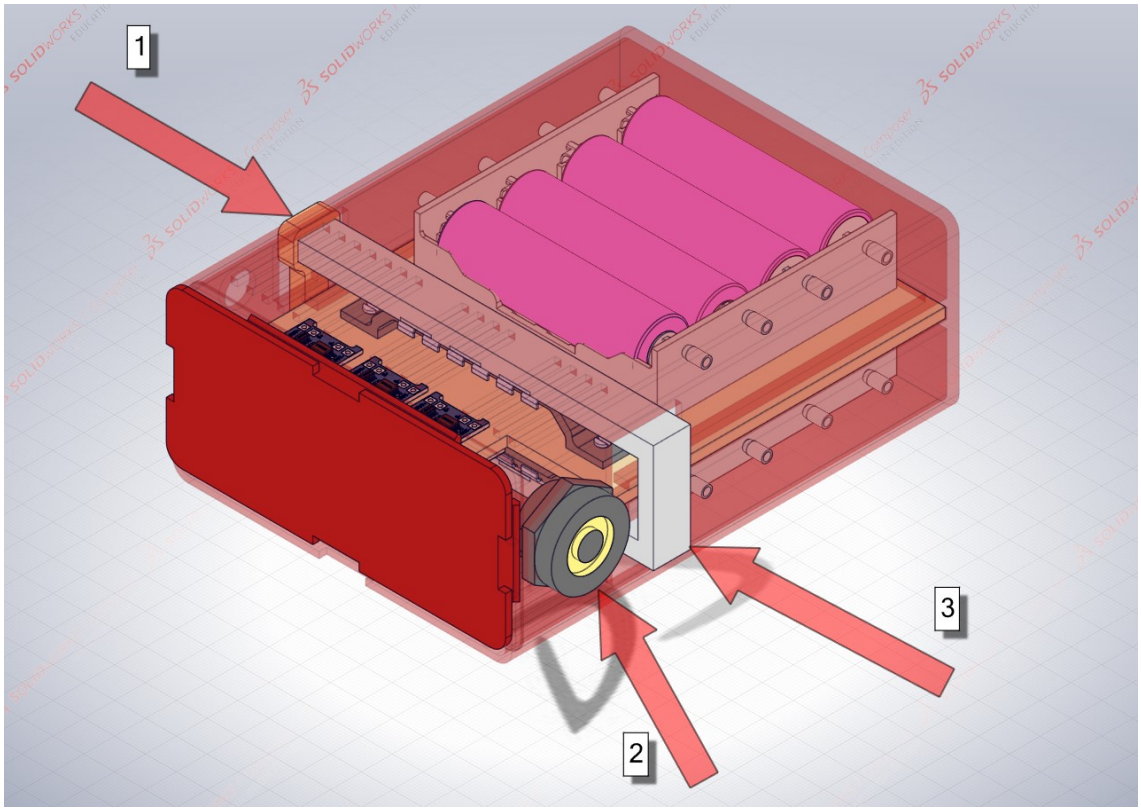


Figure J.6: Battery iteration 2. Arrow 1 points towards the holder for the mechanical switch. Arrow 2 points towards the magnetic connector. Arrow 3 points towards the mechanical switch

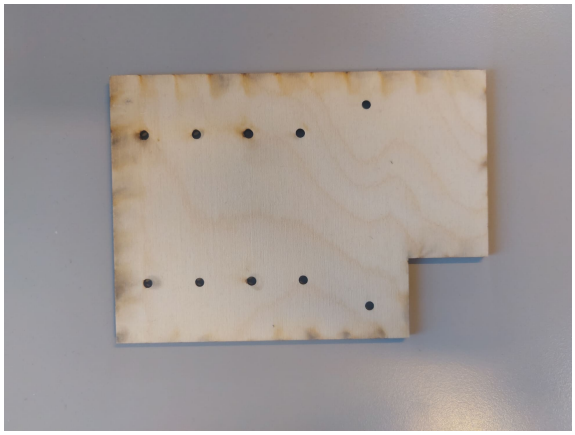


Figure J.7: Mounting plate for battery cell holders, for the second design

Figure J.8: The cover for the second iteration

- **Iteration three**
  In the third iteration, the mechanical switch received an improvement. The reason for this was that the switch was sliding over the connectors in the battery. This would have led to a short circuit every time the switch would be taken in or out.

  To prevent a short circuit when inserting or removing the switch, the opening for the switch was made larger and there was added some taps to the switch. The larger hole in the battery box made it possible to slide the switch in and through the whole box, then slide the switch into position without the connectors on the switch touching other connector plates than they are supposed to. The taps on the switch make it possible to attach the switch to the battery box, so the switch won't move while in use and disconnect.

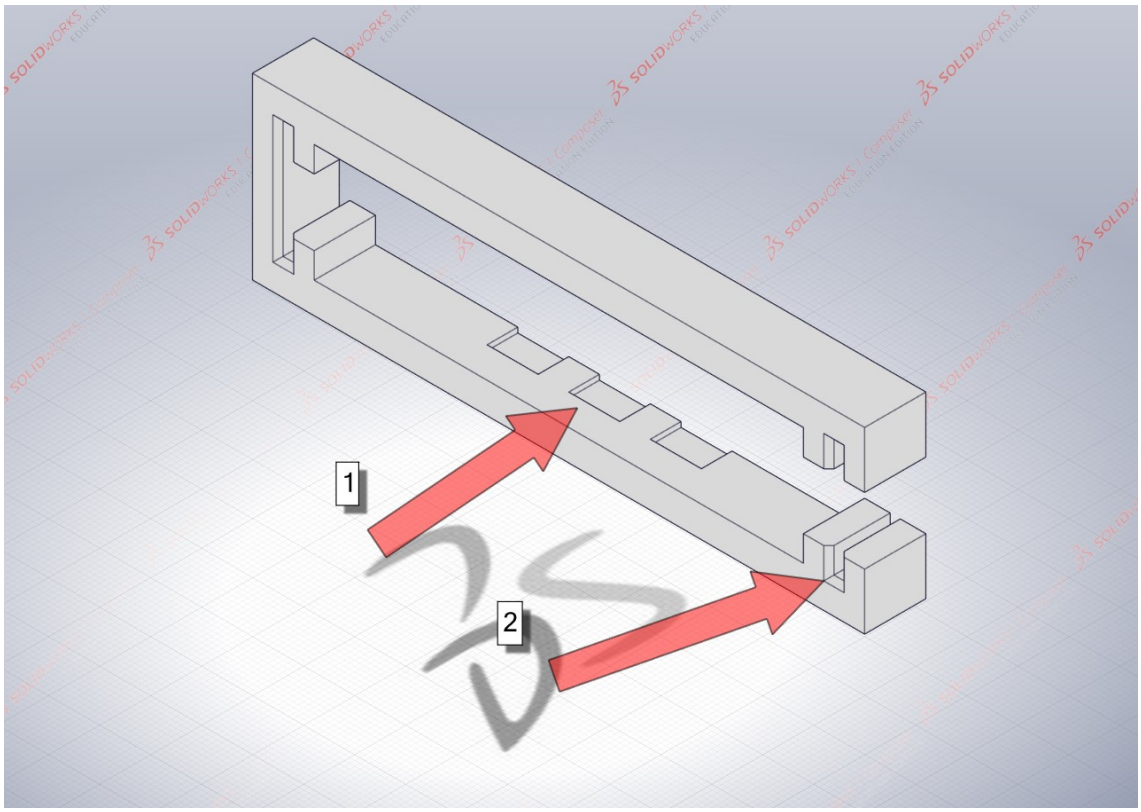


Figure J.9: The mechanical switch. Arrow 1 points towards the space for the connectors. Arrow 2 points towards the taps that keep the switch attached to the battery box.

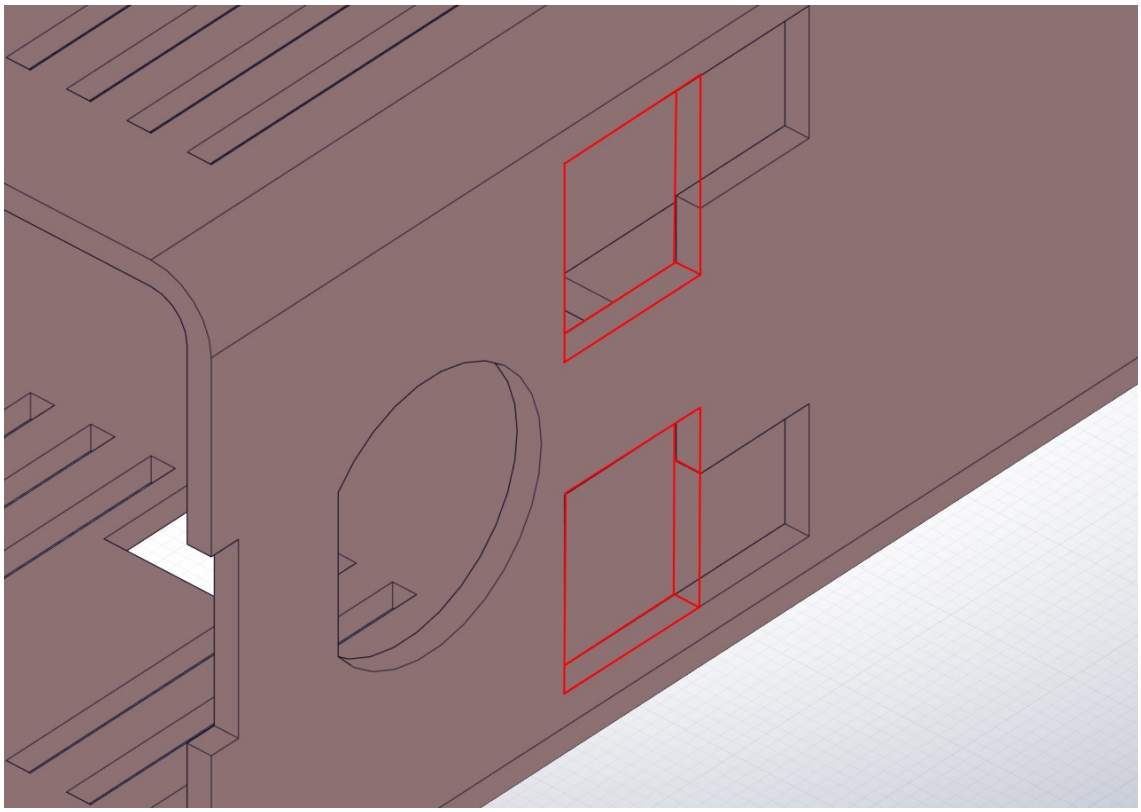


Figure J.10: The red markings show the added cut to the holes for the switch

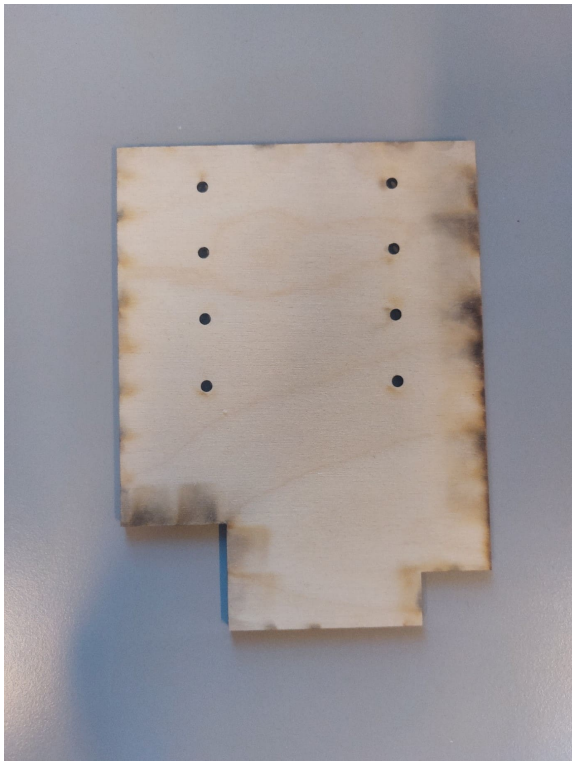


Figure J.11: The mounting plate had to be made longer for this iteration because the fork switch had to be slid in the length direction. The small cut in the bottom right corner is to make space for the notch in the battery cover lid.

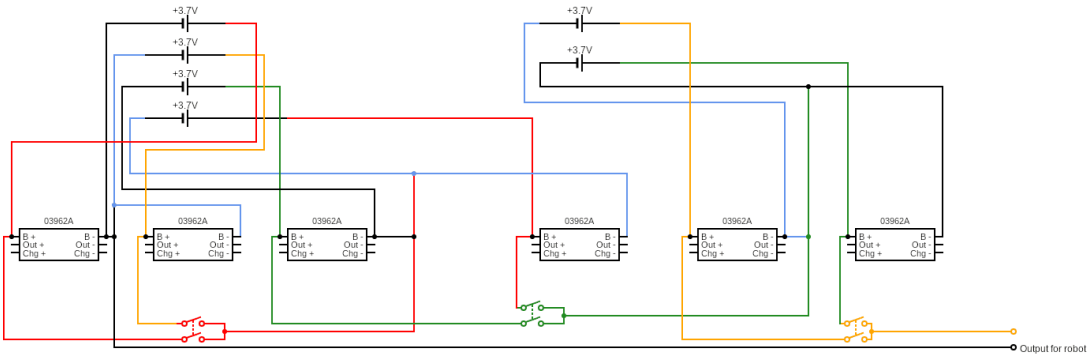


Figure J.12: The circuit for the battery with BMS in a 2P3S configuration. Wires for charging are not drawn into this circuit.

- **Iteration four**
  The fourth iteration was swapping out the large fork-like switch, to three separate switches of the type DPST. This type of switch has two separate inbuilt switches that is controlled by a single trigger. This change meant that the battery box had to get a redesign as well. The holes for the previous switch had to be removed, and new holes for the new switches had to be made. Because the new switches were placed on the top of the battery, the battery had to be made slimmer. This was because the height of the space for the battery was limited to 57 mm, and the battery height exceeded this with the new switches. See D for the reason this battery was scrapped.

Figure J.13: The battery cover for design iteration four

- **Iteration five**
  The fifth iteration was almost a complete redesign of the battery. This was because of a failed test of the previous iteration. The BMSs were removed, which meant much simpler wiring and a smaller battery. At the same time, the battery box and the separator plate got a new design. The fifth iteration ended up being almost the same as iteration one. The main difference was that the first design had a 3S2P design and this battery have a 2P3S design.

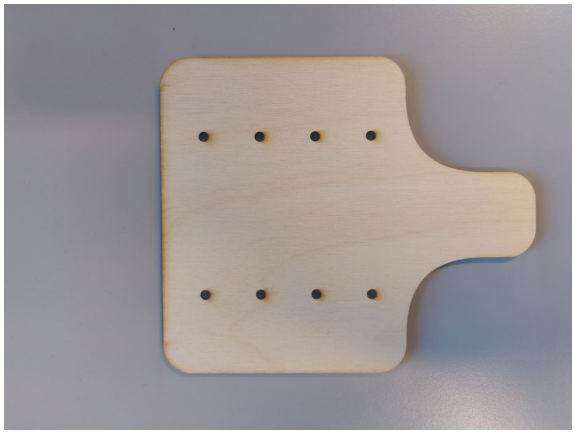


Figure J.14: Mounting plate for the final design iteration

J.3 Building the battery with BMS **HB** | *AEH*

The building process of this battery can be divided into four main steps; 3D printing, laser cutting, wiring, and assembling.

The 3D printing consisted of printing 2 parts. The cover of the battery and the lid of the battery. To prevent waste of filament the parts were sat in the position which required the least amount of support. The lid was placed flat on the printing surface, while the battery box was placed standing on the printing surface, with the back of the battery box down. This way the print only required support for the holes on what became the sides of the box while printing.

The laser-cut part is a piece of plywood that acts like a separator and mounting plate for the battery cell holders, at the same time it keeps the battery cell holders in place inside of the battery.
The wiring part was a bit more tedious process. First, all of the components that were handed out had a lot of short wires attached to them. These had to come off because many of them were way too short or would have no purpose at all in the battery. All of the wires were removed with the help of a soldering iron. After this, the process of wiring could start.

When soldering the wires together, a lot of the wires became exposed. To make sure that the wires were isolated and did not touch any other components that they were not supposed to touch, shrink tubes were placed above the exposed area and heated up with a heat gun. As the shrink tubes became hot, they shrank and attached themselves to the areas it was placed. When all of the soldering was finished, cable shoes had to be attached to the wires that should be connected to the switches. In the end, wires were attached to the magnetic connector, so it could more easily be attached and detached from the output from the battery. The wires from the connector were not yet attached to the wires from the battery itself, because it is one of the last steps of the assembly process of the battery.

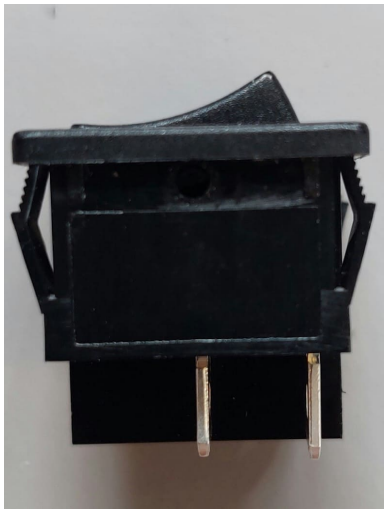


Figure J.15: The switches used for battery iteration four, in this position the switch is turned on

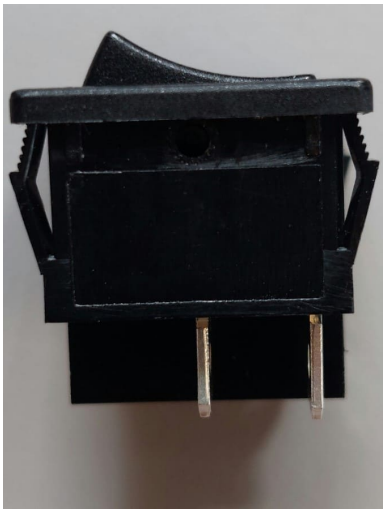


Figure J.16: The switches used for battery iteration four, in this position the switch is turned off

Assembling the battery was the last step before it could be tested. The first step of the assembly process was to attach the battery cell holders and the BMSs to the operator plate. The cell holders were attached using four M3 machine screws and nuts. To attach the BMSs to the operator plate, double-sided tape was applied. The battery so far was a mess, because the wires were soldered to the cell holders and the BMSs before they were attached to the separator plate. Because of this, the wires went everywhere.

The next step in the assembly process was inserting the cells, then insert the cattery components into the battery box.
The parts that were bought for the battery were a magnetic cable connector and some switches to change between charging mode and use mode. This is because of the single-cell BMS. If a regular 2P3S BMS had been used, there would not be a need for the switches to change between charging mode and use mode.

J.4 Building the battery without BMS **HB** | *AEH*

The battery without BMS was a backup solution after the battery with BMS didn't work as thought. This battery has a simpler circuit than the battery with BMSs because the only parts in the battery consist of the battery cells, wires, and a connector to the robot.

J.5 Test of battery cells

J.5.1 Introduction **HB** | *AEH*

To make sure that the battery cells are still alive and have enough capacity to be used, they are tested. To make sure that the capacity of the battery cells is good enough to be used, they are tested using the OPTUS BT-C3100 V2.2, which is a battery cell tester with other functions as well. If the battery cell has too little capacity left or they are dead, there would be no use for them.

J.5.2 Equipment **HB** | *AEH*

The equipment that is being used for the testing is a multimeter and a battery tester. The multimeter is a FLUKE 8010A Digital Multimeter and the battery tester is an OPUS BT-

C3100 V2.2.

Fluke 8010A Digital Multimeter
The FLUKE 8010A Digital Multimeter is a portable bench-type digital multimeter that USN got to their possession in 1992. Even though this is an old multimeter, it is reliable. It can measure voltage, current and resistance in both AC and DC. For the creation of the battery, it is only needed to use the DC voltage function. The values measured are displayed on a small $3\frac{1}{2}$ -digit LCD.

OPUS BT-C3100 V2.2
The OPUS BT-C3100 is capable of charging, discharging, restoring and testing different types of battery cells. The capacity is four battery cells at a time, and it can have different modes for each one at the same time. The different sections in the battery cell tester consist of a positive and a negative pole. The positive pole is at the far end of the tester, while the negative side is a spring-loaded metal plate that makes sure that the battery cell fits nicely and firmly in the tester.
The test sequence consists of first charging the battery cells up to 4.20 V. Then a discharging sequence is initiated, and the battery cells are discharged to 2.80 V. During this discharging sequence the battery capacity is measured, and displayed in mAh when the sequence is finished. The final step is charging the battery cells up to 4.20 V, so the battery cells don't die. All of these steps happen automatically when the device is set to test mode.

J.5.3 Testing

HB | *AEH*

The testing process is divided into two tests. The first test is to check the voltage of the batteries using a multimeter. The second test is to check the capacity that is left of the battery cells, measured in mAh. This test is carried out using a battery cell tester. To keep track of each battery cell, they are numbered from "1" and up and then logged in to a table. This is to prevent mixing them if any of them are dead and to separate the cells with the least capacity and most capacity.

The reason the voltage of the battery cells was checked is because if the cells are below 2.80 volts, the cells "die". A battery cell that has died is a battery cell that has lost all of its voltage and it is not possible to recharge the cell. Luckily only one of the given battery cells was dead. This gives 20 battery cells that could be tested for their capacity. The battery capacity test was a tedious process. The test consists of placing the cells in the battery cell tester and then switching the battery cell tester into testing mode. The way this testing process works is that the cell is charged all the way up to 4.20 volts. Then a discharging-sequence is initiated and the cells are discharged all the way down to 2.80 volts. Then the test is finished. To prevent the cells from dying, the battery cell tester automatically starts to charge the cells again after the test is finished. Because 20 battery cells had to be tested, the capacity test took at least six hours for the cells with the least capacity. The total time for testing the cells was over two and a half days to complete because the cells had to be watched when charged and tested.
As the cells was done with the capacity test, the result was logged into a table and ranged from 1 to 20, where 1 was the cell with the most capacity and 20 was the cell with the least amount of capacity.

J.5.4 Results

HB | *AEH*

The results from the battery check were quite promising. Only 1 out of 21 battery cells were dead. This meant that 20 battery cells were possible candidates for the battery pack. The results from the battery testing can be viewed in table J.2.

Battery cell test				
Cell №:	Voltage:	Tolerance:	Capacity:	Ranking:
1	3,84V	+/- 0,0038V	1805mAh	20
2	3,84V	+/- 0,0038V	1853mAh	19
3	4,09V	+/- 0,0041V	1984mAh	14
4	4,1V	+/- 0,0041V	2031mAh	7
5	4,1V	+/- 0,0041V	1984mAh	14
6	4,1V	+/- 0,0041V	2031mAh	7
7	4,1V	+/- 0,0041V	1996mAh	13
8	4,13V	+/- 0,0041V	2060mAh	4
9	4,09V	+/- 0,0041V	2023mAh	10
10	4,09V	+/- 0,0041V	2047mAh	5
11	0,01V	+/- 0,V	*	*
12	4,07V	+/- 0,0041V	2091mAh	1
13	3,88V	+/- 0,0039V	2038mAh	6
14	4,07V	+/- 0,0041V	1874mAh	18
15	3,96V	+/- 0,004V	2086mAh	2
16	4,07V	+/- 0,0041V	2082mAh	3
17	4,07V	+/- 0,0041V	2028mAh	9
18	3,96V	+/- 0,004V	2005mAh	12
19	3,81V	+/- 0,0038V	1979mAh	16
20	3,82V	+/- 0,0038V	1940mAh	17
21	3,88V	+/- 0,0039V	2019mAh	11

*The tolerance for DC-Voltage measurements is 0.1% at all ranges in FLUKE 8010A Digital Multimeter

**The voltage listed is the voltage measured before the capacity test

Table J.2: Battery cell test results. Cell number 11 was dead.

J.6 Testing of batteries

HB |

J.6.1 Battery with BMS

HB | *AEH*

The testing of the battery with BMS consisted of two stages. The first test was attaching the battery cells before the charging wires were attached. The negative and the positive wires were attached to the multimeter, then the battery was put into “use mode”.

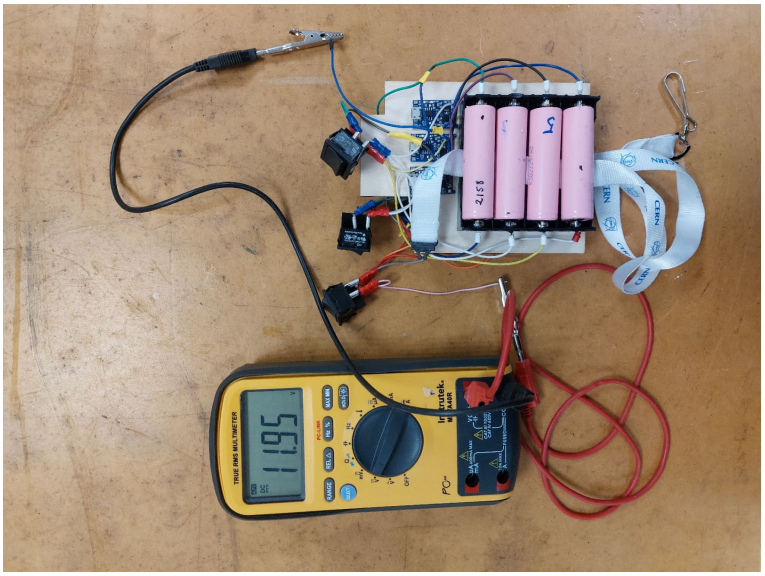


Figure J.17: The voltage test was a success. The white ribbon under the battery cells is used to detach the battery cells easily.

The second test was charging the battery. When the charging wires were attached, the battery cells were inserted and the battery was put into charge mode. The charger was plugged into the power outlet from the wall and the charging module LEDs lit up. All was fine up until 30 seconds in when a burning smell occurred. The charger was disconnected, but the component did not stop frying. So all of the cells were ripped out of the charger to prevent further damage. This incident is one of the reasons the white band is placed below the cells.

J.6.2 Battery without BMS

HB | *AEH*

The main test of the battery without BMS is checking the output voltage. This was done the same way as the battery with BMS. Inserting the cells, then measuring the output voltage. This test was a success as well.

The next test for this battery is to check if all of the components that were made, actually fit. This was done by assembling all of the parts of the battery. The battery passed this test as well, but there was some room for improvement. The positive wire going from the battery cell holder towards the magnetic connector was not long enough to attach to the clip that connects to the magnetic connector. However, this can be solved by not inserting the cell holder all the way before attaching the wire to the clip.

J.6.3 Discussion

HB | *AEH*

The reason that the battery with the BMS did not have the last test was that it failed on a functional test which it was required to pass to be able to be used. It was therefore no point in testing the parts to see if they fit.

The reason for the failed test may have been the way the BMSs are designed, they would short-circuit. Another reason could be that there was too much heat when soldering the wires to the BMSs which led to some of the conducting material in the BMS flowing and

connecting to each other, and in that way create a short circuit.

After studying the schematics of the BMS again and going through what was done when soldering the wires, it looks like the reason for the short circuit to appear was applying too much heat when soldering the wires.

J.7 Using the battery

J.7.1 General use

HB | *AEH*

To use the battery is simple. First, the cells have to be inserted into the battery, then the battery has to be assembled. After the assembly of the battery, the battery is inserted into the battery drawer in the robot. When the voltage level becomes too low, the voltage alarm on the expansion board starts sounding. Then the battery has to be swapped, or the battery cells have to be swapped.

Because the battery cells are old, the capacity varies from cell to cell. Because of this, the cells being used are the 12 cells with the highest capacity. The reason for choosing 12 cells to use is so that the battery cells can be swapped out and the battery can quickly be put into use again. The 12 cells are split into two groups, this is to make sure that the cells that are in the battery are as close to each other as possible when it comes to capacity. This is because the battery has no BMS, and if the capacity of the cells varies a lot, the cells with the lower capacity have a higher risk of getting over-discharged and dying.

Battery cluster 1	
Cell №:	Capacity
8	2060mAh
10	2047mAh
12	2091mAh
13	2038mAh
15	2086mAh
16	2082mAh
Avg:	2067mAh

Battery cluster 2	
Cell №:	Capacity
4	2031mAh
6	2031mAh
9	2023mAh
17	2028mAh
18	2005mAh
21	2019mAh
Avg:	2023mAh

Table J.3: The two groups of battery cell clusters used for the battery

J.7.2 Assembly

HB | *AEH*

To assemble the battery some components need to be assembled, then the cells must be inserted, and then the rest of the components need to be assembled.
The first step in assembling the battery is to attach the two battery cell holders and the operator plate to each other. They are kept together by four M3 screws and four matching nuts. The cell holder that keeps four of the battery's cells is located at the top of the plate. The screws are first inserted through this, then through the separator plate, and in the end, the battery cell holder that keeps two of the six battery cells.

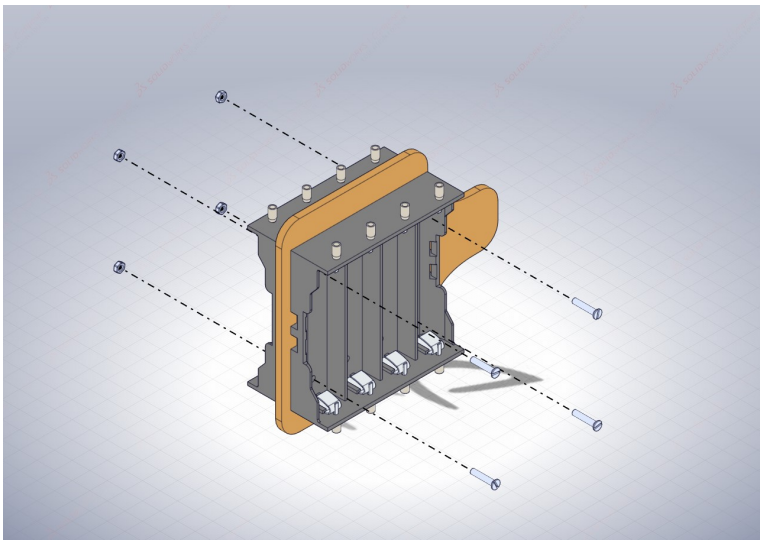


Figure J.18: Step one of assembling the battery

The second step is to wire the battery as shown in the schematics in figure J.2. This is the basic setup for a 2P3S battery.

The third step is to insert the battery cells. It is very important to insert the cells correctly. If they are inserted the wrong way around, the battery might short circuit or the polarity of the battery becomes inverted. Also, make sure that the ribbon is placed between the cells and the holder so the cells can easily be removed. This also prevents from damaging the cells.

The next step is to insert the cell holder into the battery box itself. Line the separator plate up with the guide tracks inside of the battery box. Then slide it all the way in.

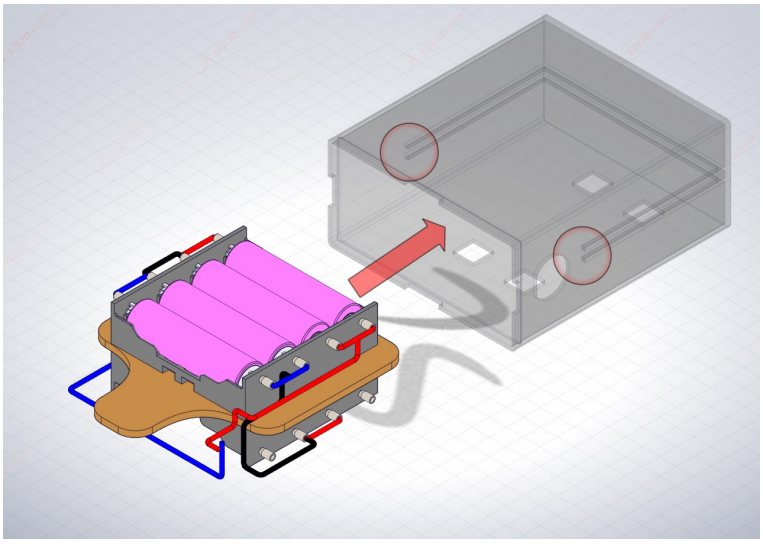


Figure J.19: Insert the battery into the battery box

The next step is to attach the magnetic connector to the battery box. Unscrew the large plastic nut from the back of the connector, insert the connector in the hole on the right side of the battery box, and then screw the nut onto the magnetic connector from the inside of the battery box.

Then attach the wires from the battery to the connector. Attach the positive wire to the middle pin, and the negative wire to the pin on the side. See the provided schematics for better understanding. It is recommended to use cable shoes or clips like VAGO to connect the wires to the magnetic connector. This is because the battery needs to be partially disassembled to switch the cells. To make the assembly/disassembly process easier, there should be attached wires to the pins on the magnetic connectors.

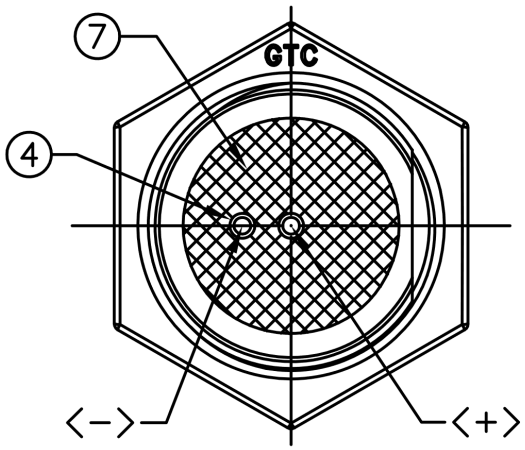


Figure J.20: Close-up picture of the back side of the connector, from the datasheet for the magnetic connector[16]

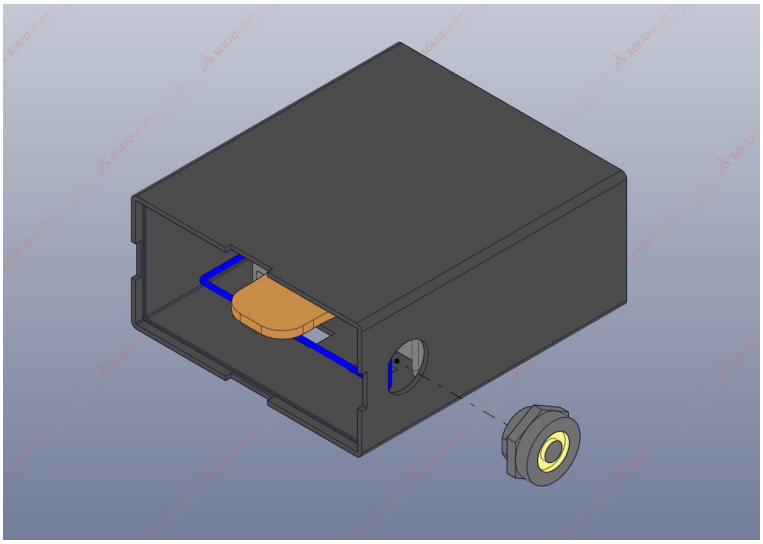


Figure J.21

The last step is to attach the battery box lid, and the battery is ready for use.

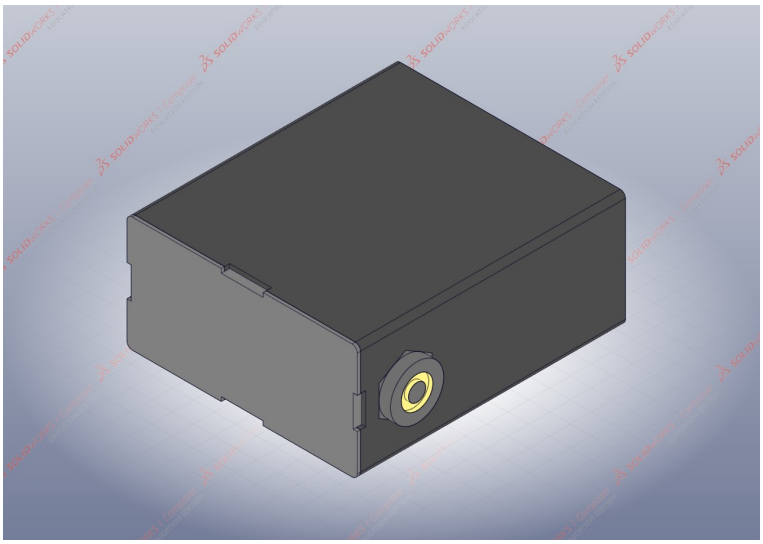


Figure J.22: The finished battery

J.7.3 Swapping cells

HB | *AEH*

To swap the battery cells, the battery has to be disassembled following the last five steps of the assembly process, in reverse. To get the lid off the battery box, it might be helpful to use a flathead screwdriver in one of the notches. To remove the cells, pull on the ribbon that is placed between the cells and the cell holders to loosen the cells. Remove the cells and insert a new set of cells, then follow the normal assembly process and the battery is ready to use again.

J.7.4 Charging

HB | *AEH*

The charging of the battery consists of removing the cells from the battery and then inserting the cells into a battery cell charger. In this case, the charger is the battery cell tester that was used to test the capacity of the cells, the OPUS BT-C3100.

K Initial software implementation for arm

OM | *AEH*

The first implementation of robotic arm movements was simple but good enough for a demonstration. The VR operator could only control servo 1, 4 and 6 (see 4.49).

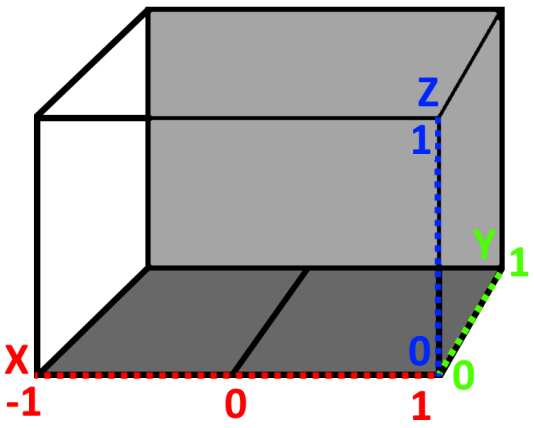


Figure K.1: VR hand control area

If the operator's hand was in the origin, the end-effector (pinch mechanism) was pointing forwards
For (point) the end effector would point -

- (0, Y, 0) forwards (0 degrees)
- (1, Y, 0) 90 degrees to the right
- (-1, Y, 0) 90 degrees to the left
- (X, Y, 1) 90 degrees upwards
- (0.5, Y, 0.5) 45 degrees to the right and 45 degrees upwards

For any point in between the end-effector would point accordingly. The Y-axis was irrelevant, only the X-Z plane affected the position of the arm. The code uses X and Y values, but Y is mapped to Z in the diagram, as this was easier to work with.

```
class Controller:
    # ... code

    @staticmethod
    def convert_x_to_angle_difference(x: float) -> int:
        """Converts x to angle difference.

        Args:
            x: x

        Returns:
            angle difference
```

```
        """
        return int(x * 90)

    @staticmethod
    def convert_y_to_angle_difference(y: float) -> int:
        """Converts y to angle difference.

        Args:
            y: y

        Returns:
            angle difference
        """
        y_abs = abs(y)

        if y_abs > 1:
            return 90

        return int(y_abs * 90)

    @set_last_message
    def handle_vr_hand(self, msg) -> None:
        """Handles VRHand messages.

        Args:
            msg: VRHand message
        """
        x, y, strength = msg.x, msg.y, msg.strength

        print(f"got {x=} {y=} {strength=}")
        x_angle = self.convert_x_to_angle_difference(x)
        self.robot.set_arm_rotation_difference(x_angle)

        y_angle = self.convert_y_to_angle_difference(y)
        self.robot.set_arm_tilt(y_angle)

        pinch_angle = self.convert_pinch_to_angle(strength)
        self.robot.set_pinch(pinch_angle)

        print(f"- {x_angle=} {y_angle=} {pinch_angle=}")
```

Corresponding unittests to the code written above.

```
class TestController(TestCase):
    @classmethod
    def setUpClass(cls):
        cls.c = Controller(production=False)

    def test_angle_x_conversion(self):
        self.assertEqual(self.c.convert_x_to_angle_difference(0), 0)
        self.assertEqual(self.c.convert_x_to_angle_difference(1), 90)
```

```
        self.assertEqual(self.c.convert_x_to_angle_difference(-1), -90)
        self.assertEqual(self.c.convert_x_to_angle_difference(0.5), 45)
        self.assertEqual(self.c.convert_x_to_angle_difference(-0.5), -45)

    def test_angle_y_conversion(self):
        self.assertEqual(self.c.convert_y_to_angle_difference(0), 0)
        self.assertEqual(self.c.convert_y_to_angle_difference(1), 90)
        self.assertEqual(self.c.convert_y_to_angle_difference(-1), 90)
        self.assertEqual(self.c.convert_y_to_angle_difference(0.5), 45)
        self.assertEqual(self.c.convert_y_to_angle_difference(-0.5), 45)
        self.assertEqual(self.c.convert_y_to_angle_difference(1.1), 90)
```

L Electronics interface description

L.1 Motor expansion board YB-ERF01-V1.0 interface description

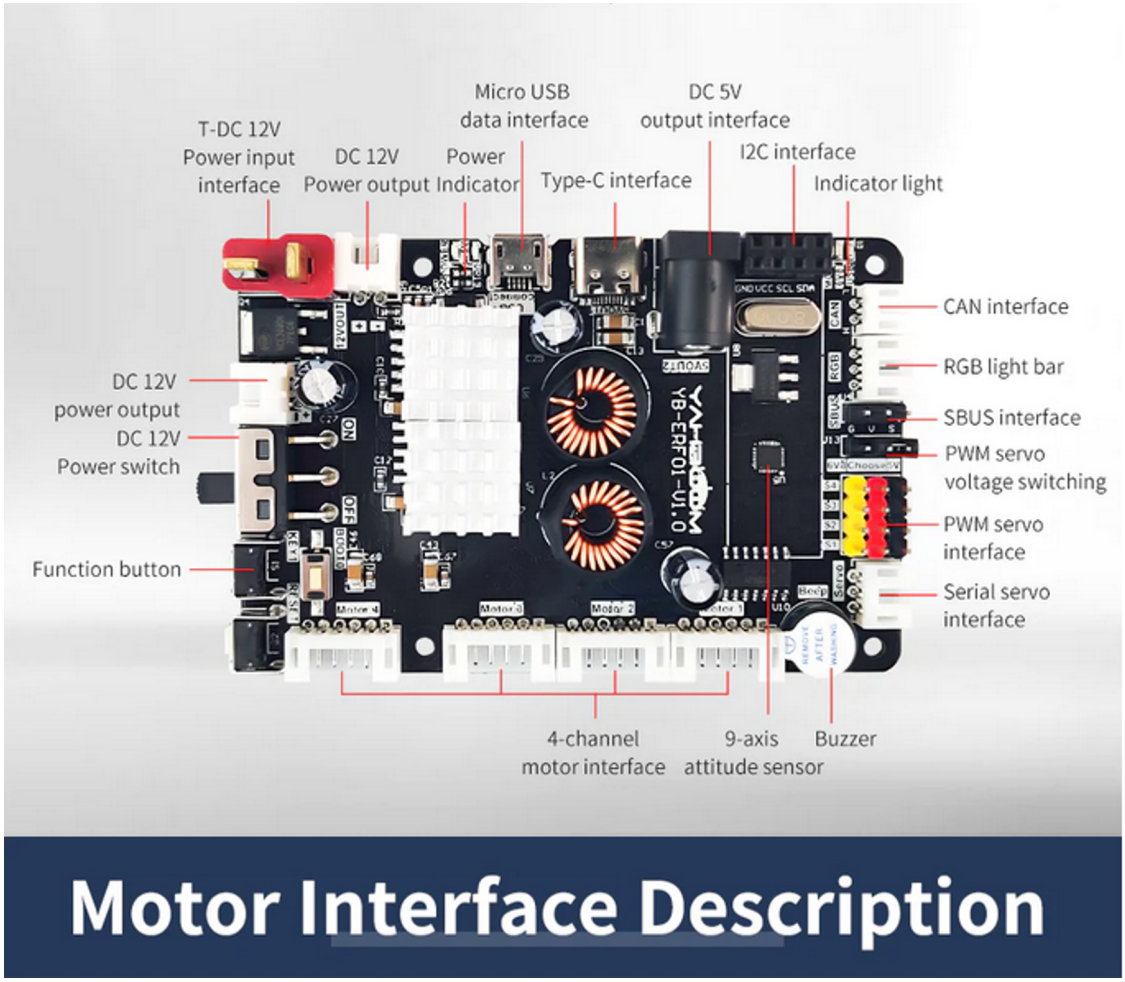


Figure L.1: Motor expansion board YB-ERF01-V1.0 interface description

M Earlier iterations of object detection

In this section, we detailed our initial exploration using a pre-trained object detection model from the TensorFlow Lite Model Zoo. Building upon those results, we went through a second iteration aimed at creating a more tailored and potentially higher-performing model. This section delves into the process of developing a custom object detection model using TensorFlow's *tflite_model_maker* library, leveraging the concept of transfer learning. We outlined our dataset preparation, model training, and deploying the model on a RPi connected to a camera. We have written about why we did not continue with using *tflite_model_maker* for making the custom object detection model in the Challenges section 9, sub-sub section 9.2.3.

M.1 First iteration

AD | *AEH*

Doing some quick research we found out that there were quite a few pre-trained models on the official TFLite website hub. Some of these models were also trained on a dataset called the COCO (Common Objects in Context) dataset, which is a large-scale object detection dataset [99]. The COCO dataset has several objects of interest, and they also provide a label file. The model used was MobileNet Version 2. It is a popular CNN algorithm in object detection. Using the COCO dataset label file, we aligned them together on the raspberry Pi, achieving satisfying results as seen in Figure M.1.

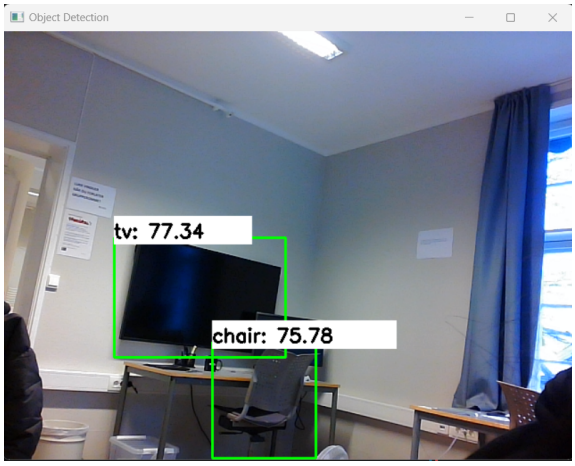


Figure M.1: Object detection with the COCO dataset and MobilenetV2

M.2 Second iteration

AD | *AEH*

Building upon the results from deploying the pre-trained model, we progressed to tailoring a custom object detection model to suit our project's specific requirements better. TensorFlow provides a library within its framework called *tflite model maker* which simplifies the model-making process.
The *tflite model maker* library uses a concept called transfer learning. **Transfer learning** is a technique that shortcuts the training of ML models by taking hold of a model that has already been trained on a related task and reusing it in a new model [108]. Figure M.2 represents the architecture for doing this.

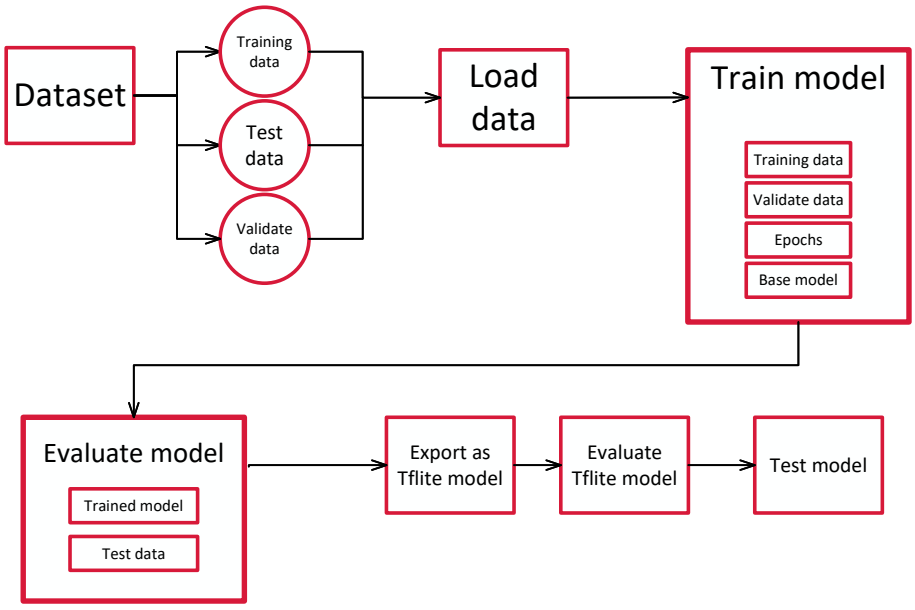


Figure M.2: System architecture for Transfer Learning

As shown in the above diagram, we needed to start by creating a dataset with accompanying bounding box coordinates that define the boundaries of a bounding box around an object within an image or a video frame. A bounding box is a rectangular box that can be drawn around the detected object to identify its location. The coordinates are usually defined in terms of the box's corners or the top-left corner along with the box's width and height, often annotated as(x_min, y_min, x_max, y_max). They are also normalized values ranging from 0 to 1. As we wanted to work out how the library worked, we leveraged a comprehensive dataset from TensorFlow Hub, consisting of 3000 images of people, each annotated with precise bounding box coordinates. The dataset was to be divided into three sections:

1. **Training**: 80 per cent of the data (roughly 2400 pictures) were used for training the model.
2. **Test**: 10 per cent of the data (roughly 300 pictures) were used for testing the model.
3. **Validation**: 10 per cent of the data (roughly 300 pictures) were used for validating the model.

The coordinates were listed in a comma-separated values (CSV) file, and table M.1 shows the file format with an example.

Set name	Image Path	Label	X Min	Y Min	X Max	Y Min
Training	0679.jpg	Person	0.25	0.29282	0.40820	0.4758

Table M.1: Example of the dataset CSV file looks like

The data has to be loaded now. This means the training data is divided into smaller batches so that the model can learn more quickly. The data is also resized and shuffled so that the model gets random data.
The next step was the biggest one, to train the data. The package function that trains the model requires some prerequisites such as the training data, test data, and the base model that the custom model is to be trained upon. Our group chose to start with the model that TensorFlow recommended, *efficient det* (version 0) [109]. It also requires the number

of epochs the model needs to train on, which is typically set to 50. While training, the package outputs the total loss function. This is the difference between the actual output and the predictions made by the model. It is the metric for evaluating how well the model is learning [6, p. 710].
The model needs to be tested. This is quite straightforward, as the model maker provides a function for it, using the test data. The model is then exported as a TFLite model, and again evaluated. Finally, we test the model by deploying it on the RPi which is connected to a camera.

M.3 Testing of the custom model

AD | *AEH*

Testing the model in itself is a big process. Luckily the TensorFlow website provides help for deploying the model, and also how to display the detection on a camera and stream to a web browser. Figure M.3 represents how the testing works with the help of a sequence diagram. A sequence diagram shows the sequence of messages passed between objects or functions in a program. There are three main objects: the detection object, the camera, and the model.

The detection object waits for the ROS master node to activate object detection. After it receives confirmation, it initializes the camera video capture. Then a sequence of functions occurs in a loop; the camera frames are read, captured, and sent to the model object. The model object has interpreted the model and has retrieved the input and the output details. The **output details** contain information about the prediction accuracy (ranging from 0 to 1), the bounding box coordinates, and the object label. The **input details** contain information about the data dimensions that the model expects [165]. After receiving the frame, the model object interprets the frame and adds the predicted bounding box coordinates, label, and prediction score. It sends the frame back to the camera for display.

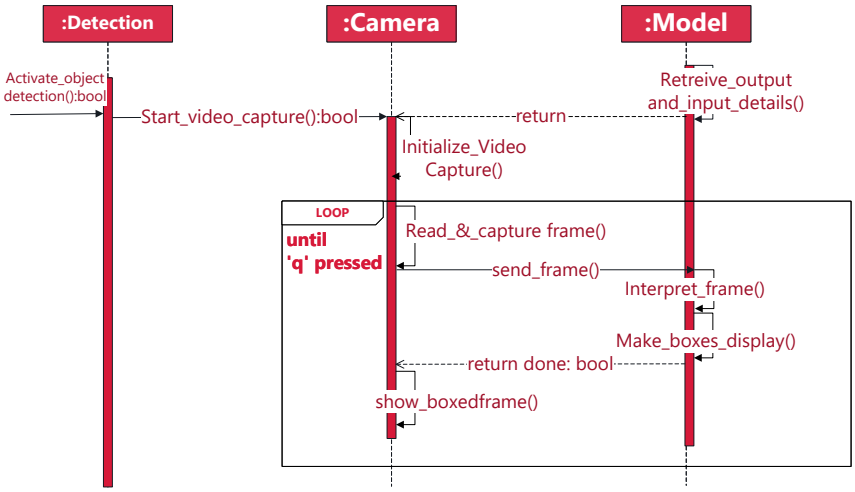


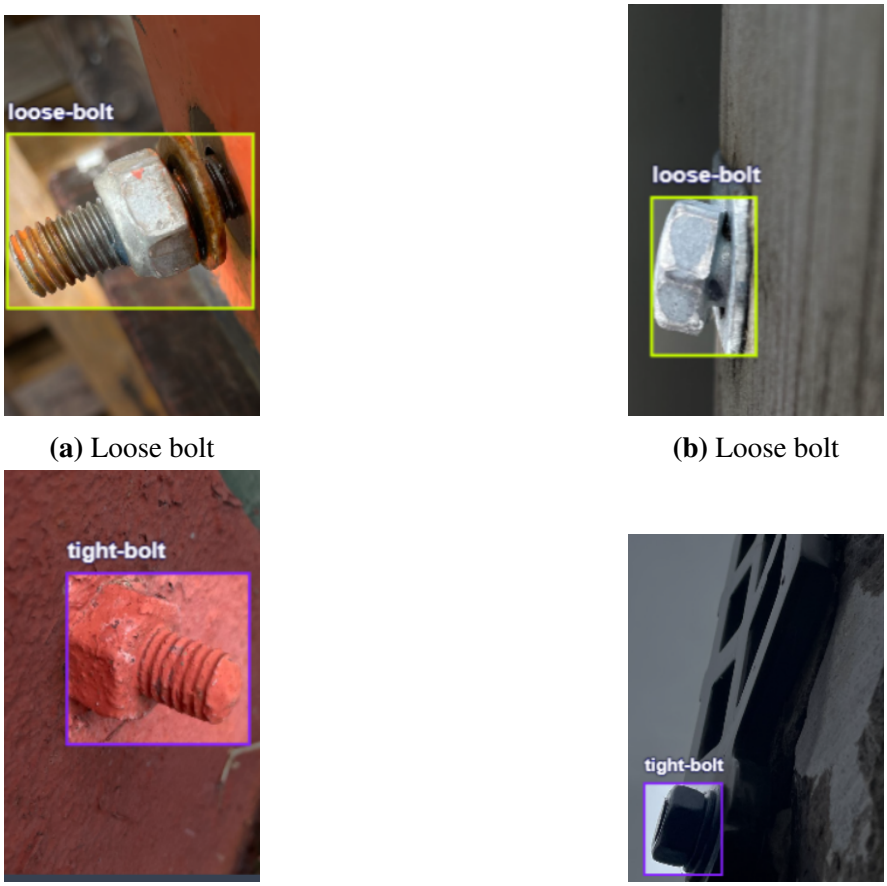
Figure M.3: Testing the custom-made model

M.4 Previous bolt detection dataset

AD | *SO*

As mentioned in 4.6, our group took upon an additional task of automated bolt inspection and fastening. In an initial exploration of this task, a dataset was constructed by capturing images of bolts found in the local environment. These images were categorized as either “loose” or “tight” bolts and subsequent labelling and testing were conducted. 82 pictures were collected, and after data augmenting with saturation and noise, the dataset consisted

of 198 images. However, the resulting model performance proved to be sub-optimal, exhibiting signs of overfitting due to limited variation within the dataset. Furthermore, the practical applicability of this dataset was questioned as the robot's fastening capabilities were not suitable for the small bolt sizes predominantly found in the collected images. The dataset can be found here :(robowflow/old-bolt-dataset).



(a) Loose bolt

(b) Loose bolt

(c) Tight bolt

(d) Tight bolt

Figure M.4: Example images from dataset

Consequently, a decision was made to abandon this initial dataset. Instead, a new dataset was generated utilizing 3D-printed bolts, offering greater control over bolt size and configuration. The concept of differentiating between "loose" and "tight" bolts was also discarded due to challenges in reliably distinguishing these states based on visual cues alone, especially given the varied orientations in which bolts might be encountered in the robot's operating environment. Moreover, the ability of the robotic arm to successfully tighten a bolt depended heavily on the initial positioning and alignment of the bolt, which further complicated the task of automating the process.
Therefore, the focus shifted towards a simpler yet more achievable objective: developing a model capable of accurately detecting the presence of a bolt, regardless of its tightness. This streamlined approach offered a more practical and robust solution for the robot's intended functionality. This approach aimed to address the shortcomings of the previous dataset and provide a more reliable dataset for training an object detection model tailored to the specific requirements of the project. Read more about the final implementation in 4.6.4.

N Implementation of detection caching

Every time the endpoint is requested (think while true) a new frame is read from the camera. This frame gets saved inside the class. There is a thread running all the time which analyzes the currently saved frame and the result of this analysis is also saved within the class. So when the actual frame is given as a response to the request the currently saved result of the analysis is added onto the frame before sending it.

This code can be improved by checking if the current frame within the `def detect_in_frame` is equal to the last, this way the code does not analyze a frame which is already analyzed. If there is no new frame set (the VR application is off for instance), the current code will continue to analyze the current frame over and over even though this frame does not change, and the result would be the same all the time, therefore wasting resources.

```
from threading import Thread

import cv2

from flask import Flask, Response

import numpy as np

import tflite_runtime.interpreter as tflite


# from PIL import Image

app = Flask(__name__)
# Load model and labels
model_path = "/home/pi/Obj-detection-pi/custom-transfer-learning/tflite_models/
people_detection_2.tflite"
label_path = "/home/pi/Obj-detection-pi/custom-transfer-learning/labels/labels.txt"
interpreter = tflite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()


# Function to load labels from the labels file
def load_labels(label_path):
    """Loads the labels file. Supports files with or without index numbers.

    Args:
        label_path: path to the labels file.

    Returns:
        A list with the labels.

    If the file contains index numbers, then the index number is removed from the
    label.
    """
```

```
    with open(label_path, "r") as file:
        labels = [line.strip() for line in file.readlines()]
    return labels


labels = load_labels(label_path)


cap = cv2.VideoCapture(-1)


class AIBox:

    def __init__(self):
        self.ret = None
        self.frame = None
        self.num_detections = 0
        self.scores = {}
        self.threshold = 0.4
        self.boxes = {}
        self.classes = {}
        self.height = 480
        self.width = 640

    def set_frame(self, ret, frame):
        self.ret = ret
        self.frame = frame

    def get_drawn_frame(self) -> bytes:
        """Draw bounding boxes on the frame.

        Args:
            frame: the frame to draw on.
            num_detections: the number of detections.
            boxes: the bounding boxes.
            classes: the class of the detected object.
            scores: the confidence scores of the detected object.
            labels: the labels of the detected object.
            threshold: the confidence threshold to use.
        """

        for i in range(self.num_detections):
            if self.scores[i] < self.threshold:
                continue

            y_min, x_min, y_max, x_max = self.boxes[i]
            x_min = int(x_min * self.width)
            x_max = int(x_max * self.width)
            y_min = int(y_min * self.height)
            y_max = int(y_max * self.height)
```

```
            cv2.rectangle(self.frame, (x_min, y_min), (x_max, y_max),
            (0, 255, 0), 2)

            object_name = labels[int(self.classes[i])]
            label = "%s" % (object_name)
            label_size, base_line = cv2.getTextSize(
                label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2
            )  # Get font size
            y_min = max(y_min, label_size[1])
            cv2.rectangle(
                self.frame,
                (int(x_min), int(y_min - round(1.5 * label_size[1]))),
                (int(x_min + round(1.5 * label_size[0])), int(y_min + base_line)),
                (255, 255, 255),
                cv2.FILLED,
            )
            cv2.putText(
                self.frame,
                label,
                (int(x_min), int(y_min)),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.7,
                (0, 0, 0),
                2,
            )

        # Convert the frame to JPEG format
        ret, buffer = cv2.imencode(".jpg", self.frame)

        if not ret:
            return None

        return buffer.tobytes()
        # do calculation

    def detect_in_frame(self):
        height = input_details[0]["shape"][1]
        width = input_details[0]["shape"][2]
        # Prepare the frame for model input

        input_frame = cv2.resize(self.frame, (width, height))
        input_frame = np.expand_dims(input_frame, axis=0).astype(np.uint8)

        # Run inference
        interpreter.set_tensor(input_details[0]["index"], input_frame)
        interpreter.invoke()

        self.num_detections = int(interpreter.get_tensor(output_details[2]["index"]
        self.scores = interpreter.get_tensor(output_details[0]["index"])[0]
        self.boxes = interpreter.get_tensor(output_details[1]["index"])[0]
```

```
        self.classes = interpreter.get_tensor(output_details[3]["index"])[0]

    def analyze(self):
        while True:
            if self.frame is None or self.ret is None:
                continue

            self.detect_in_frame()


ai_box = AIBox()
thread = Thread(target=ai_box.analyze, daemon=True)
thread.start()


def get_frame() -> tuple:
    ret, frame = cap.read()
    return ret, frame


@app.route("/snapshot")
def snapshot():
    ret, frame = get_frame()
    ai_box.set_frame(ret, frame)

    if frame is None:
        return "Failed to capture frame", 400
    return Response(ai_box.get_drawn_frame(), mimetype="image/jpeg")


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

O Robot: Detailed Implementation and Configuration

O.1 Example code a ROS 2 node

OM | *AEH*

A basic Python ROS 2 node that fills in a message and publishes it to a topic can be seen below.

```
# example_publisher
from interfaces.msg import RobotData

import rclpy
from rclpy.node import Node


class MyNode(Node):

    def __init__(self):
        super().__init__(self.__class__.__name__)

        self.pub_robot_data = self.create_publisher(
            RobotData, "robot_data", 1
        )

        self.create_timer(1, self.publish_robot_data)

    def publish_robot_data(self):
        msg = RobotData()
        msg.gyroscope.x = 1.0
        msg.gyroscope.y = -2.0
        msg.gyroscope.z = 3.0
        msg.voltage = 12.4
        # more data...

        self.pub_robot_data.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

First, the interfaces and the **rclpy** Python package are imported. **rclpy** is the Python Application Programming Interface (API) for ROS 2, enabling ROS functionality. The `class MyNode` is the ROS node that publishes to the `robot_data` topic with the interface type RobotData. The `1` indicates the Quality of Service (QoS) policy [166]. Without any subscriptions or loops, this node would not publish anything, so a timer is created to execute the `def publish_robot_data` method every second. Inside this method, a message is created, populated with data, and then published to the `robot_data` topic. The main method initializes the node and keeps it running using the `spin(node)` function.

This node cannot receive data externally as it does not subscribe to anything. An example of a subscriber written in Python is shown below.

```
# example_subscriber
from interfaces.msg import RobotData

import rclpy
from rclpy.node import Node


class MySecondNode(Node):

    def __init__(self):
        super().__init__(self.__class__.__name__)

        self.sub_robot_data = self.create_subscription(
            RobotData, "robot_data", self.handle_robot_data, 1
        )

    def handle_robot_data(self, msg):
        voltage, mode = msg.voltage, msg.mode
        print(f"Received {voltage=}, {mode=}")

        # do something with the data...


def main(args=None):
    rclpy.init(args=args)
    node = MySecondNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

This node works similarly to the publisher but instead subscribes to the `robot_data` topic. Every time a message is sent to this topic, the `def handle_robot_data` method is triggered, which prints some of the data to the terminal.

O.2 Example of Rosmaster Library code modifications OM | *AEH*

This is the original code created by Yahboom, which assigns a particular angle to one of the servo motors in the robotic arm.

```
# Set bus steering gear Angle interface: s_id:[1,6], s_angle:
# 1-4:[0, 180], 5:[0, 270], 6:[0, 180], set steering gear to
# move to the Angle.
# run_time indicates the running time (ms). The shorter the time,
# the faster the steering gear rotates.  The minimum value is 0
# and the maximum value is 2000
def set_uart_servo_angle(self, s_id, s_angle, run_time=500):
    try:
        if s_id == 1:
```

```
            if 0 <= s_angle <= 180:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_1 set error!")
        elif s_id == 2:
            if 0 <= s_angle <= 180:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_2 set error!")
        elif s_id == 3:
            if 0 <= s_angle <= 180:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_3 set error!")
        elif s_id == 4:
            if 0 <= s_angle <= 180:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_4 set error!")
        elif s_id == 5:
            if 0 <= s_angle <= 270:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_5 set error!")
        elif s_id == 6:
            if 0 <= s_angle <= 180:
                value = self.__arm_convert_value(s_id, s_angle)
                self.set_uart_servo(s_id, value, run_time)
            else:
                print("angle_6 set error!")
    except:
        print('---set_uart_servo_angle error! ID=%d---' % s_id)
        pass
```

The absence of type annotations and the lack of comments in the form of a docstring make it challenging for users to understand the limitations of each servo without delving into the source code. By incorporating docstrings, users could conveniently access the documentation by hovering over the function name in their IDE. Moreover, the function's logic appears repetitive and less adaptable to potential changes.

The revised code maintains the same functionality while reducing redundancy. It includes type hints and a docstring to facilitate user comprehension during development. Additionally, the `try`/`except` block could be eliminated, as the only potential failure point within the `self.set_uart_servo` function is already covered by `try`/`except` functionality.

```
def set_uart_servo_angle(
```

```
        self, s_id: int, s_angle: int, run_time: int = 500
    ) -> None:
        """Sets the angle of one of the servos (on the arm).
        Degrees: 1-4, 6:[0, 180], 5:[0, 270]

        1 - Rotation of the arm
        2 - Shoulder of the arm
        3 - Elbow of the arm
        4 - Tilt of the arm
        5 - Wrist of the arm (rotation)
        6 - Fingers of the arm (pinch)

        Args:
            s_id: the id of the servo
            s_angle: the angle of the servo
            run_time: the running time in milliseconds
        """
        for i in range(1, 6 + 1):
            if s_id != i:
                continue

            # out of range for all servos
            if 0 > s_angle or s_angle > 270:
                print(f"angle_{i} set error!")
                return

            # only servo 5 can rotate up to 270 degrees
            if s_angle > 180 and i != 5:
                print(f"angle_{i} set error!")
                return

            value = self.__arm_convert_value(s_id, s_angle)
            self.set_uart_servo(s_id, value, run_time)
            return
```

O.3 Installing Docker

OM | *AEH*

To install Docker on RPiOS 64-bit, the following script was used.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg
    -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] h
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
```

```
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
    docker-buildx-plugin docker-compose-plugin -y
```

O.4 Permissions to use Docker

OM | *AEH*

To avoid the need to use sudo every time when executing Docker commands, the following commands were utilized to mitigate this issue.

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

O.5 Dockerfile

OM | *AEH*

Below is the Dockerfile used, which includes all the necessary packages and libraries required for the Docker container and the robot to function. For a comprehensive list of all Python packages, see O.6.

```
FROM arm64v8/ros:humble

ARG USERNAME=ros
ARG USER_UID=1000
ARG USER_GID=${USER_UID}

# create a non-root user
RUN groupadd --gid ${USER_GID} ${USERNAME} \
    && useradd -s /bin/bash --uid ${USER_UID}
        --gid ${USER_GID} -m ${USERNAME} \
    && mkdir /home/${USERNAME}/.config &&
        chown ${USER_UID}:${USER_GID} /home/${USERNAME}/.config \
    && apt-get update \
    # give sudo
    && apt-get install -y sudo \
    && echo $USERNAME ALL=\(root\) NOPASSWD:ALL >
        /etc/sudoers.d/$USERNAME \
    && chmod 0440 /etc/sudoers.d/$USERNAME
RUN apt-get update --fix-missing && apt-get upgrade -y
RUN apt-get install ros-humble-magic-enum -y
RUN apt-get install ros-humble-camera-info-manager -y
RUN apt-get install ros-humble-vision-opencv -y
RUN apt-get install ros-humble-image-pipeline -y
RUN apt-get install libuvc-dev -y
RUN apt-get install nlohmann-json3-dev -y
RUN apt-get install libgoogle-glog-dev -y
RUN apt-get install python3-opencv -y
RUN apt-get install python3-pip -y

ENV SHELL /bin/bash

# install python packages
```

```
WORKDIR /robot
COPY requirements.txt .
RUN python3 -m pip install -r requirements.txt

WORKDIR /robot/src
```

O.6 requirements.txt

OM | *AEH*

Below are all the utilized package installer for Python (pip) [167] packages necessary for the robot to operate.

```
pyserial
numpy
brotli
pymongo
mediapipe
websockets
```

O.7 Expansion board symlink

OM | *AEH*

The following steps were carried out according to the instructions provided by Yahboom under "2. Python basic control", "2.0 Bind PCB port devices" using Method-1 [168].

Firstly, a rules file was created:

```
sudo nano /etc/udev/rules.d/expbrd.rules
```

Subsequently, a symlink was added based on vendor, product, and kernel:

```
KERNEL=="ttyUSB*", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="7523",
MODE:="0777", SYMLINK+="expbrd"
```

Ensuring that the symlink can be accessed:

```
sudo chmod a+x /etc/udev/rules.d/expbrd.rules
```

Finally, services were restarted for the changes to take place:

```
sudo udevadm trigger
sudo service udev reload
sudo service udev restart
```

O.8 MongoDB database implementation

O.8.1 Installing the Docker image

OM | *AEH*

MongoDB provides a Docker image, which can be installed using the following command:

```
docker pull mongodb/mongodb-community-server:latest
```

O.8.2 Running the container **OM** | *AEH*

To establish a connection with the database for data insertion or retrieval, the MongoDB image needs to run within a container. The container can be initiated with the following command:

```
docker run --name mongodb
    --network host -d mongodb/mongodb-community-server:latest
```

The container should operate using the host's network to enable connection. Additionally, the default port for MongoDB is 27017, which can be exposed if needed. Underlyingly, the container employs a volume or a similar technology to persist data, ensuring that data is retained even when the container is terminated. Without this persistence mechanism, data would be lost each time the container is terminated, including reboot of the system.

O.9 Measuring the speed of the robot

O.9.1 Speed at 0% **OM** | *AEH*

If speed is set to 0%, the robot stands still or $0 cm/s$.

O.9.2 Speed at 50% **OM** | *AEH*

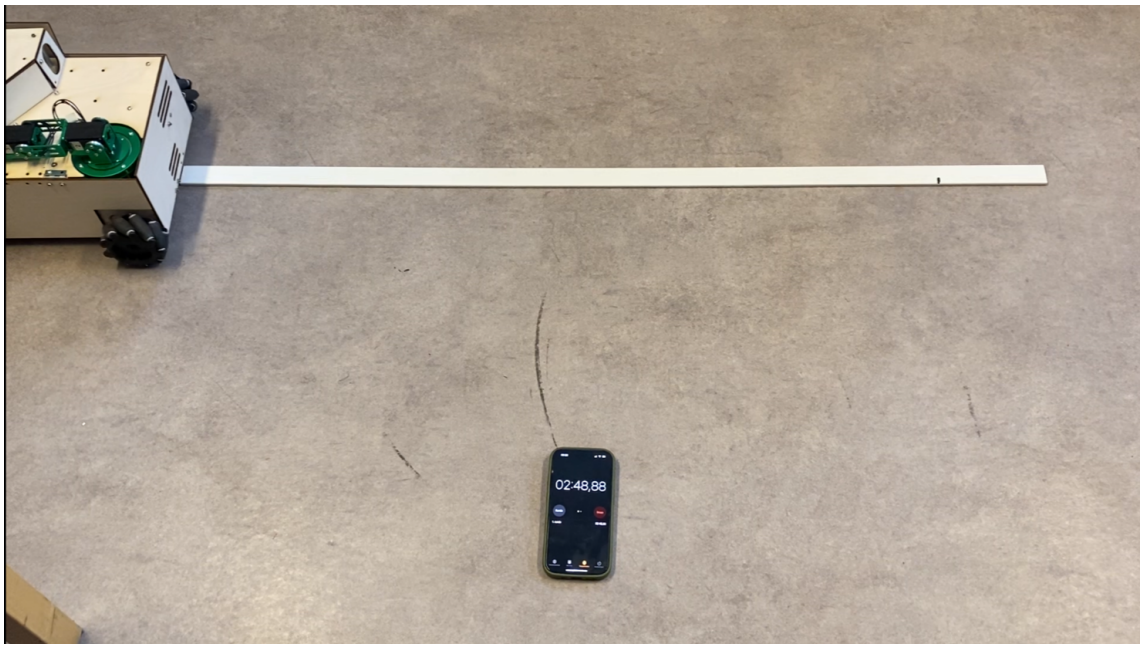


Figure O.1: 50% speed: start time

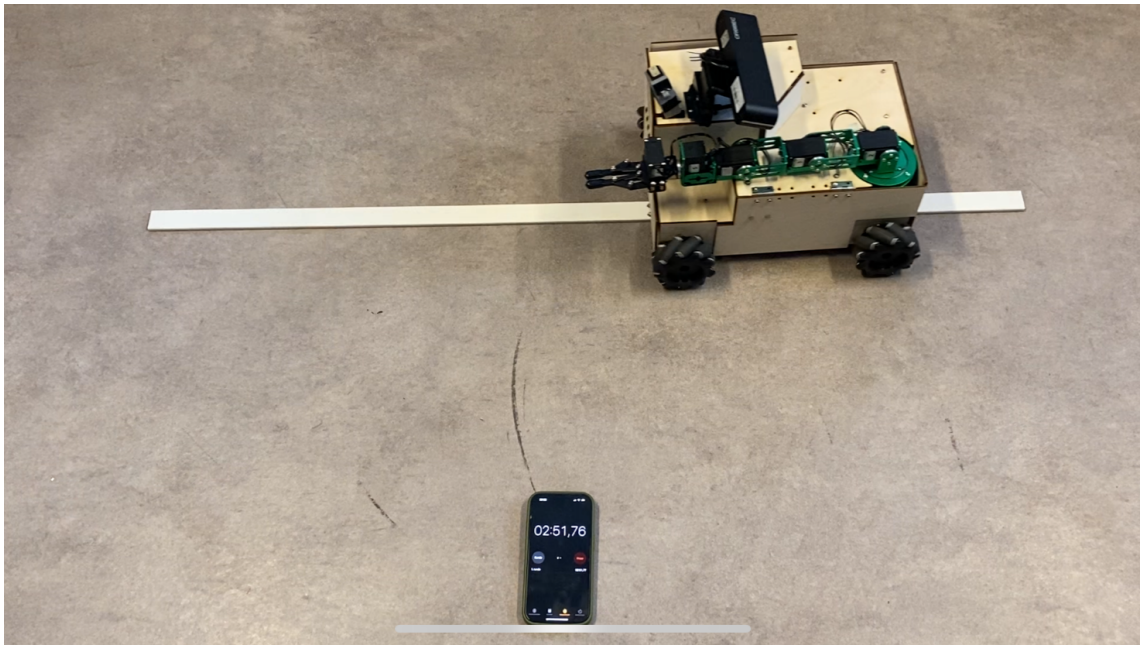


Figure O.2: 50% speed: end time

$$2:48,88 - 2:51,76 = 2,88$$
$$\frac{100cm}{2.88s} \approx 34.72cm/s$$

O.9.3 Speed at 100% OM | *AEH*

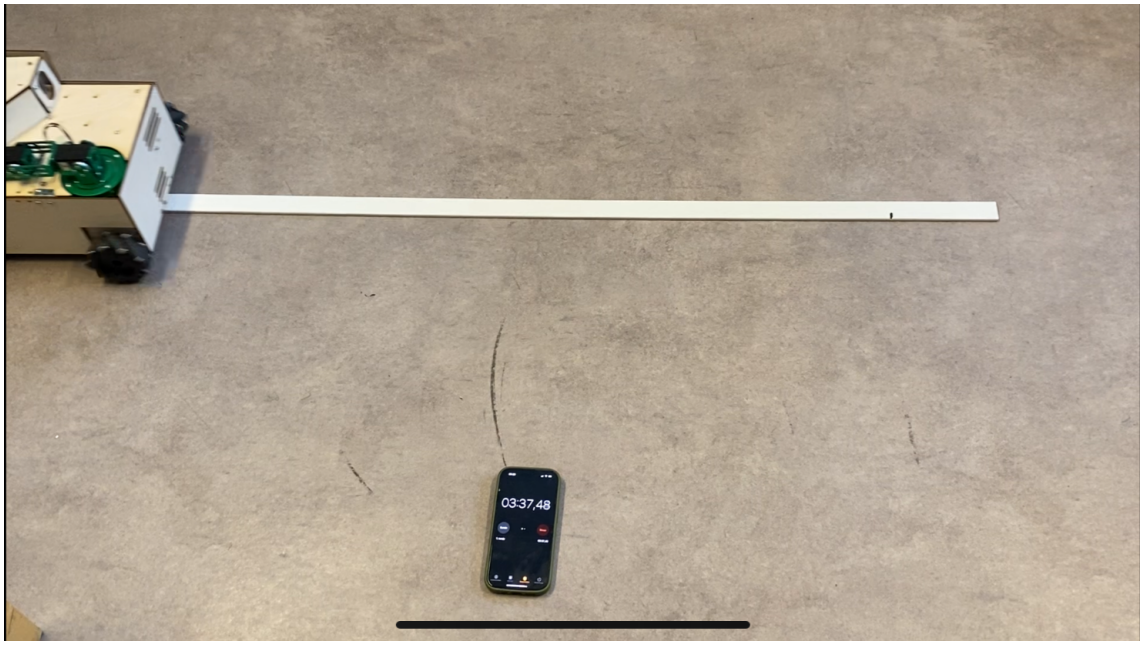


Figure O.3: 100% speed: start time

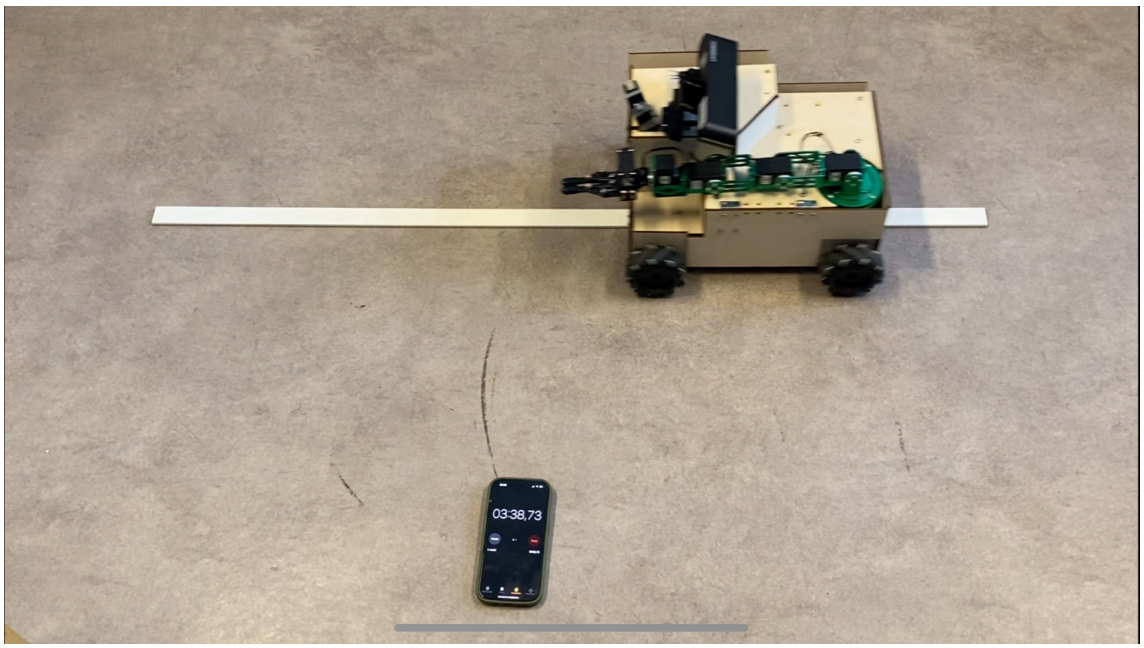


Figure O.4: 100% speed: end time

$$3:37,48 - 3:38,73 = 1,25$$

$$\frac{100cm}{1.25s} = 80cm/s$$

O.9.4 Errors with estimation **OM** | *AEH*

These tests were done prior to the completion of the final robot, which is heavier than the one used for measurements. This difference in weight may have had an impact on the robot's speed. It is anticipated that the current version of the robot is likely slower than the one tested during these trials. Furthermore, it should be noted that the measurements obtained are not highly accurate, but they are considered sufficient for estimation purposes. There were no specific requirements regarding the accuracy of speed estimation.

O.10 Finding arm camera angle **OM** | *AEH*

The camera arm angle was determined through a trial-and-error process while observing the screen with the connected camera. When satisfied with the angle, a picture was taken. Initially, the horizontal line was drawn on the image, and the image was rotated until this line was close to 0 degrees rotation. Subsequently, another line was added to the image, and the angle of this line was measured. All measurements were conducted using paint.net software.

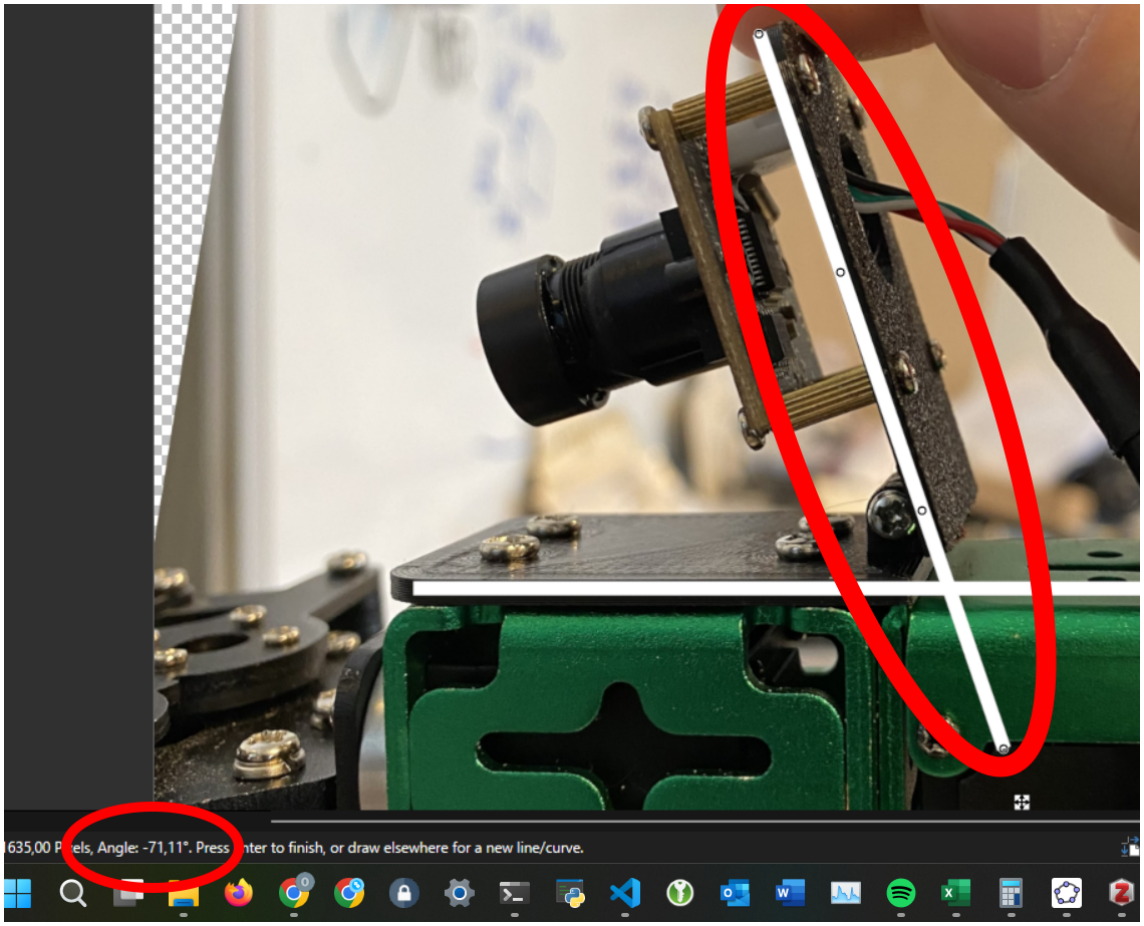


Figure O.5: Enter Caption

$$-71,11 + 180 \approx 109$$

In this representation, the horizontal line corresponds to 0 degrees, while the angle of 109 degrees is positively oriented counterclockwise.

O.11 Finding and drawing end effector point

OM | *AEH*

To address the discrepancy between the centre of the image and the location of the end effector, this point was determined.

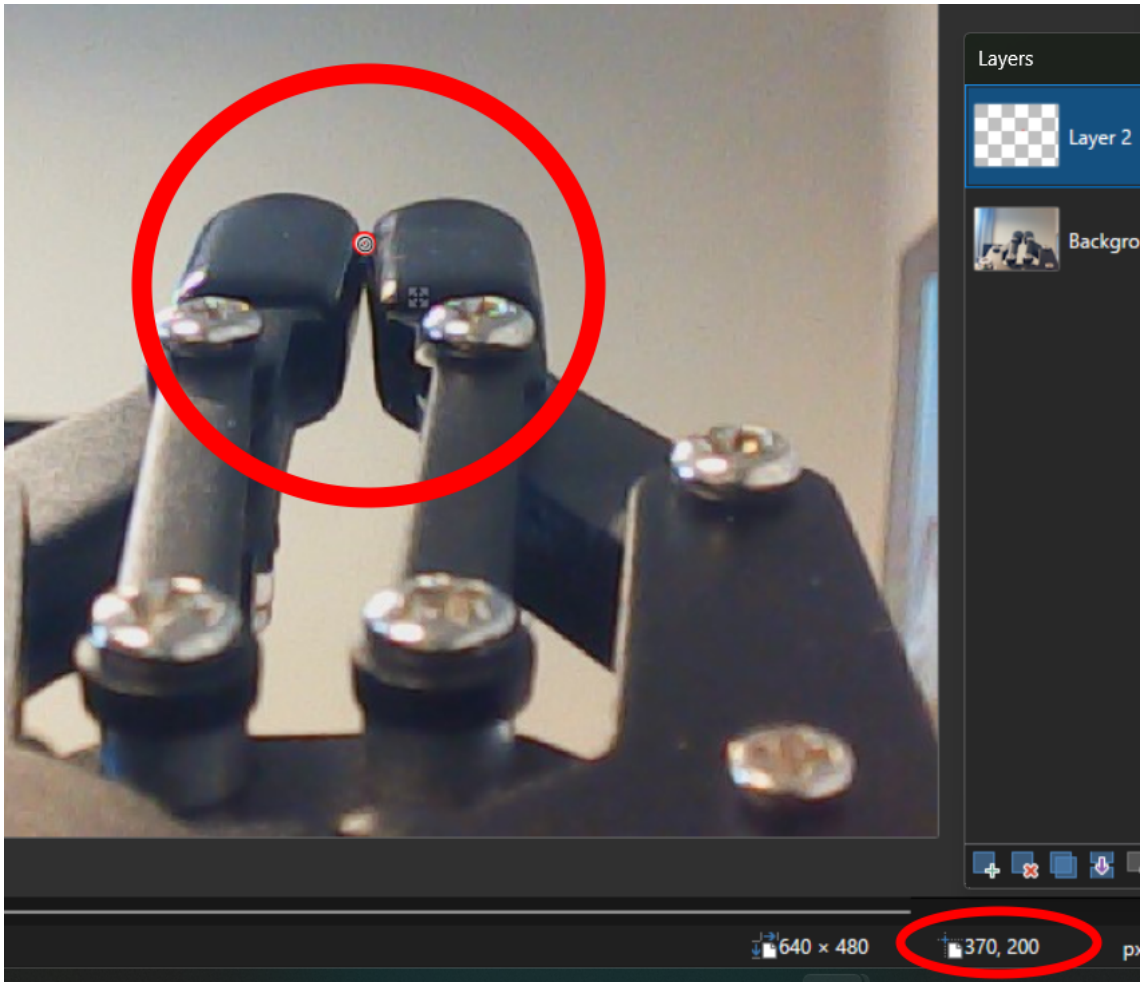


Figure O.6: Enter Caption

The identified point was found to be (370, 200), measured from the top left corner, consistent with OpenCV's coordinate system [169]. These coordinates were utilized to plot the dot in the camera feed when in arm mode.

O.12 Mapping servo ids to names

OM | *AEH*

when the angles are gathered the json looks like this

```
{
    "1": 170,
    "2": 90,
    "3": 80,
    "4": 70,
    "5": 60,
    "6": 50,
}
```

by having an enum with the stored names and IDs, the mapping can be done dynamically as shown below

```
class Arm(enum.IntEnum):
    ROTATION = 1
    SHOULDER = 2
```

```
    ELBOW = 3
    TILT = 4
    WRIST = 5
    PINCH = 6

class Controller(Node):
    ...

    def get_arm_angles(self) -> None:
        """Gets the arm angles and publishes them."""
        msg = ArmAngles()
        angles = self.controller.robot.get_stored_angles()

        for servo in angles.copy():
            arm_name = Arm(int(servo)).name.lower()
            value = angles[servo]
            setattr(msg, arm_name, int(value))

        self.pub_arm_angles.publish(msg)
```

the message published will then look like what shown below if the angles are as shown in the json above.

```
ArmAngles()
```

O.13 Mesh plotting code

OM | *AEH*

Below is the code for creating mesh objects using Open3D. However, during testing on theRPi, this code was inconsistent in producing `.obj` files.

```
import open3d as o3d
import numpy as np


# Convert point cloud data to NumPy array
point_cloud_array = np.array(
    [-1.2974001169204712, -0.9725425243377686, 2.31600022315979],
    [-1.2933393716812134, -0.9725425243377686, 2.31600022315979],
    ...
)

# Create a point cloud object
point_cloud = o3d.geometry.PointCloud()
point_cloud.points = o3d.utility.Vector3dVector(point_cloud_array)

# Compute normals
point_cloud.estimate_normals(
    search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.1, max_nn=30)
)

# Run Poisson surface reconstruction
```

```
mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(
    point_cloud, depth=9
)

# Simplify the mesh
mesh = mesh.simplify_quadric_decimation(target_number_of_triangles=5_000)

# Save the mesh as a .obj file
o3d.io.write_triangle_mesh("mesh.obj", mesh)
```

O.14 Dealing with different paths

OM | *AEH*

Issues arise from the structure outlined in 4.3.1, particularly when locating specific imports during unit testing.

```
robot/src/
    interfaces/
        msg/
            RobotData.msg
            VRDrive.msg
    master/
        master/
            __init__.py
            master_node.py
        test/
            __init__.py
            test_master.py
```

The issue stems from the fact that during regular code execution, the interfaces are found and function correctly. However, during unit testing, the path to the interfaces is not resolved correctly. This was addressed by employing a try-except block and setting a variable within the master node being tested.

```
try:
    from interfaces.msg import RobotData, VRDrive
    UNITTEST = False
except ModuleNotFoundError:
    UNITTEST = True
```

As a result of these challenges, most of the nodes are structured such that the logic is separated into another file that does not import the interfaces directly.

```
robot/src/
    controller/
        controller/
            __init__.py
            controller_node.py
            controller.py
            ... more files
        test/
            __init__.py
            test_controller.py
```

O.15 Flask video stream implementation **OM** | *AEH*

Originally, the solution opted for involved hosting a HTTP server using Flask. This server provided a `/snapshot` endpoint, which served the most recent frame captured by the camera. By having the VR headset request this GET endpoint frequently, it effectively worked as a video stream. However, this approach had several drawbacks. Firstly, it introduced considerable overhead and unnecessary delays due to the amount of requests. Additionally, it may have contributed to a memory leak within the VR application, ultimately leading to crashes after prolonged use.

The server was implemented to run in its own thread. This was necessary because the `rcply.spin(node)` function is blocking, preventing other operations from executing, the same goes for the running of the Flask server.

O.16 Missing packages for the Astra driver **OM** | *AEH*

The specified packages were:

```
ros-humble-magic-enum
libuvc-dev
```

Meanwhile, the missing packages included:

```
ros-humble-camera-info-manager
ros-humble-vision-opencv
ros-humble-image-pipeline
nlohmann-json3-dev
libgoogle-glog-dev
```

Identifying and resolving these missing dependencies took some time. It involved examining the source code, building, running, and debugging to pinpoint where crashes occurred. Additionally, determining the correct ROS packages added another layer of complexity to the task. Adjustments were also made to the CMake file, to get the Astra node to work.

O.17 Docker robot container options **OM** | *AEH*

```
docker run --rm -it --user ros --network host --name robot
    -v /dev/:/dev -v $PWD/robot:/robot ros2_humble
```

The Docker image named `ros2_humble` is used in the command. The `--rm` flag is specified to remove the container after exiting, preventing stopped containers from accumulating. The `-it` flag indicates that the container should be interactive with the command line. The container is named “robot”. Two volumes are shared: the `/robot/` directory containing the source code, and the `/dev/` directory providing access to connected USB devices and alike.

O.18 Building ROS 2 from source **OM** | *AEH*

Using a community solution, ROS 2 was successfully compiled from source and operational on RPiOS 64-bit [170]. However, attempting the same steps on a second RPi did not result in a successful installation. The decision to opt for a Docker solution was made

instead.

The compilation process on aRPi 5 8GB took approximately 2-3 hours.

```
sudo apt install -y git colcon python3-rosdep2 vcstool wget
    python3-flake8-docstrings python3-pip python3-pytest-cov
    python3-flake8-blind-except python3-flake8-builtins
    python3-flake8-class-newline python3-flake8-comprehensions
    python3-flake8-deprecated python3-flake8-import-order
    python3-flake8-quotes python3-pytest-repeat
    python3-pytest-rerunfailures python3-vcstools
mkdir -p ~/ros2_humble/src
cd ~/ros2_humble
vcs import
    --input https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos src
sudo rm /etc/ros/rosdep/sources.list.d/20-default.list
sudo apt upgrade
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src -y
    --skip-keys "fastcdr rti-connext-dds-6.0.1 urdfdom_headers"
colcon build --symlink-install
```

O.19 Building code documentation

OM | *AEH*

The `docs/` folder is placed outside of the shared volume for the robot container. This means that when entering the container, this folder is inaccessible. Since the container is the only place where the packages are installed, mock packages must be specified to build the documentation without errors. This is configured in the `conf.py` file that Sphinx reads. The mock packages are as follows:

```
autodoc_mock_imports = [
    "rclpy",
    "interfaces",
    "numpy",
    "cv2",
    "sensor_msgs",
    "flask",
    "brotli",
    "pymongo",
    "mediapipe",
    "serial",
    "websockets",
]
```

Another issue arises from the implementation of function decorators, which are used to avoid repetitive code while maintaining the same security. A function decorator might look something like this:

```
def in_production_mode(func):
    def function_wrapper(self, *args, **kwargs):
```

```
        if self.production:
            return func(self, *args, **kwargs)

        raise NotInProductionMode("Enable production mode")

    return function_wrapper
```

The Python interpreter runs the decorator before the actual function. If the decorator `@in_production_mode` returns nothing or throws an exception, the code inside the `set_arm_shoulder` method is never executed.

```
class Robot:
    ...

    @in_production_mode
    def set_arm_shoulder(self, angle: int) -> None:
        """Sets the arm shoulder angle.

        Args:
            angle: angle
        """
        self.ros_master.set_uart_servo_angle(Arm.SHOULDER, angle)
```

However, when building the documentation with Sphinx, there are difficulties in determining the arguments of the original method due to the decorator obscuring them. This leads to the generated documentation missing crucial information. The result is:

```
set_arm_shoulder(*args, **kwargs)
```

The arguments should be `angle: int`, not `*args, **kwargs`, which are the arguments of the `def function_wrapper`. Additionally, due to the incorrect arguments, the formatted docstring is not shown. To resolve this, all wrapper functions need to include `@functools.wraps(x)`.

```
def in_production_mode(func):
    @functools.wraps(func)
    ...
```

This adjustment ensures that the code documentation builds successfully and correctly displays the method signature and docstring.

```
set_arm_shoulder(angle: int) → None
    Sets the arm shoulder angle.

    PARAMETERS:
        angle - angle
```

O.20 Robot repository timeline & lines of code

OM | *AEH*

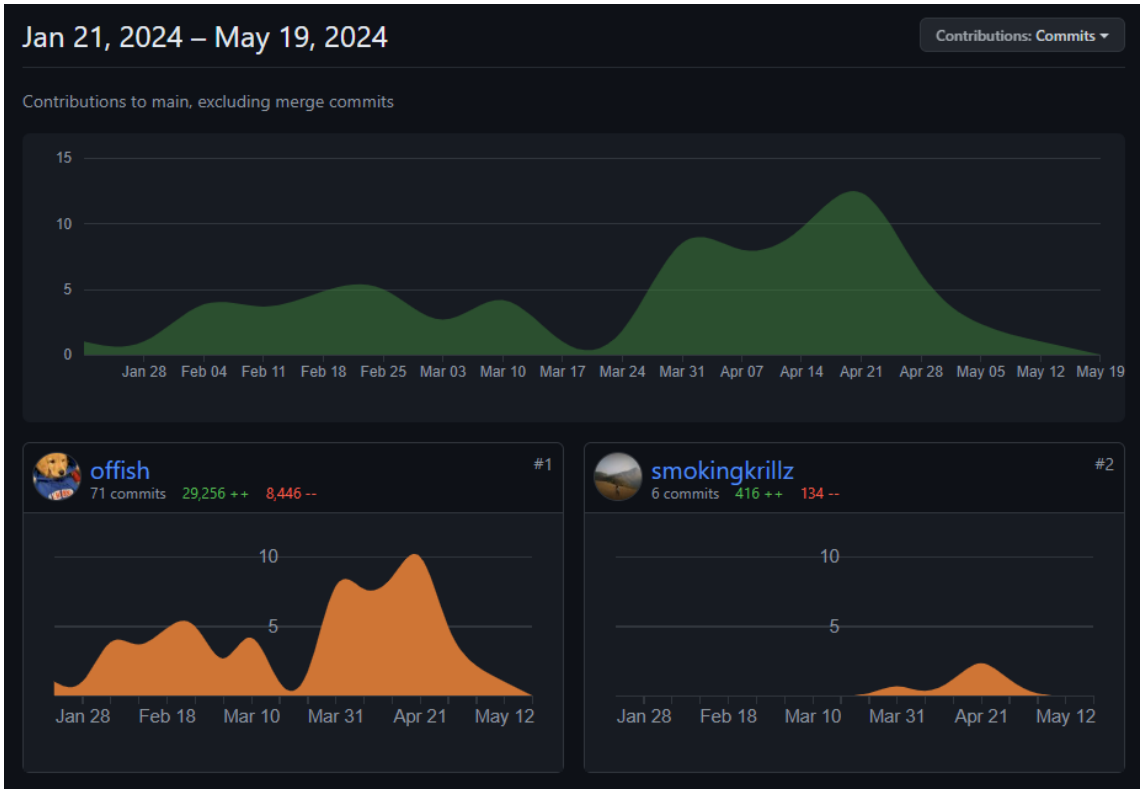


Figure O.7: Commits timeline [17]

The aliases “smokingkrillz” and “offish” correspond to Aditi and Oscar, respectively.

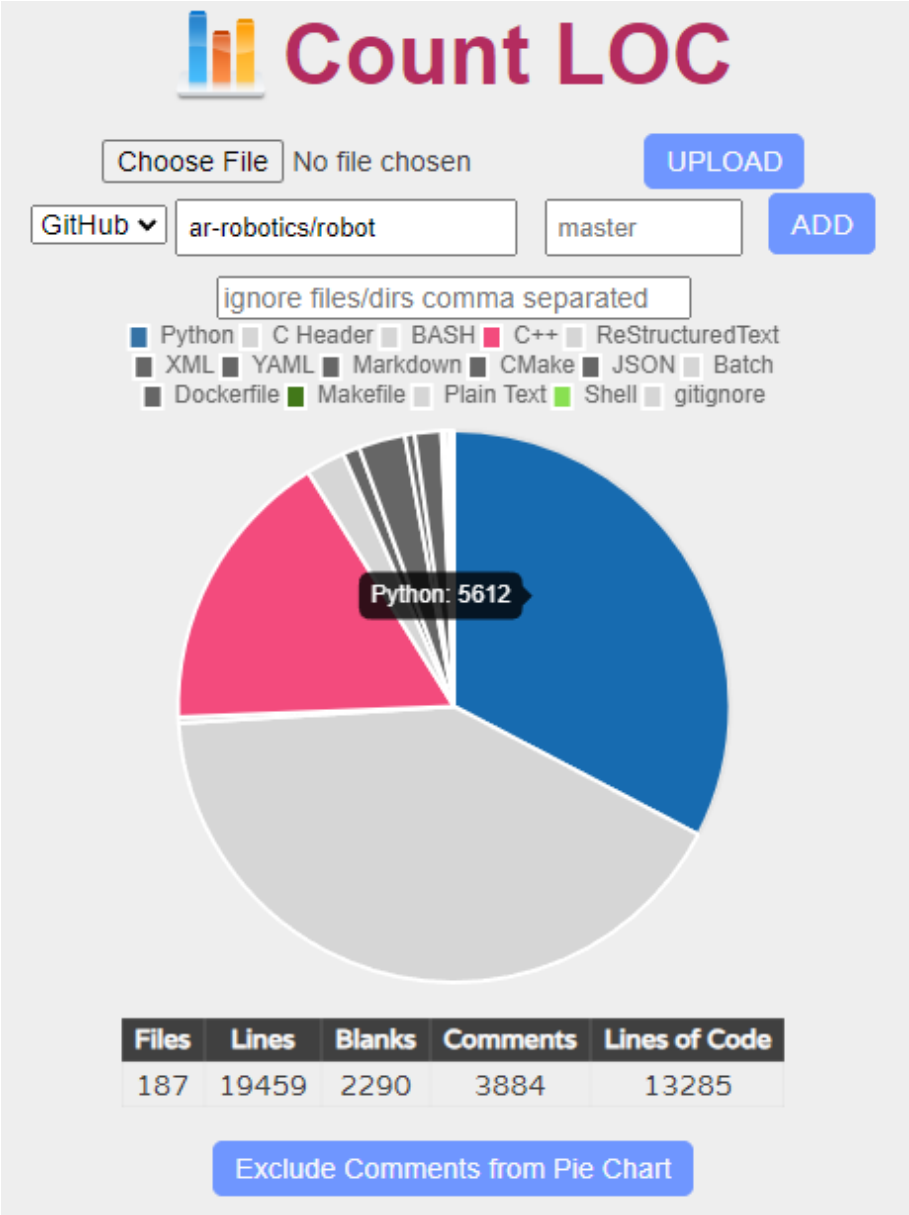


Figure O.8: Lines of code [18]

Most of the C/C++ code is from the Astra driver as mentioned in 4.6.1.

P VR application

The VR application is the visual interface for controlling the robot in a 3D space using a headset. It contains visual elements such as screens, menus, and control elements to interact with the robot through hand gestures. The main responsibility of the application is to translate hand movements from the operator into interface-specific data and send it to the robot through the robot control logic and provide the operator an intuitive way to control the robot.

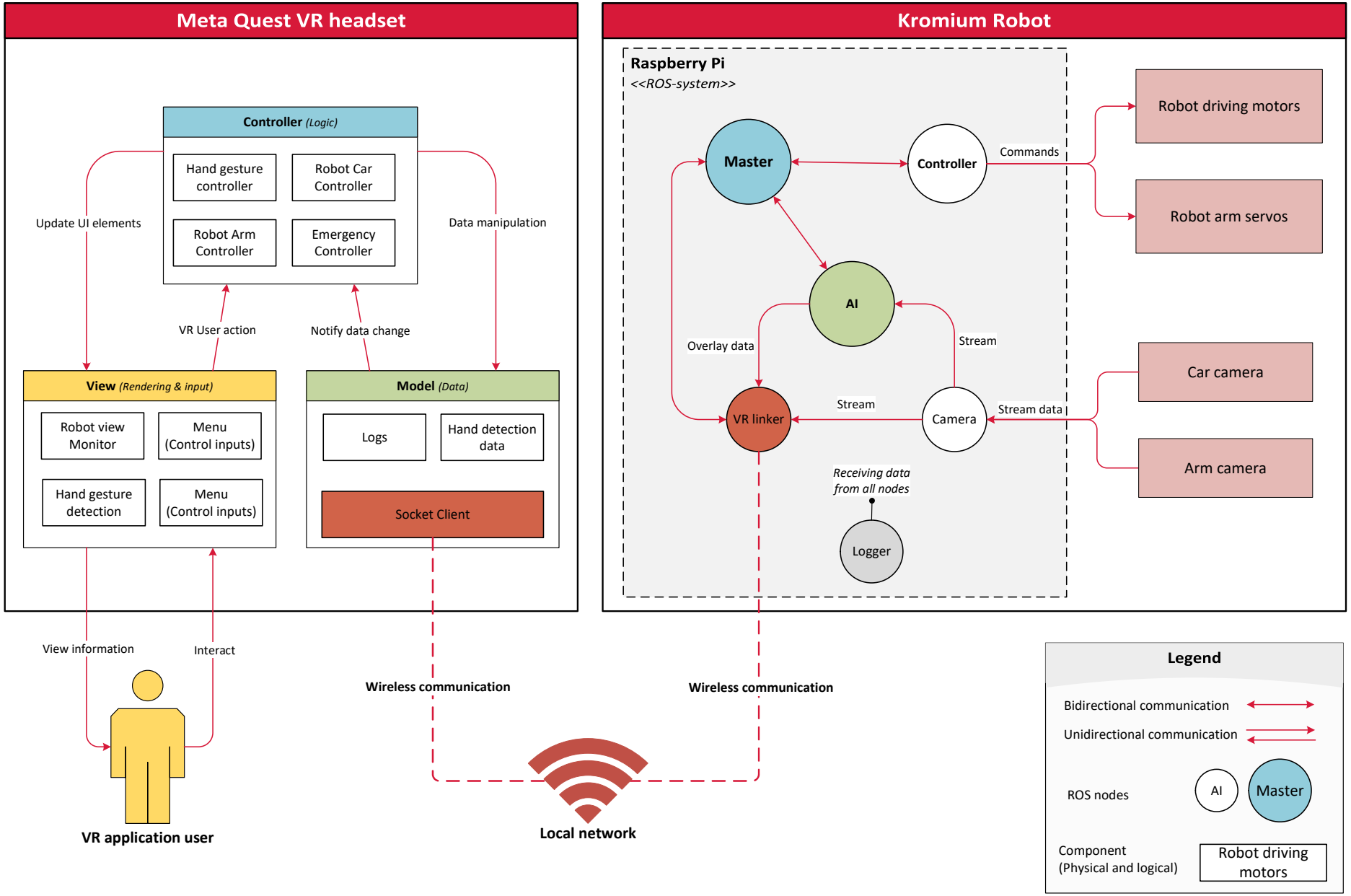


Figure P.1: High-level system architecture

P.1 Unity Scene and Game Object Elements **SO** | *AD*

This subsection provides an overview of key elements in Unity that are fundamental to building VR applications, including scenes, game objects, and prefabs.

P.1.1 Scene **SO** | *AEH*

A scene in VR applications is the graphical space that contains all or part of the application. Figure P.2 shows the default scene in Unity that shows a camera and directional light. A scene is where one builds application contents such as buttons, screens, text, menus, and other custom-designed graphical elements. An application can contain a simple scene, while large applications can contain multiple scenes representing different environments, for example, if one creates a game inside a house with multiple rooms, one can create a scene for each room [19].

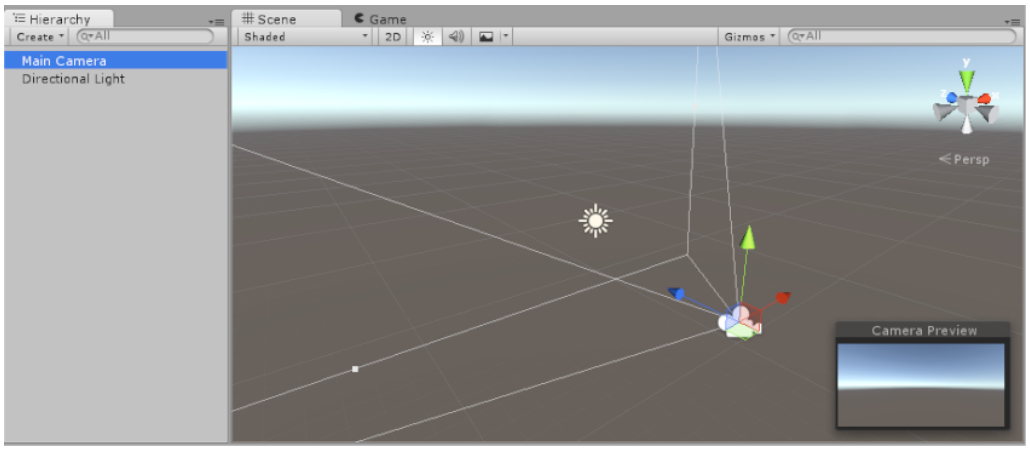


Figure P.2: Default Unity empty scene [19]

P.1.2 Game objects **SO** | *HB*

In Unity, designed primarily for game application development, every object in the application is a game object. A game object is the fundamental element in Unity scenes and can represent a wide range of entities, from characters to advanced 3D models such as robots. However, a game object on its own is inert; it acts as a container for components that implement functionality [20]. It is a common misconception that game objects only represent visual elements. A game object can also represent purely software components, which can provide interfaces and functionalities that other game objects or components can utilize. A prime example of such an object is a singleton network manager class. This class can be added as a component to a non-visual game object, allowing other objects to access the singleton instance for communication purposes. Examples of visual game objects can be seen in Figure P.3.

P.1.3 Prefab **SO** | *HB*

A prefab in Unity represents a modular and reusable asset that encapsulates a game object along with its components, properties, values, and child objects. Prefabs serve as predefined and configured objects from which developers can instantiate instances across various scenes or projects. Any game object in Unity can be converted into a prefab, facilitating uniform behaviour and characteristics across different scenes and projects [21]. An example of a prefab representing a tree is shown in Figure P.4.

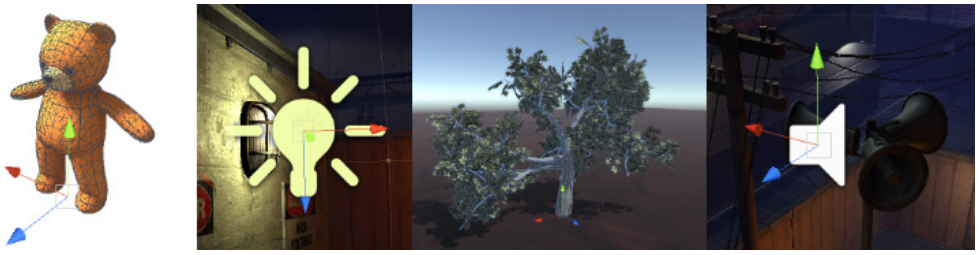


Figure P.3: Different game objects: animated character, a light, a tree, and an audio source [20]

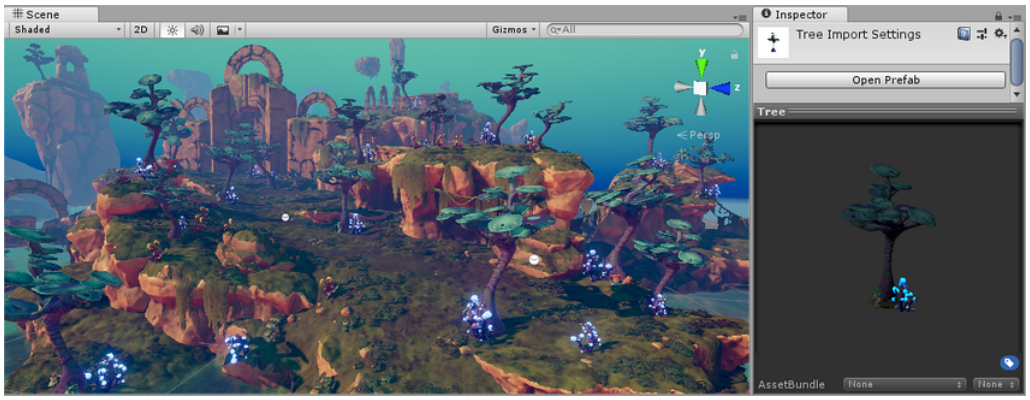


Figure P.4: Tree prefab example in a scene [21]

P.2 First Two Iterations

SO | *AD*

The first two iterations for the development of the VR application are detailed here. These iterations cover the initial stages of the project, including setup, foundational elements, and primary functionalities.

P.2.1 First iteration: Initial design

SO | *AD*

Figure P.5 illustrates the initial iteration. In this phase, we successfully established communication between the VR application and the robot, including showing an external video feed inside the VR application, tracking the user's hand movements, and converting the tracked motion into driving commands for the robot. The learning process was also an important aspect of the iteration, given that the group lacked experience in creating VR applications or using C# as the programming language. The development of each feature involved programming on both the VR and the robot components, followed by multiple testing and debugging sessions. We proceeded to the next feature only after the current system had passed all tests.

P.2.2 Conceptualization

SO | *AEH*

Meta Quest 3 offers various functionalities through different SDK. Hand tracking is one of the features provided by the Interaction SDK which grants VR applications access to a wide range of user data such as estimated hand size and hand pose data. [171]. Using the Interaction SDK, we were able to track the hand motion and bring to life the initial concept of converting tracked hand motion into driving commands, as illustrated in Figure P.6.

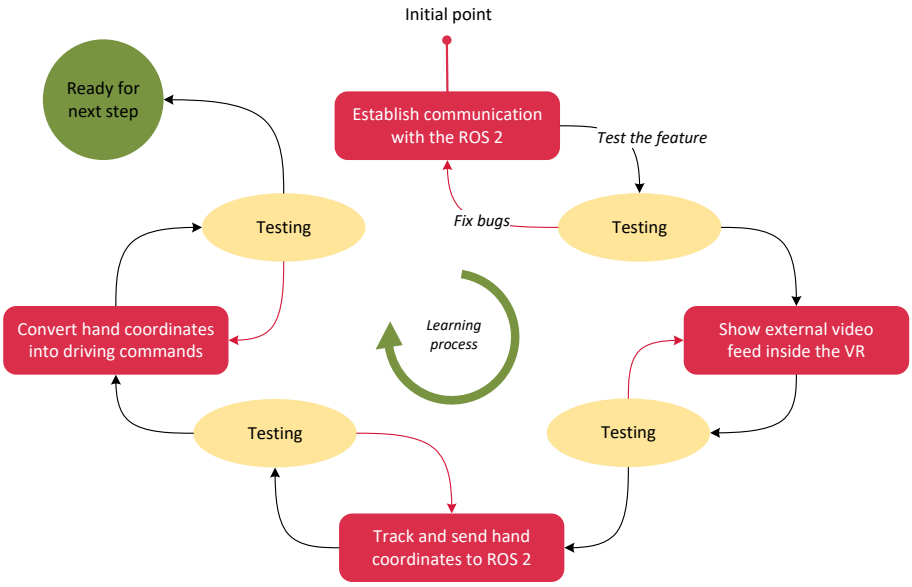


Figure P.5: Initial iteration of VR application development

The **robot control boundary** within the VR application is conceptualized as a rectangular zone, divided into four different areas for directional control: Left (L), Forward (F), Right (R), and Stop (S). When the user's hand enters this boundary, its position marked by a **red dot** within the boundary box, the application dispatches corresponding commands to the robot. This approach enables dynamic command adjustment by continuously monitoring and recalculating the hand's location relative to the boundary and giving visual feedback to the user through the red dot. This approach proved to be sufficient for the testing phase, enabling early hands-on experience with controlling the robot via the VR headset. It was also important for understanding the user experience and collecting valuable insights for future improvements.

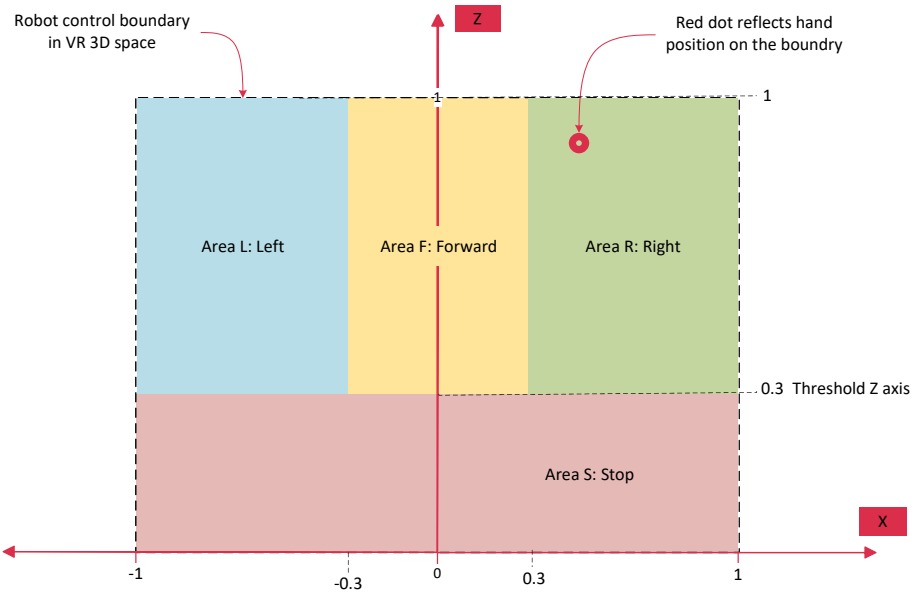


Figure P.6: Command conversion from Hand tracking

P.2.3 Second Iteration: New feature and improvements SO | *AEH*

The second iteration is an extension of the first iteration where we used collected knowledge and experience to improve our functionalities and implement new functions based on our customer requirements. Figure P.7 illustrates the process of the second iteration.

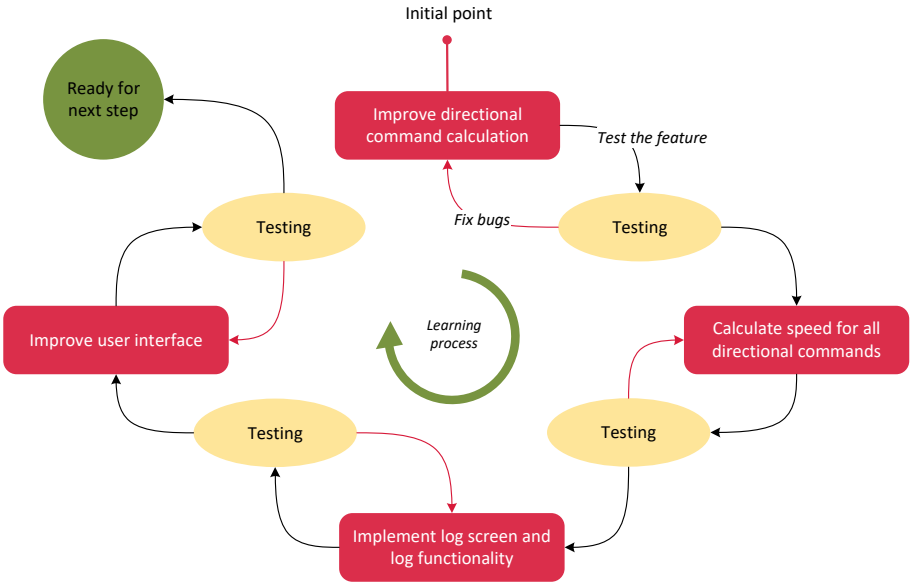


Figure P.7: Second iteration of VR application development

Improving Directional Command Calculation

The system's capability to calculate hand position within the designated control boundary for the robot (see Figure P.6) and to translate them into driving commands is crucial. The directional calculation process is important for achieving a high level of accuracy and precision, enabling the translation of the position into robot commands. Figures P.8 and P.9 illustrate the differences between the first and second versions. In the first version, we used the entire hand as an object to calculate its position relative to the control boundary, then mapped this position into X, and Z values on the boundary where $X \in [-1, 1]$ and $Z \in [0, 1]$. These X and Z values were then used to create driving commands based on the area of the hand, as indicated in P.6. The precision was not the optimal solution, since a small change in hand rotation or tilt resulted in significant changes in the hand position data. In Figure P.8, you can see an example of the relative distance between the hand and the left edge of the control boundary, where the distance varies between 0.8 and 1.2, based on a slight left and right tilt of the hand.

In the second iteration, we decided to track just one specific point on the hand instead of the whole hand. With this approach, we increased the precision of our calculations as illustrated in P.9. Focusing on one point, like the tip of the index finger, helped us avoid problems caused by small hand movements or rotations that used to make our data less precise.
This method has several benefits:

- It simplifies the tracking process, making it faster and more accurate.
- It makes our measurements more stable and consistent.
- We can control the robot with more precise and accurate data.

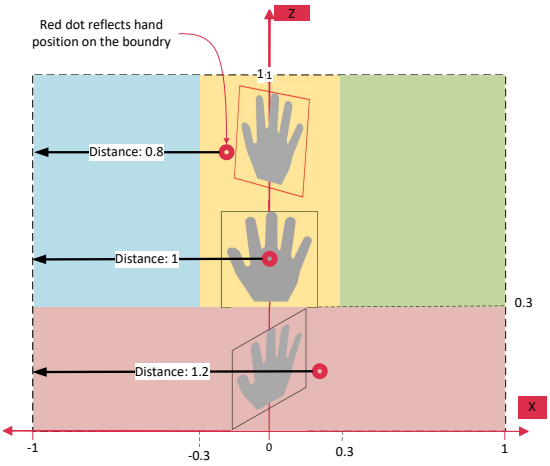


Figure P.8: Hand tracking version one

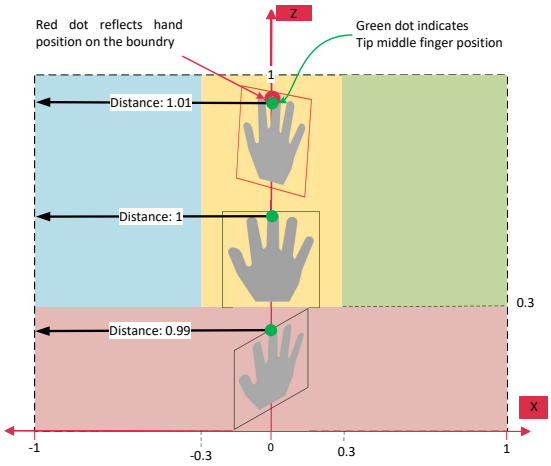


Figure P.9: Hand tracking version two

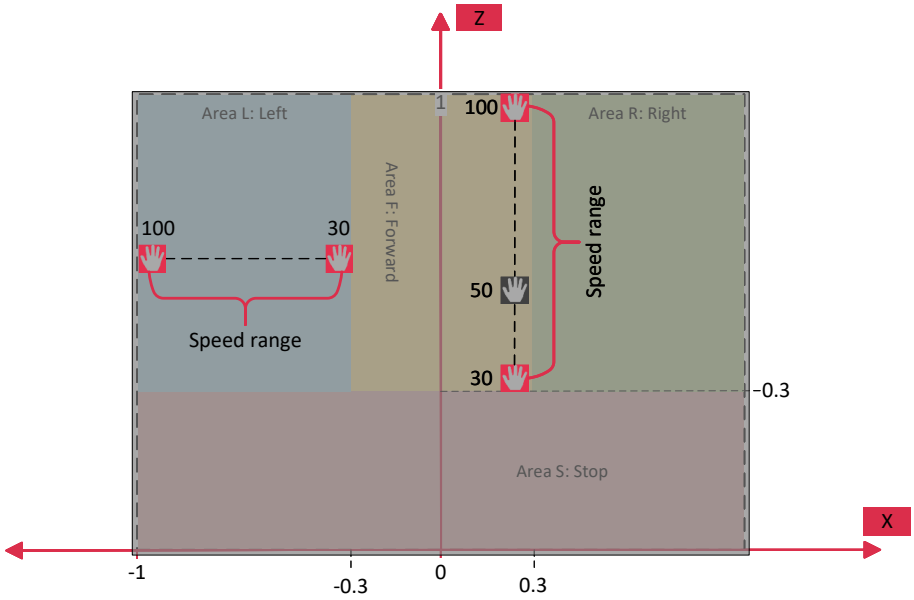


Figure P.10: Speed control

Q Calculations

Q.1 Degrees of freedom for robot arm

AEH | *HB*

] In the information about the Rosmaster x3 plus [74] it states that the robotic arm has six degrees of freedom (DOF). We wanted to understand better what DOF mean in the context of robotic arms and verify that the information is correct.
In this context, the arm has six DOF means it can exhibit six independent motions. To calculate the degrees of freedom we can use Grüblers formula (all constraints independent) [75]:

$$N = number\,of\,bodies,\,including\,ground$$

$$J = number\,of\,joints$$

$$m = 6\,for\,spatial\,bodies,\,3\,for\,planar$$

$$C = constraints\,between\,two\,rigid\,bodies$$

$$f_i = m - C$$

$$dof = m(N-1-J) + \sum_{i=1}^{J} f_i$$

Calculation of the degrees of freedom in our robotic arm:

$$N = 7, J = 6, m = 6$$

$$dof = 6(7-1-6) + \sum_{i=1}^{6} 1$$

$$dof = 6(7-1-6) + 6 = \underline{\underline{6}}$$

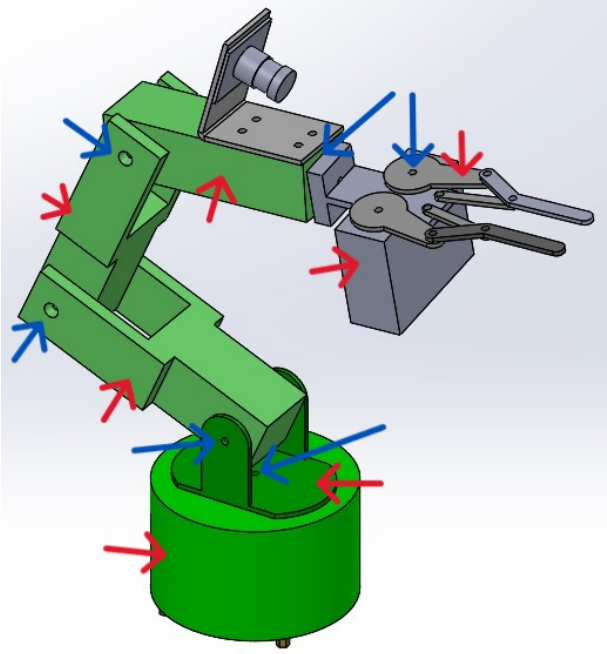


Figure Q.1: Robot arm simple drawing of joints and bodies. Joints (J) = blue & bodies (N) = Red

Q.2 Battery duration calculation

HB | *AEH*

The duration of the two different battery clusters is calculated after they both completed a 30-minute test each. The duration of the battery clusters is calculated by charging the battery cells fully. This way the battery reaches a voltage of 12.6 volts. After the battery had been used for 30 minutes, the voltage was measured. The battery is assumed empty at 9.6 volts when the voltage alarm on the expansion board starts beeping. By knowing these values, the battery duration can be assumed by calculating how many volts the battery is from triggering the alarm after the elapsed time. Below is how the battery duration was calculated.

$$U_f = Maximum\,voltage\,of\,the\,battery\,(12.60v)$$

$$U_e = Minimum\,voltage\,of\,the\,battery\,(9.60v)$$

$$U_s = Battery\,voltage\,at\,the\,end\,of\,the\,test$$

$$t = duration\,of\,the\,test\,in\,hours\,(0.5h)$$

The equation to find the voltage difference between full and empty battery

$$U_d = U_f - U_e = 12.60v - 9.60v = 3.00v$$

The voltage after $\frac{1}{2}$hour

$$U_f - U_s = U_{0.5h}$$
$$\frac{U_{0.5h}}{t} = U_{drop\,per\,hour}[\frac{v}{h}]$$
$$Duration = \frac{U_d[v]}{U_{drop\,per\,hour}[\frac{v}{h}]}$$

Calculation of the duration of “Battery cluster 1”

$$U_{0.5h} = 12.60v - 12.05v$$
$$U_{drop\,per\,hour} = \frac{0.55v}{0.5h} = 1.10v/h$$
$$Duration = \frac{3.00v}{1.10v/h} = 2.73h$$
$$Duration \approx 2h\,44min$$

Calculation of the duration of “Battery cluster 2”

$$U_{0.5h} = 12.60v - 11.98$$
$$U_{drop\,per\,hour} = \frac{0.62v}{0.5h} = 1.24v/h$$
$$Duration = \frac{3.00v}{1.24v/h} = 2.42h$$
$$Duration \approx 2h\,25min$$

The duration of “Battery Cluster 1” is estimated to last 2 hours and 44 minutes, while the duration of “Battery Cluster 2” is estimated to last 2 hours and 25 minutes.

R Code statistics

Add additional experimental contents here, like images, tables and equations that support the experimental outcome.

R.1 Repositories timeline

OM | *AD*

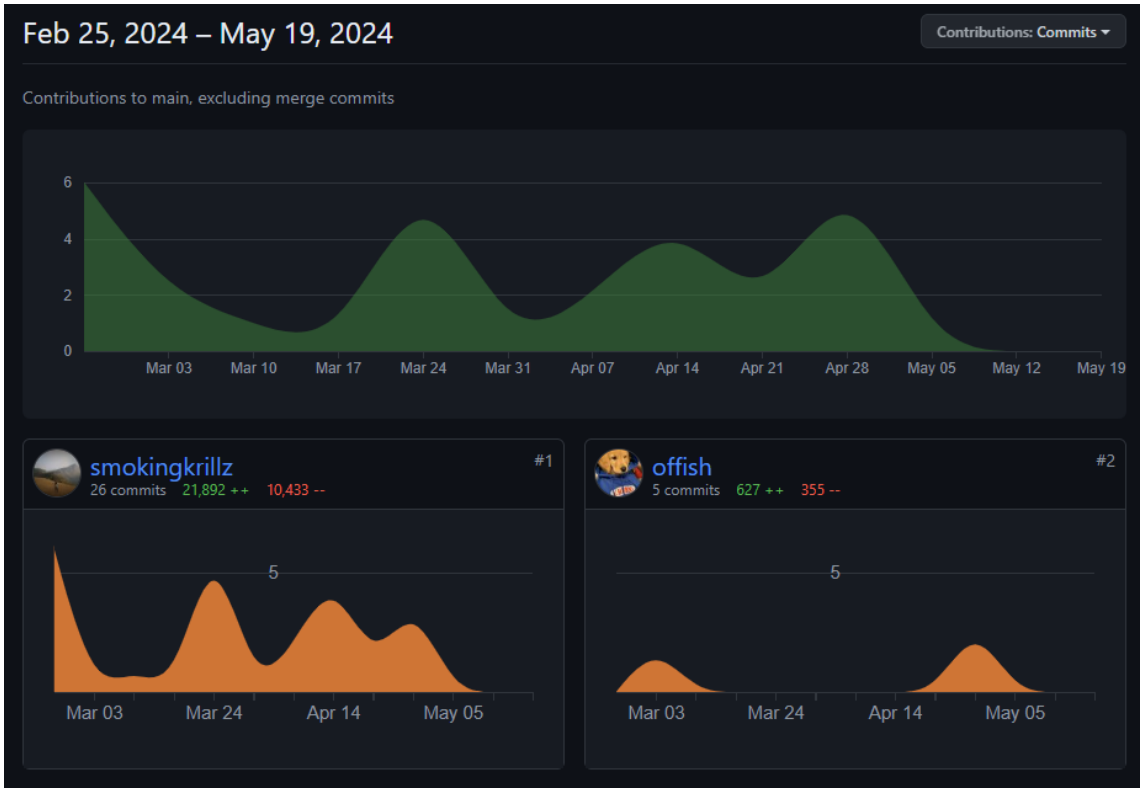


Figure R.1: transfer-learning-training commits timeline [22]

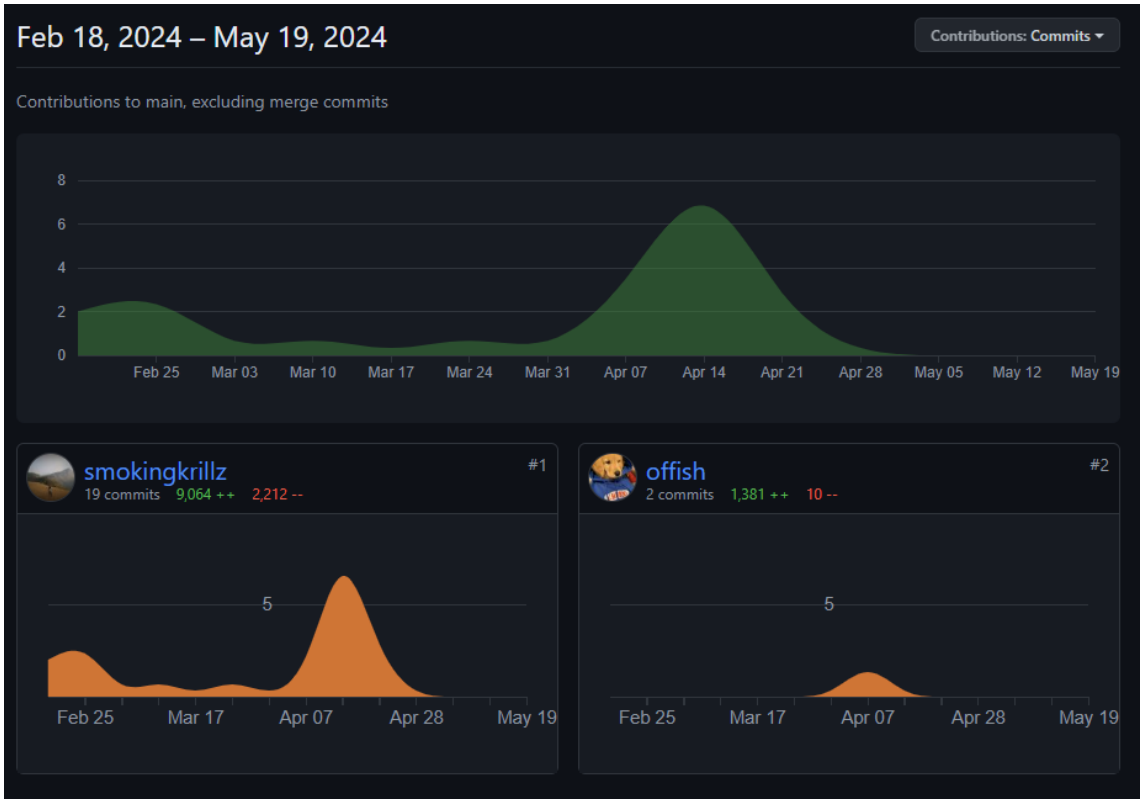
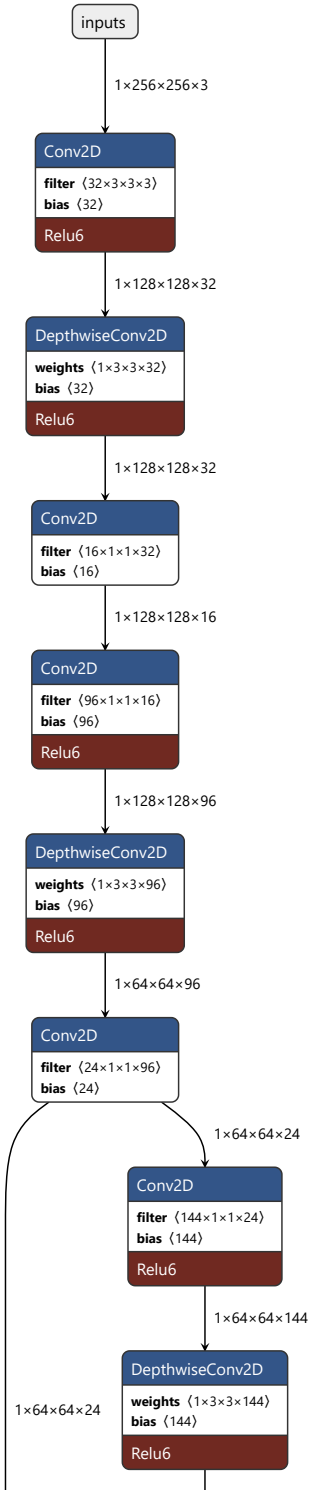


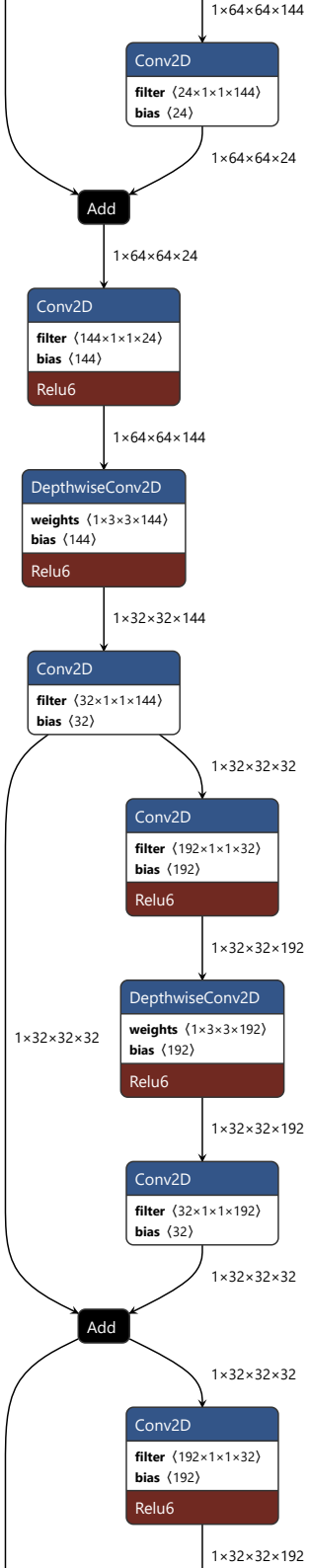
Figure R.2: obj-detection-pi commits timeline [23]

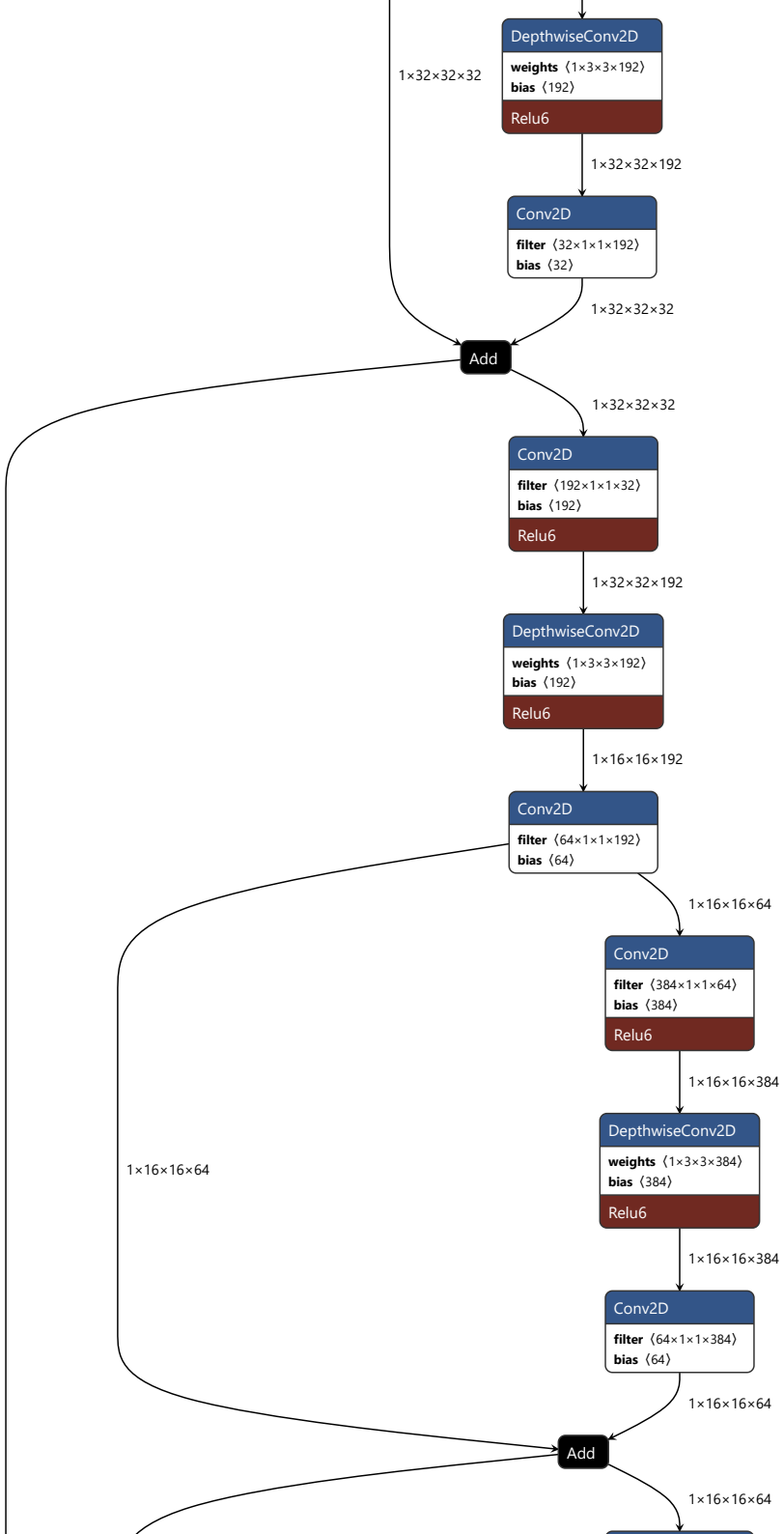
The aliases “smokingkrillz” and “offish” correspond to Aditi and Oscar, respectively.

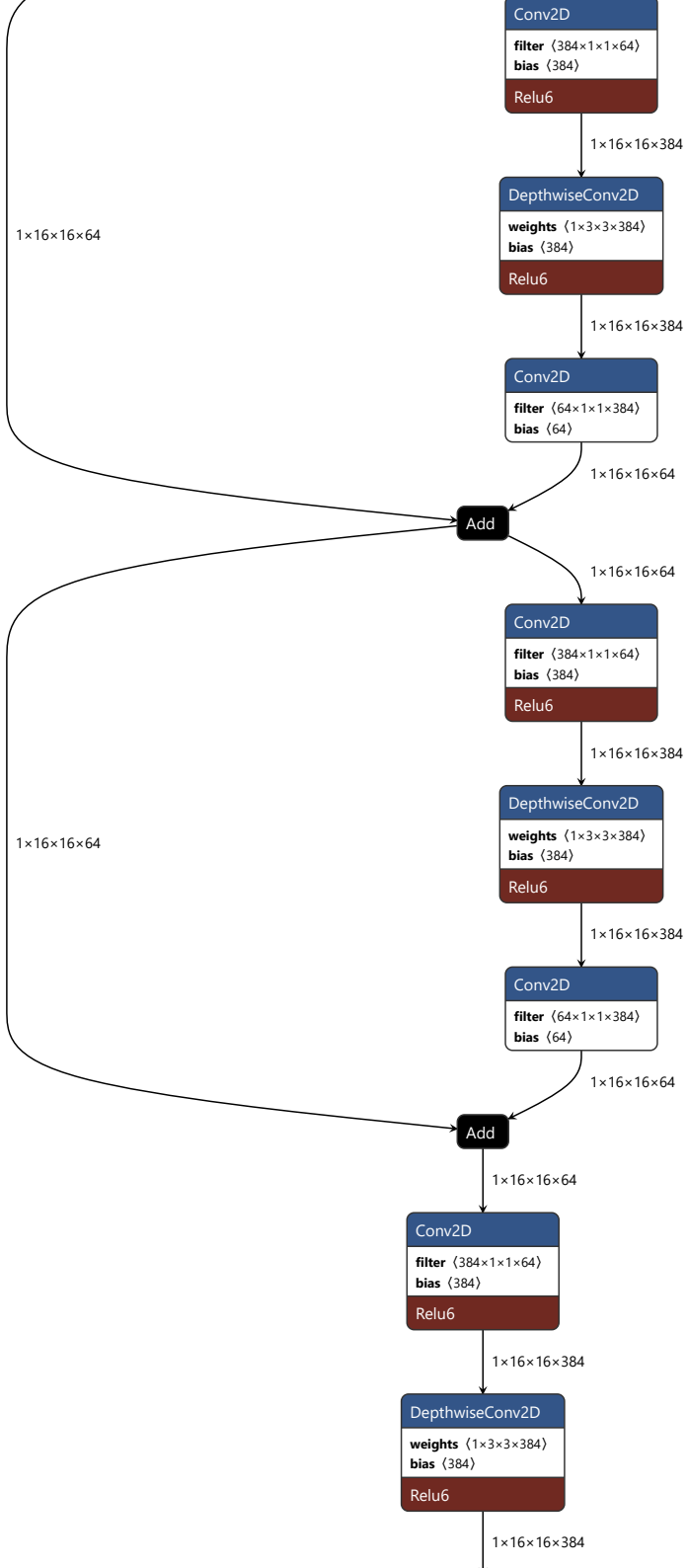
S Netron Object detection model analyzation

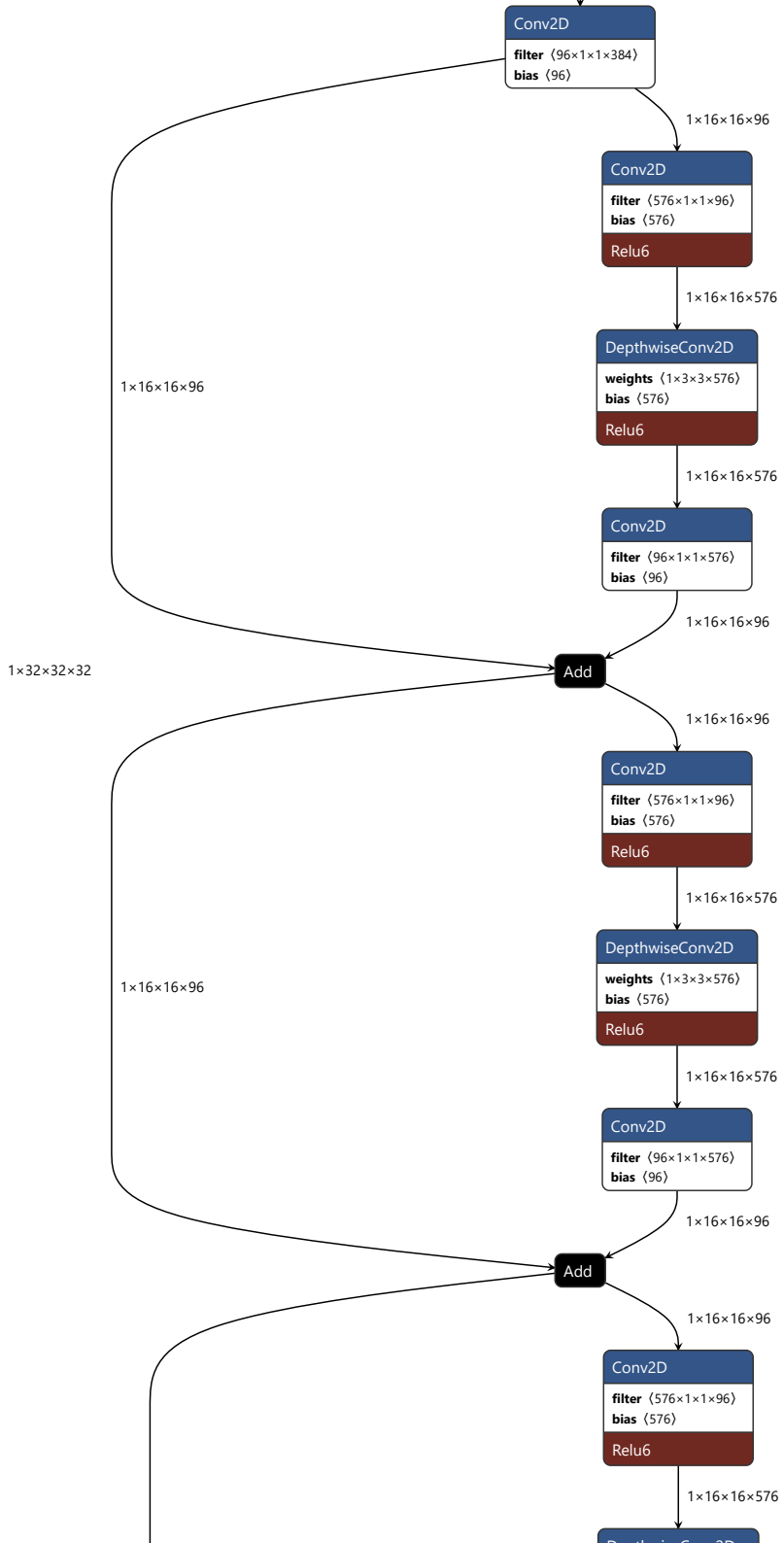
S.1 Bolt detection

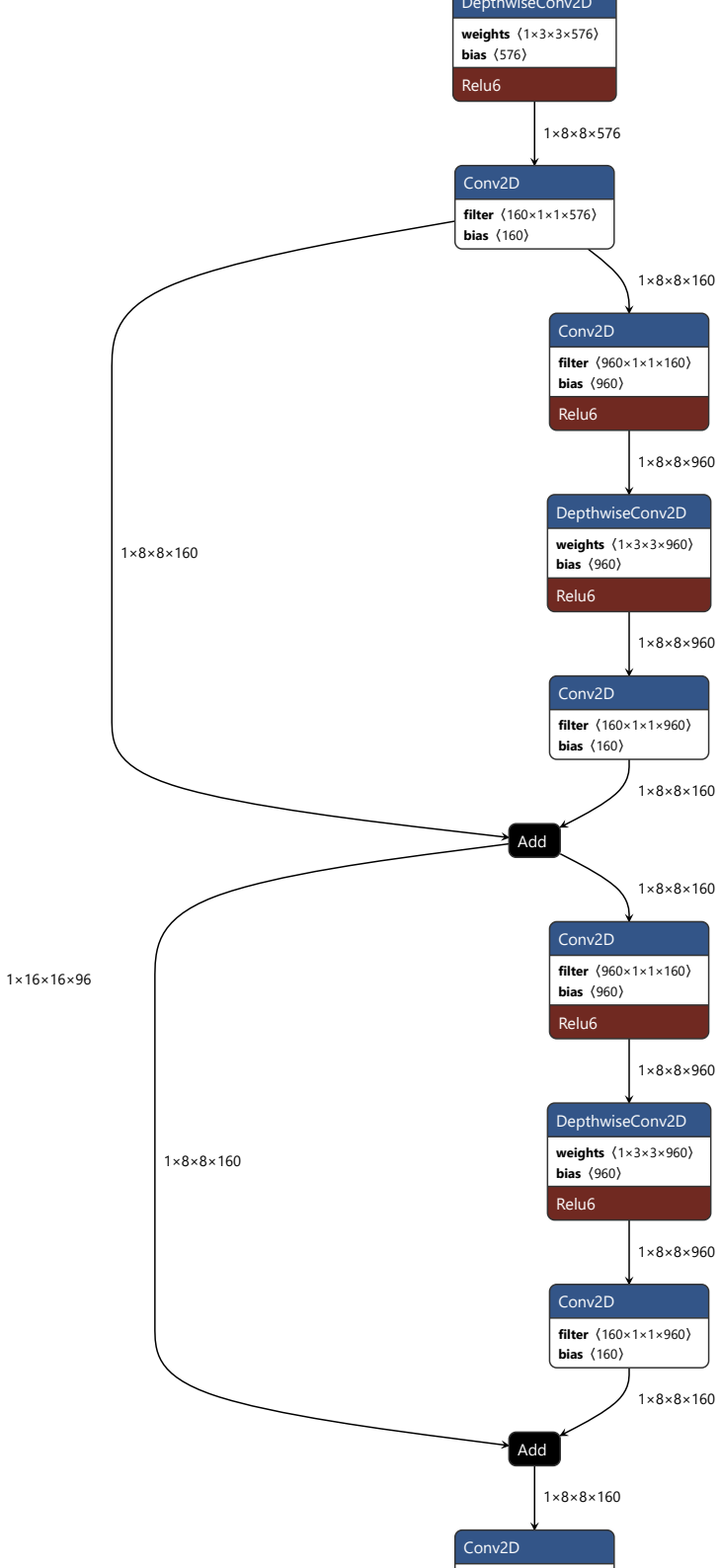

inputs
1×256×256×3
Conv2D
filter 〈32×3×3×3〉
bias 〈32〉
Relu6
1×128×128×32
DepthwiseConv2D
weights 〈1×3×3×32〉
bias 〈32〉
Relu6
1×128×128×32
Conv2D
filter 〈16×1×1×32〉
bias 〈16〉
1×128×128×16
Conv2D
filter 〈96×1×1×16〉
bias 〈96〉
Relu6
1×128×128×96
DepthwiseConv2D
weights 〈1×3×3×96〉
bias 〈96〉
Relu6
1×64×64×96
Conv2D
filter 〈24×1×1×96〉
bias 〈24〉
1×64×64×24
Conv2D
filter 〈144×1×1×24〉
bias 〈144〉
Relu6
1×64×64×144
DepthwiseConv2D
weights 〈1×3×3×144〉
bias 〈144〉
Relu6
1×64×64×24

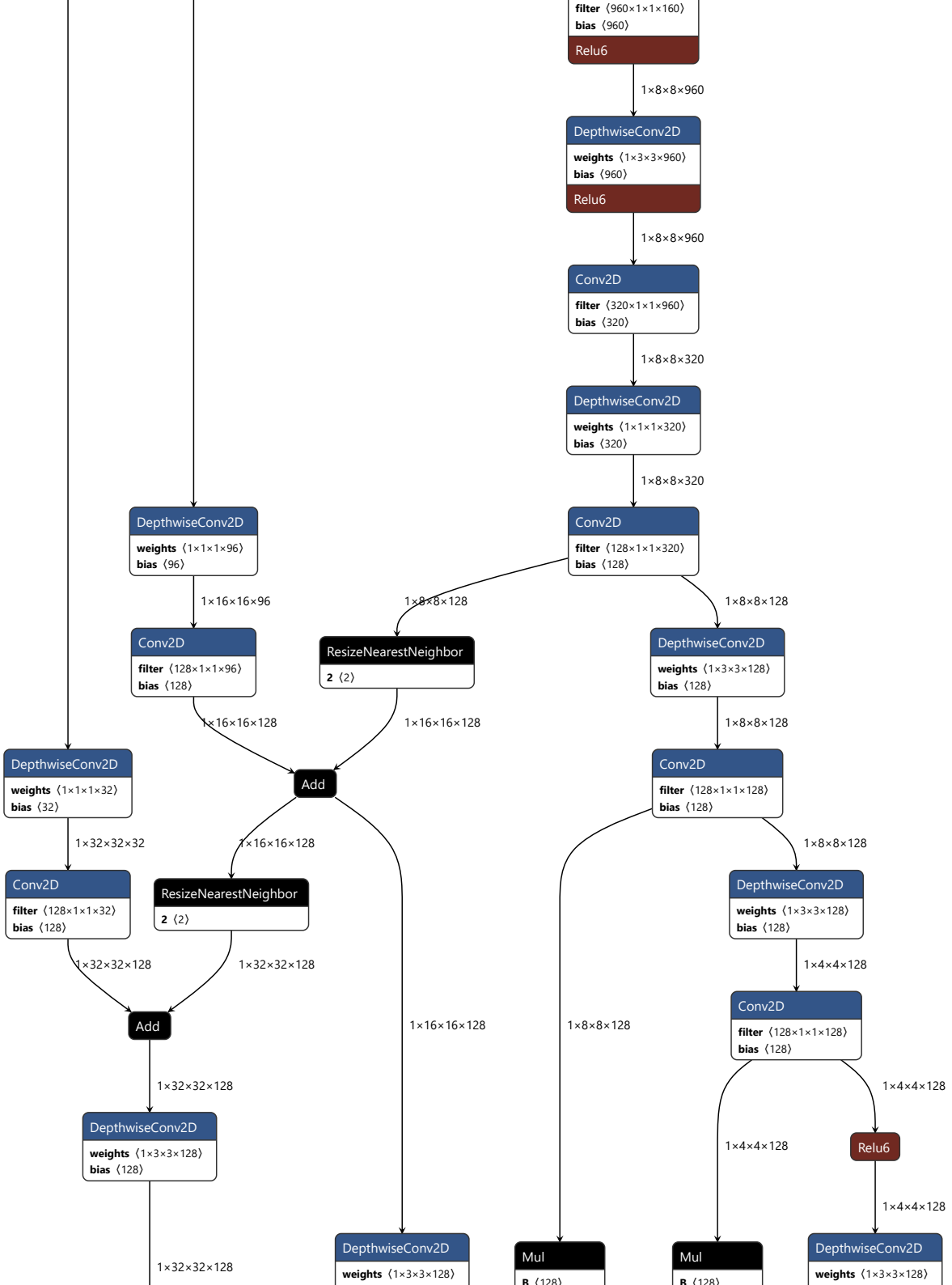

1×64×64×144
Conv2D
filter 〈24×1×1×144〉
bias 〈24〉
1×64×64×24
Add
1×64×64×24
Conv2D
filter 〈144×1×1×24〉
bias 〈144〉
Relu6
1×64×64×144
DepthwiseConv2D
weights 〈1×3×3×144〉
bias 〈144〉
Relu6
1×32×32×144
Conv2D
filter 〈32×1×1×144〉
bias 〈32〉
1×32×32×32
Conv2D
filter 〈192×1×1×32〉
bias 〈192〉
Relu6
1×32×32×192
DepthwiseConv2D
weights 〈1×3×3×192〉
bias 〈192〉
Relu6
1×32×32×32
1×32×32×192
Conv2D
filter 〈32×1×1×192〉
bias 〈32〉
1×32×32×32
Add
1×32×32×32
Conv2D
filter 〈192×1×1×32〉
bias 〈192〉
Relu6
1×32×32×192

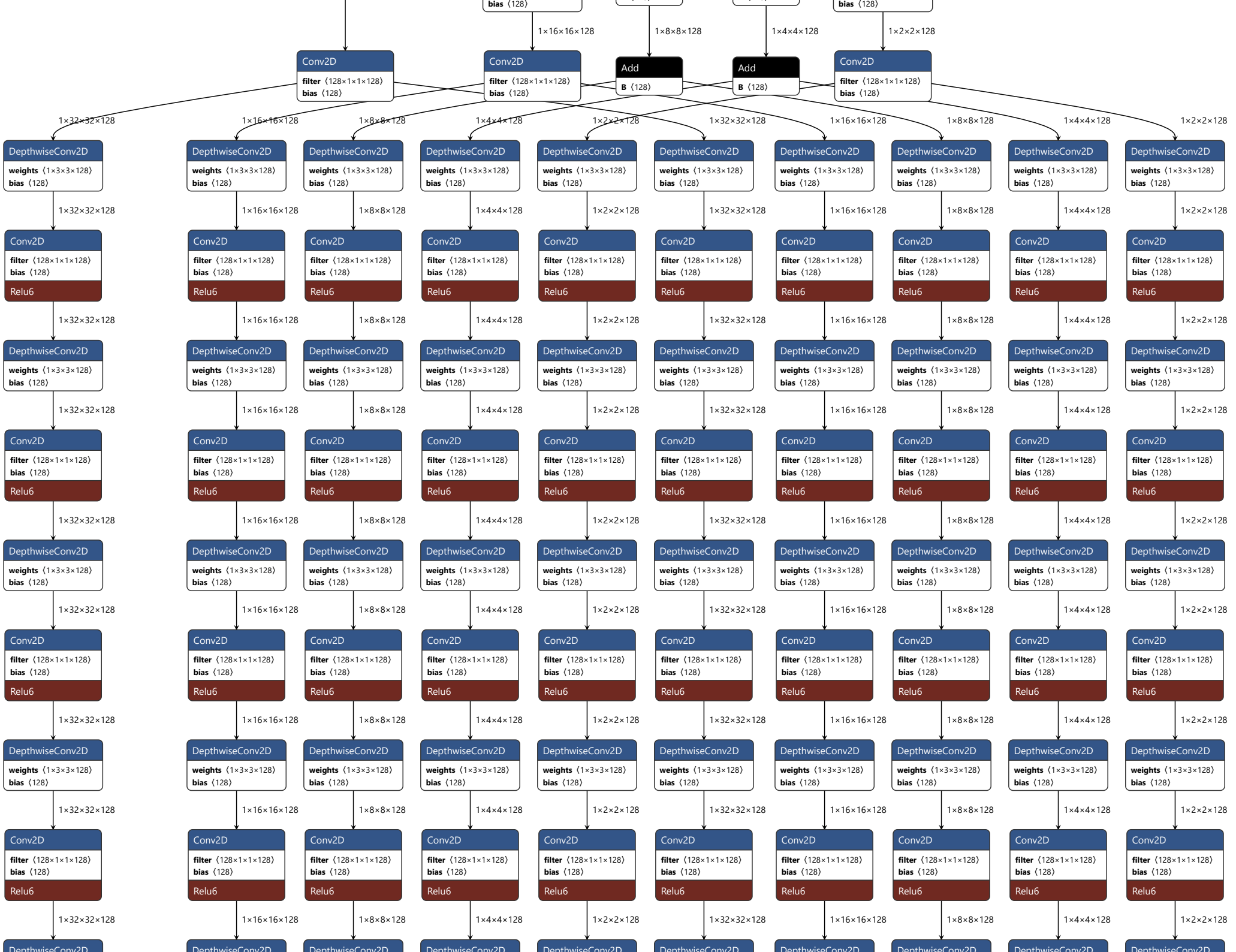

1×32×32×32
DepthwiseConv2D
weights 〈1×3×3×192〉
bias 〈192〉
Relu6
1×32×32×192
Conv2D
filter 〈32×1×1×192〉
bias 〈32〉
1×32×32×32
Add
1×32×32×32
Conv2D
filter 〈192×1×1×32〉
bias 〈192〉
Relu6
1×32×32×192
DepthwiseConv2D
weights 〈1×3×3×192〉
bias 〈192〉
Relu6
1×16×16×192
Conv2D
filter 〈64×1×1×192〉
bias 〈64〉
1×16×16×64
Conv2D
filter 〈384×1×1×64〉
bias 〈384〉
Relu6
1×16×16×384
DepthwiseConv2D
weights 〈1×3×3×384〉
bias 〈384〉
Relu6
1×16×16×384
Conv2D
filter 〈64×1×1×384〉
bias 〈64〉
1×16×16×64
1×16×16×64
Add
1×16×16×64

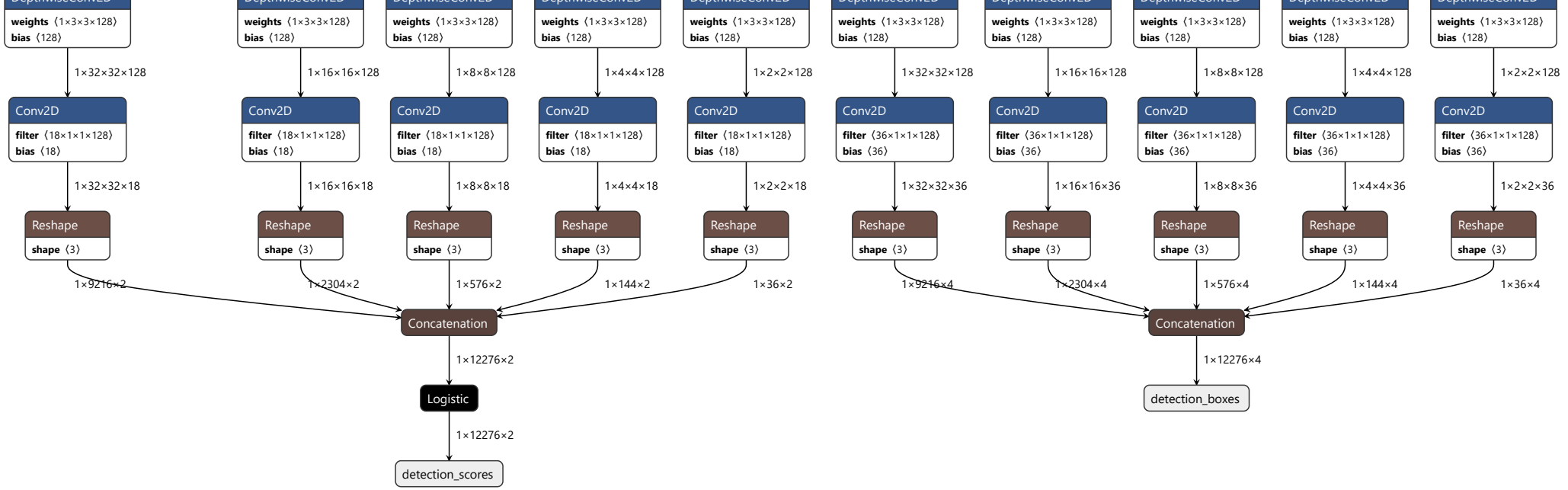

Conv2D
filter ⟨384×1×1×64⟩
bias ⟨384⟩
Relu6
1×16×16×384
DepthwiseConv2D
weights ⟨1×3×3×384⟩
bias ⟨384⟩
Relu6
1×16×16×384
Conv2D
filter ⟨64×1×1×384⟩
bias ⟨64⟩
1×16×16×64
1×16×16×64
Add
1×16×16×64
Conv2D
filter ⟨384×1×1×64⟩
bias ⟨384⟩
Relu6
1×16×16×384
DepthwiseConv2D
weights ⟨1×3×3×384⟩
bias ⟨384⟩
Relu6
1×16×16×384
Conv2D
filter ⟨64×1×1×384⟩
bias ⟨64⟩
1×16×16×64
1×16×16×64
Add
1×16×16×64
Conv2D
filter ⟨384×1×1×64⟩
bias ⟨384⟩
Relu6
1×16×16×384
DepthwiseConv2D
weights ⟨1×3×3×384⟩
bias ⟨384⟩
Relu6
1×16×16×384


Conv2D
filter 〈96×1×1×384〉
bias 〈96〉
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576
DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×16×16×576
Conv2D
filter 〈96×1×1×576〉
bias 〈96〉
1×16×16×96
1×16×16×96
1×32×32×32
Add
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576
DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×16×16×576
Conv2D
filter 〈96×1×1×576〉
bias 〈96〉
1×16×16×96
1×16×16×96
Add
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576

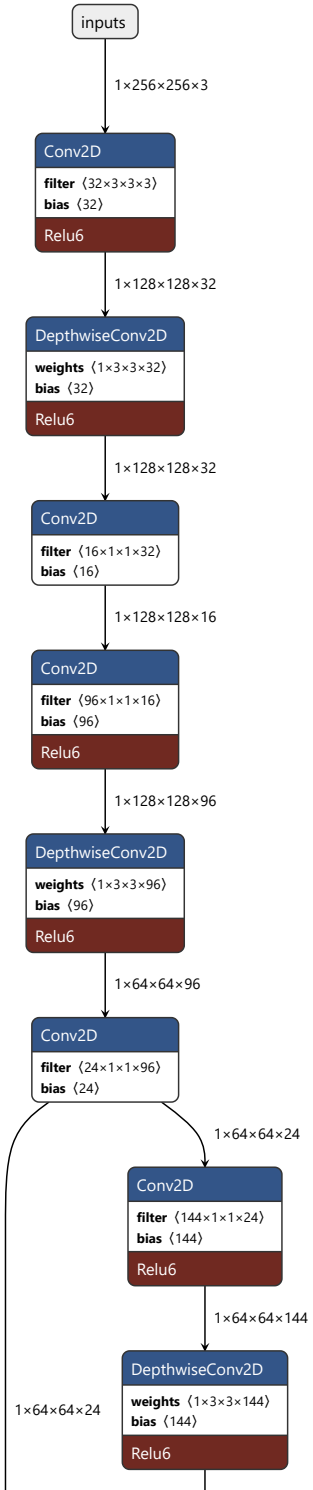

DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×8×8×576
Conv2D
filter 〈160×1×1×576〉
bias 〈160〉
1×8×8×160
Conv2D
filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈160×1×1×960〉
bias 〈160〉
1×8×8×160
1×8×8×160
Add
1×8×8×160
1×16×16×96
Conv2D
filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈160×1×1×960〉
bias 〈160〉
1×8×8×160
1×8×8×160
Add
1×8×8×160
Conv2D

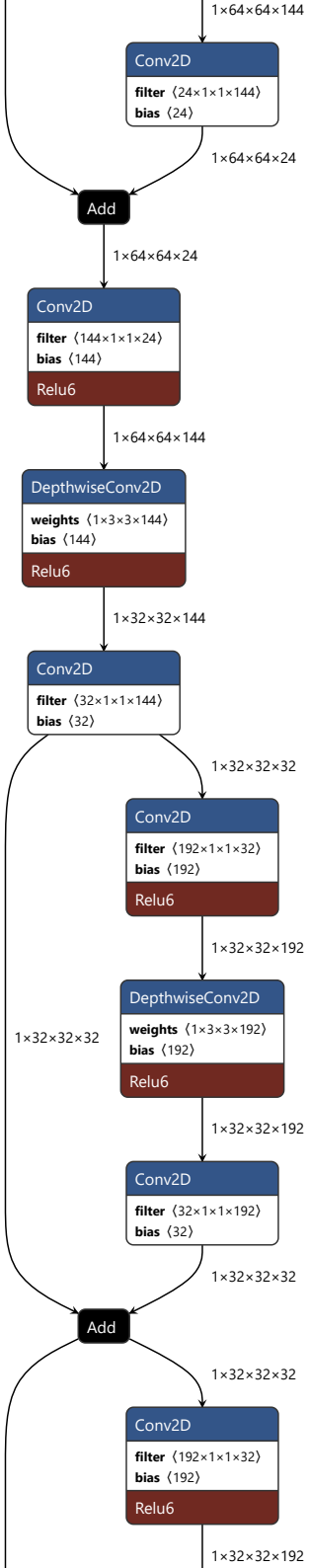

filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈320×1×1×960〉
bias 〈320〉
1×8×8×320
DepthwiseConv2D
weights 〈1×1×1×320〉
bias 〈320〉
1×8×8×320
Conv2D
filter 〈128×1×1×320〉
bias 〈128〉
1×8×8×128
1×8×8×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×8×8×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
1×8×8×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×4×4×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
1×4×4×128
1×4×4×128
Relu6
1×4×4×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
DepthwiseConv2D
weights 〈1×1×1×96〉
bias 〈96〉
1×16×16×96
Conv2D
filter 〈128×1×1×96〉
bias 〈128〉
1×16×16×128
ResizeNearestNeighbor
2 〈2〉
1×16×16×128
DepthwiseConv2D
weights 〈1×1×1×32〉
bias 〈32〉
1×32×32×32
Add
1×16×16×128
Conv2D
filter 〈128×1×1×32〉
bias 〈128〉
1×32×32×128
ResizeNearestNeighbor
2 〈2〉
1×32×32×128
Add
1×32×32×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×32×32×128
1×16×16×128
Mul
B 〈128〉
1×8×8×128
Mul
B 〈128〉
DepthwiseConv2D
weights 〈1×3×3×128〉

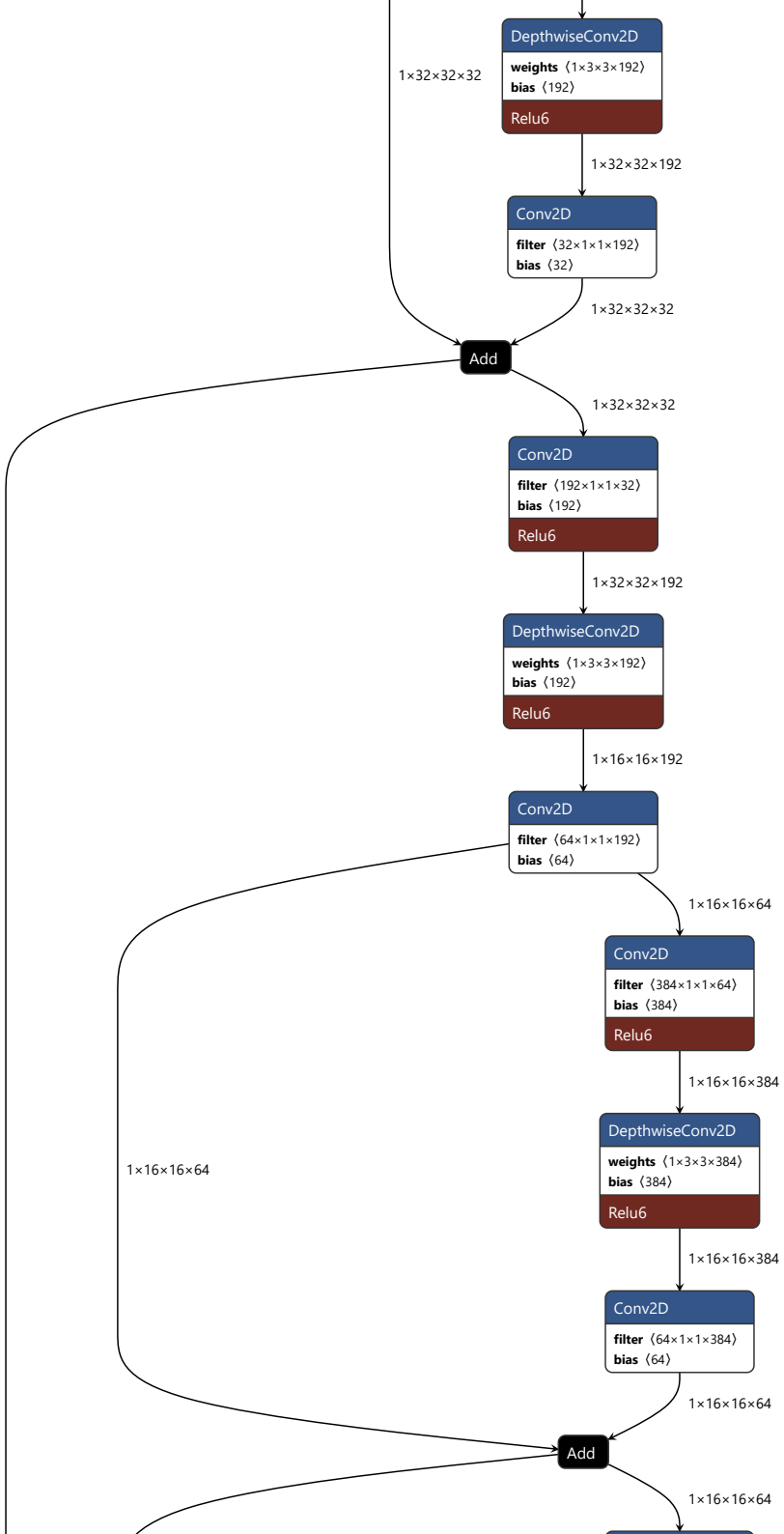

bias 〈128〉
1×16×16×128
1×8×8×128
1×4×4×128
1×2×2×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
Add
B 〈128〉
1×32×32×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
Relu6

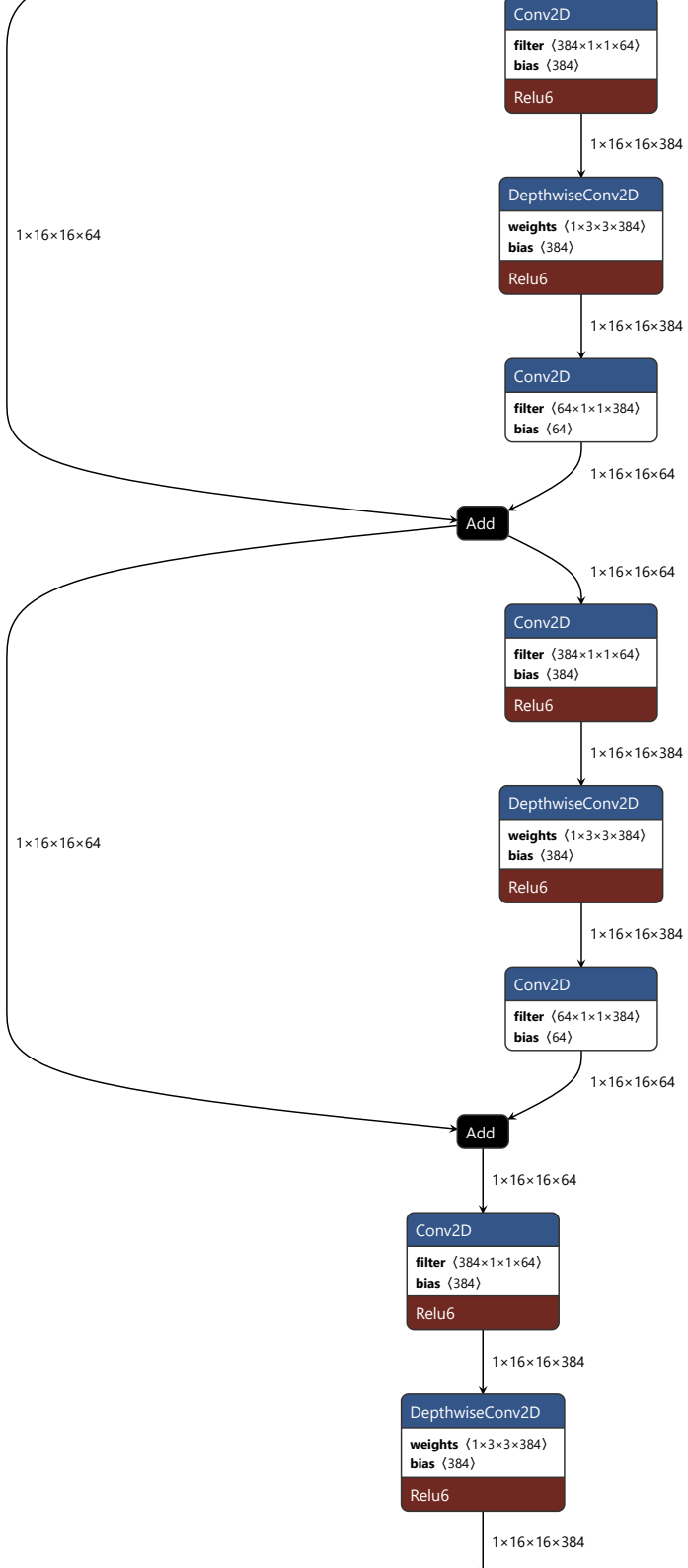

weights ⟨1×3×3×128⟩
bias ⟨128⟩
1×32×32×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×32×32×18
Reshape
shape ⟨3⟩
1×9216×2
1×16×16×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×16×16×18
Reshape
shape ⟨3⟩
1×2304×2
1×8×8×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×8×8×18
Reshape
shape ⟨3⟩
1×576×2
1×4×4×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×4×4×18
Reshape
shape ⟨3⟩
1×144×2
1×2×2×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×2×2×18
Reshape
shape ⟨3⟩
1×36×2
Concatenation
1×12276×2
Logistic
1×12276×2
detection_scores
1×32×32×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×32×32×36
Reshape
shape ⟨3⟩
1×9216×4
1×16×16×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×16×16×36
Reshape
shape ⟨3⟩
1×2304×4
1×8×8×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×8×8×36
Reshape
shape ⟨3⟩
1×576×4
1×4×4×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×4×4×36
Reshape
shape ⟨3⟩
1×144×4
1×2×2×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×2×2×36
Reshape
shape ⟨3⟩
1×36×4
Concatenation
1×12276×4
detection_boxes

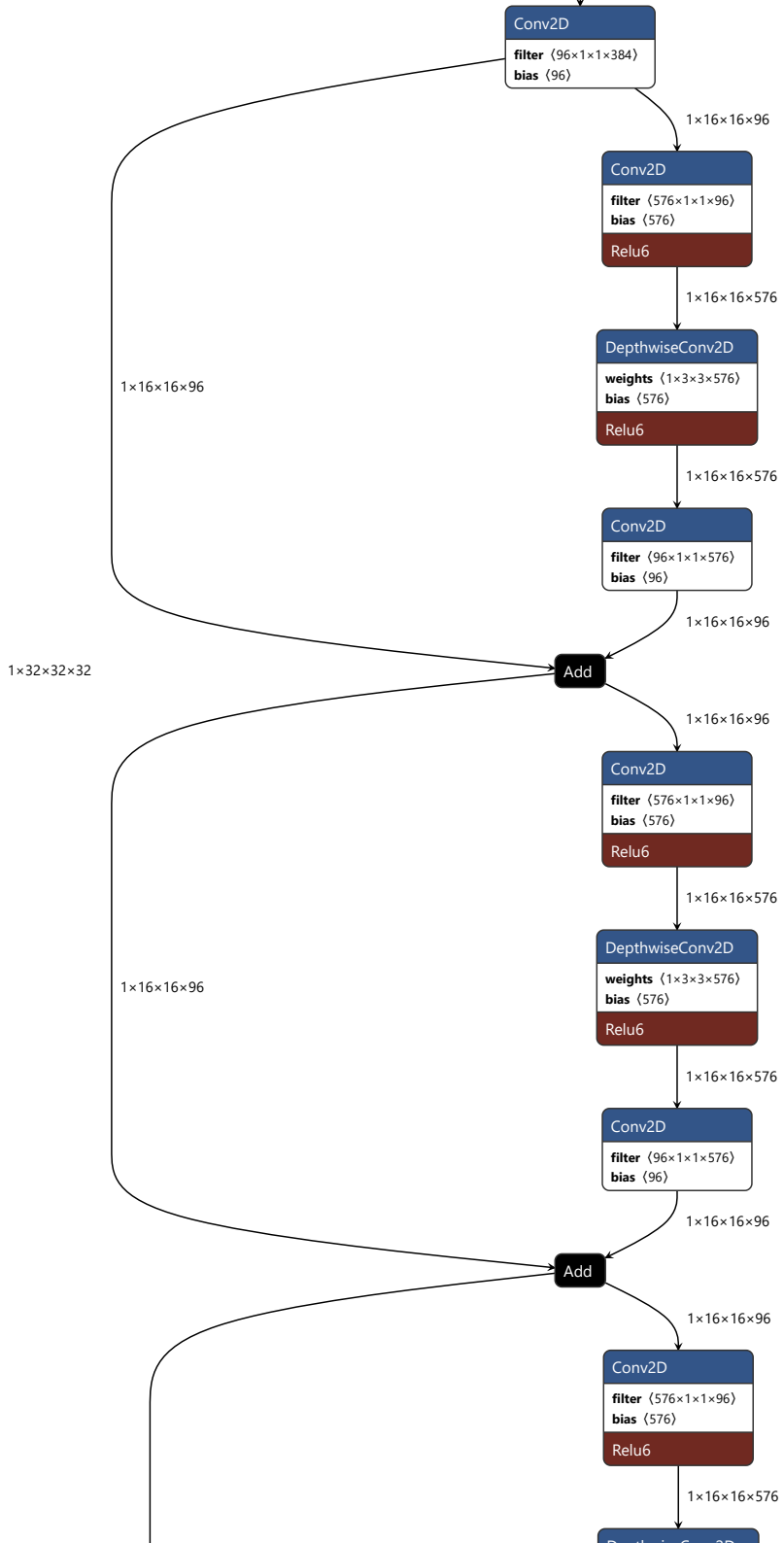
S.2 People detection

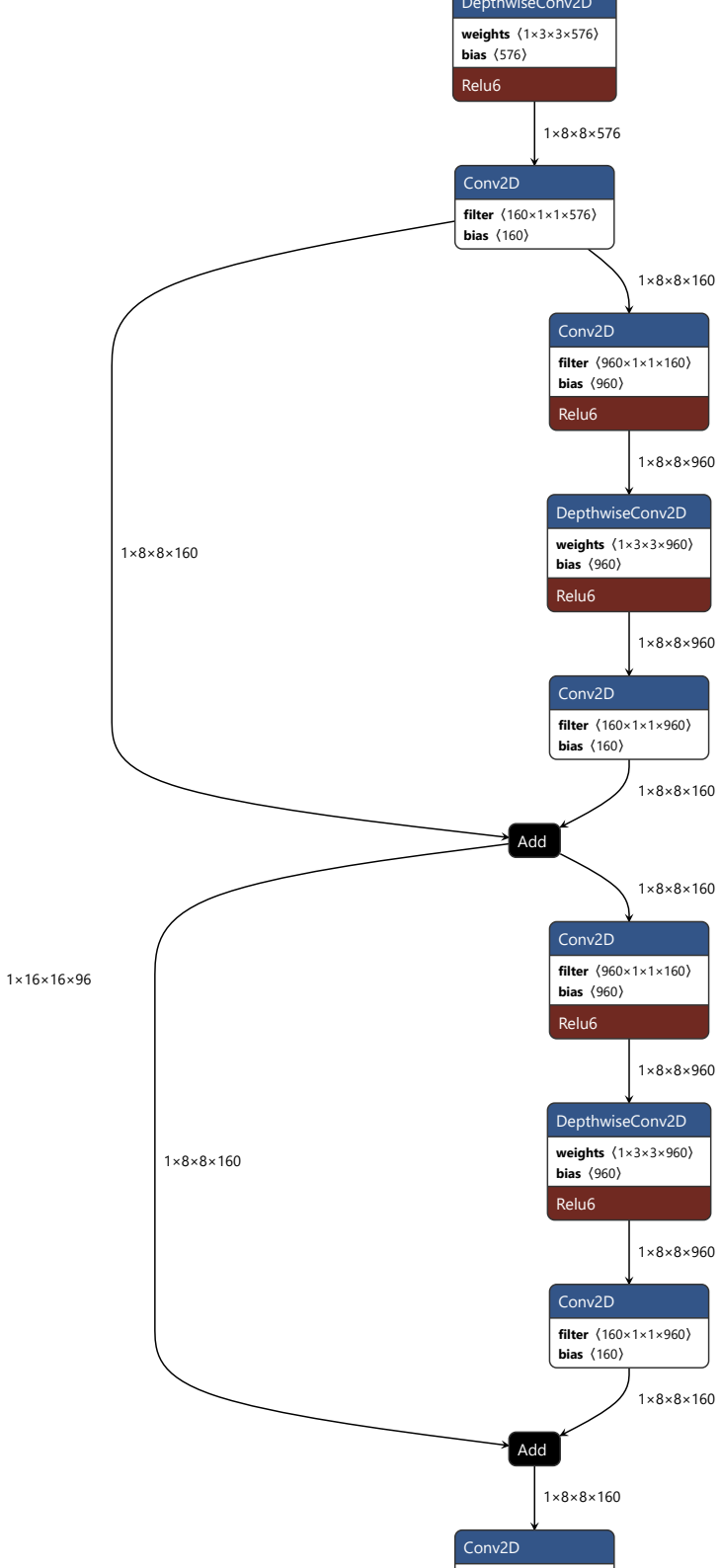

inputs
1×256×256×3
Conv2D
filter 〈32×3×3×3〉
bias 〈32〉
Relu6
1×128×128×32
DepthwiseConv2D
weights 〈1×3×3×32〉
bias 〈32〉
Relu6
1×128×128×32
Conv2D
filter 〈16×1×1×32〉
bias 〈16〉
1×128×128×16
Conv2D
filter 〈96×1×1×16〉
bias 〈96〉
Relu6
1×128×128×96
DepthwiseConv2D
weights 〈1×3×3×96〉
bias 〈96〉
Relu6
1×64×64×96
Conv2D
filter 〈24×1×1×96〉
bias 〈24〉
1×64×64×24
Conv2D
filter 〈144×1×1×24〉
bias 〈144〉
Relu6
1×64×64×144
DepthwiseConv2D
weights 〈1×3×3×144〉
bias 〈144〉
Relu6
1×64×64×24

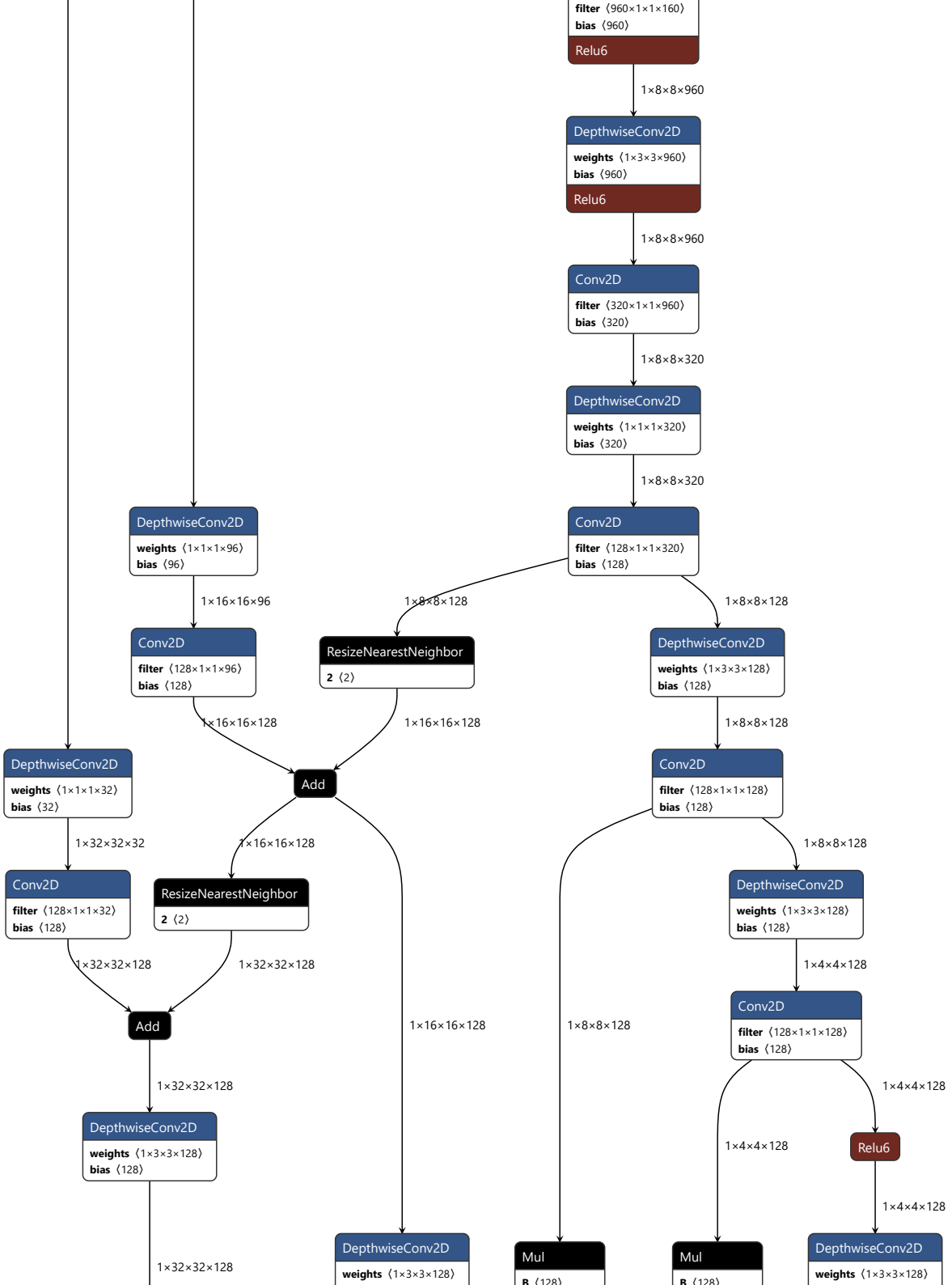

1×64×64×144
Conv2D
filter〈24×1×1×144〉
bias〈24〉
1×64×64×24
Add
1×64×64×24
Conv2D
filter〈144×1×1×24〉
bias〈144〉
Relu6
1×64×64×144
DepthwiseConv2D
weights〈1×3×3×144〉
bias〈144〉
Relu6
1×32×32×144
Conv2D
filter〈32×1×1×144〉
bias〈32〉
1×32×32×32
Conv2D
filter〈192×1×1×32〉
bias〈192〉
Relu6
1×32×32×192
DepthwiseConv2D
weights〈1×3×3×192〉
bias〈192〉
Relu6
1×32×32×32
1×32×32×192
Conv2D
filter〈32×1×1×192〉
bias〈32〉
1×32×32×32
Add
1×32×32×32
Conv2D
filter〈192×1×1×32〉
bias〈192〉
Relu6
1×32×32×192

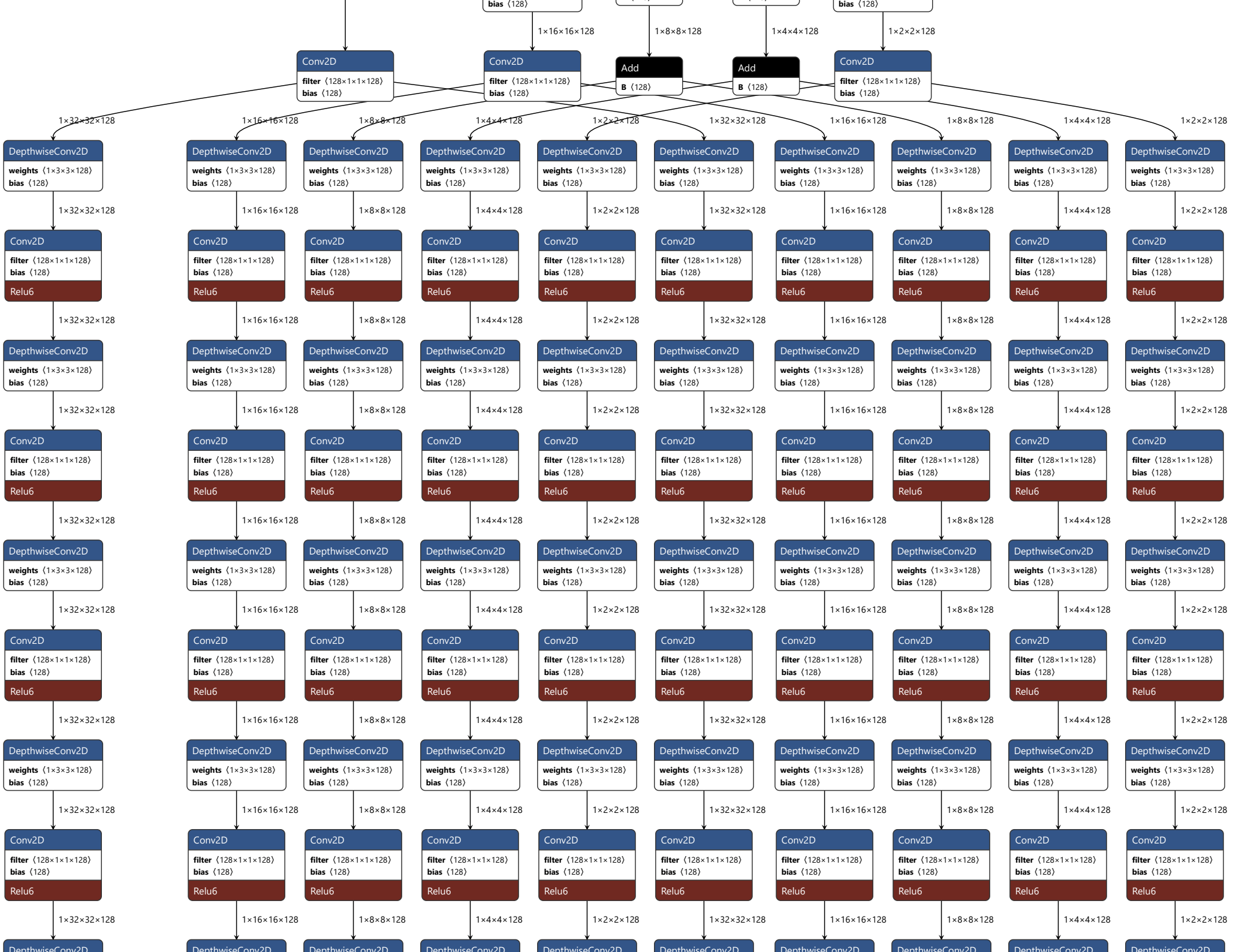

DepthwiseConv2D
weights 〈1×3×3×192〉
bias 〈192〉
Relu6
1×32×32×32
1×32×32×192
Conv2D
filter 〈32×1×1×192〉
bias 〈32〉
1×32×32×32
Add
1×32×32×32
Conv2D
filter 〈192×1×1×32〉
bias 〈192〉
Relu6
1×32×32×192
DepthwiseConv2D
weights 〈1×3×3×192〉
bias 〈192〉
Relu6
1×16×16×192
Conv2D
filter 〈64×1×1×192〉
bias 〈64〉
1×16×16×64
Conv2D
filter 〈384×1×1×64〉
bias 〈384〉
Relu6
1×16×16×384
DepthwiseConv2D
weights 〈1×3×3×384〉
bias 〈384〉
Relu6
1×16×16×384
Conv2D
filter 〈64×1×1×384〉
bias 〈64〉
1×16×16×64
1×16×16×64
Add
1×16×16×64

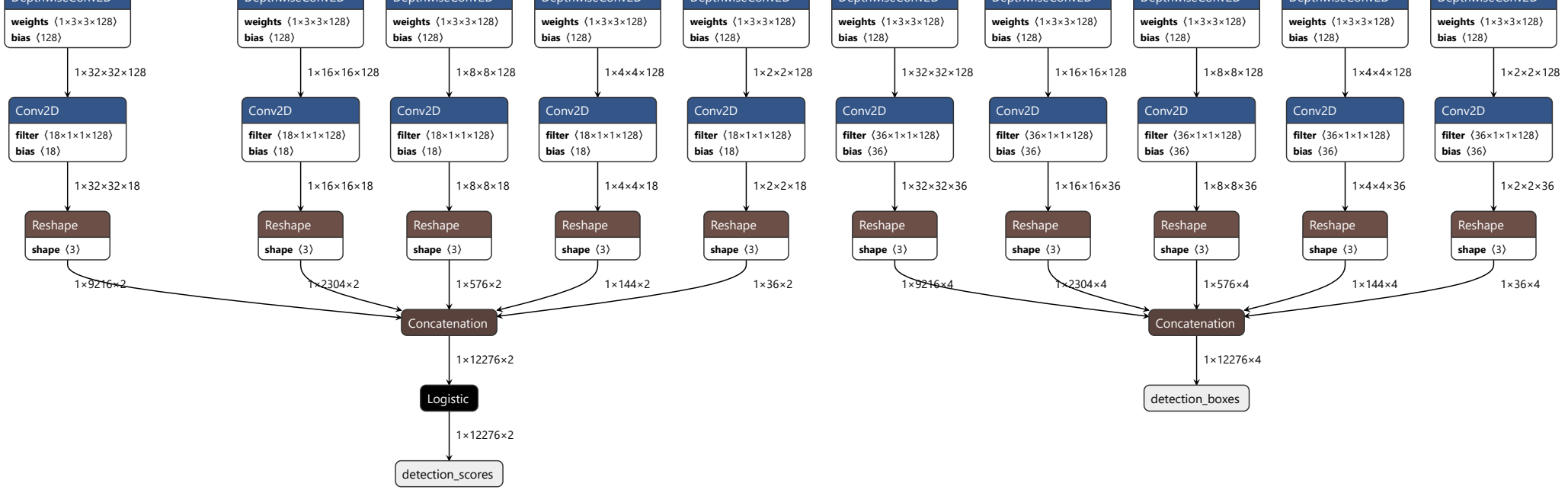

Conv2D
filter 〈384×1×1×64〉
bias 〈384〉
Relu6
1×16×16×384
DepthwiseConv2D
weights 〈1×3×3×384〉
bias 〈384〉
Relu6
1×16×16×384
Conv2D
filter 〈64×1×1×384〉
bias 〈64〉
1×16×16×64
1×16×16×64
Add
1×16×16×64
Conv2D
filter 〈384×1×1×64〉
bias 〈384〉
Relu6
1×16×16×384
DepthwiseConv2D
weights 〈1×3×3×384〉
bias 〈384〉
Relu6
1×16×16×384
Conv2D
filter 〈64×1×1×384〉
bias 〈64〉
1×16×16×64
1×16×16×64
Add
1×16×16×64
Conv2D
filter 〈384×1×1×64〉
bias 〈384〉
Relu6
1×16×16×384
DepthwiseConv2D
weights 〈1×3×3×384〉
bias 〈384〉
Relu6
1×16×16×384


Conv2D
filter 〈96×1×1×384〉
bias 〈96〉
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576
DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×16×16×576
Conv2D
filter 〈96×1×1×576〉
bias 〈96〉
1×16×16×96
1×16×16×96
1×32×32×32
Add
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576
DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×16×16×576
Conv2D
filter 〈96×1×1×576〉
bias 〈96〉
1×16×16×96
1×16×16×96
Add
1×16×16×96
Conv2D
filter 〈576×1×1×96〉
bias 〈576〉
Relu6
1×16×16×576


DepthwiseConv2D
weights 〈1×3×3×576〉
bias 〈576〉
Relu6
1×8×8×576
Conv2D
filter 〈160×1×1×576〉
bias 〈160〉
1×8×8×160
Conv2D
filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈160×1×1×960〉
bias 〈160〉
1×8×8×160
1×8×8×160
Add
1×8×8×160
1×16×16×96
Conv2D
filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈160×1×1×960〉
bias 〈160〉
1×8×8×160
1×8×8×160
Add
1×8×8×160
Conv2D


filter 〈960×1×1×160〉
bias 〈960〉
Relu6
1×8×8×960
DepthwiseConv2D
weights 〈1×3×3×960〉
bias 〈960〉
Relu6
1×8×8×960
Conv2D
filter 〈320×1×1×960〉
bias 〈320〉
1×8×8×320
DepthwiseConv2D
weights 〈1×1×1×320〉
bias 〈320〉
1×8×8×320
Conv2D
filter 〈128×1×1×320〉
bias 〈128〉
1×8×8×128
1×8×8×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×8×8×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
1×8×8×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×4×4×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
1×4×4×128
1×4×4×128
Relu6
1×4×4×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
DepthwiseConv2D
weights 〈1×1×1×96〉
bias 〈96〉
1×16×16×96
Conv2D
filter 〈128×1×1×96〉
bias 〈128〉
1×16×16×128
ResizeNearestNeighbor
2 〈2〉
1×16×16×128
DepthwiseConv2D
weights 〈1×1×1×32〉
bias 〈32〉
1×32×32×32
Add
1×16×16×128
Conv2D
filter 〈128×1×1×32〉
bias 〈128〉
1×32×32×128
ResizeNearestNeighbor
2 〈2〉
1×32×32×128
Add
1×32×32×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
1×32×32×128
1×16×16×128
Mul
1×8×8×128
Mul


bias 〈128〉
1×16×16×128
1×8×8×128
1×4×4×128
1×2×2×128
Conv2D
filter 〈128×1×1×128〉
bias 〈128〉
Add
B 〈128〉
1×32×32×128
DepthwiseConv2D
weights 〈1×3×3×128〉
bias 〈128〉
Relu6


weights ⟨1×3×3×128⟩
bias ⟨128⟩
1×32×32×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×32×32×18
Reshape
shape ⟨3⟩
1×9216×2
1×16×16×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×16×16×18
Reshape
shape ⟨3⟩
1×2304×2
1×8×8×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×8×8×18
Reshape
shape ⟨3⟩
1×576×2
1×4×4×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×4×4×18
Reshape
shape ⟨3⟩
1×144×2
1×2×2×128
Conv2D
filter ⟨18×1×1×128⟩
bias ⟨18⟩
1×2×2×18
Reshape
shape ⟨3⟩
1×36×2
Concatenation
1×12276×2
Logistic
1×12276×2
detection_scores
1×32×32×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×32×32×36
Reshape
shape ⟨3⟩
1×9216×4
1×16×16×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×16×16×36
Reshape
shape ⟨3⟩
1×2304×4
1×8×8×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×8×8×36
Reshape
shape ⟨3⟩
1×576×4
1×4×4×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×4×4×36
Reshape
shape ⟨3⟩
1×144×4
1×2×2×128
Conv2D
filter ⟨36×1×1×128⟩
bias ⟨36⟩
1×2×2×36
Reshape
shape ⟨3⟩
1×36×4
Concatenation
1×12276×4
detection_boxes

T Technical drawings

T.1 Robot design iteration one

T.1.1 Arm cover iteration one

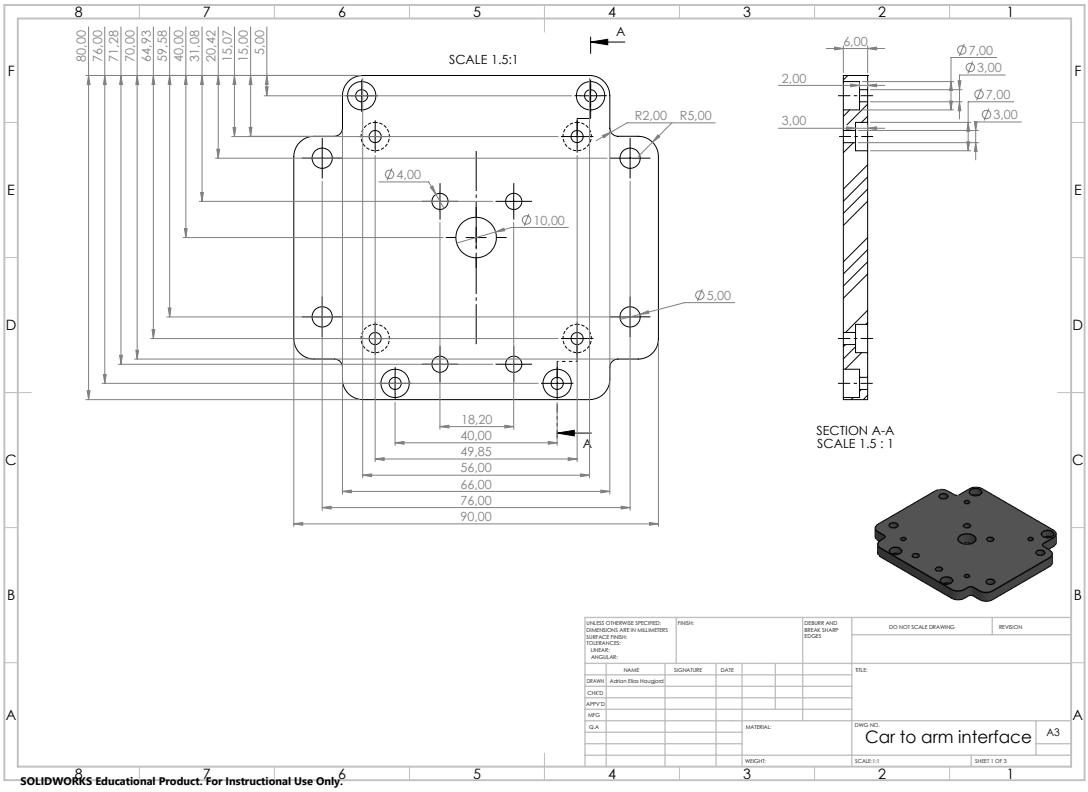


Figure T.1: Car to arm interface technical drawing

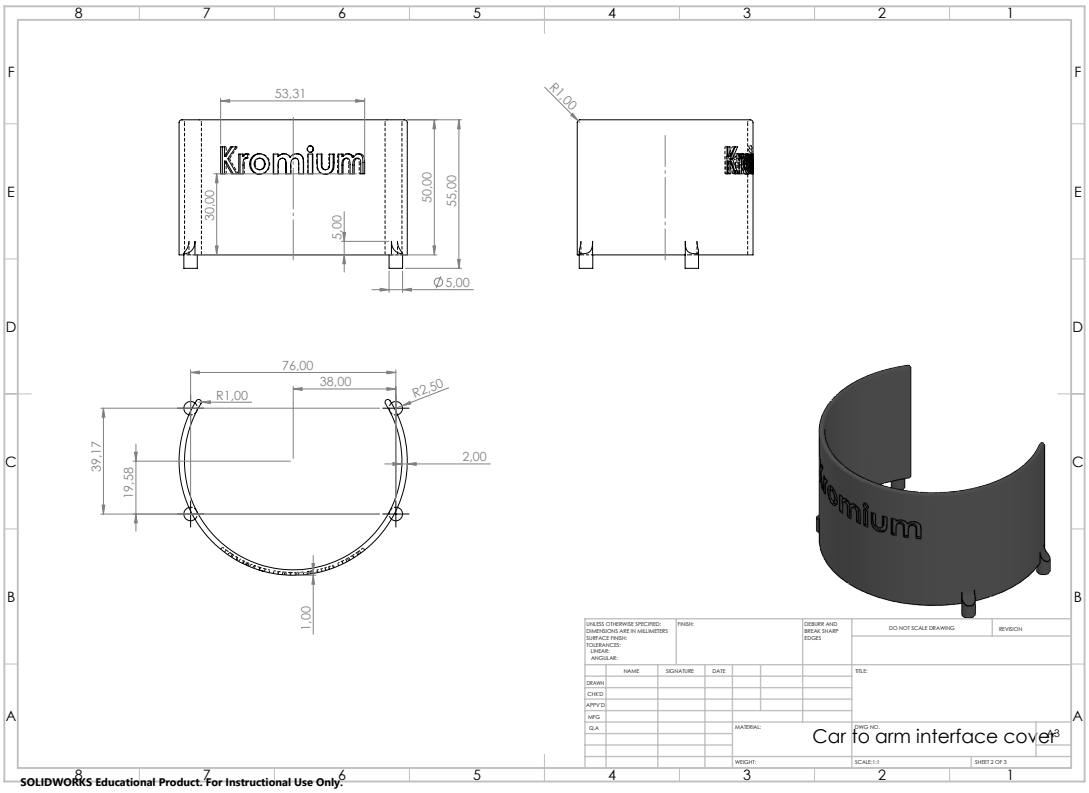


Figure T.2: Car to arm interface cover technical drawing

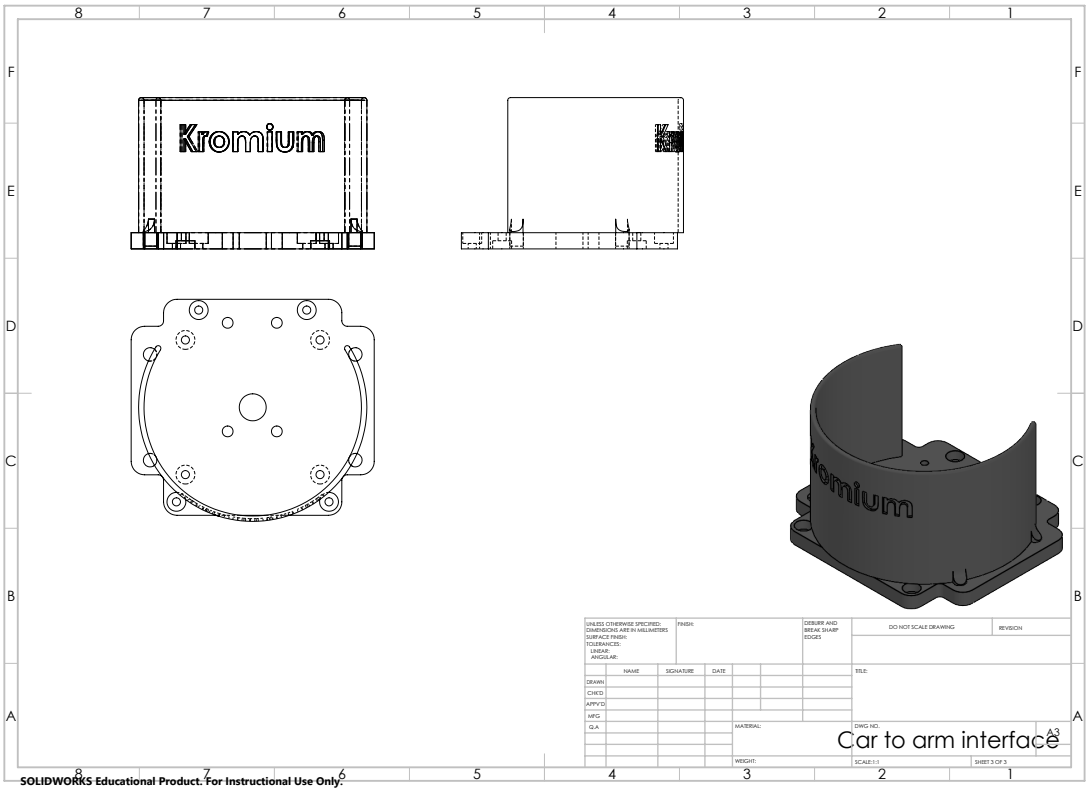


Figure T.3: Car to arm interface assembly technical drawing

T.1.2 Car cover iteration one

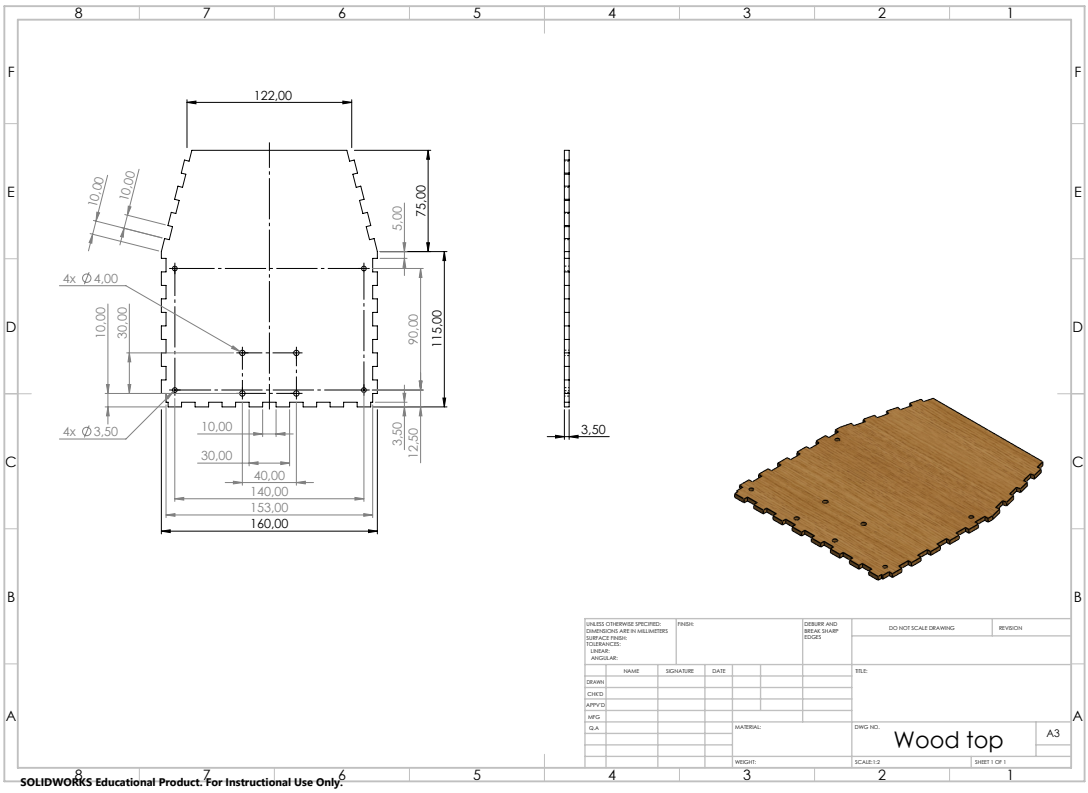


Figure T.4: Top of the temporary cover

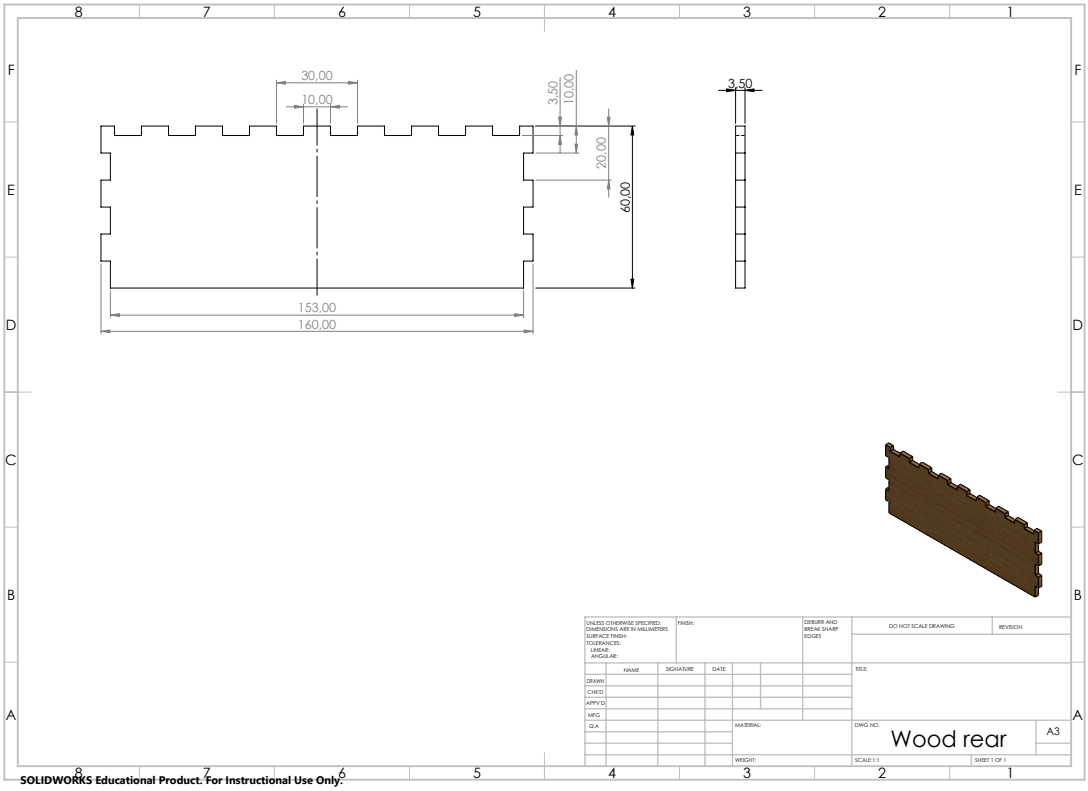


Figure T.5: Rear part of the temporary cover

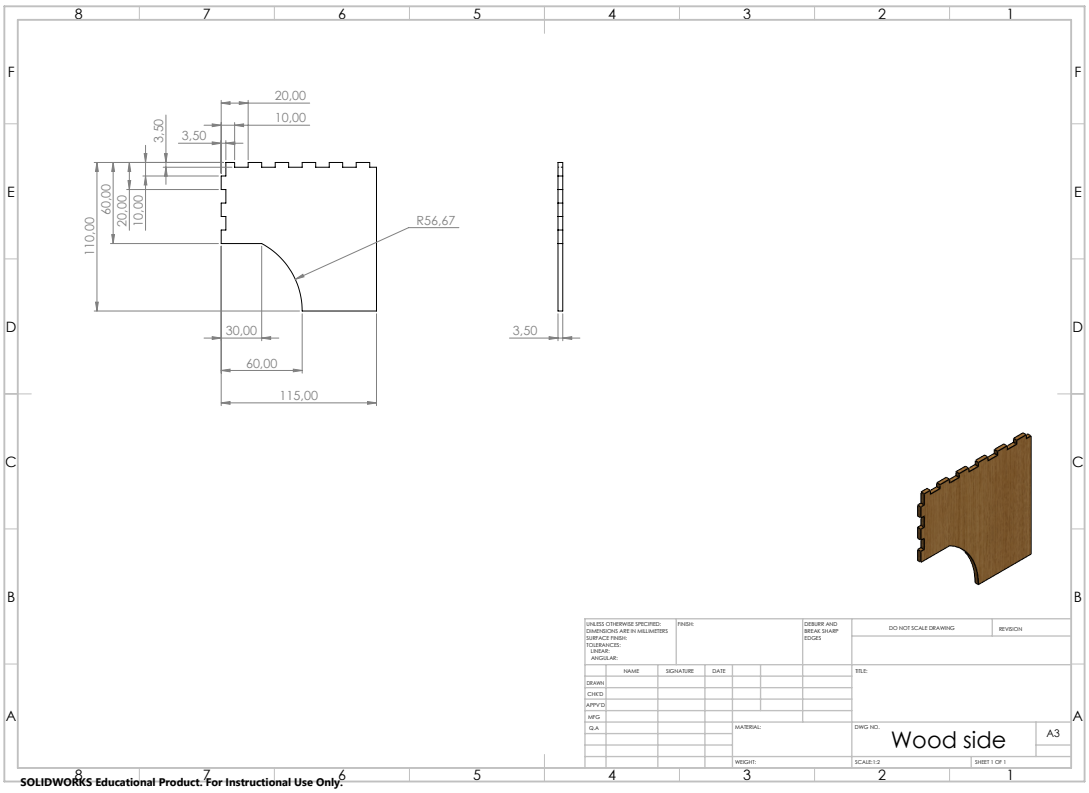


Figure T.6: Rear side piece of the temporary cover

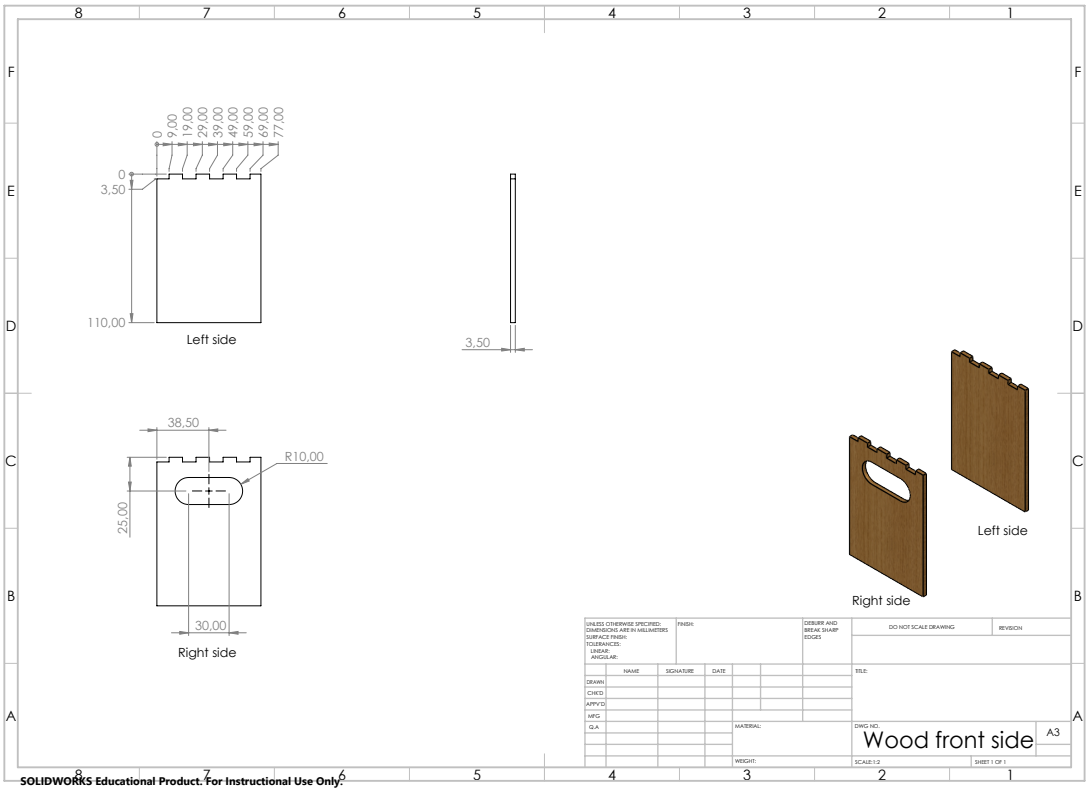


Figure T.7: Front side part of the temporary cover

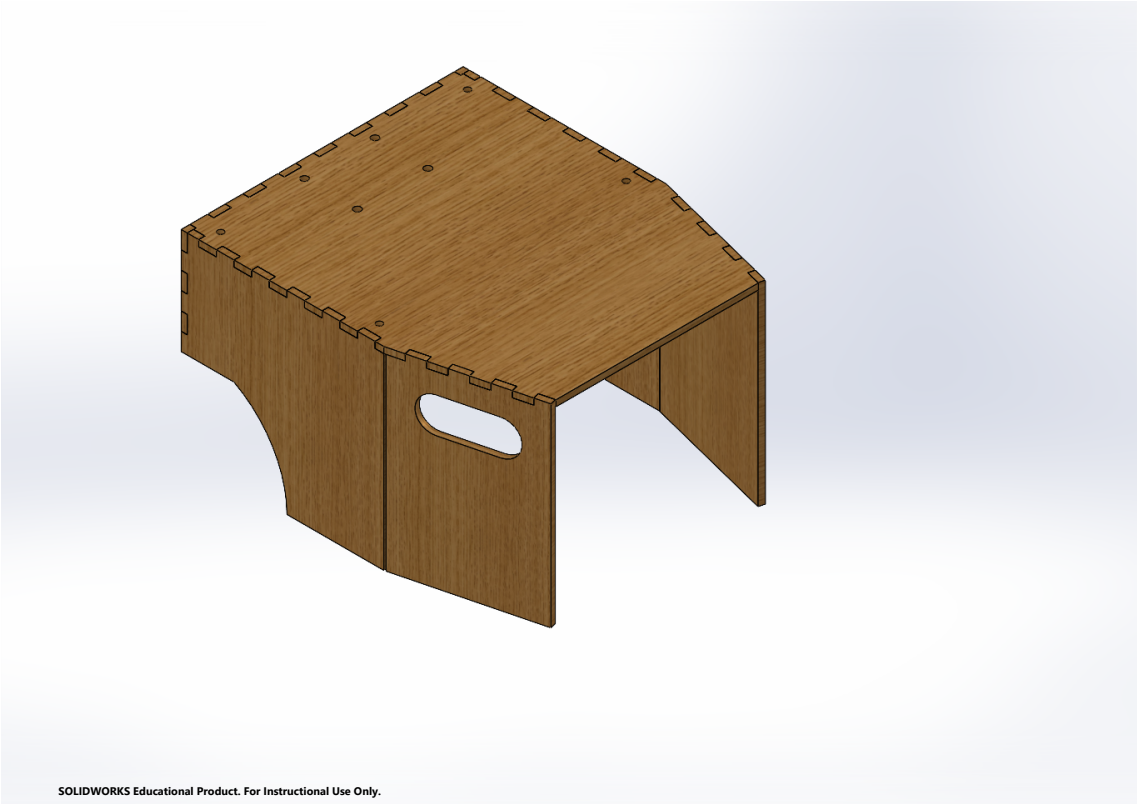


Figure T.8: Assembly of the temporary cover

T.1.3 Camera holder iteration one

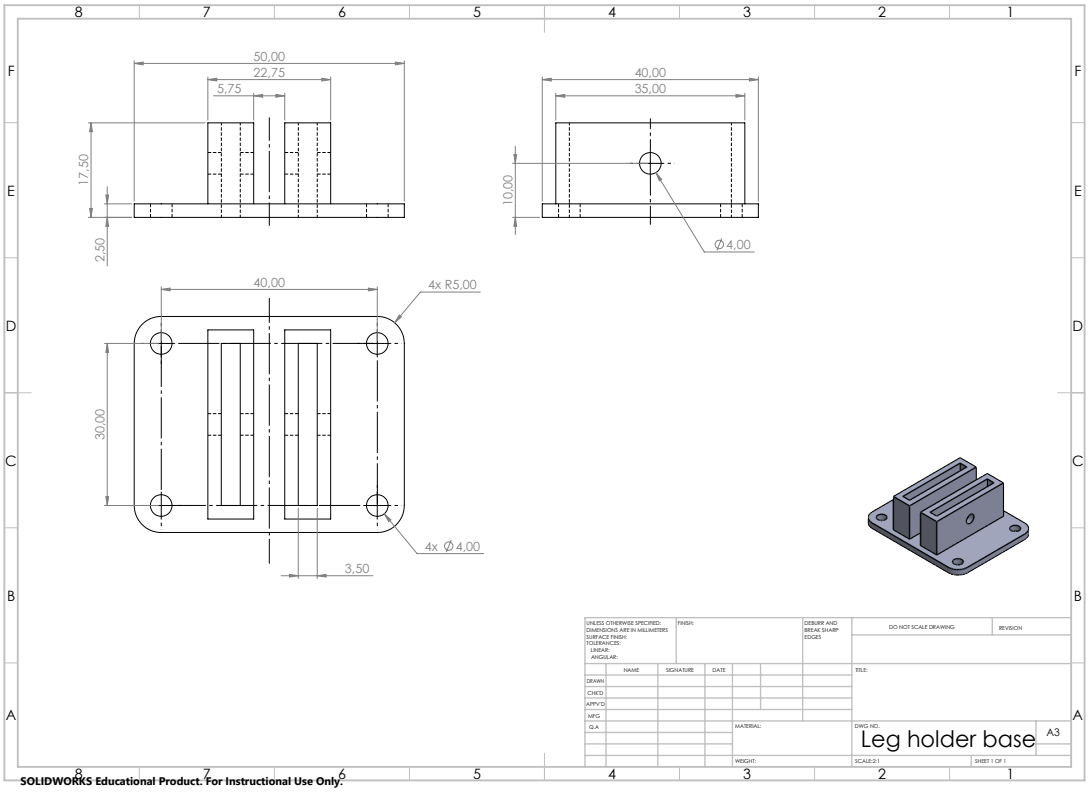


Figure T.9: Base for temporary camera support

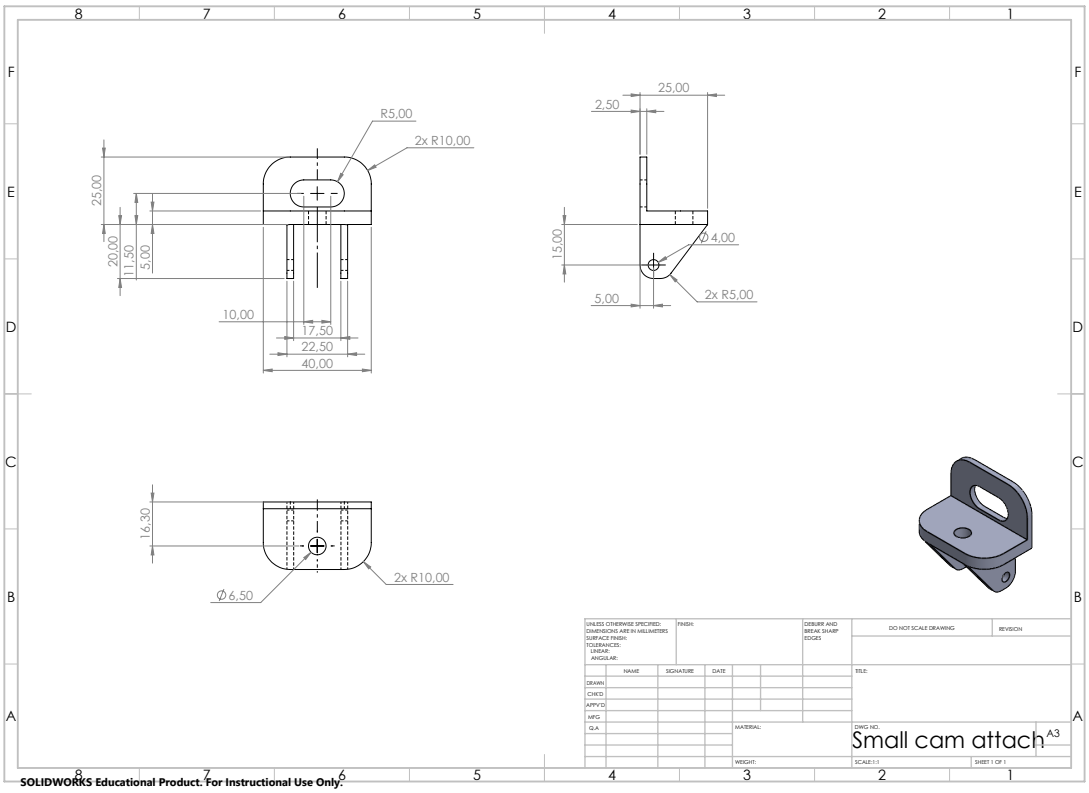


Figure T.10: Holder for the temporary camera

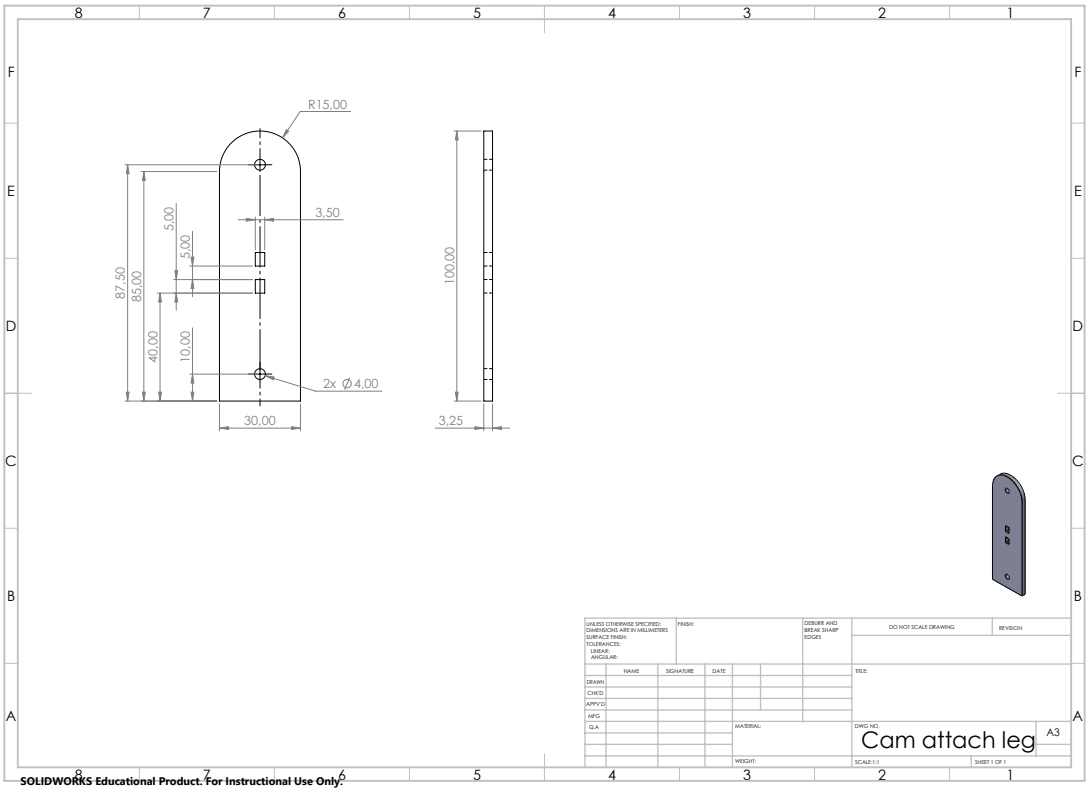


Figure T.11: Legs for the temporary camera

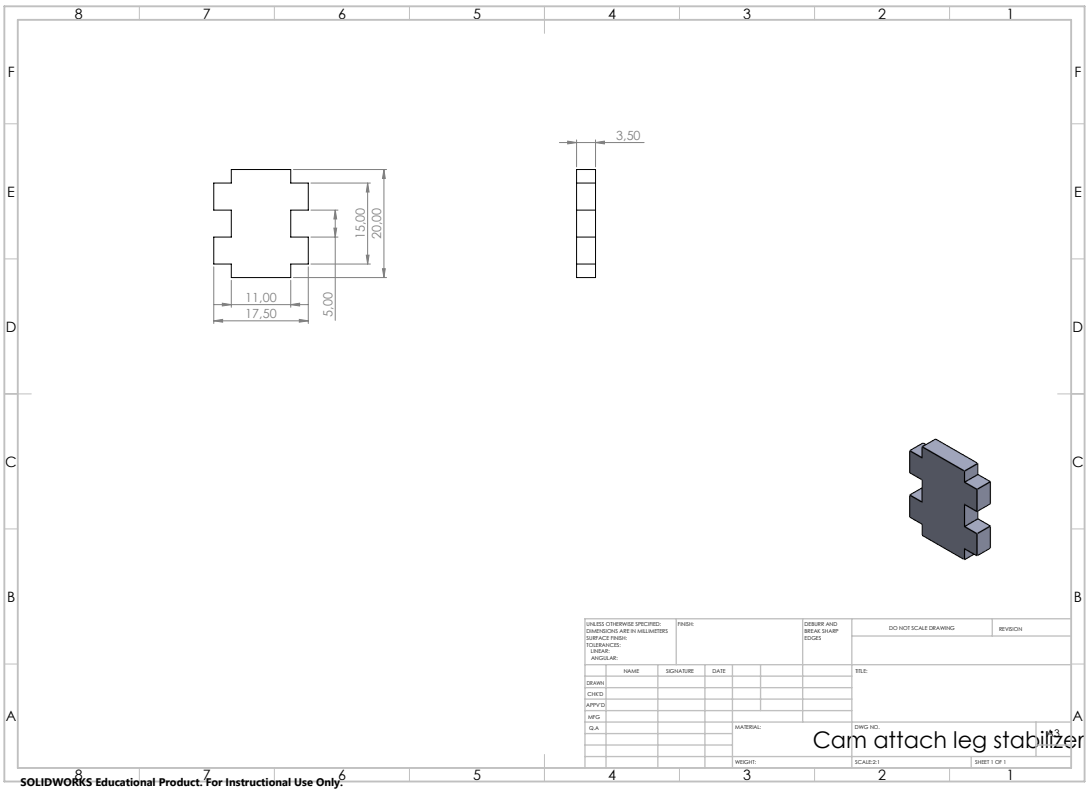


Figure T.12: Stabilizer for the camera legs

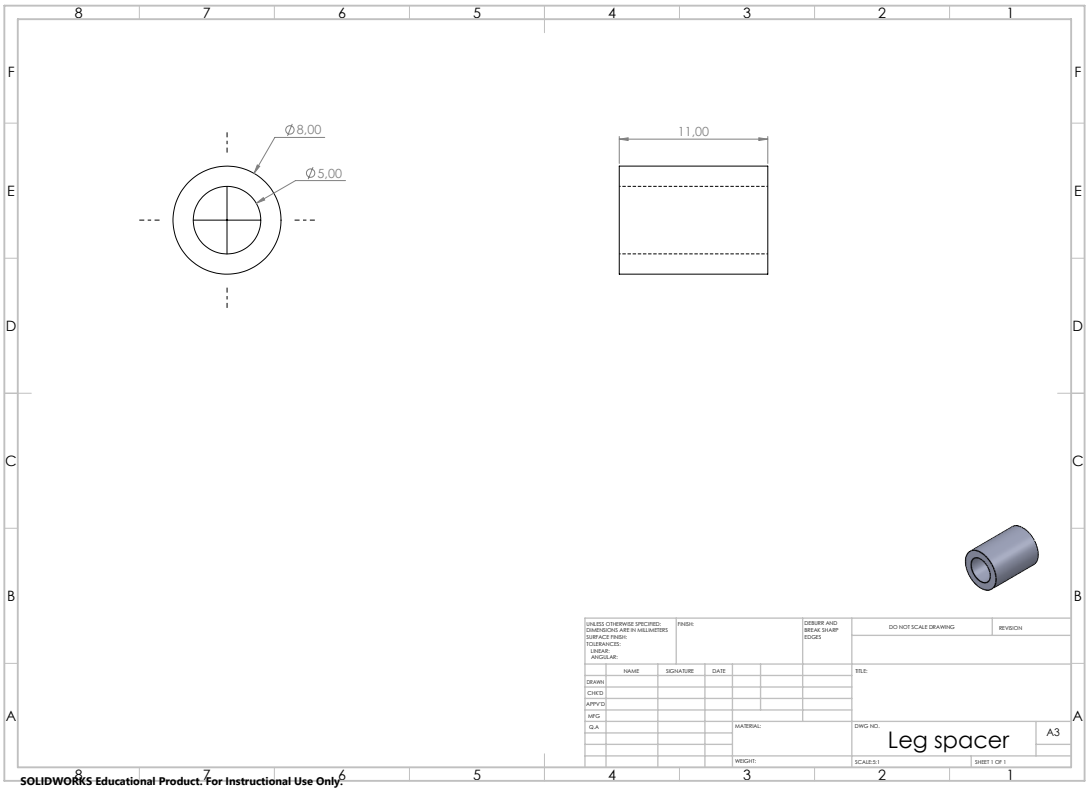


Figure T.13: Spacer for the camera legs

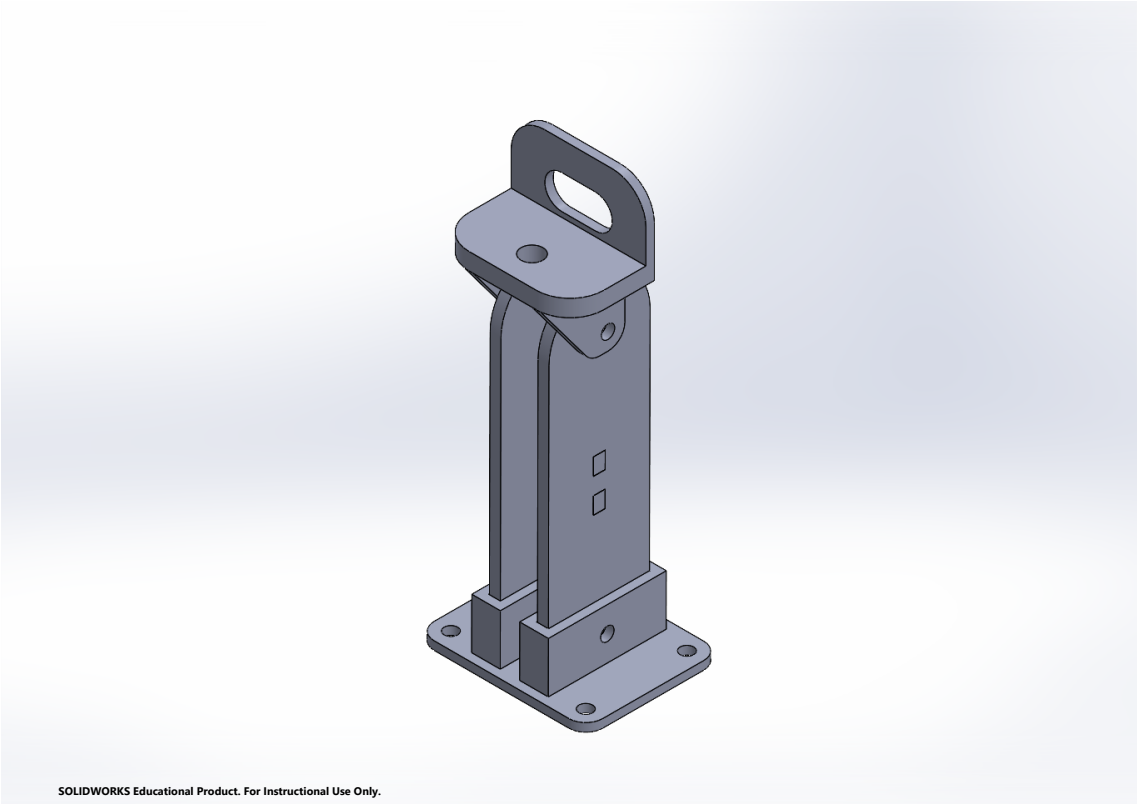


Figure T.14: Temporary camera support assembly

T.1.4 Shock absorber iteration one

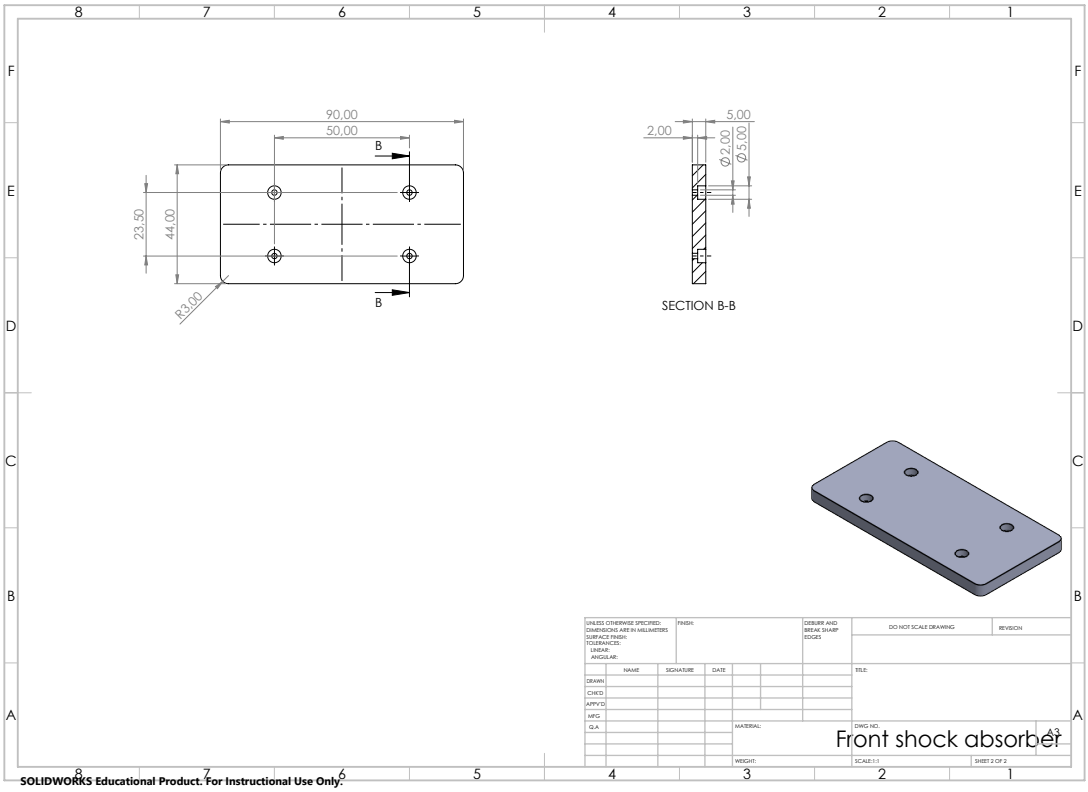


Figure T.15: Shock absorber front short

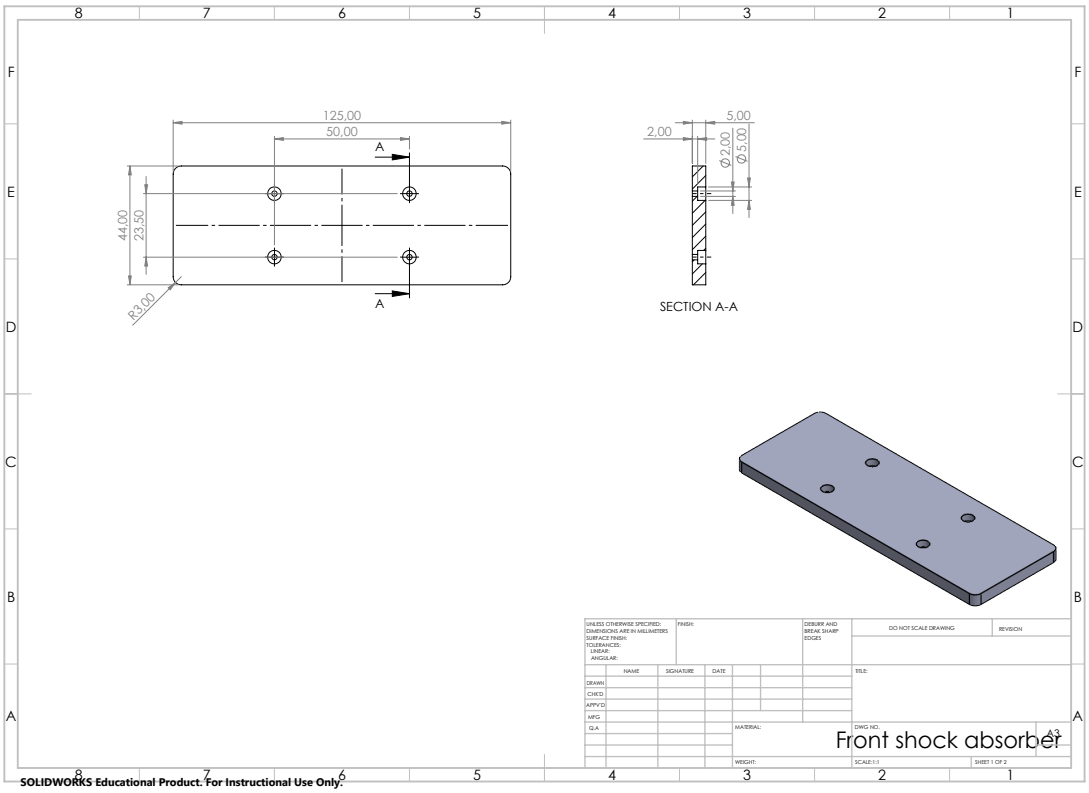


Figure T.16: Shock absorber front long

T.2 Electronics

T.2.1 Motor expansion board YB-ERF01-V1.0 dimensions

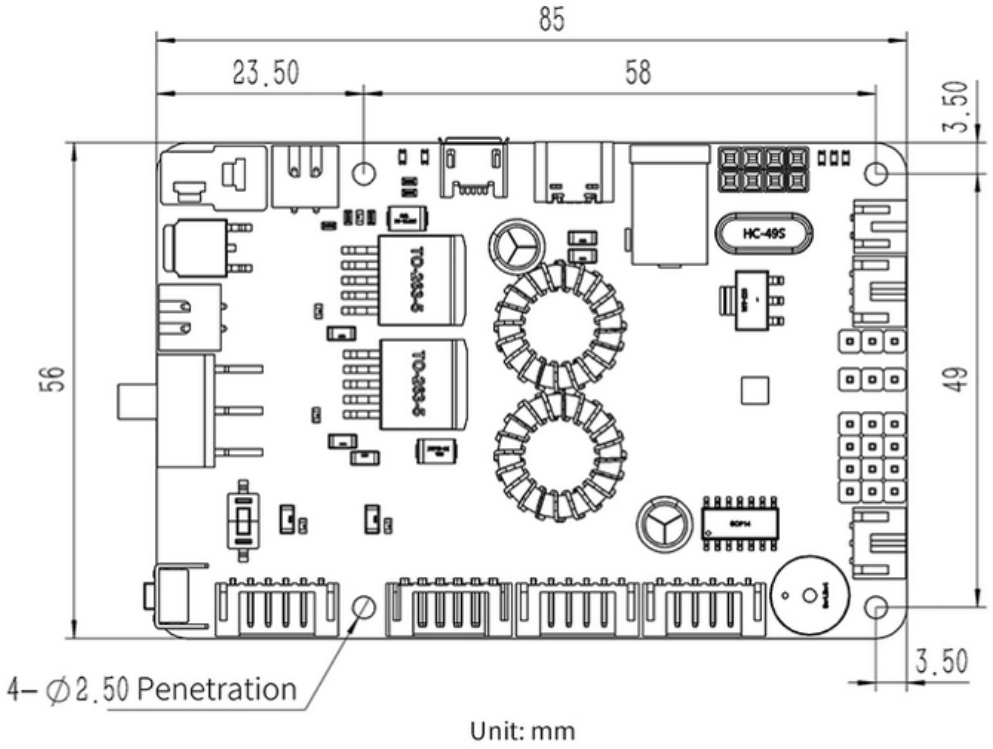


Figure T.17: Caption

T.3 Final parts technical drawings

T.3.1 Floors

TAG	X LOC	Y LOC	SIZE
A1	8	10	Ø3,50 THRU
A2	8	40	Ø3,50 THRU
A3	8	260	Ø3,50 THRU
A4	8	290	Ø3,50 THRU
A5	32	10	Ø3,50 THRU
A6	32	40	Ø3,50 THRU
A7	32	260	Ø3,50 THRU
A8	32	290	Ø3,50 THRU
A9	148	10	Ø3,50 THRU
A10	148	40	Ø3,50 THRU
A11	148	260	Ø3,50 THRU
A12	148	290	Ø3,50 THRU
A13	172	10	Ø3,50 THRU
A14	172	40	Ø3,50 THRU
A15	172	260	Ø3,50 THRU
A16	172	290	Ø3,50 THRU
B1	19,95	56,25	Ø3,10 THRU
B2	19,95	243,75	Ø3,10 THRU
B3	62,50	17,50	Ø3,10 THRU
B4	62,50	282,50	Ø3,10 THRU
B5	72,50	17,50	Ø3,10 THRU
B6	72,50	282,50	Ø3,10 THRU
B7	107,50	17,50	Ø3,10 THRU
B8	107,50	282,50	Ø3,10 THRU
B9	117,50	17,50	Ø3,10 THRU
B10	117,50	282,50	Ø3,10 THRU
B11	130,05	56,25	Ø3,10 THRU
B12	130,05	243,75	Ø3,10 THRU
B13	162,50	80	Ø3,10 THRU
B14	162,50	90	Ø3,10 THRU
B15	162,50	210	Ø3,10 THRU
B16	162,50	220	Ø3,10 THRU
C1	30	55	Ø3,00 THRU
C2	30	245	Ø3,00 THRU
C3	90	8	Ø3,00 THRU
C4	90	292	Ø3,00 THRU
C5	155	245	Ø3,00 THRU
C6	174	70	Ø3,00 THRU

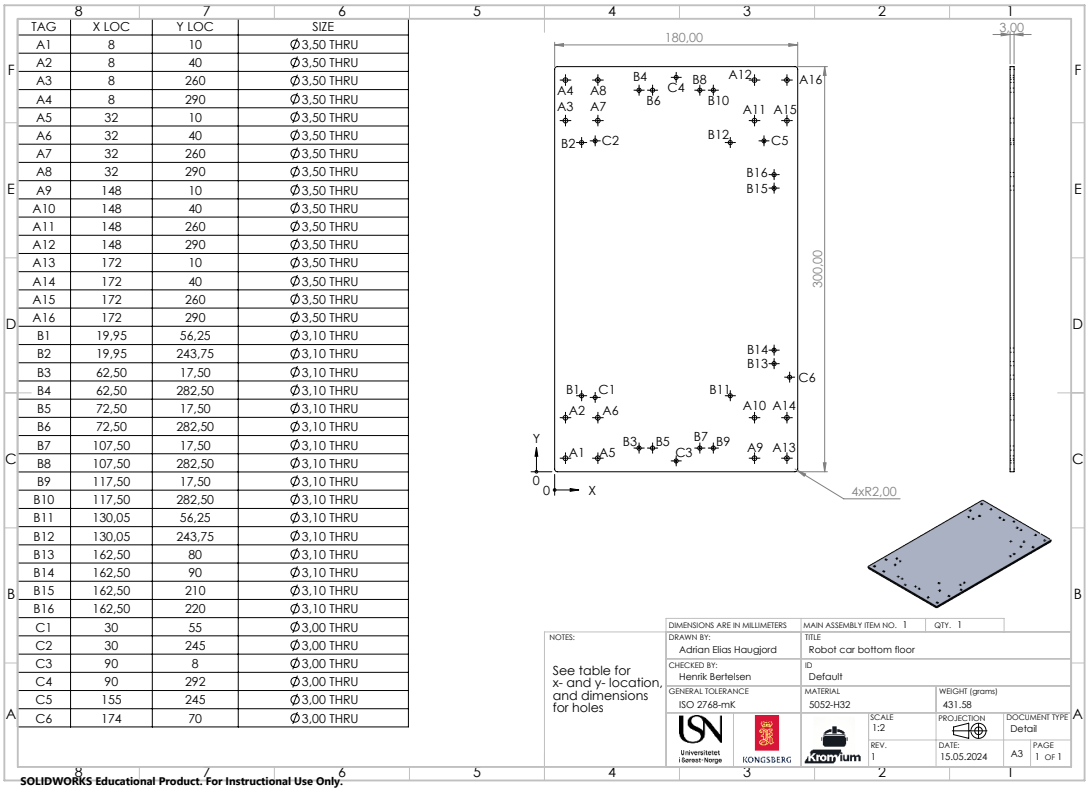


Figure T.18: Robot car bottom floor

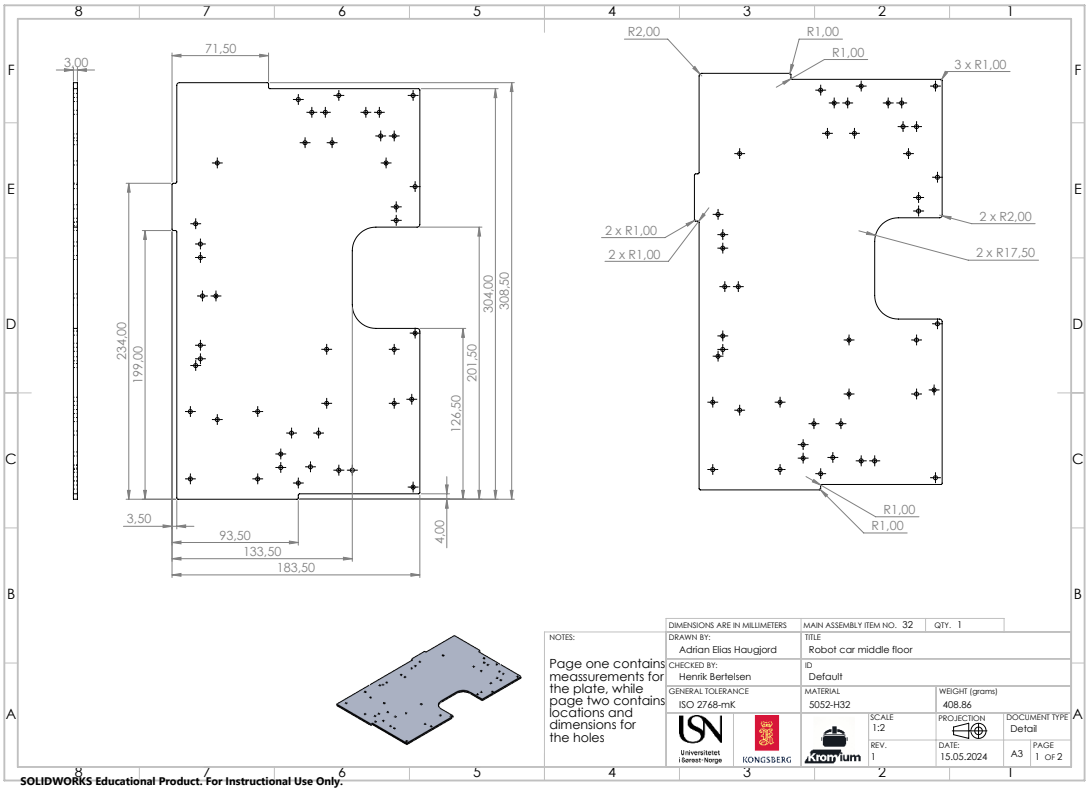
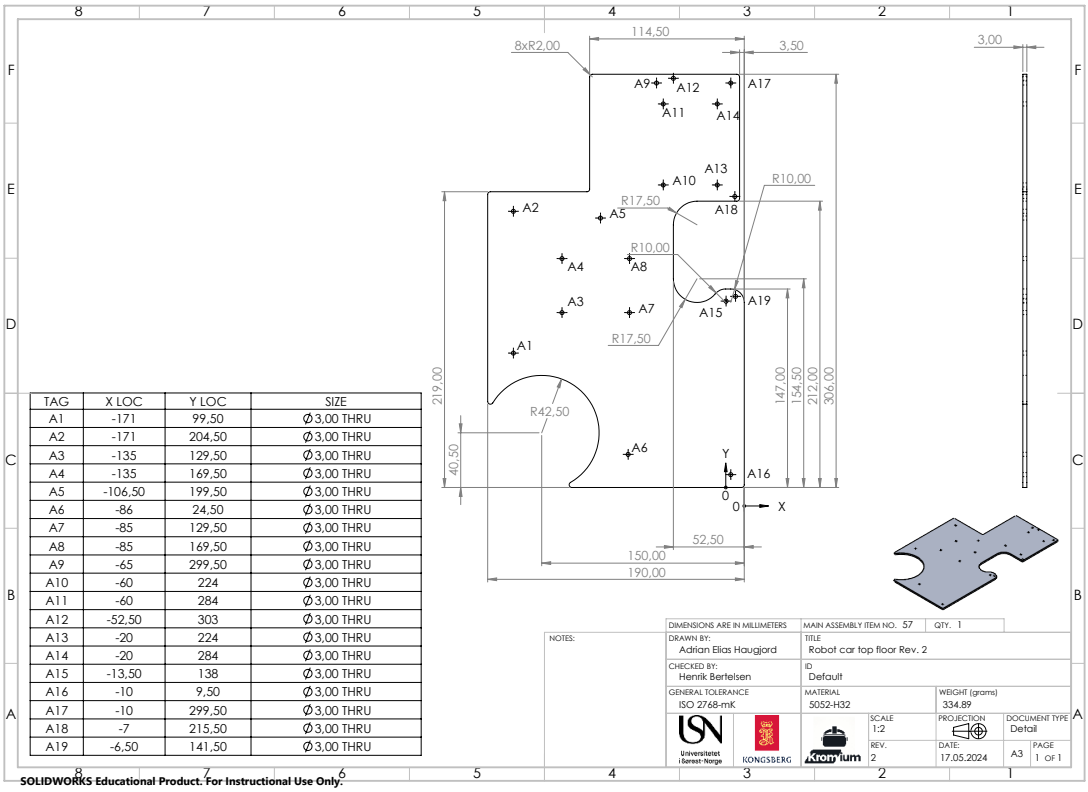


Figure T.19: Robot car middle floor



TAG	X LOC	Y LOC	SIZE
A1	-171	99,50	Ø3,00 THRU
A2	-171	204,50	Ø3,00 THRU
A3	-135	129,50	Ø3,00 THRU
A4	-135	169,50	Ø3,00 THRU
A5	-106,50	199,50	Ø3,00 THRU
A6	-86	24,50	Ø3,00 THRU
A7	-85	129,50	Ø3,00 THRU
A8	-85	169,50	Ø3,00 THRU
A9	-65	299,50	Ø3,00 THRU
A10	-60	224	Ø3,00 THRU
A11	-60	284	Ø3,00 THRU
A12	-52,50	303	Ø3,00 THRU
A13	-20	224	Ø3,00 THRU
A14	-20	284	Ø3,00 THRU
A15	-13,50	138	Ø3,00 THRU
A16	-10	9,50	Ø3,00 THRU
A17	-10	299,50	Ø3,00 THRU
A18	-7	215,50	Ø3,00 THRU
A19	-6,50	141,50	Ø3,00 THRU

Figure T.20: Robot car top floor

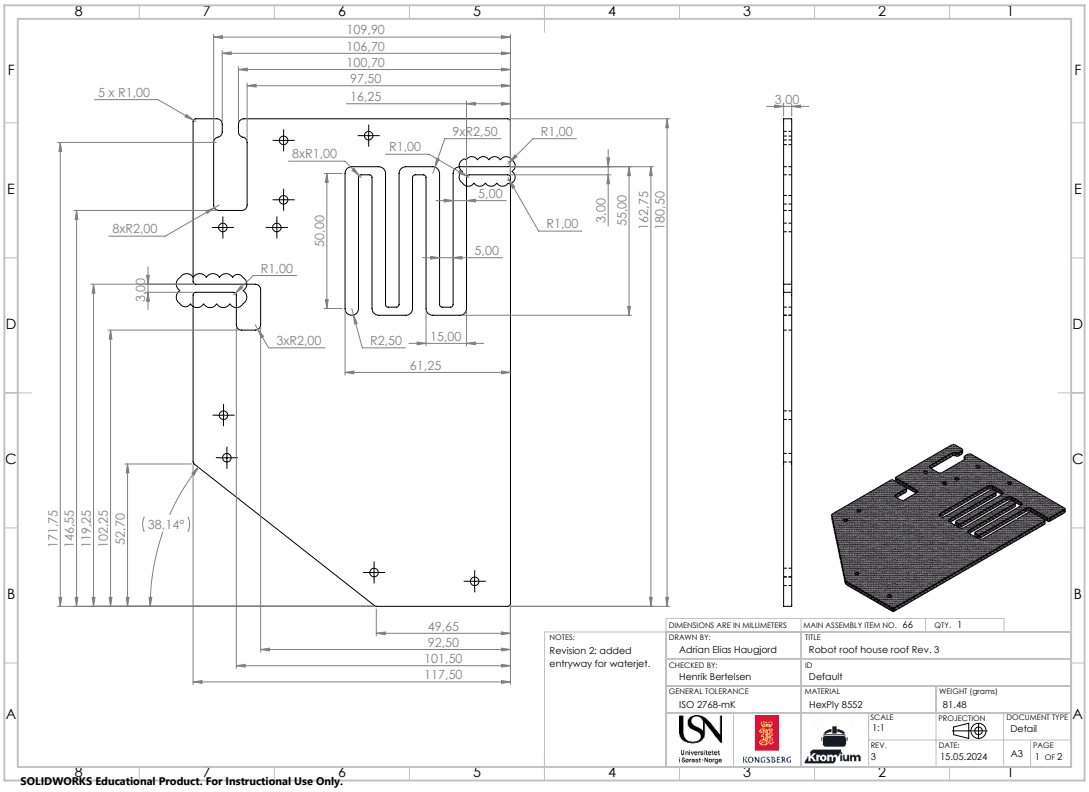


Figure T.21: Robot car roof

T.3.2 Walls

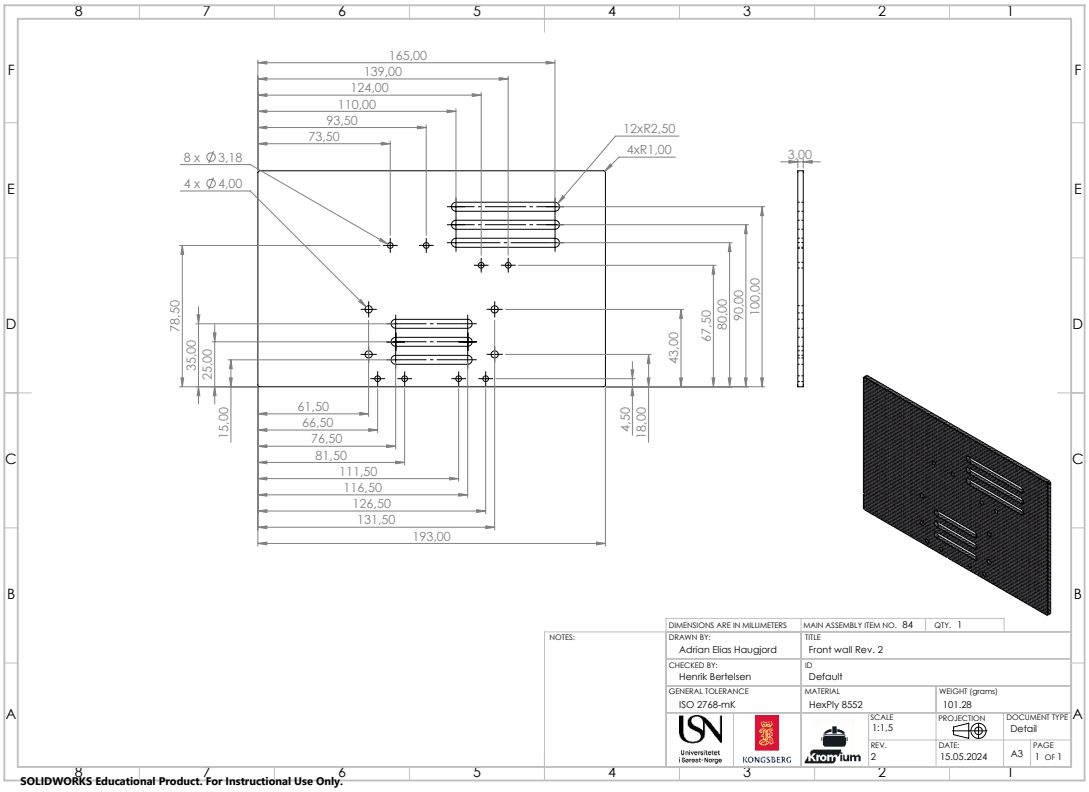


Figure T.22: Robot car front wall

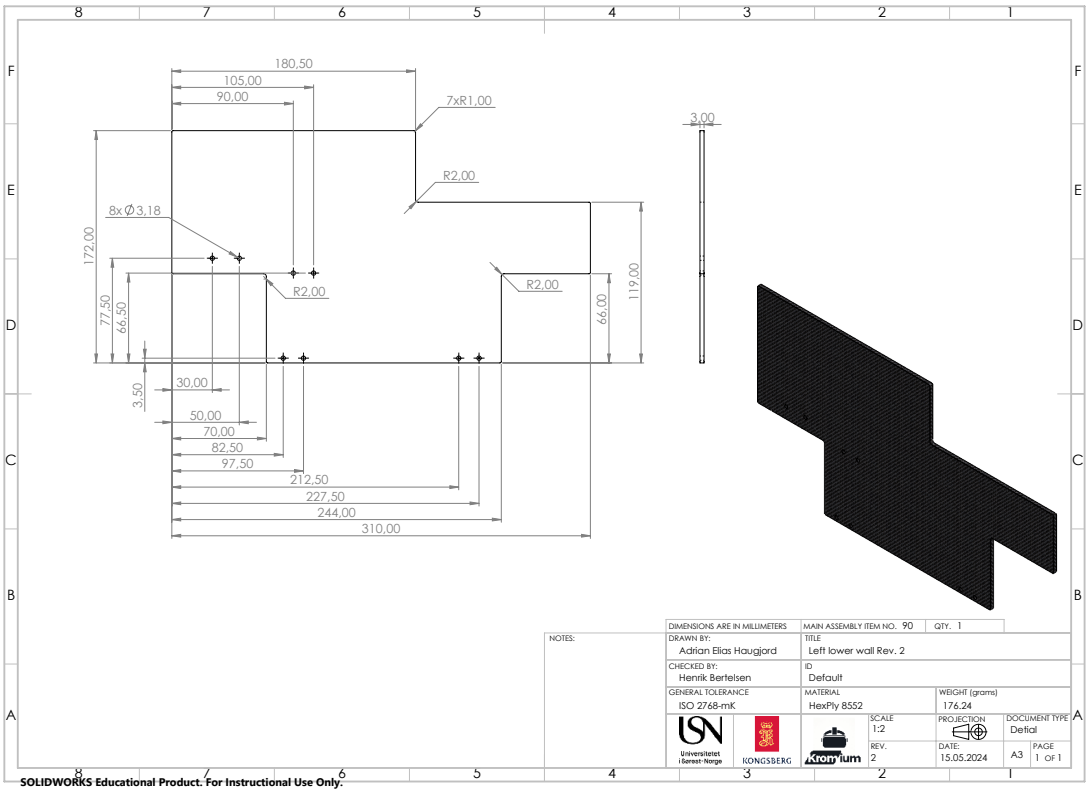


Figure T.23: Robot car left wall

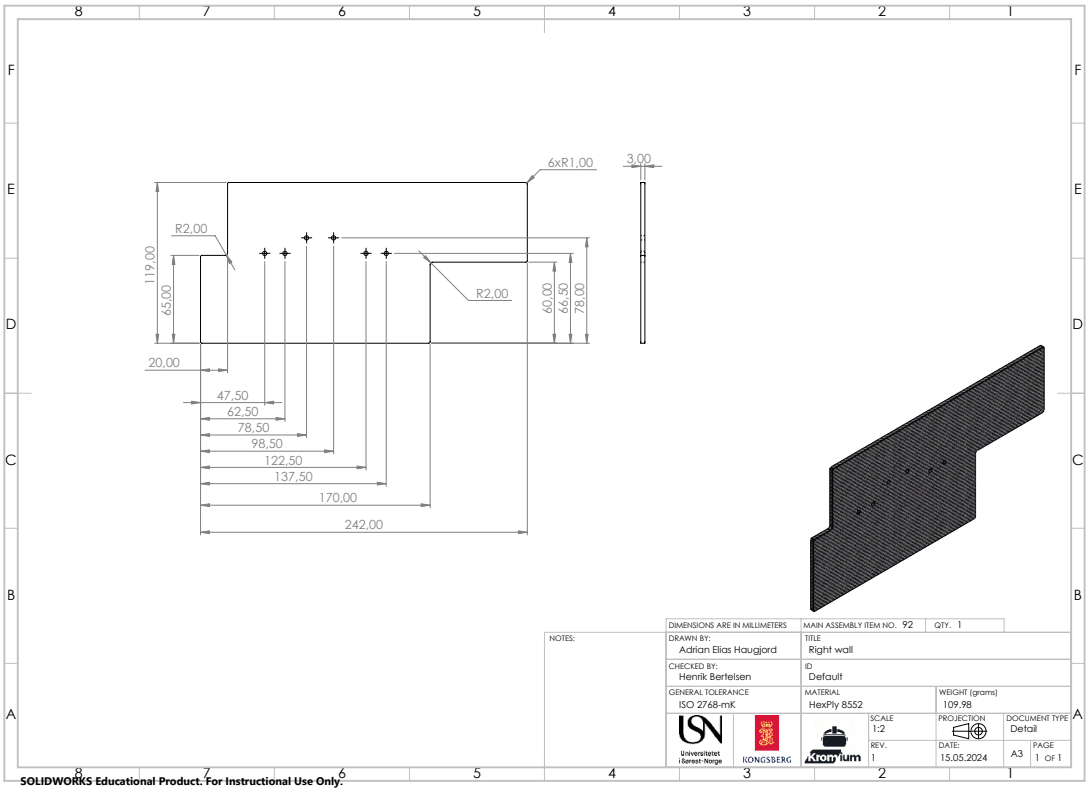


Figure T.24: Robot car right wall

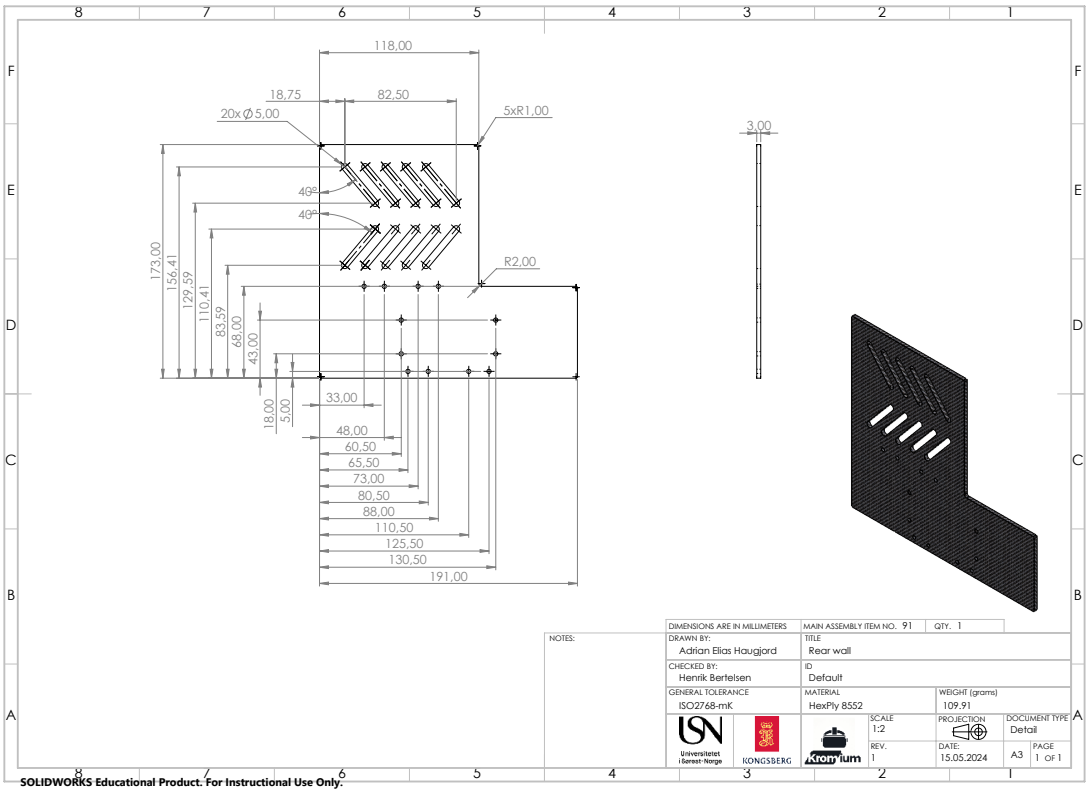


Figure T.25: Robot car rear wall

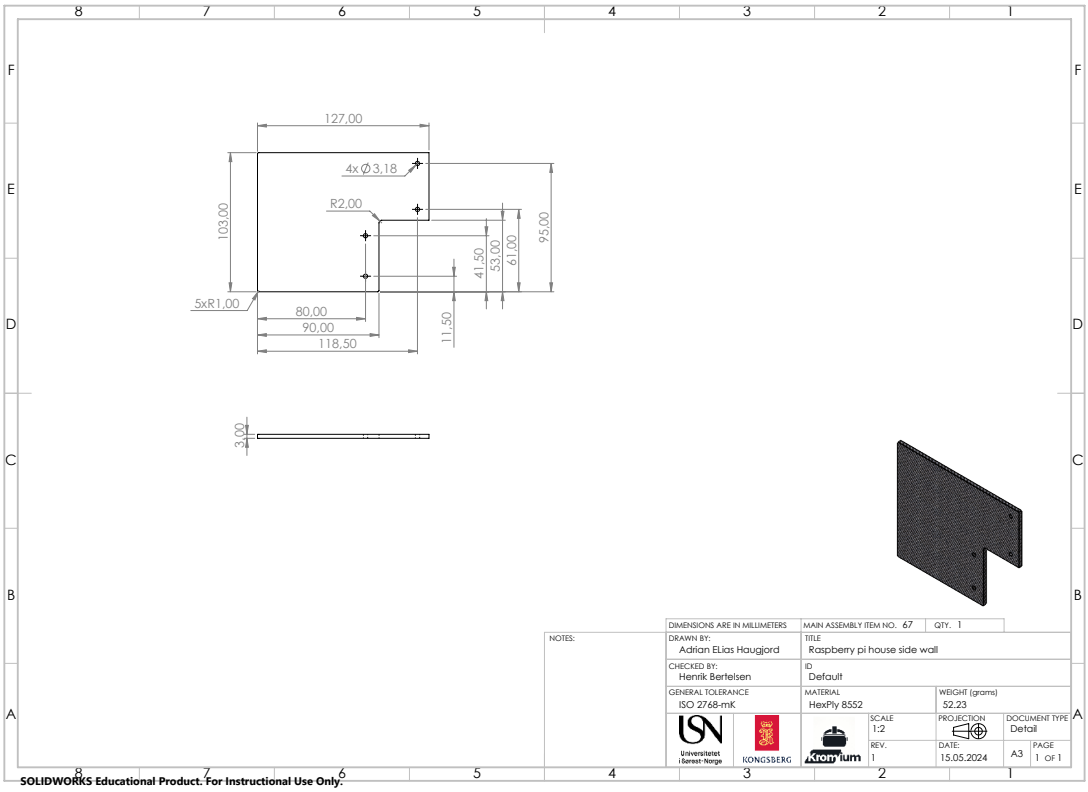


Figure T.26: Raspberry Pi house large wall

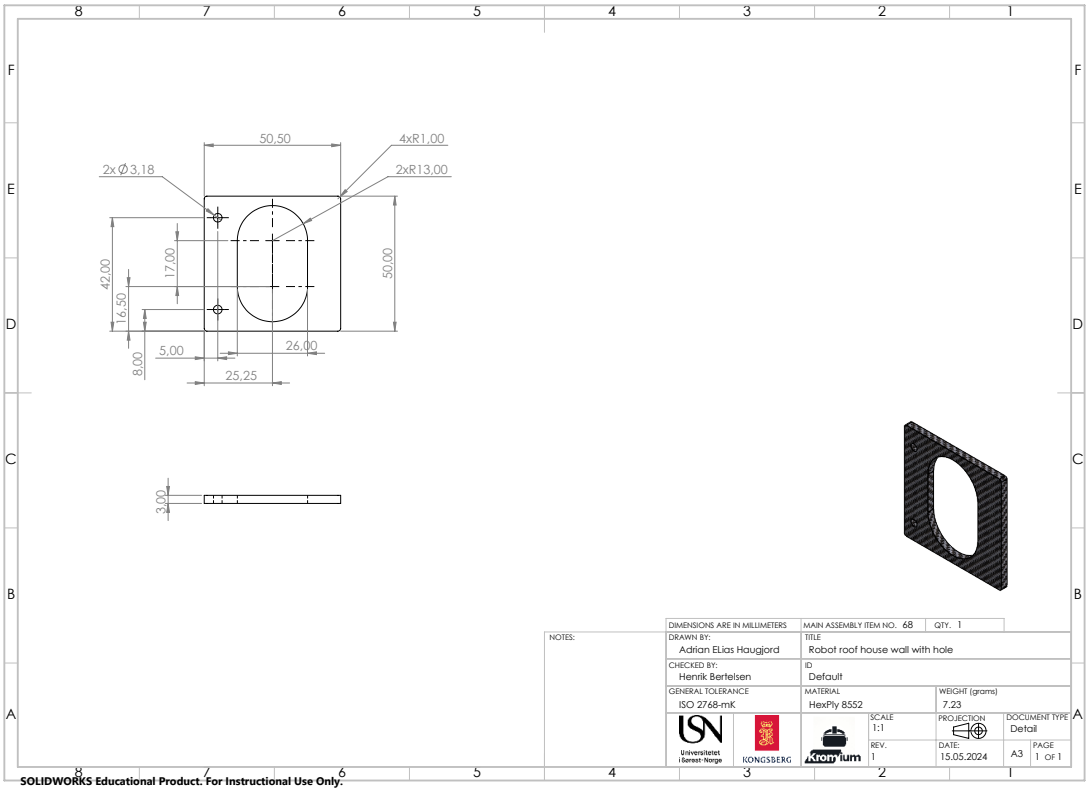


Figure T.27: Raspberry Pi house open wall

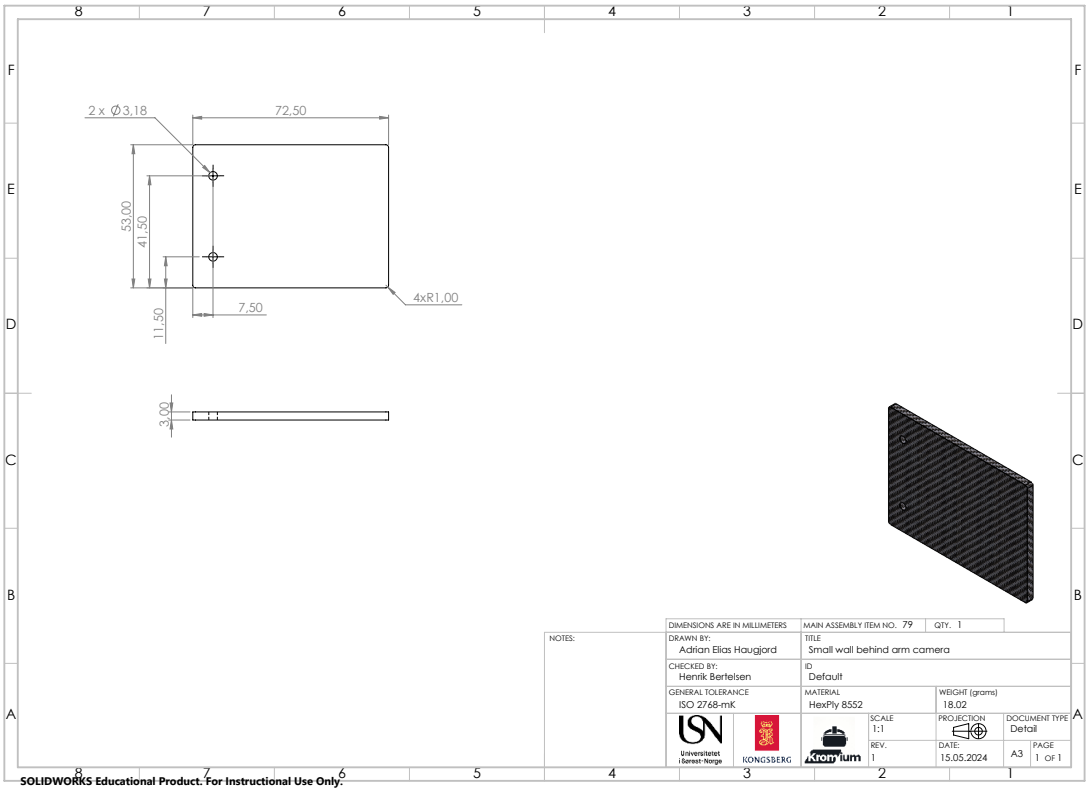


Figure T.28: Wall behind robot arm camera

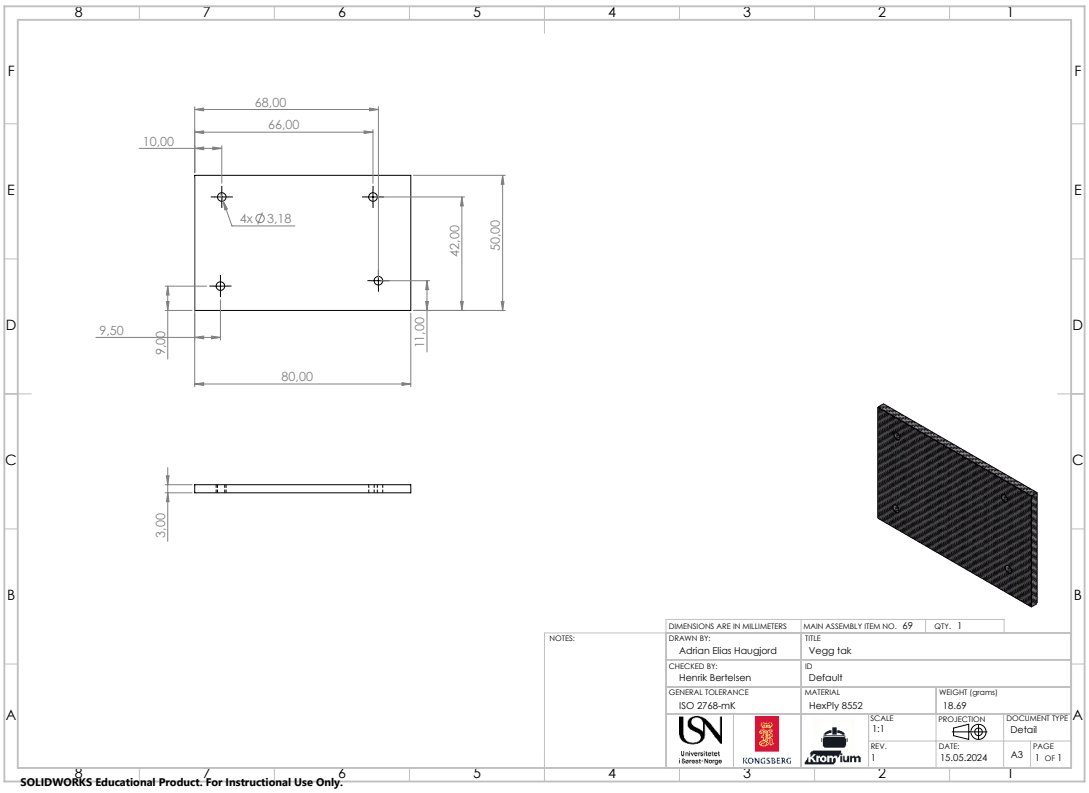


Figure T.29: Raspberry Pi house middle wall

T.3.3 Expansion board brackets

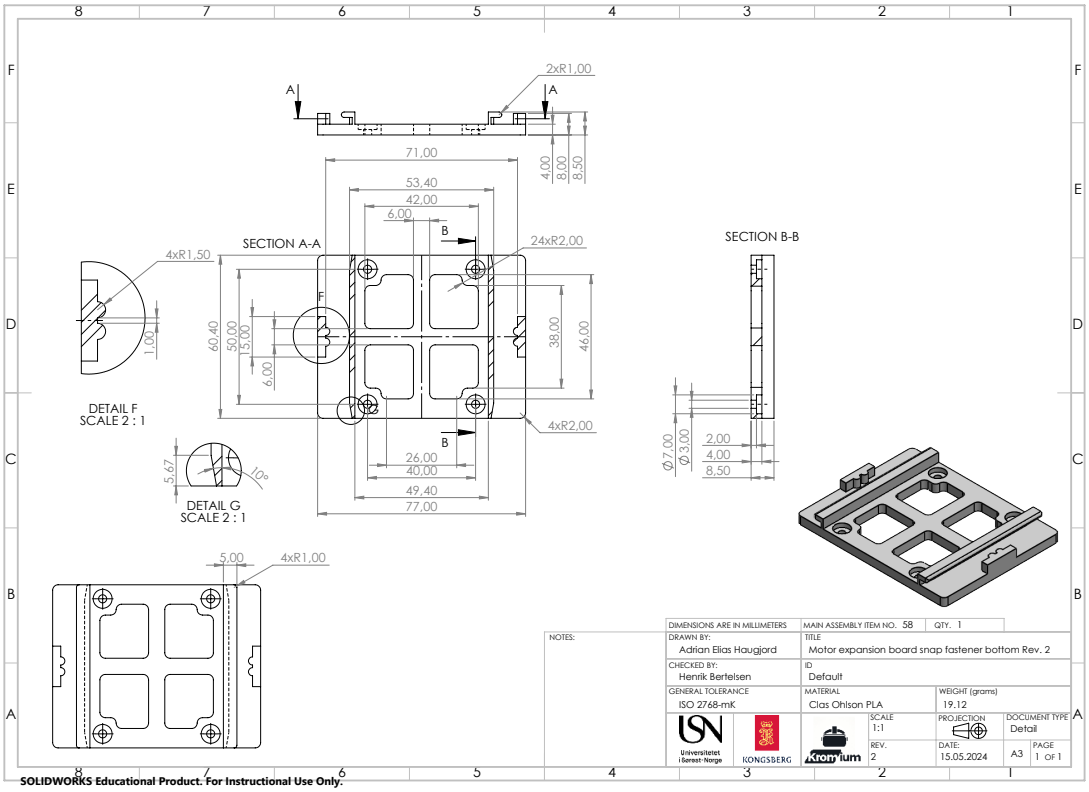


Figure T.30: Motor expansion board snap bracket bottom

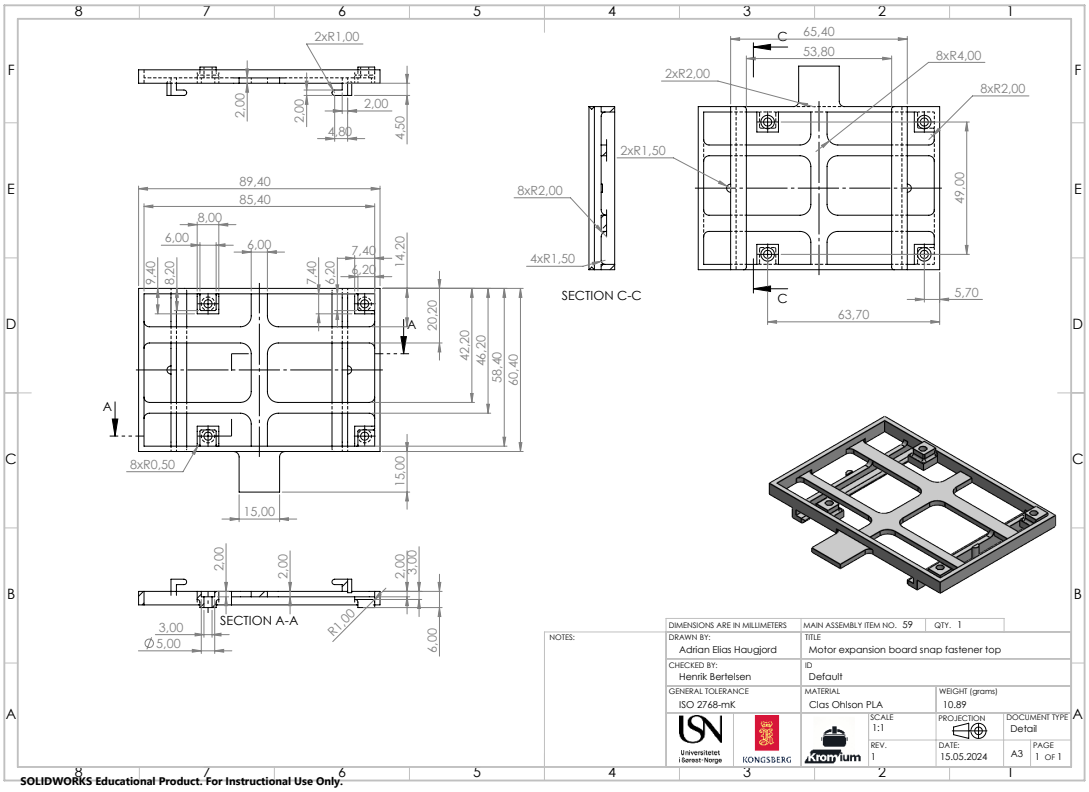


Figure T.31: Motor expansion board snap bracket top

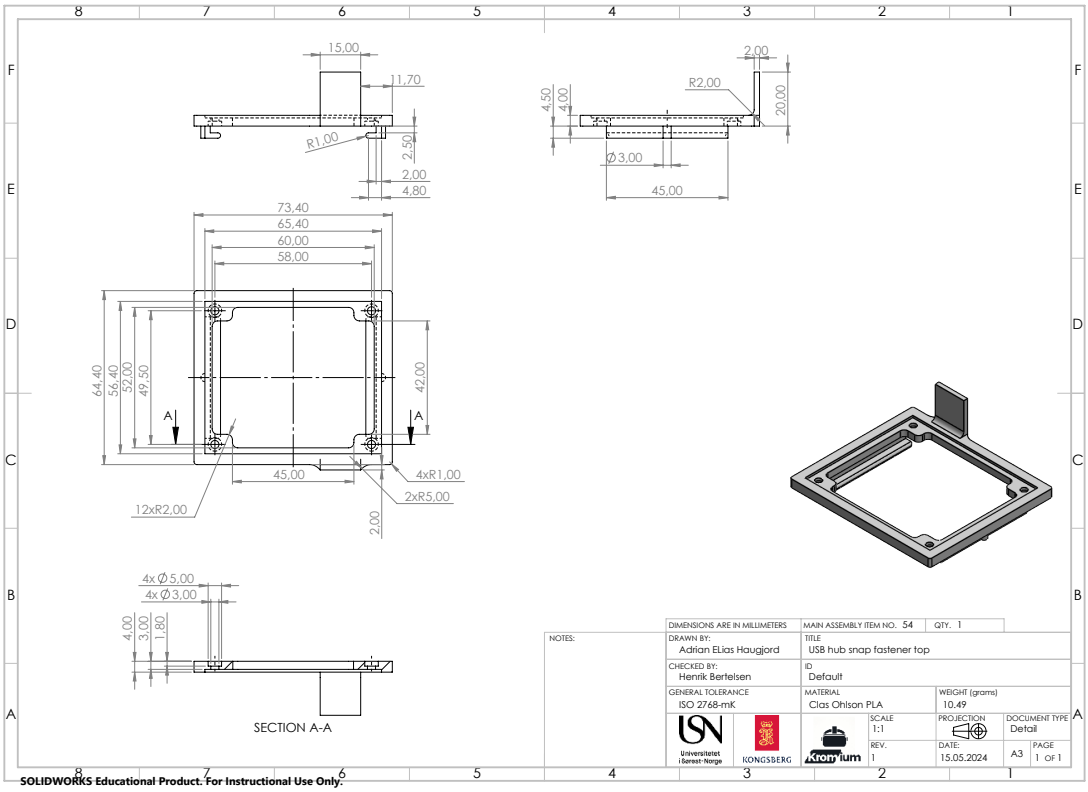


Figure T.32: USB hub snap bracket top

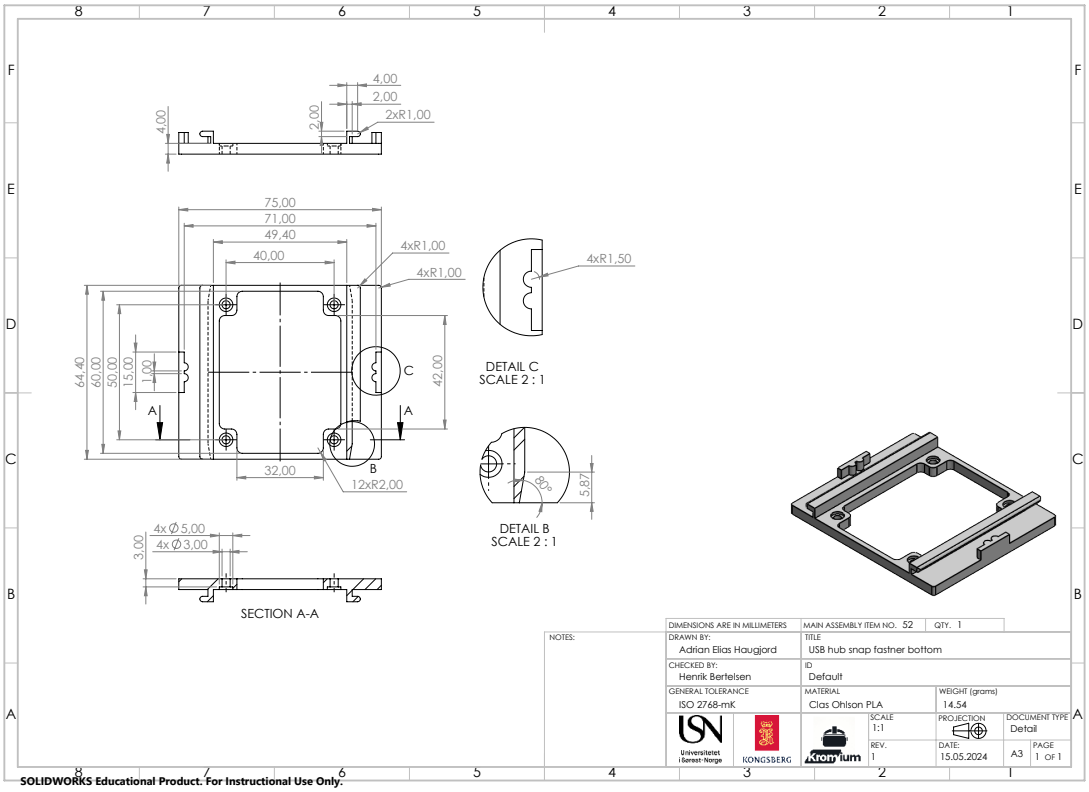


Figure T.33: USB hub snap bracket bottom

T.3.4 Raspberry Pi case bracket

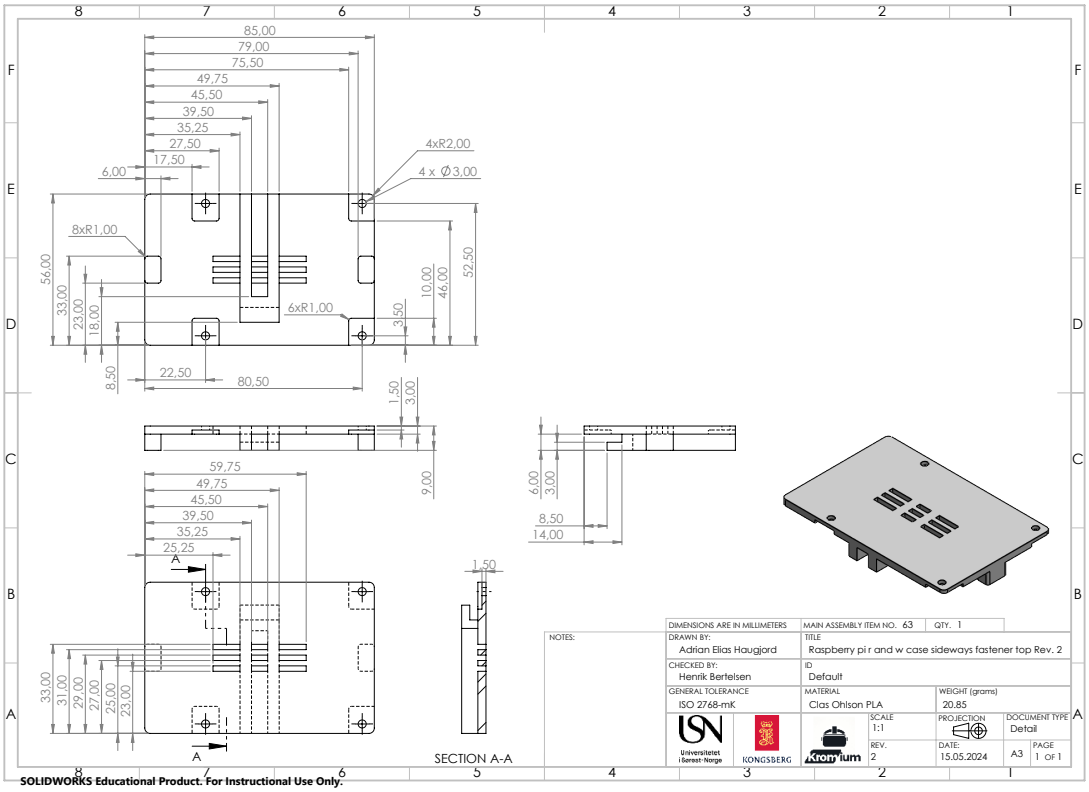


Figure T.34: Raspberry Pi bracket top

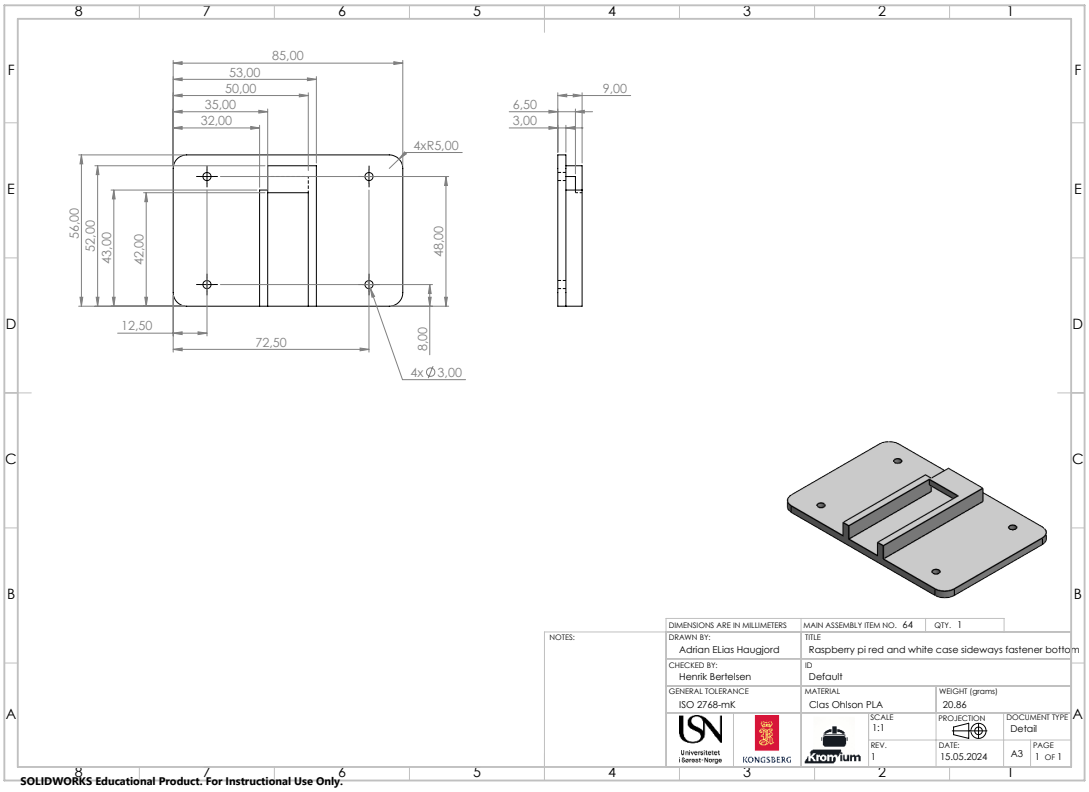


Figure T.35: Raspberry Pi bracket bottom

T.3.5 Magnet brackets

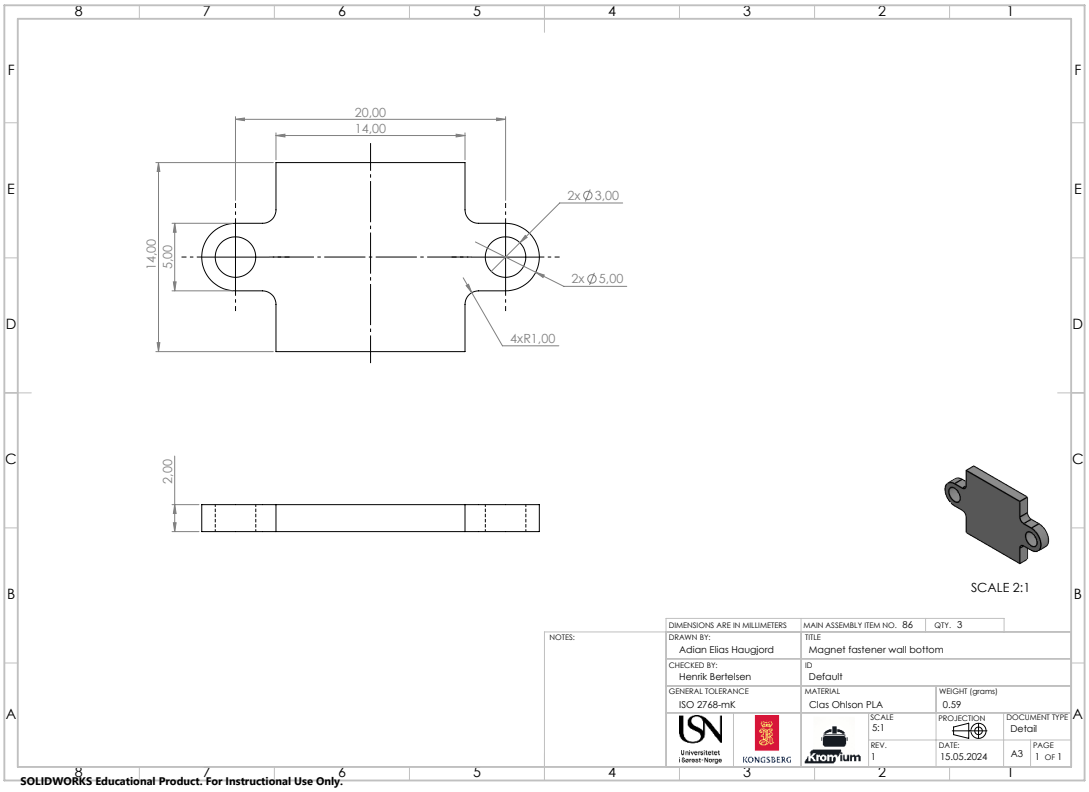


Figure T.36: Magnet bracket wall bottom

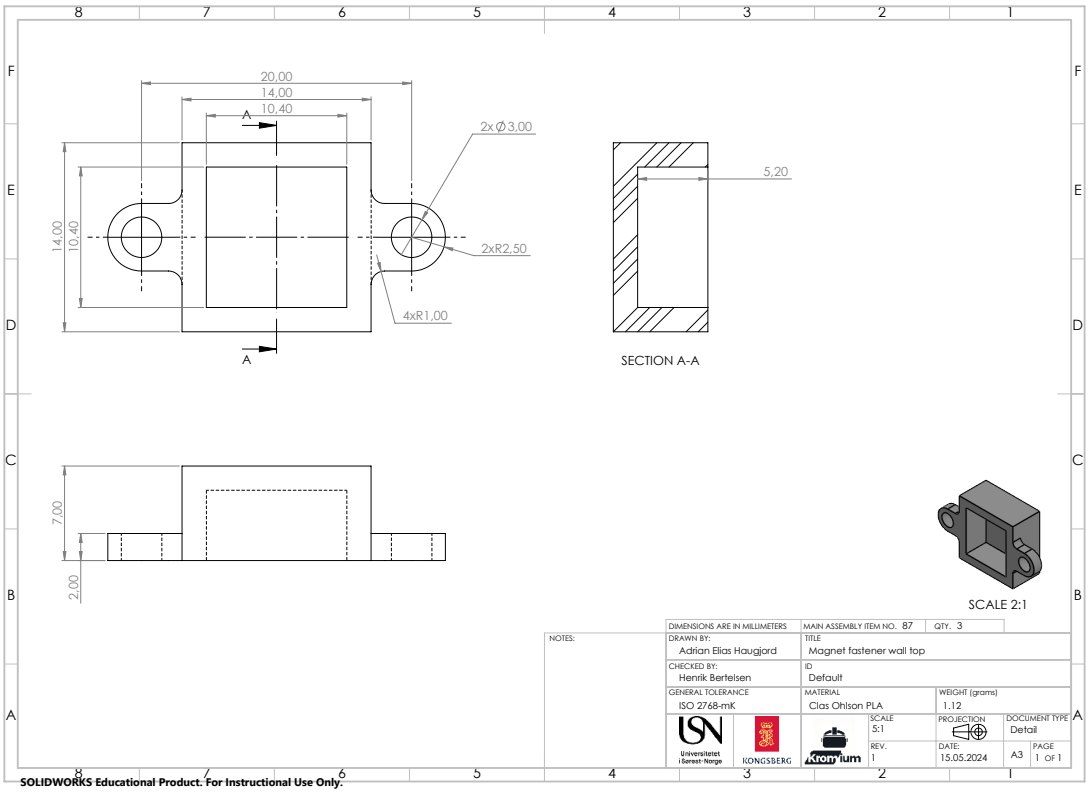


Figure T.37: Magnet bracket wall top

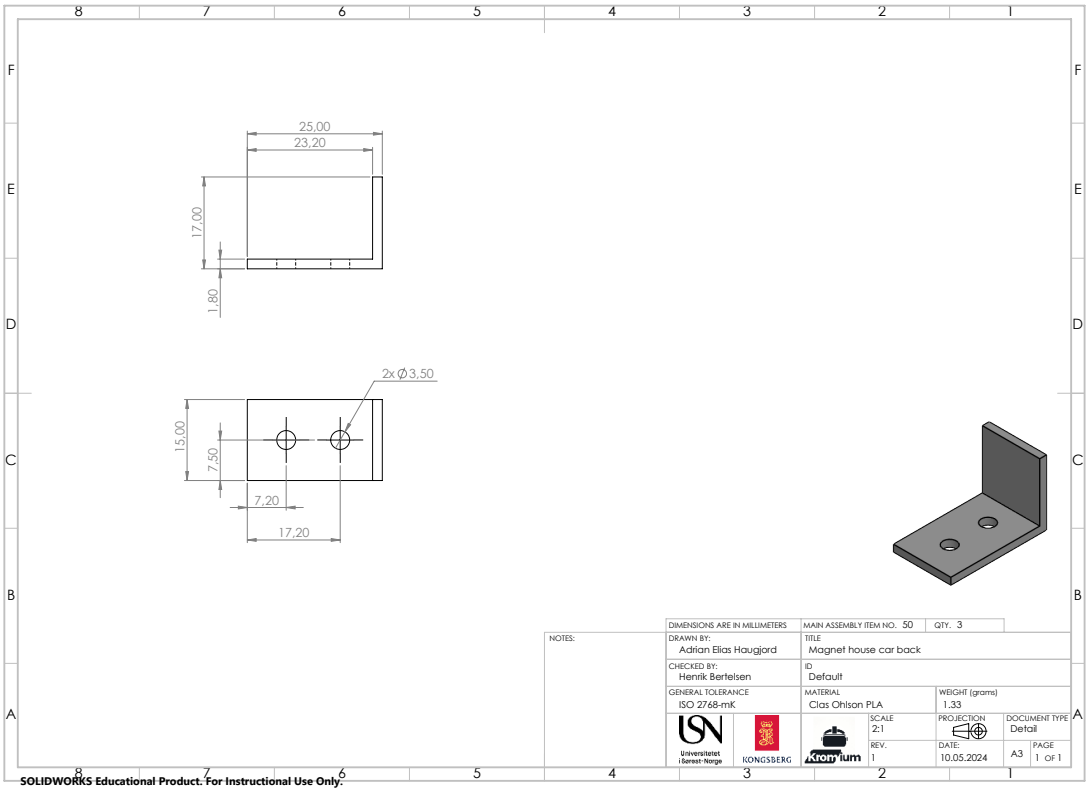


Figure T.38: Magnet bracket car back

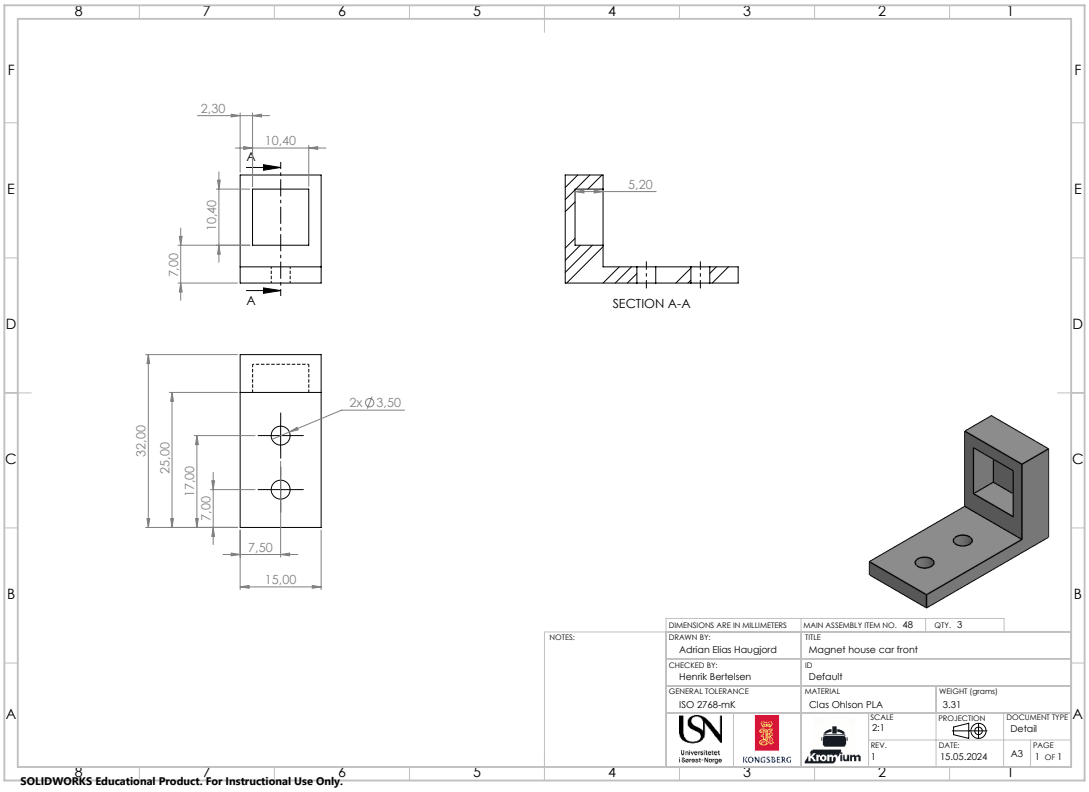


Figure T.39: Magnet bracket car front

T.3.6 Brackets

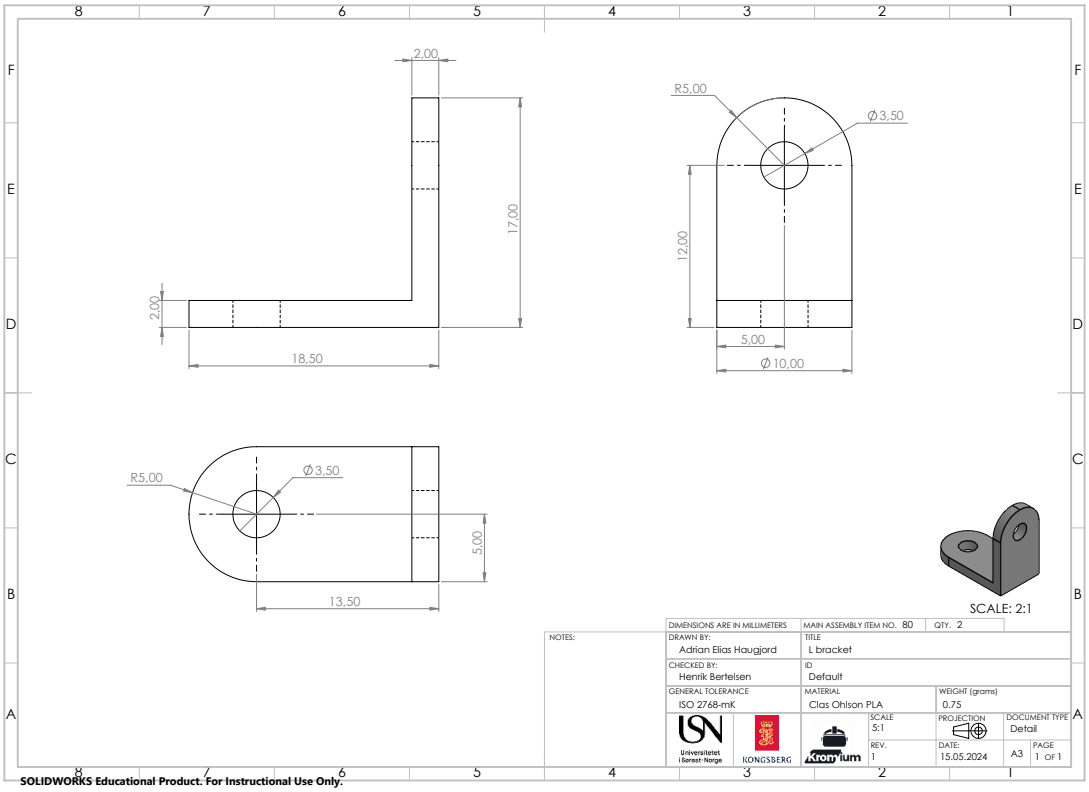


Figure T.40: L bracket

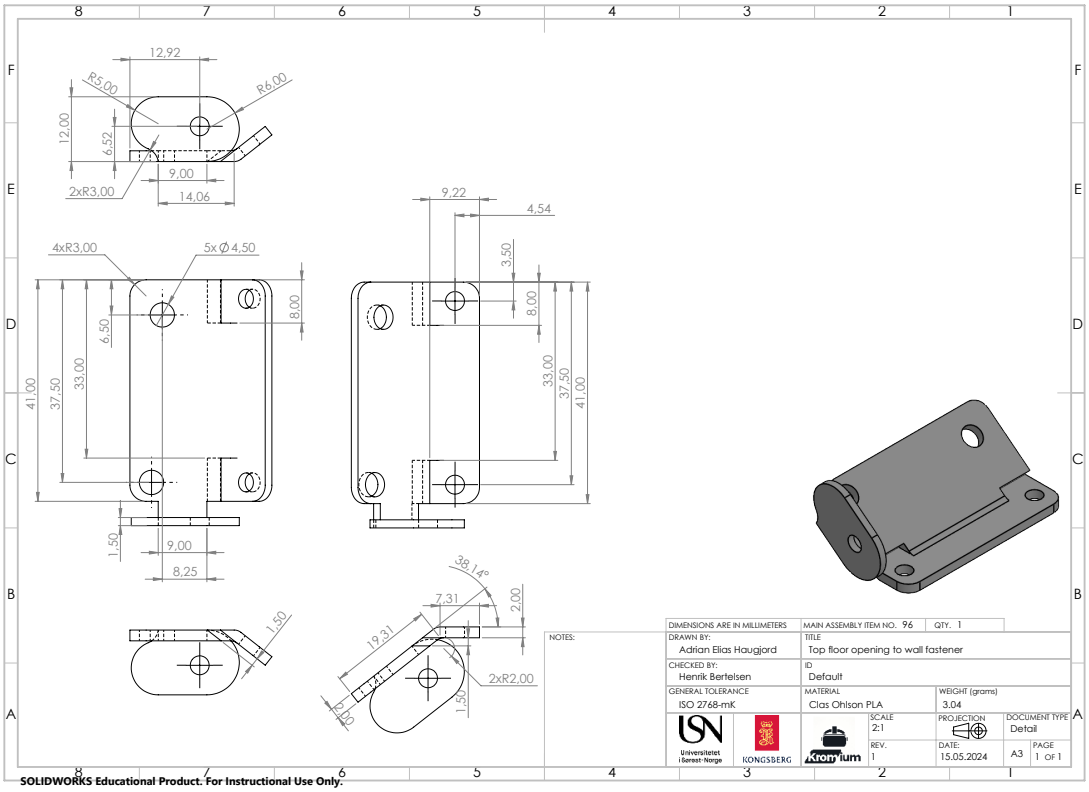


Figure T.41: Top floor opening to wall bracket

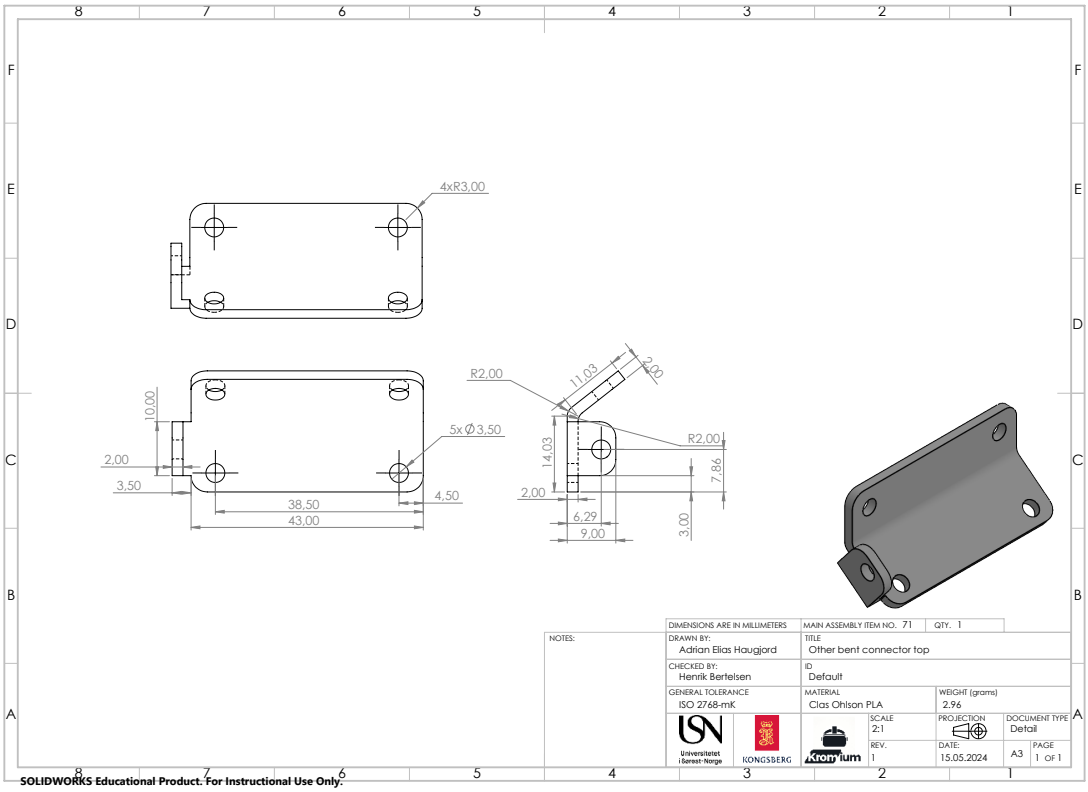


Figure T.42: Top floor other bracket

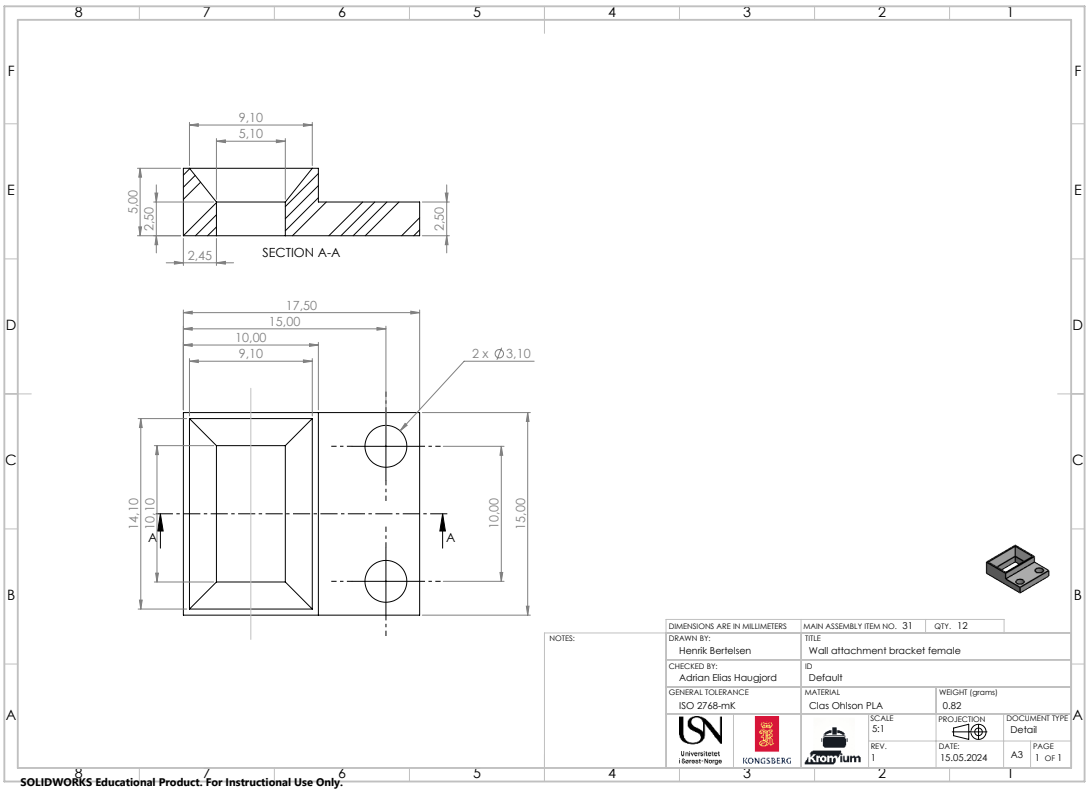


Figure T.43: Wall attachment bracket female

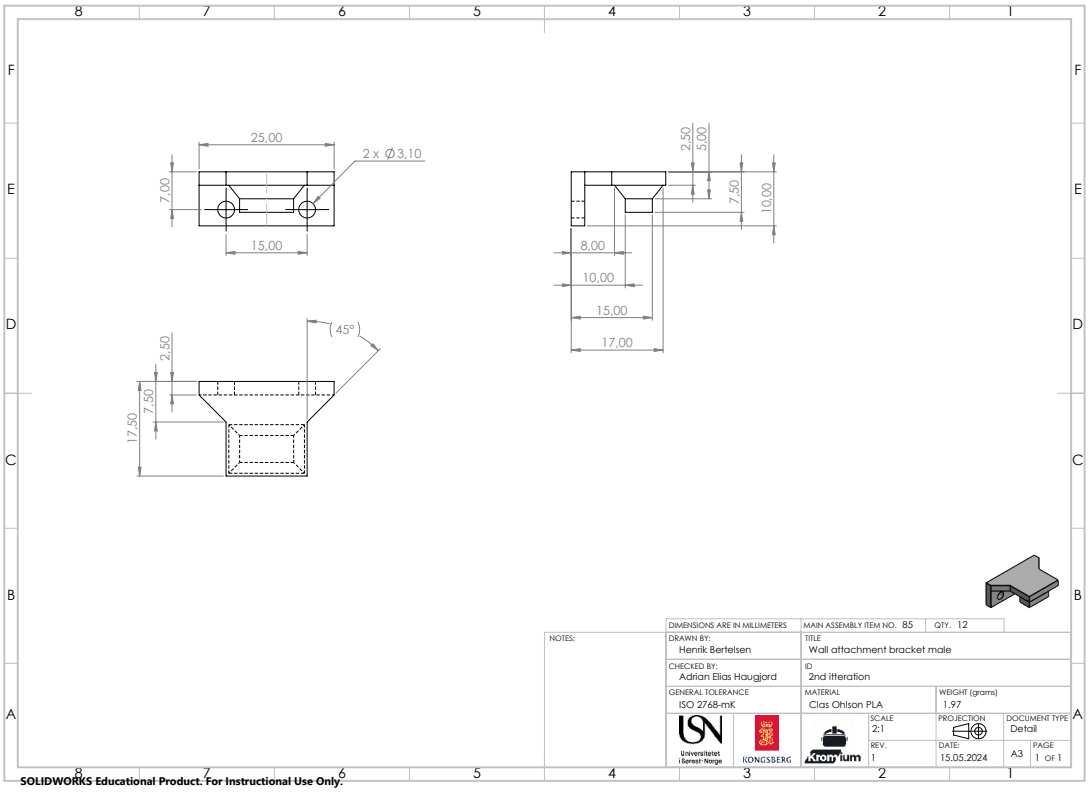


Figure T.44: Wall attachment bracket male

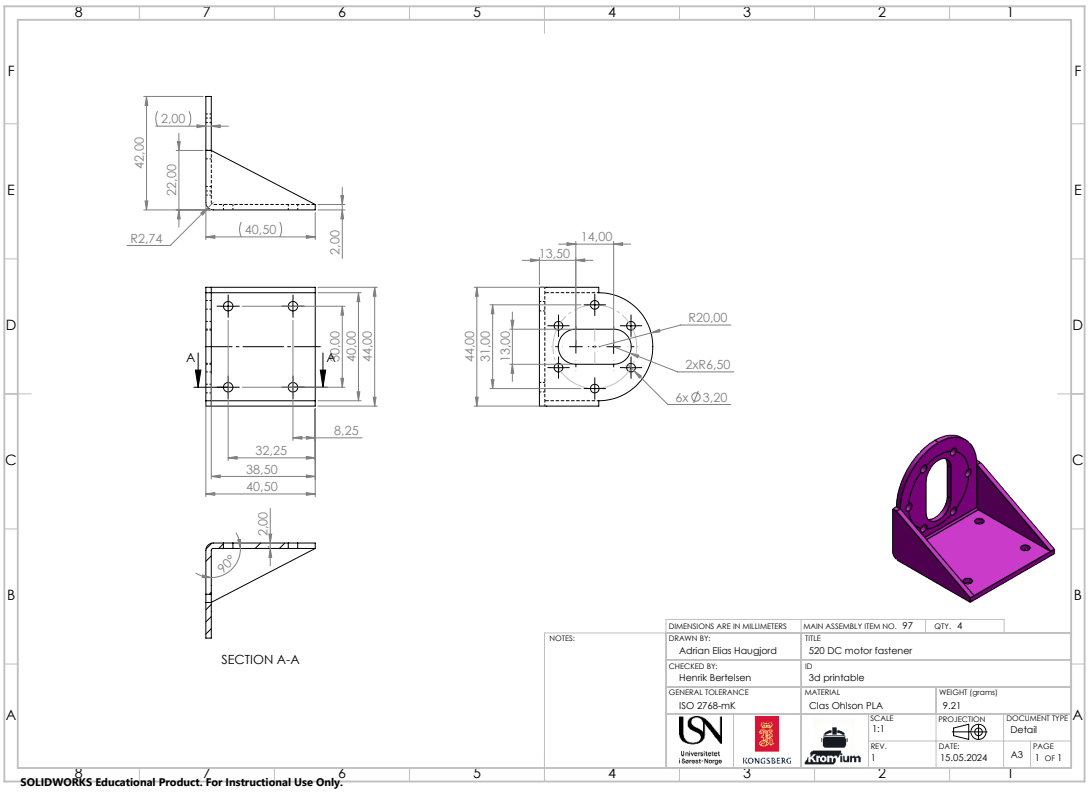


Figure T.45: 520 DC motor bracket

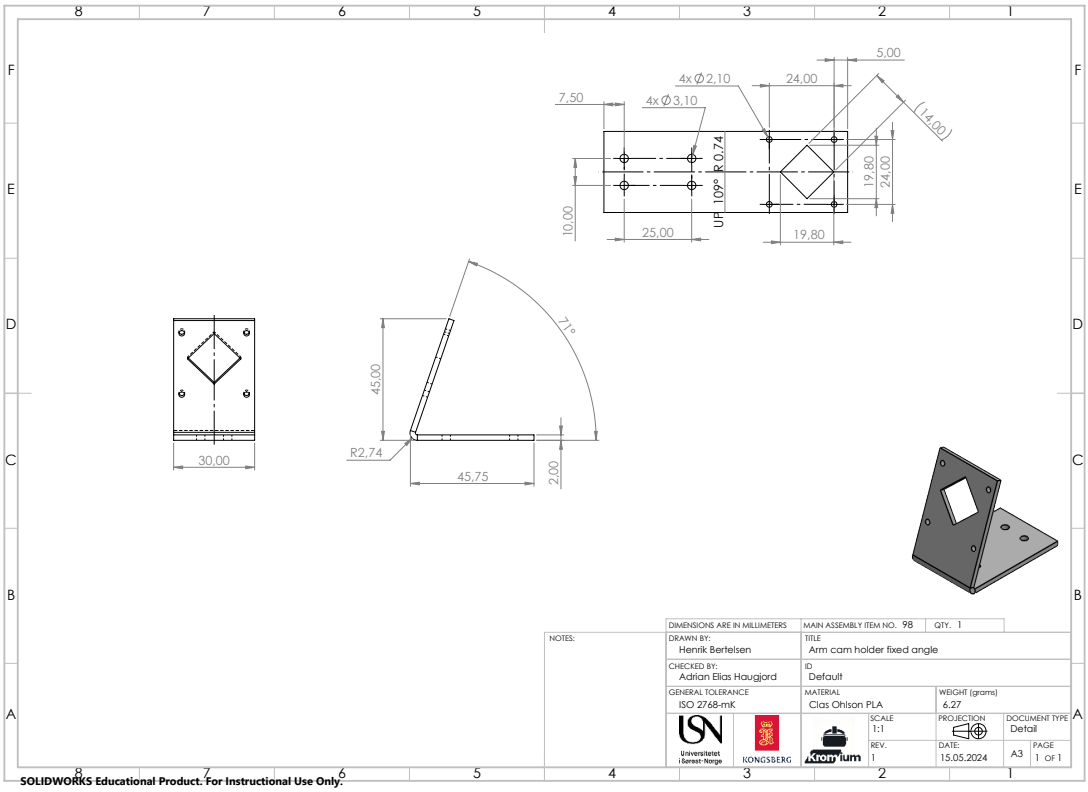


Figure T.46: Arm cam holder fixed angle

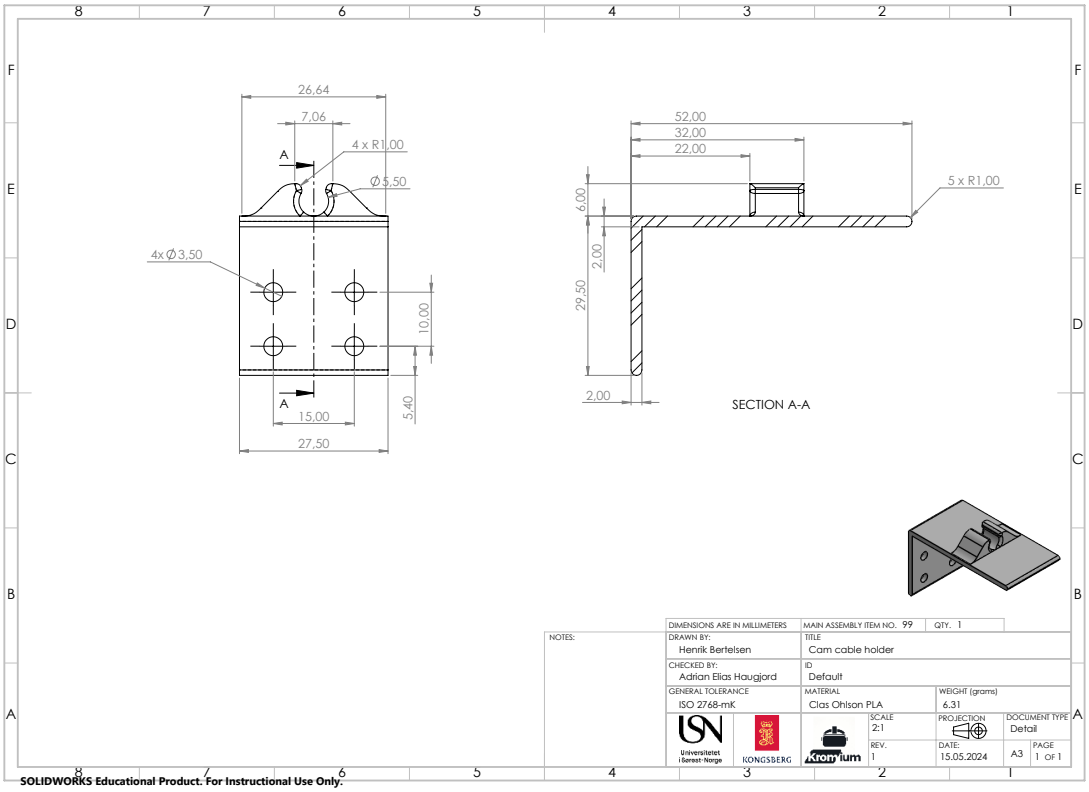


Figure T.47: Cam cable holder

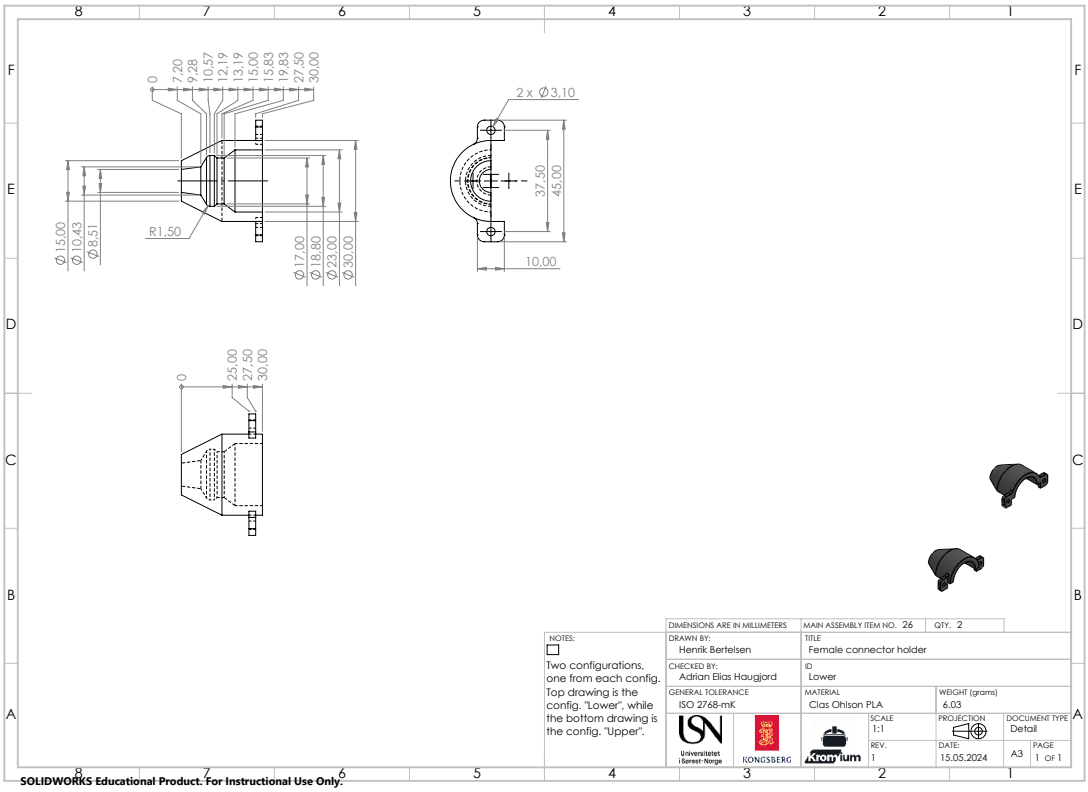


Figure T.48: Female magnet connector bracket

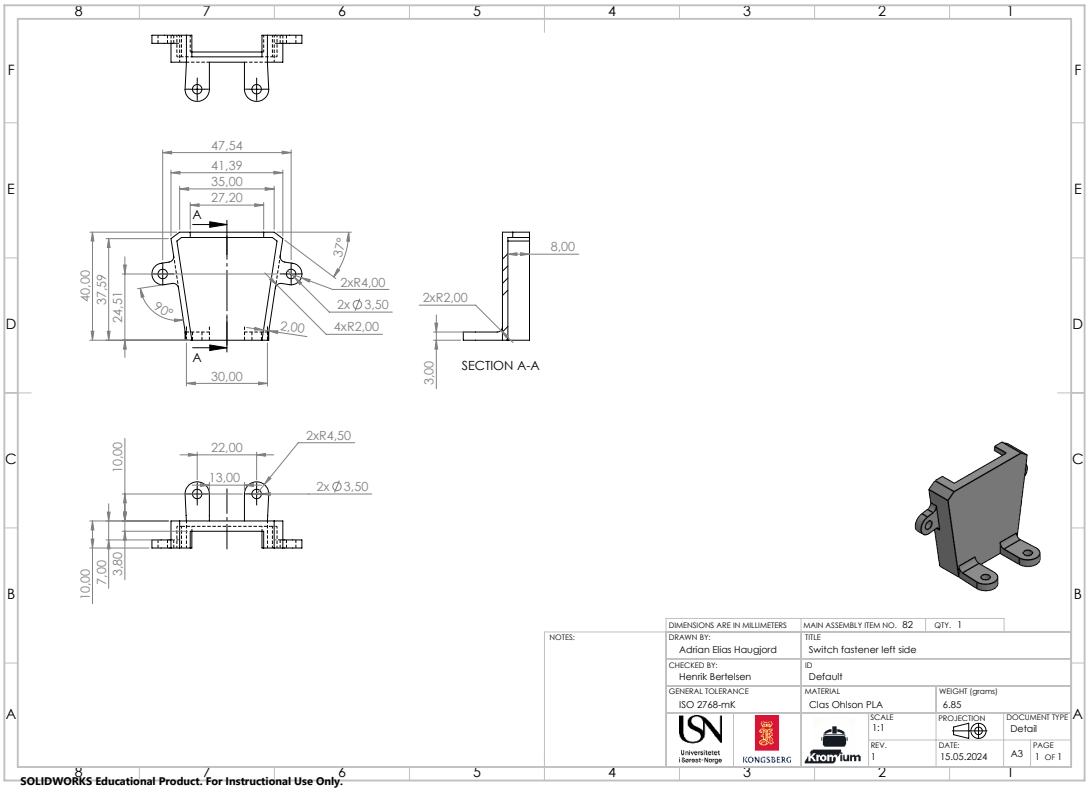


Figure T.49: Power switch bracket left side

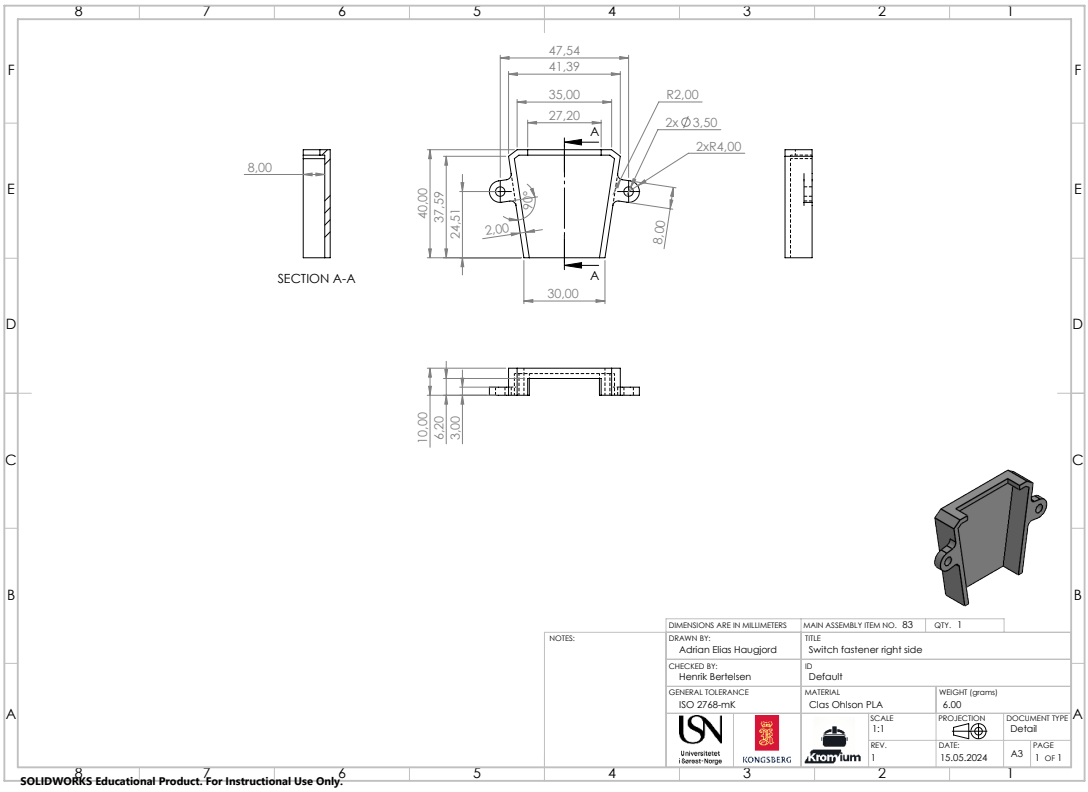


Figure T.50: Power switch bracket right side

T.3.7 Battery drawer

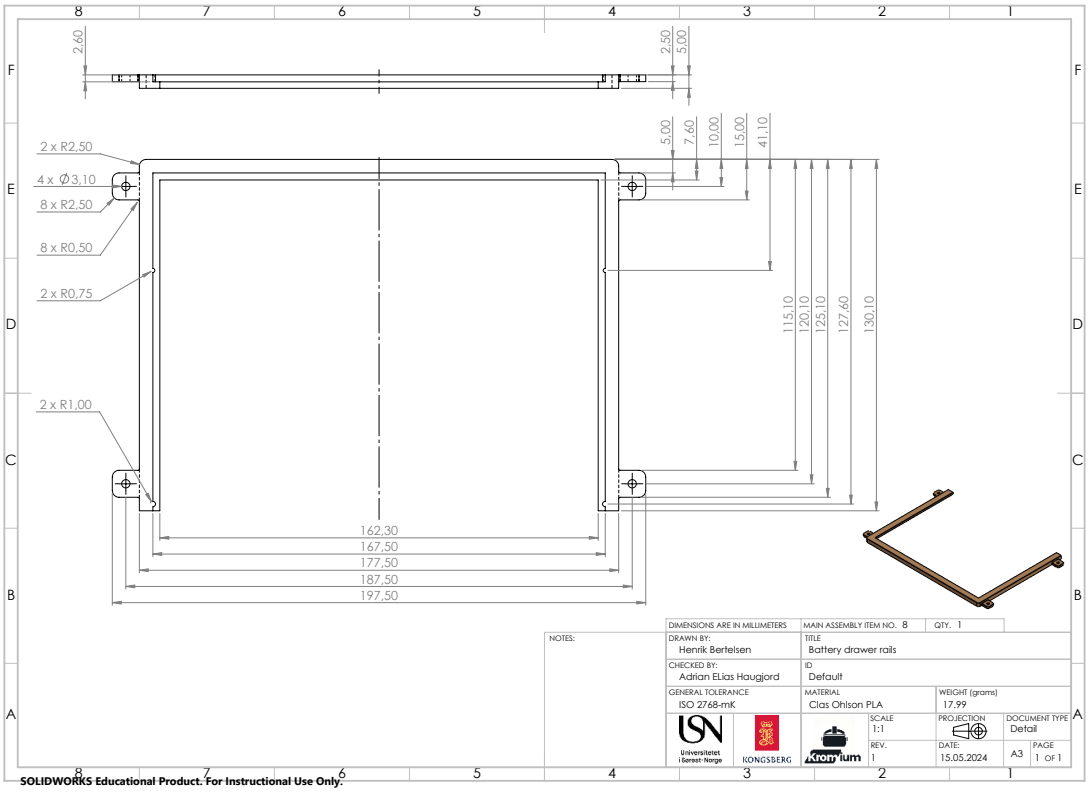


Figure T.51: Battery drawer rails

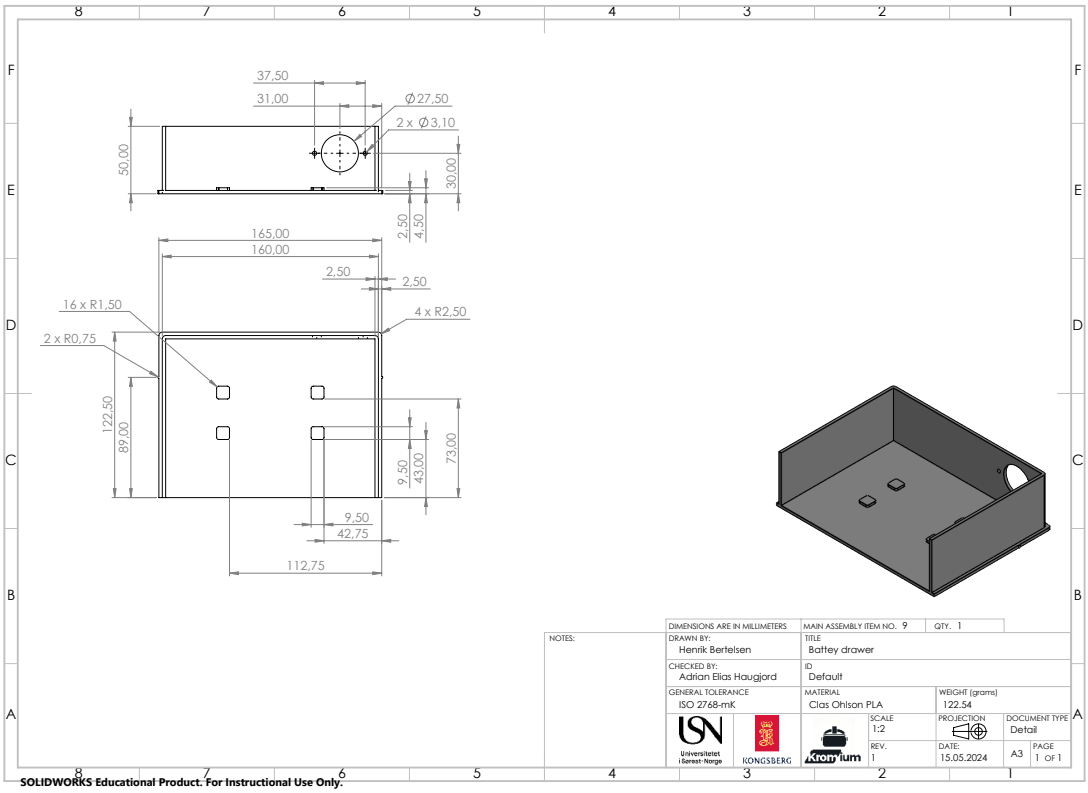


Figure T.52: Battery drawer

T.3.8 Yahboom battery case

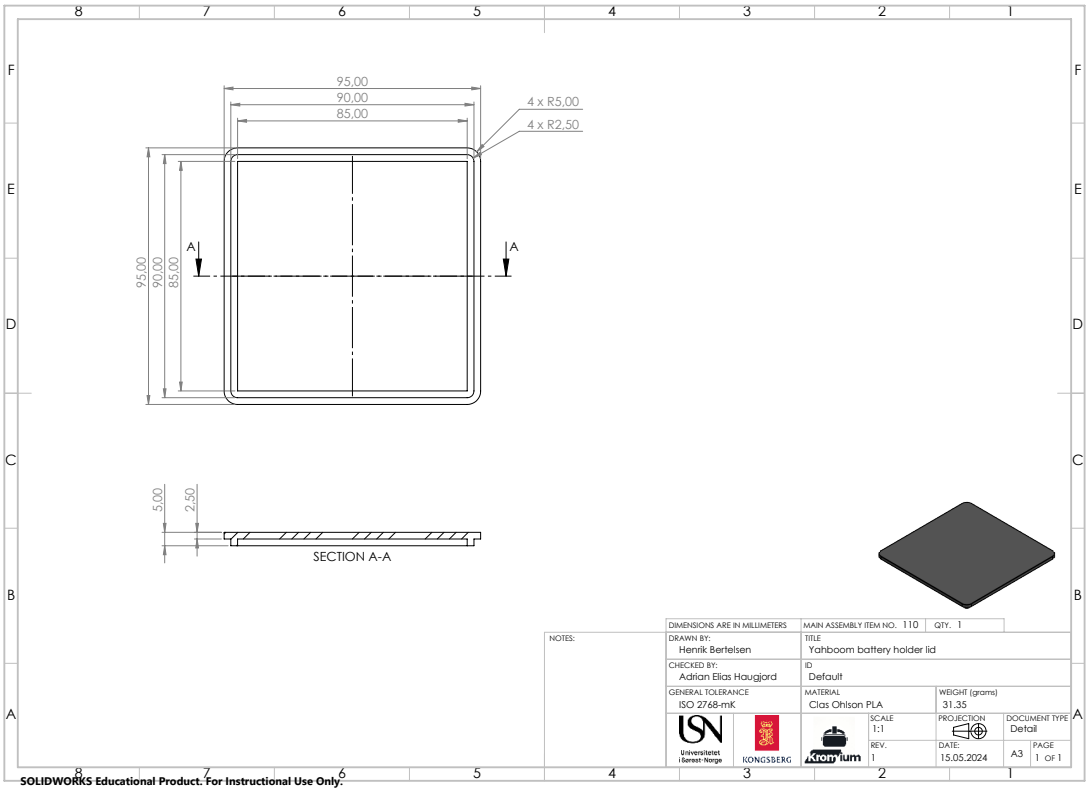


Figure T.53: Yahboom battery case lid

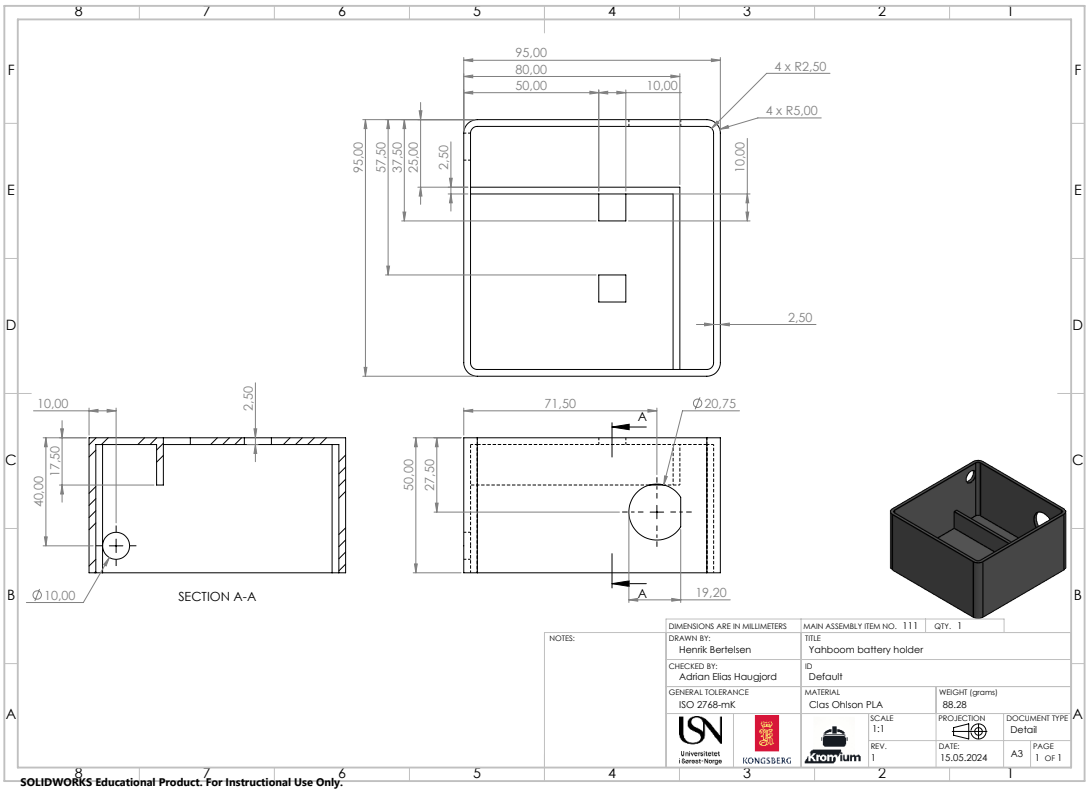


Figure T.54: Yahboom battery case

T.3.9 KROMIUM battery

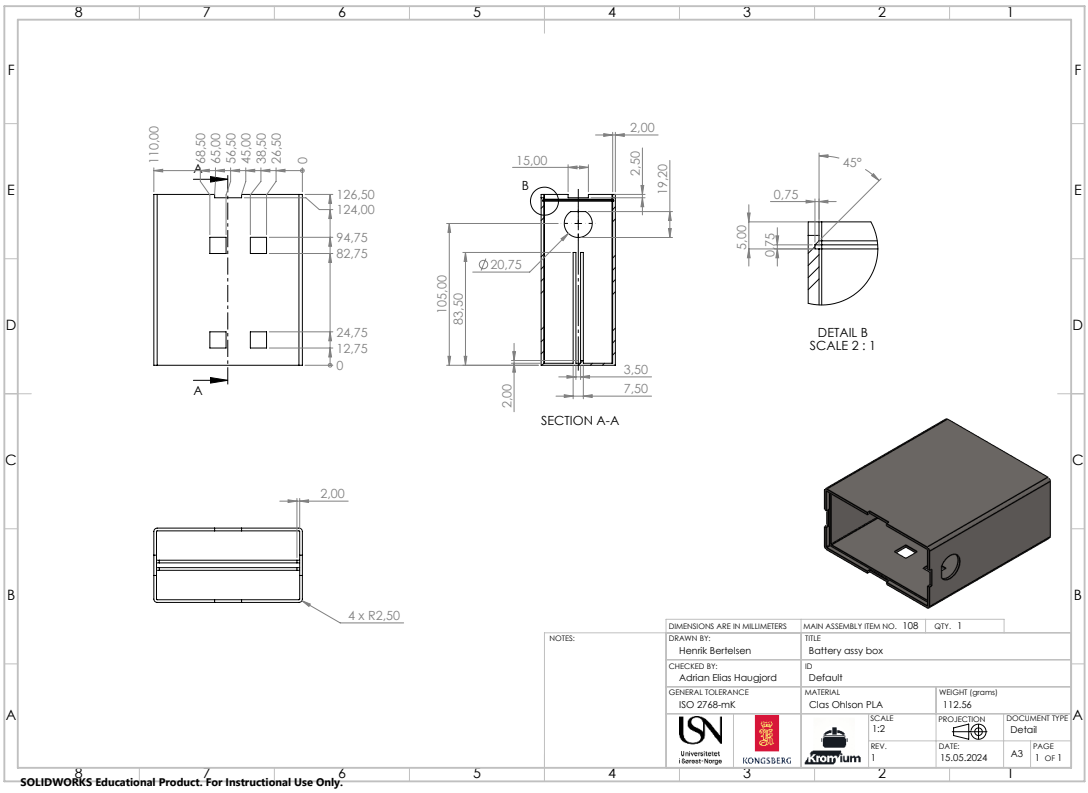


Figure T.55: KROMIUM battery case

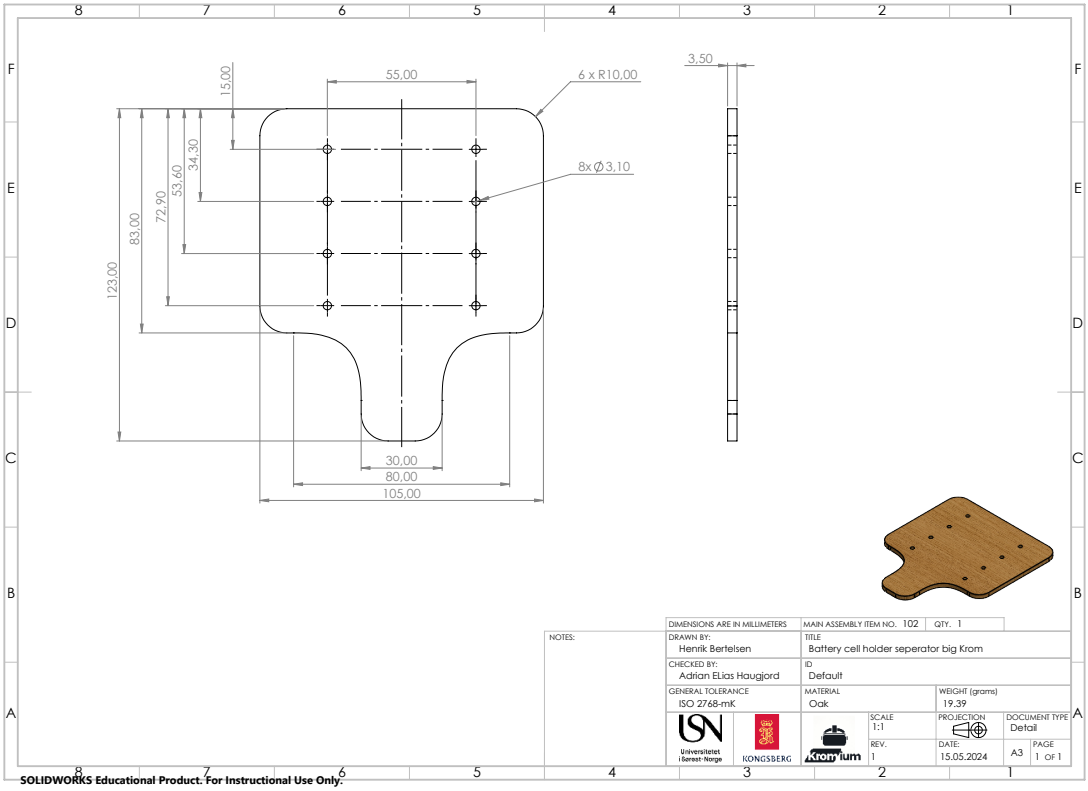


Figure T.56: KROMIUM battery cell separator holder

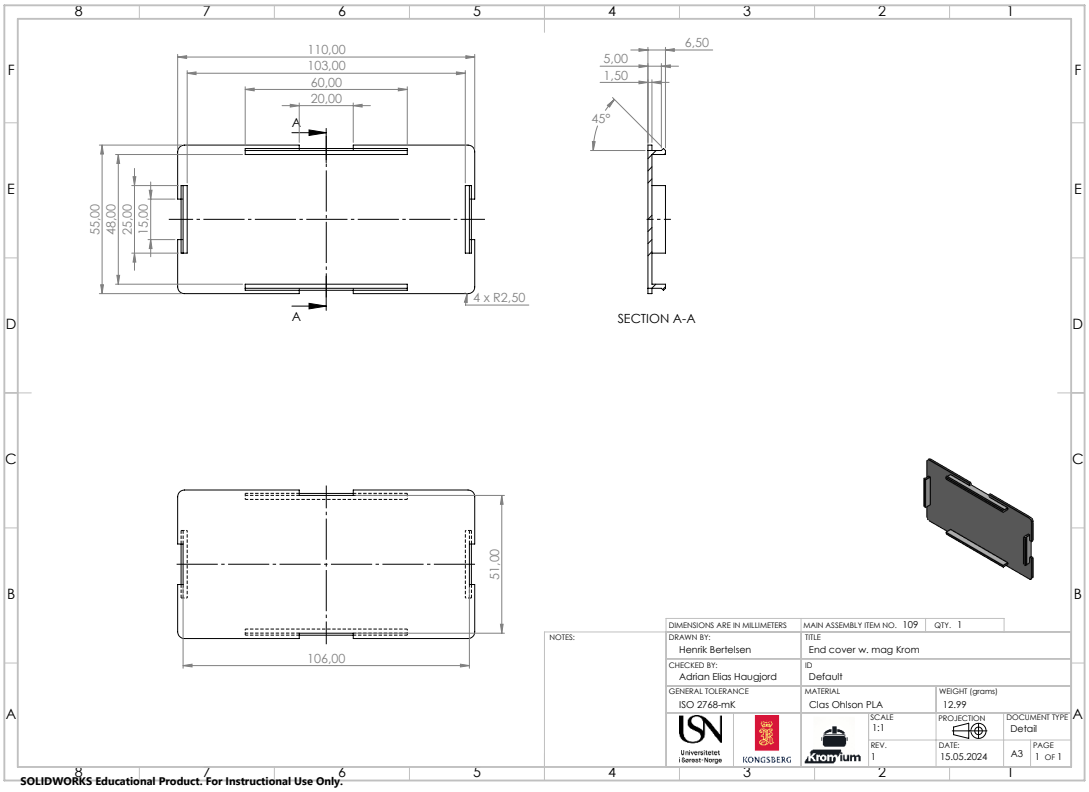


Figure T.57: KROMIUM battery case lid

T.4 Subassembly drawings

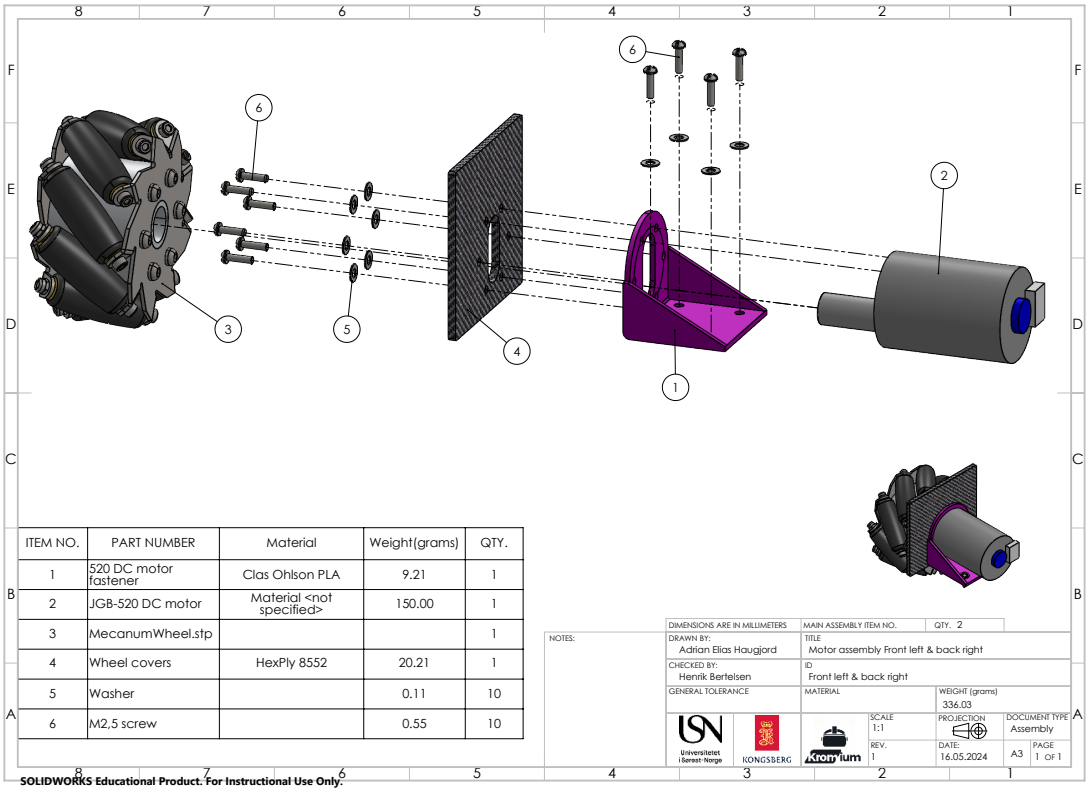
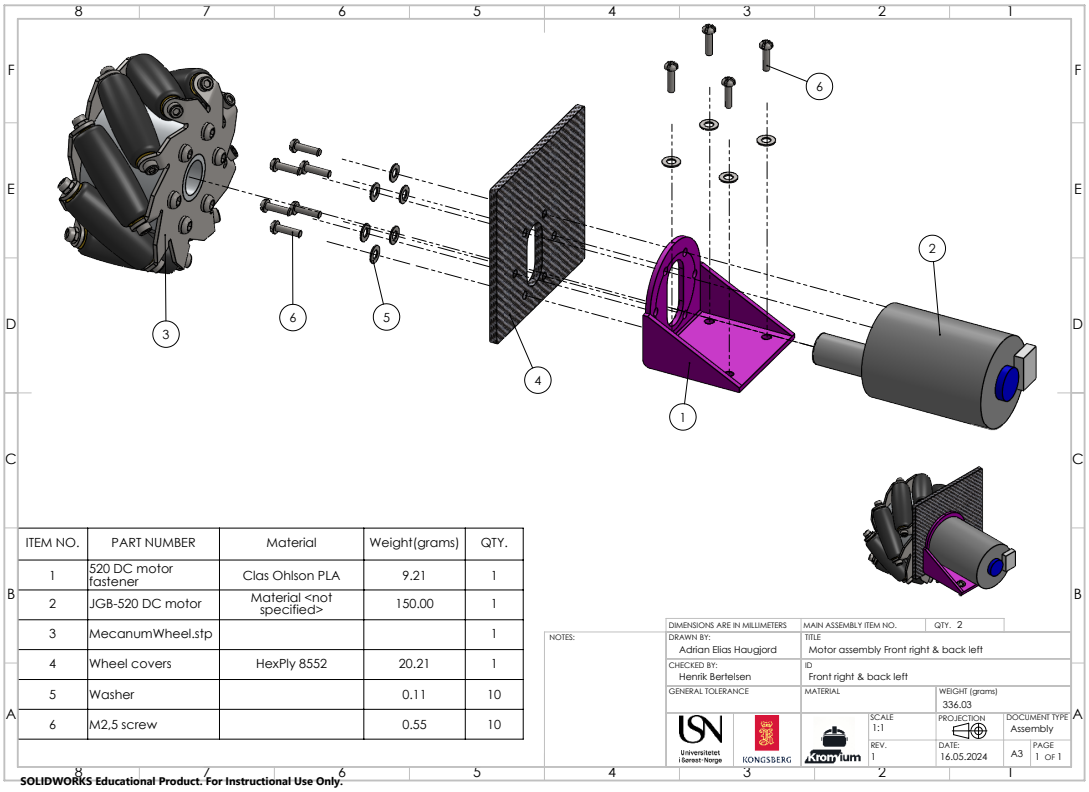


Figure T.58: Motor assembly Front left & back right

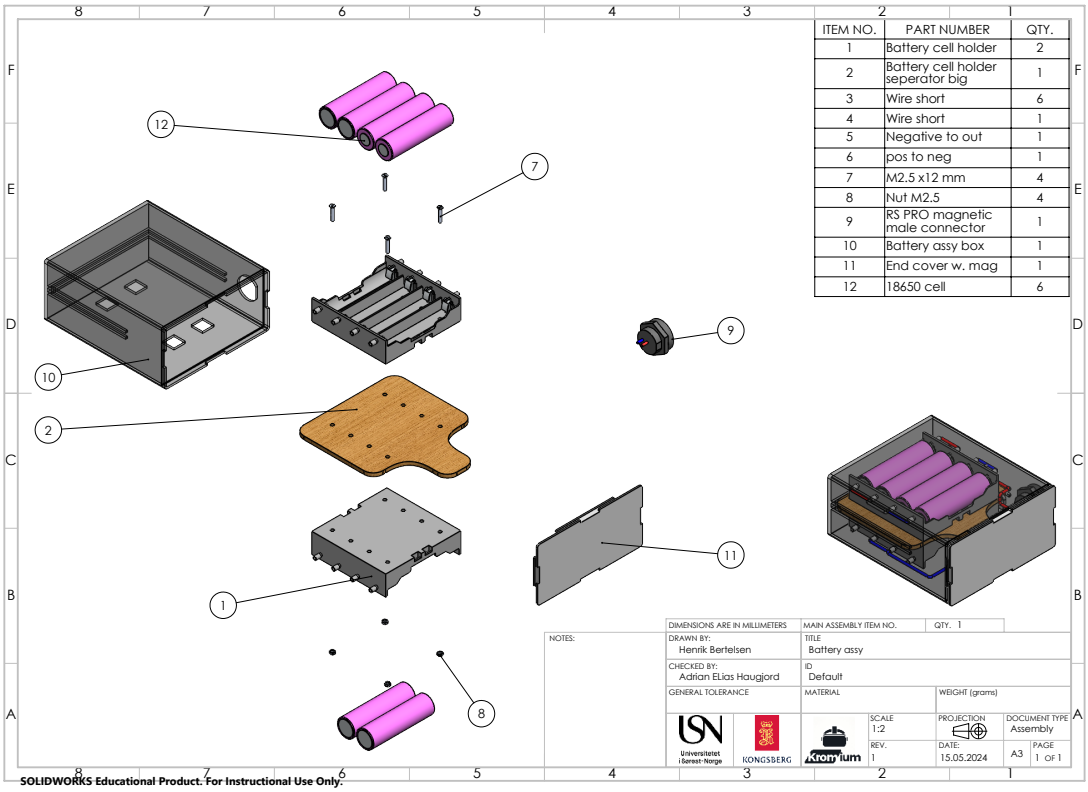


ITEM NO.	PART NUMBER	Material	Weight(grams)	QTY.
1	520 DC motor fastener	Clas Ohlson PLA	9.21	1
2	JGB-520 DC motor	Material <not specified>	150.00	1
3	MecanumWheel.stp			1
4	Wheel covers	HexPly 8552	20.21	1
5	Washer		0.11	10
6	M2,5 screw		0.55	10

NOTES:

DIMENSIONS ARE IN MILLIMETERS | MAIN ASSEMBLY ITEM NO. | QTY. 2
DRAWN BY: Adrian Elias Haugjord | TITLE: Motor assembly Front right & back left
CHECKED BY: Henrik Bertelsen | ID: Front right & back left
GENERAL TOLERANCE | MATERIAL | WEIGHT (grams): 336.03
SCALE: 1:1 | PROJECTION | DOCUMENT TYPE: Assembly
REV: 1 | DATE: 16.05.2024 | A3 | PAGE: 1 OF 1

Figure T.59: Motor assembly Front right & back left



ITEM NO.	PART NUMBER	QTY.
1	Battery cell holder	2
2	Battery cell holder seperator big	1
3	Wire short	6
4	Wire short	1
5	Negative to out	1
6	pos to neg	1
7	M2.5 x12 mm	4
8	Nut M2.5	4
9	RS PRO magnetic male connector	1
10	Battery assy box	1
11	End cover w. mag	1
12	18650 cell	6

NOTES:

DIMENSIONS ARE IN MILLIMETERS | MAIN ASSEMBLY ITEM NO. | QTY. 1
DRAWN BY: Henrik Bertelsen | TITLE: Battery assy
CHECKED BY: Adrian Elias Haugjord | ID: Default
GENERAL TOLERANCE | MATERIAL | WEIGHT (grams)
SCALE: 1:2 | PROJECTION | DOCUMENT TYPE: Assembly
REV: 1 | DATE: 15.05.2024 | A3 | PAGE: 1 OF 1

Figure T.60: KROMIUM battery assembly

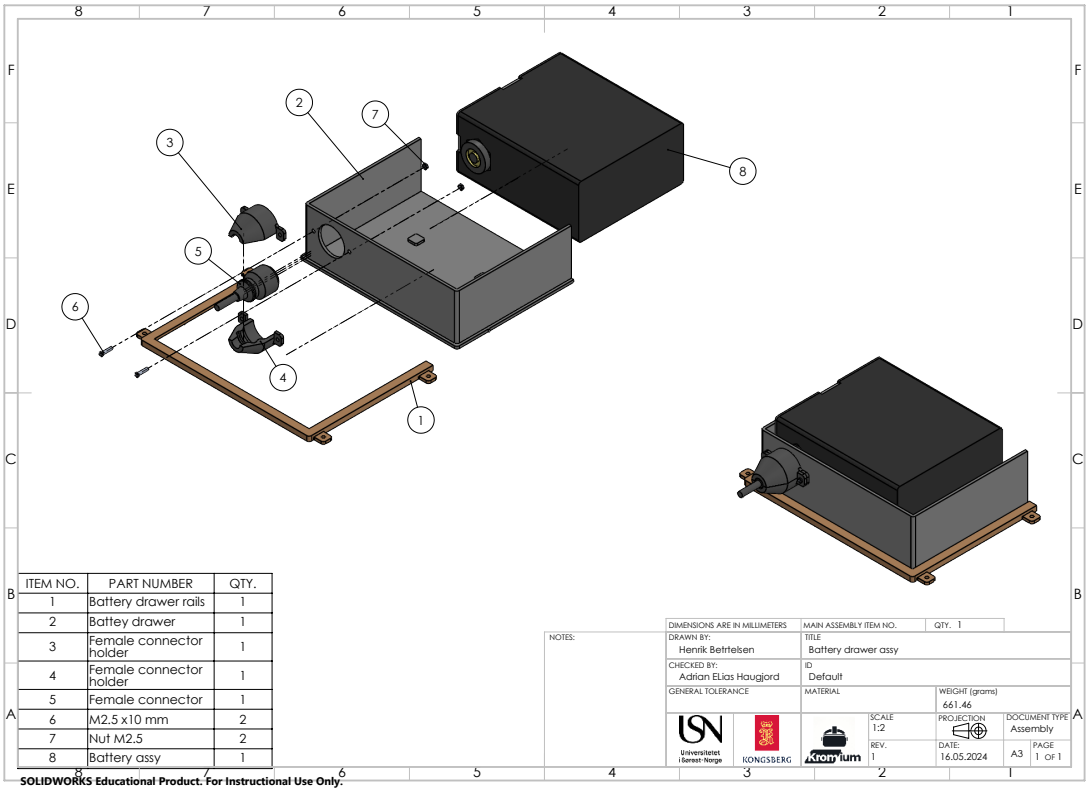


Figure T.61: KROMIUM battery drawer assembly

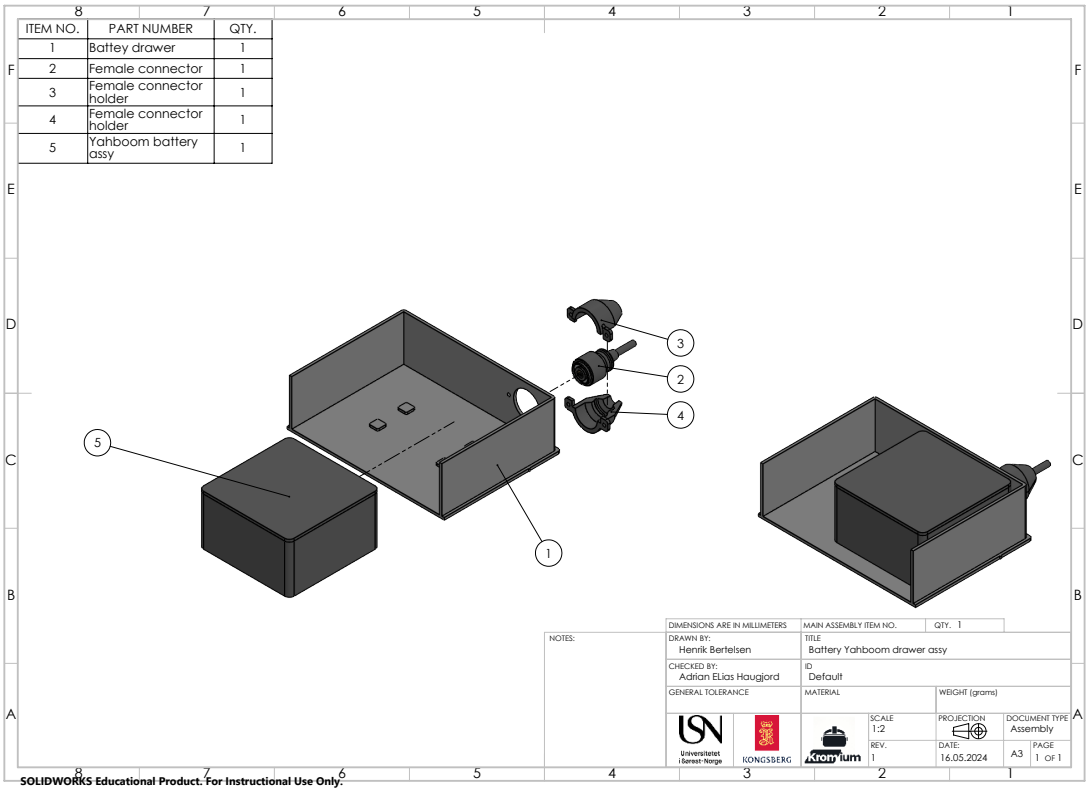


Figure T.62: Yahboom battery drawer assembly

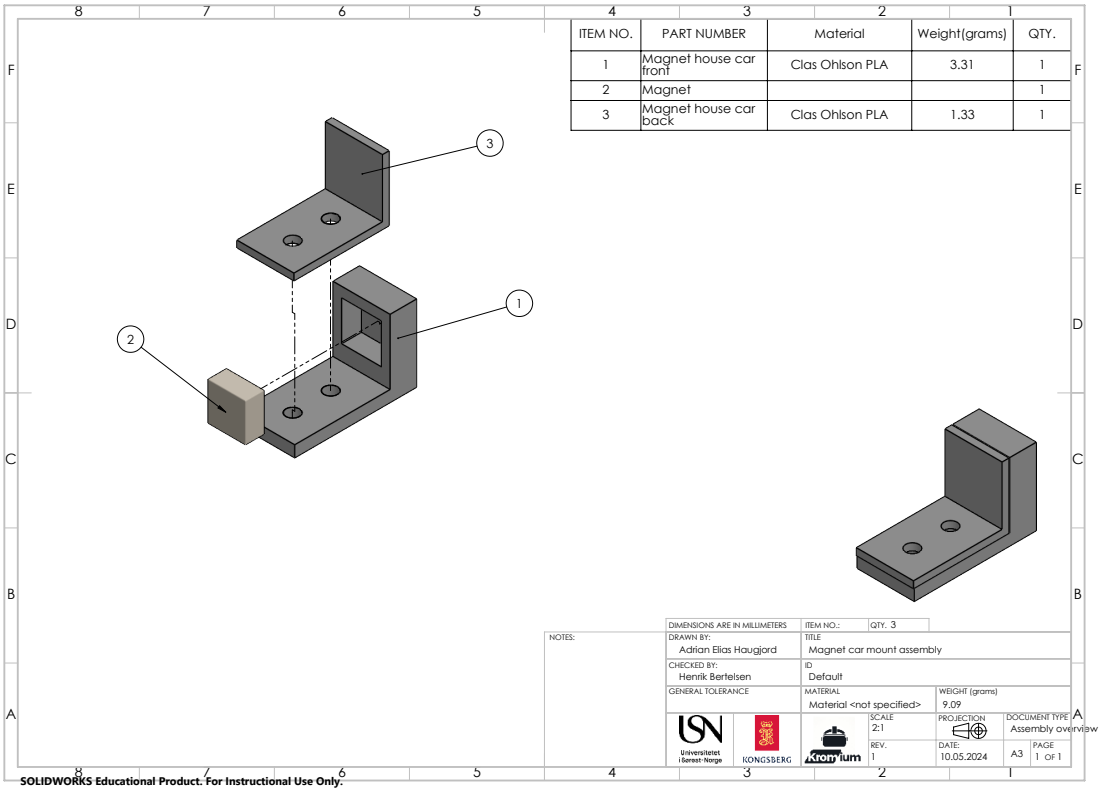


Figure T.63: Magnet car mount assembly

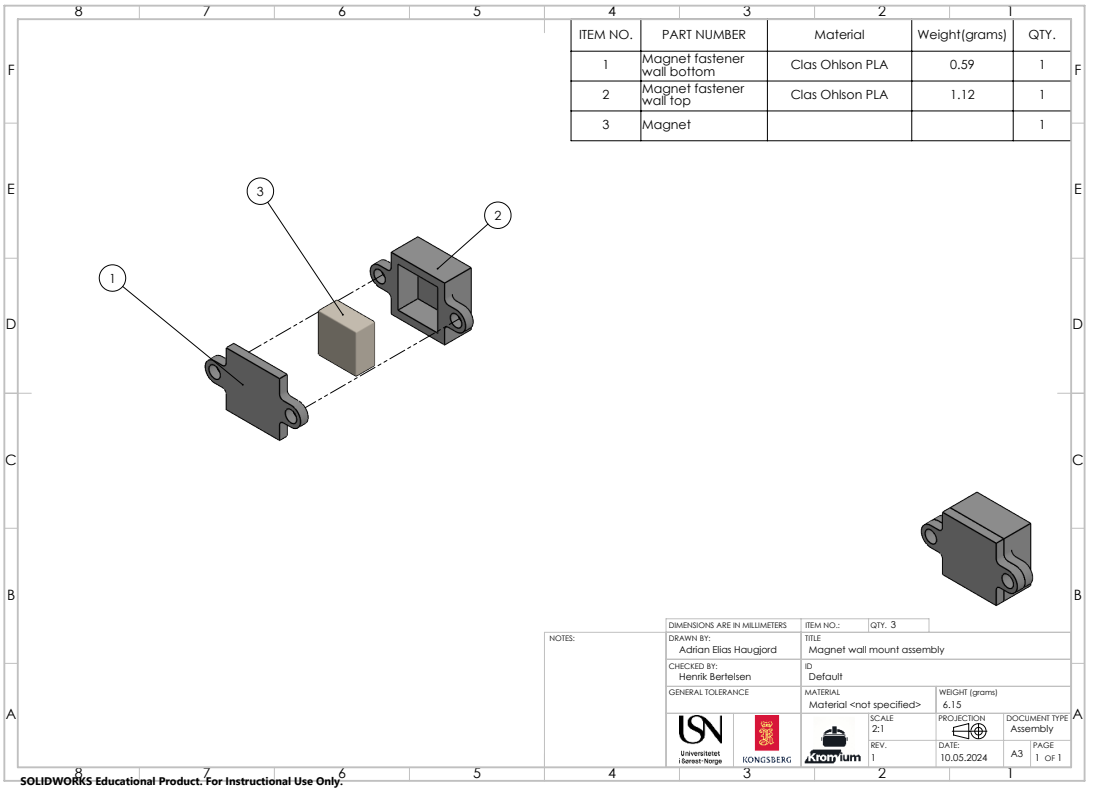


Figure T.64: Magnet wall mount assembly

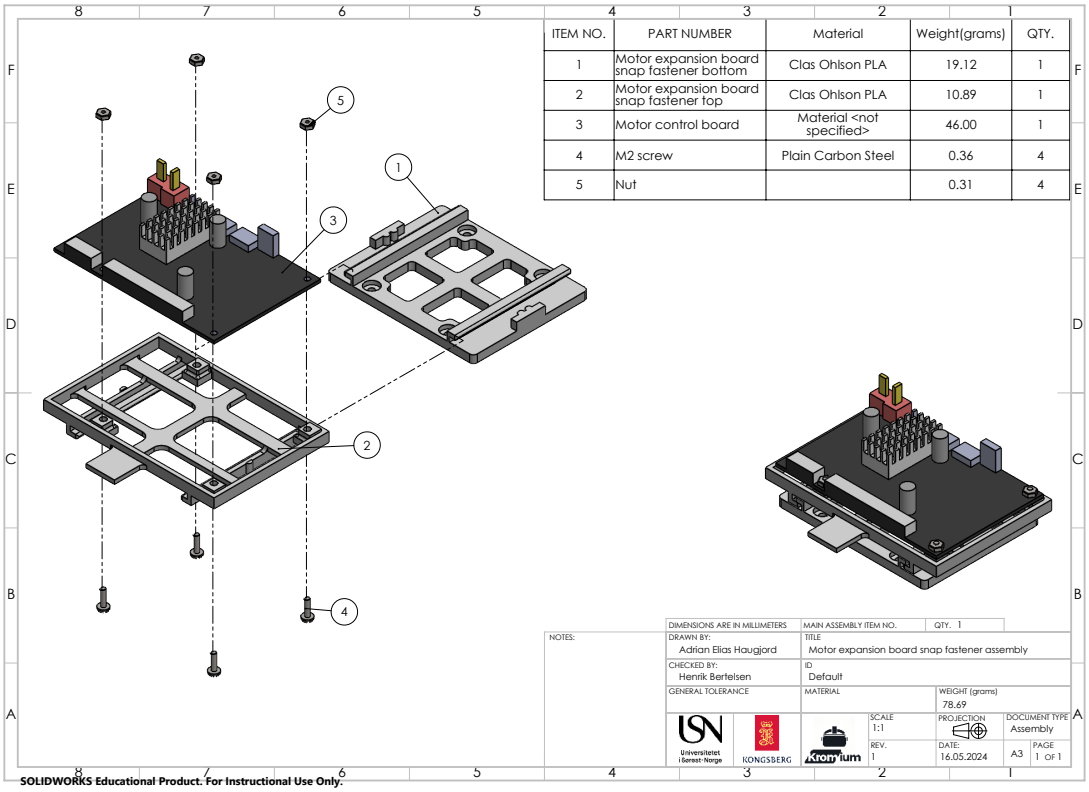


Figure T.65: Motor expansion board snap bracket assembly

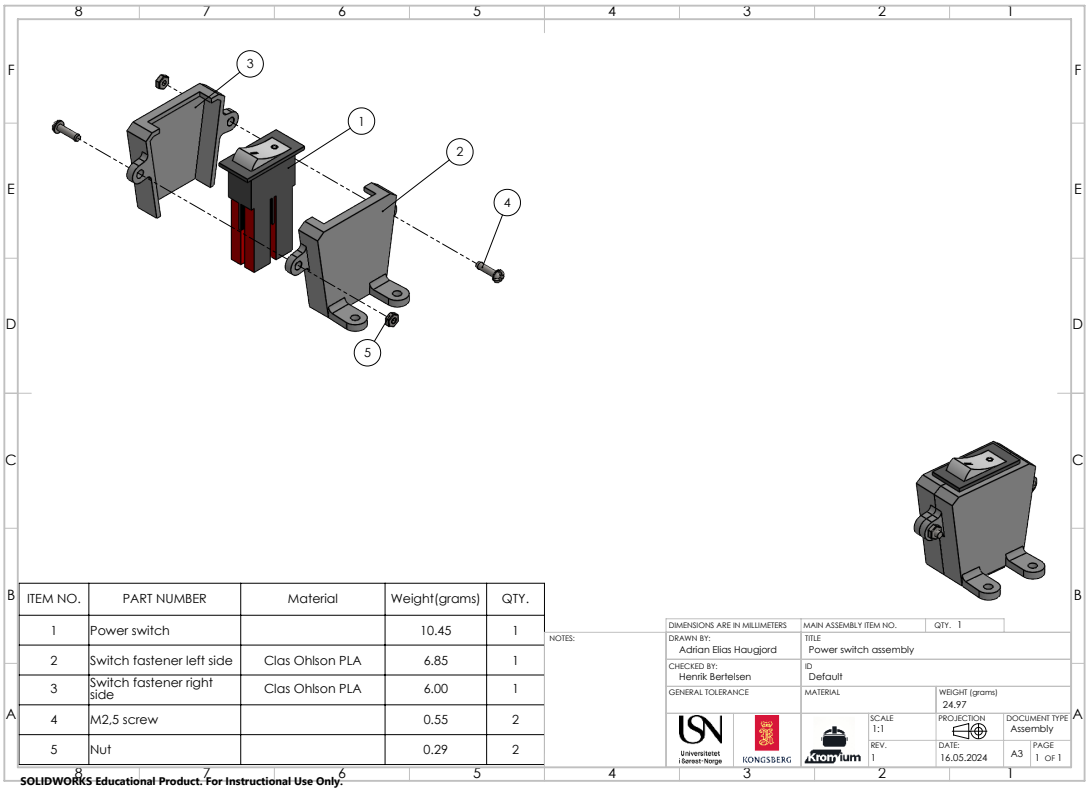


Figure T.66: Power switch bracket assembly

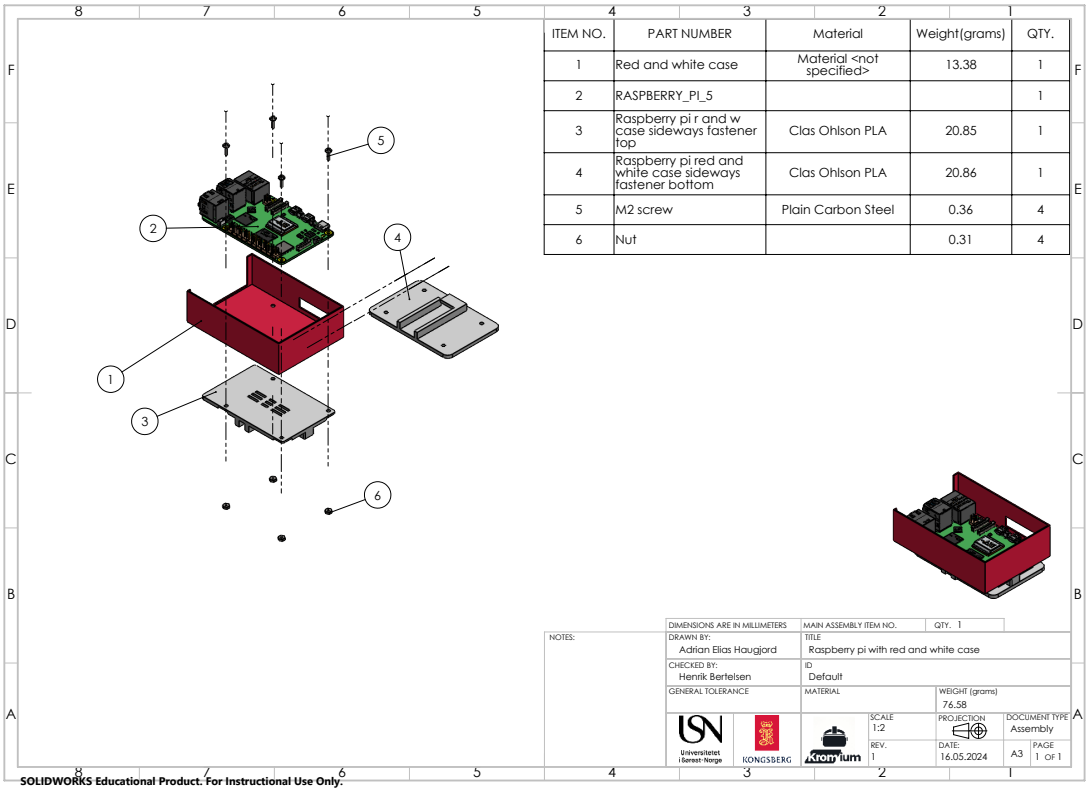


Figure T.67: Raspberry Pi bracket assembly

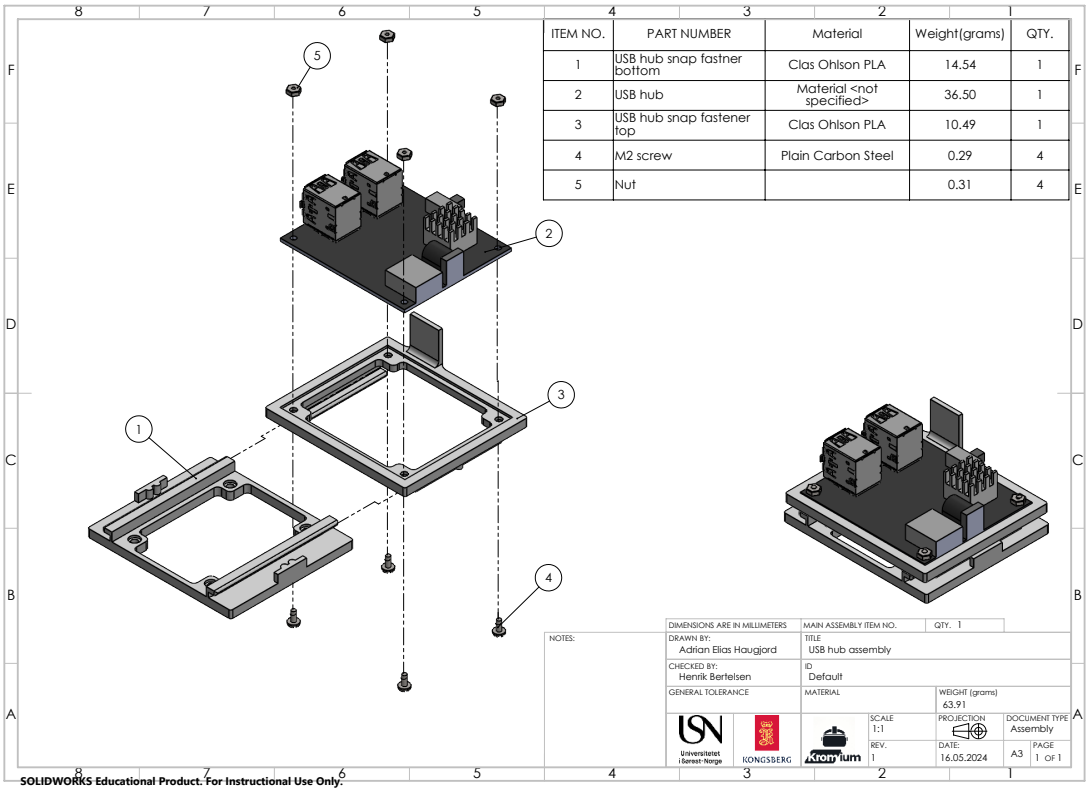


Figure T.68: USB hub snap bracket assembly

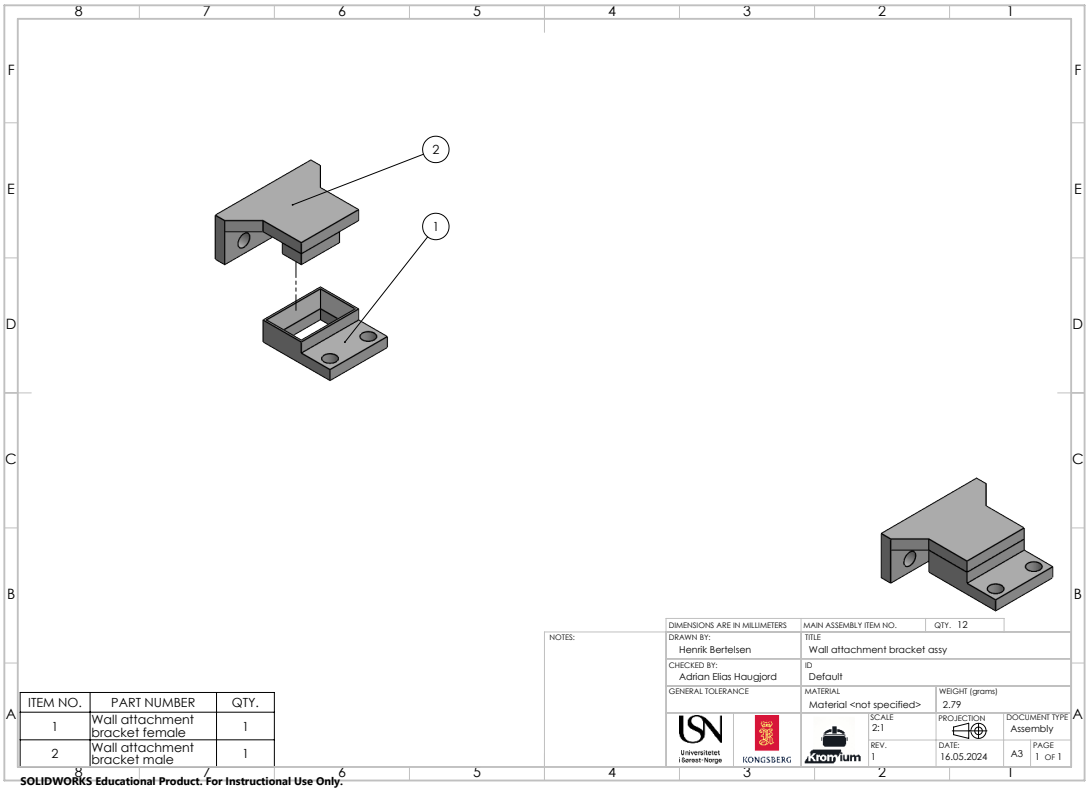


Figure T.69: Wall attachment bracket assembly

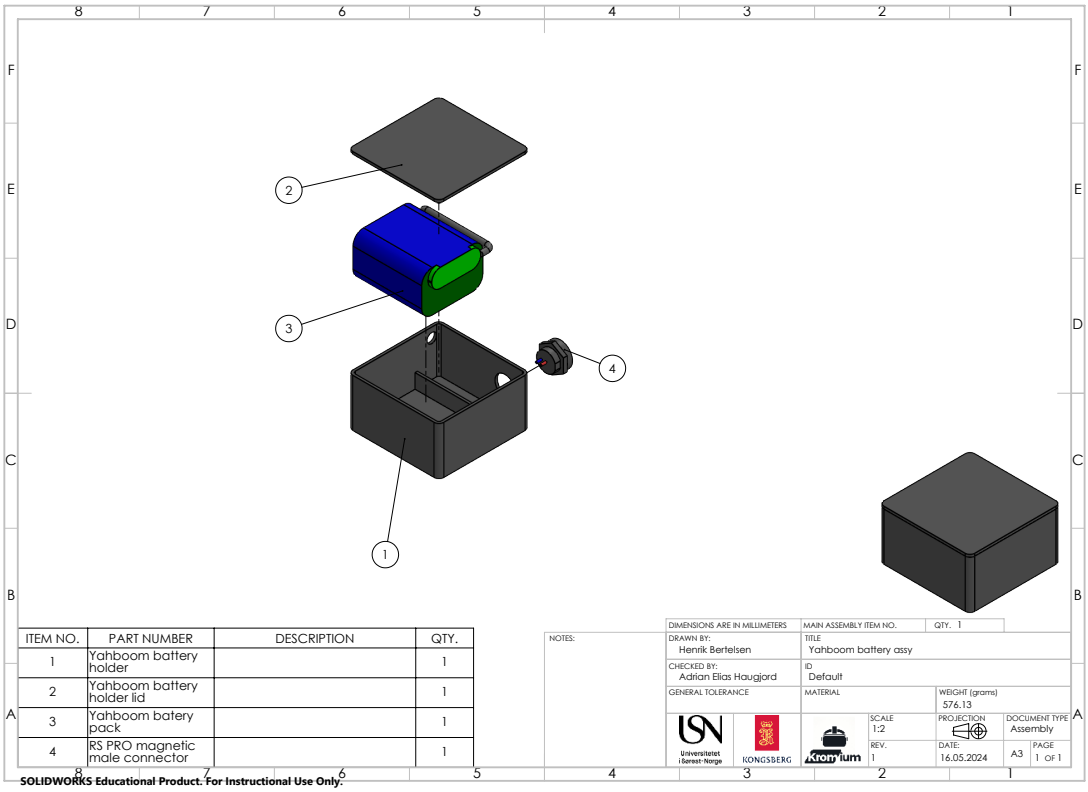


Figure T.70: Yahboom battery case assembly

T.5 Floor assemblies

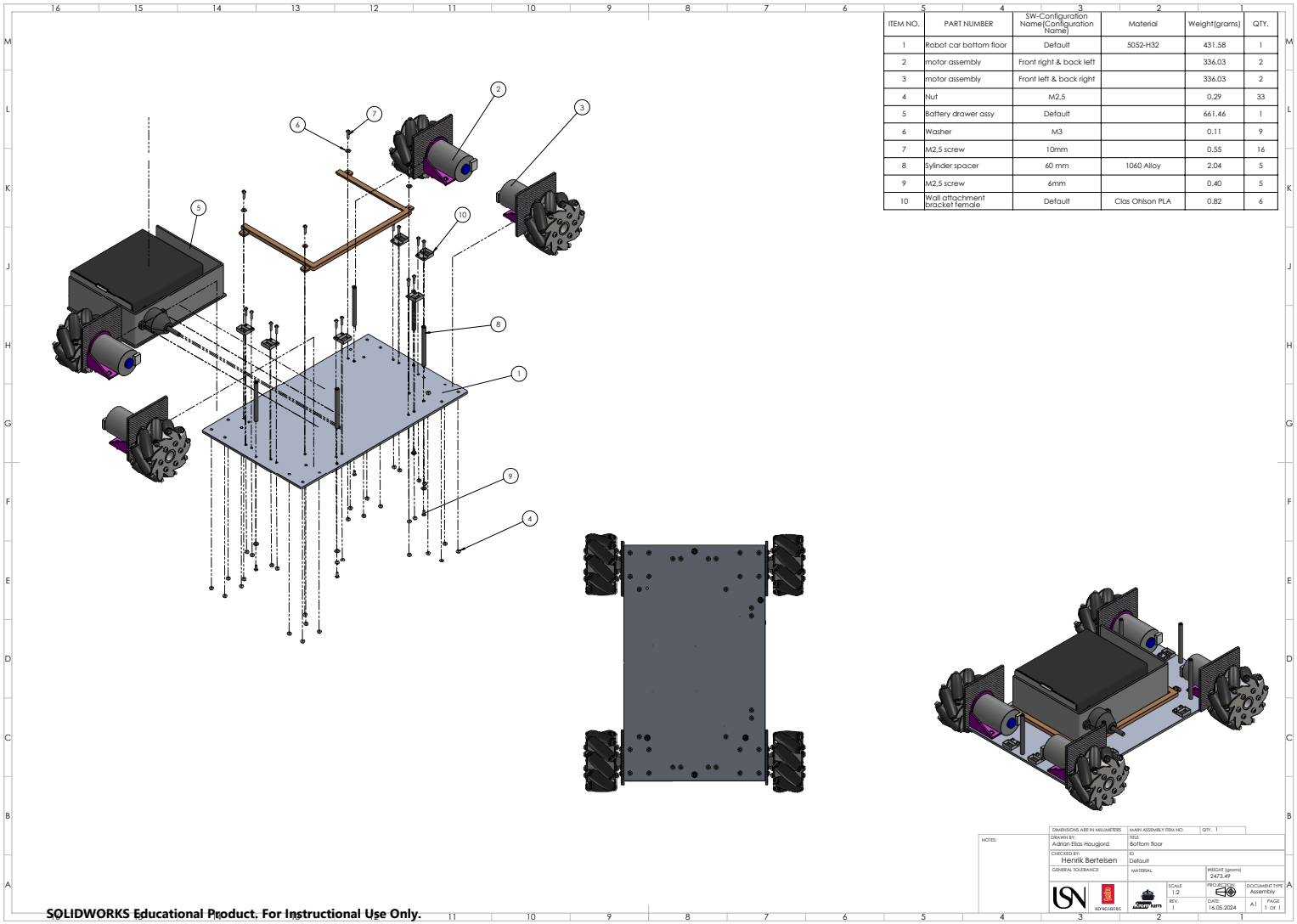


Figure T.71: Bottom floor assembly

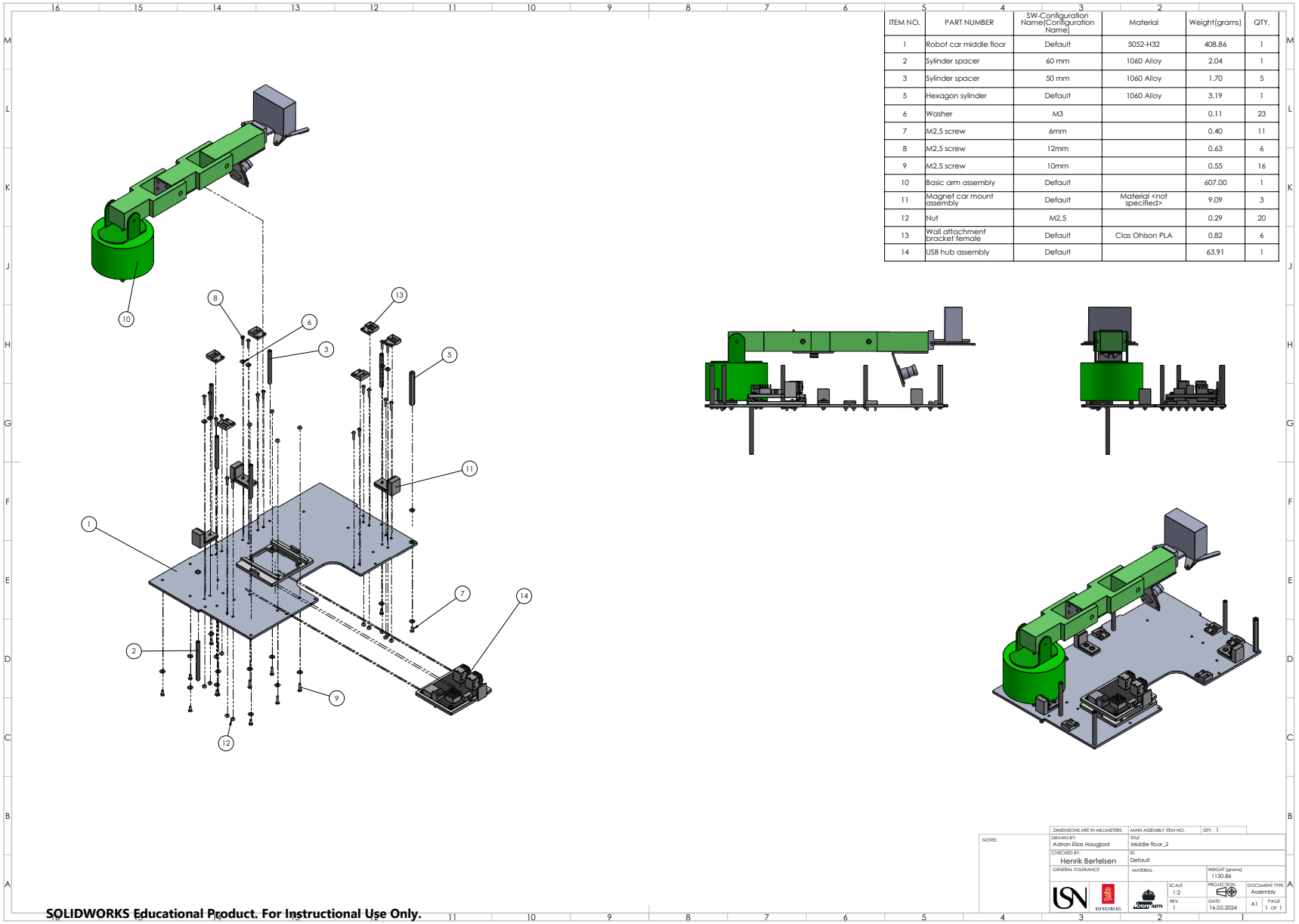


Figure T.72: Middle floor assembly

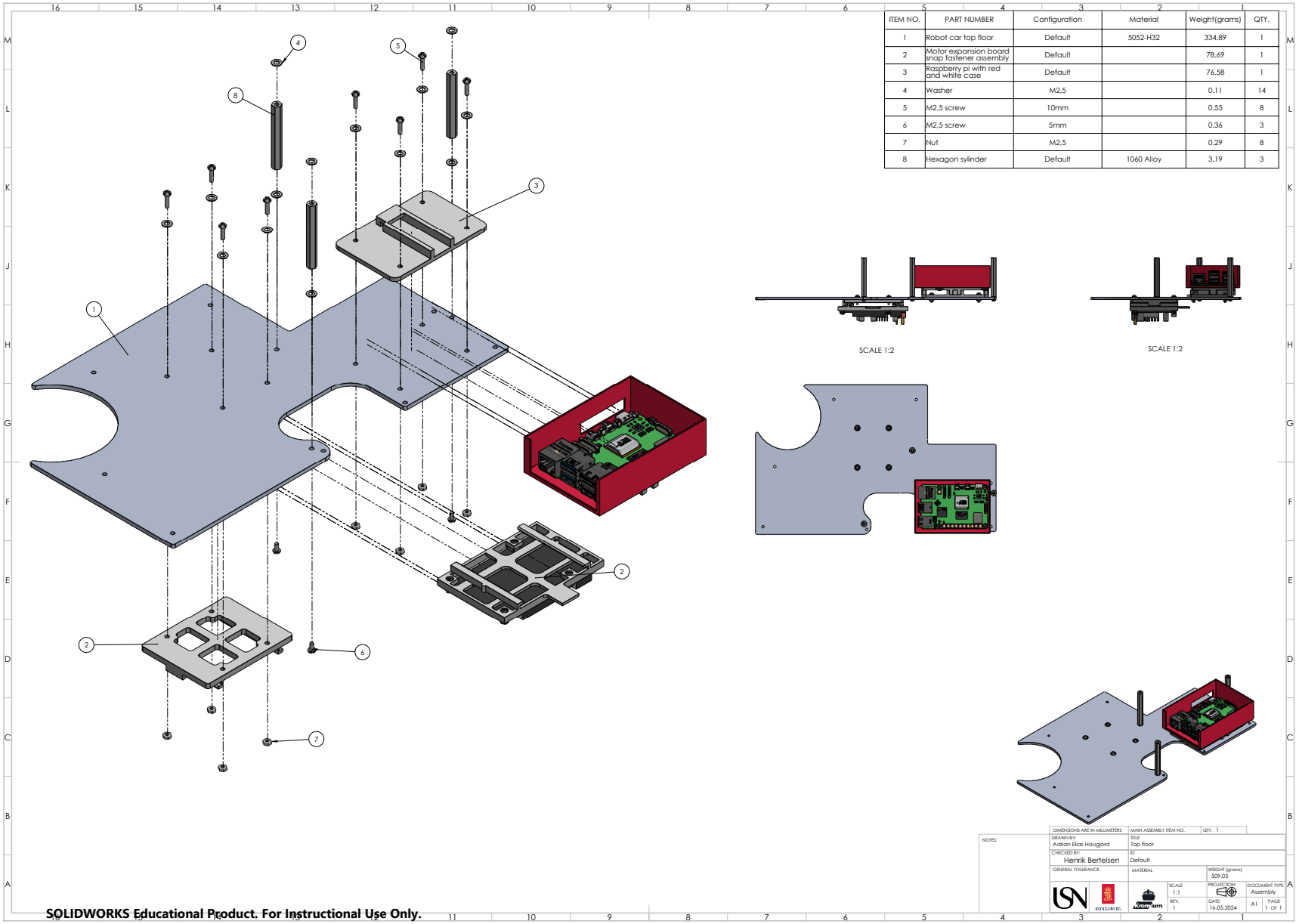
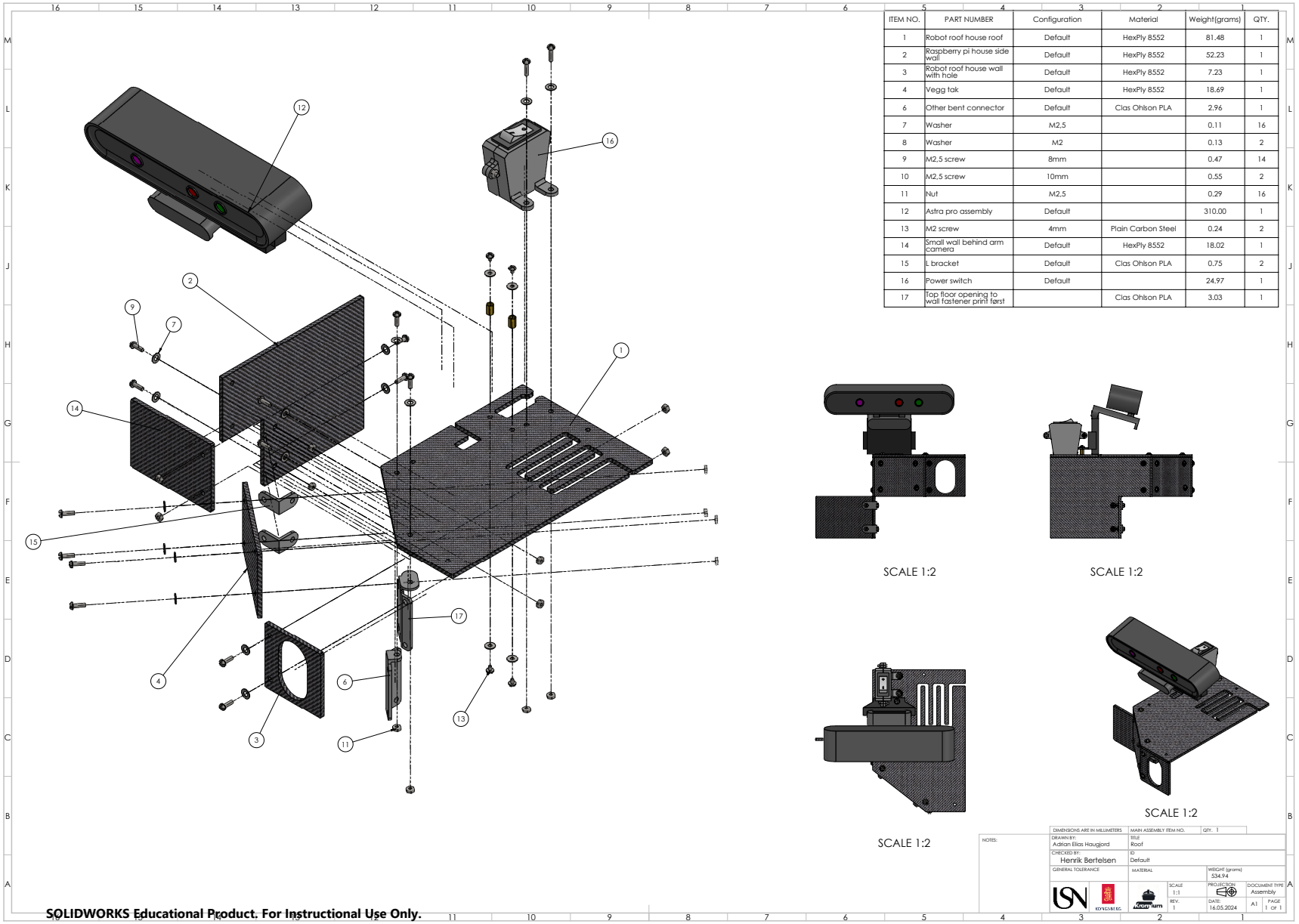


Figure T.73: Top floor assembly

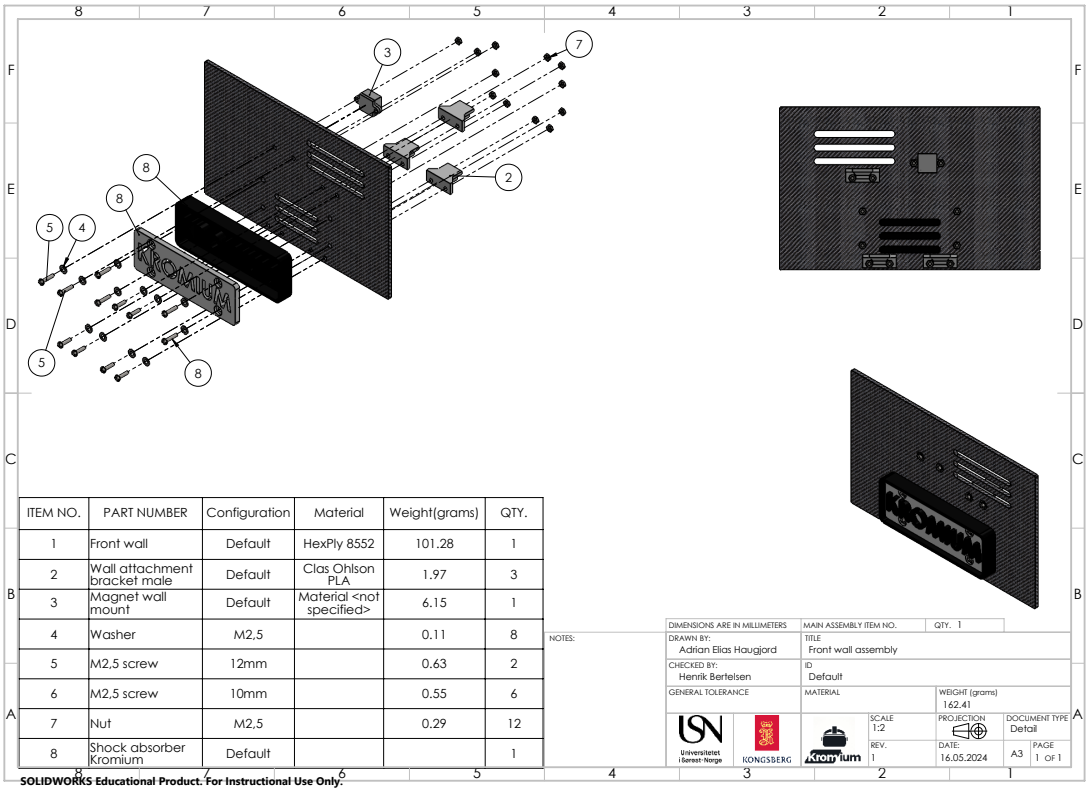


ITEM NO.	PART NUMBER	Configuration	Material	Weight(grams)	QTY.
1	Robot roof house roof	Default	HexPly 8552	81.48	1
2	Raspberry pi house side wall	Default	HexPly 8552	52.23	1
3	Robot roof house wall with hole	Default	HexPly 8552	7.23	1
4	Vegg tak	Default	HexPly 8552	18.69	1
6	Other bent connector	Default	Clas Ohlson PLA	2.96	1
7	Washer	M2,5		0.11	16
8	Washer	M2		0.13	2
9	M2,5 screw	8mm		0.47	14
10	M2,5 screw	10mm		0.55	2
11	Nut	M2,5		0.29	16
12	Astra pro assembly	Default		310.00	1
13	M2 screw	4mm	Plain Carbon Steel	0.24	2
14	Small wall behind arm camera	Default	HexPly 8552	18.02	1
15	L bracket	Default	Clas Ohlson PLA	0.75	2
16	Power switch	Default		24.97	1
17	Top floor opening to wall fastener print først		Clas Ohlson PLA	3.03	1

DIMENSIONS ARE IN MILLIMETERS	MAIN ASSEMBLY ITEM NO.	QTY. 1	
DRAWN BY: Adrian Elias Haugjord	TITLE: Roof		
CHECKED BY: Henrik Bertelsen	ID: Default		
GENERAL TOLERANCE	MATERIAL	WEIGHT (grams): 534.94	
	SCALE: 1:1	PROJECTION	DOCUMENT TYPE: Assembly
	REV: 1	DATE: 16.05.2024	A1 PAGE 1 OF 1

Figure T.74: Robot roof assembly

T.6 Wall assemblies



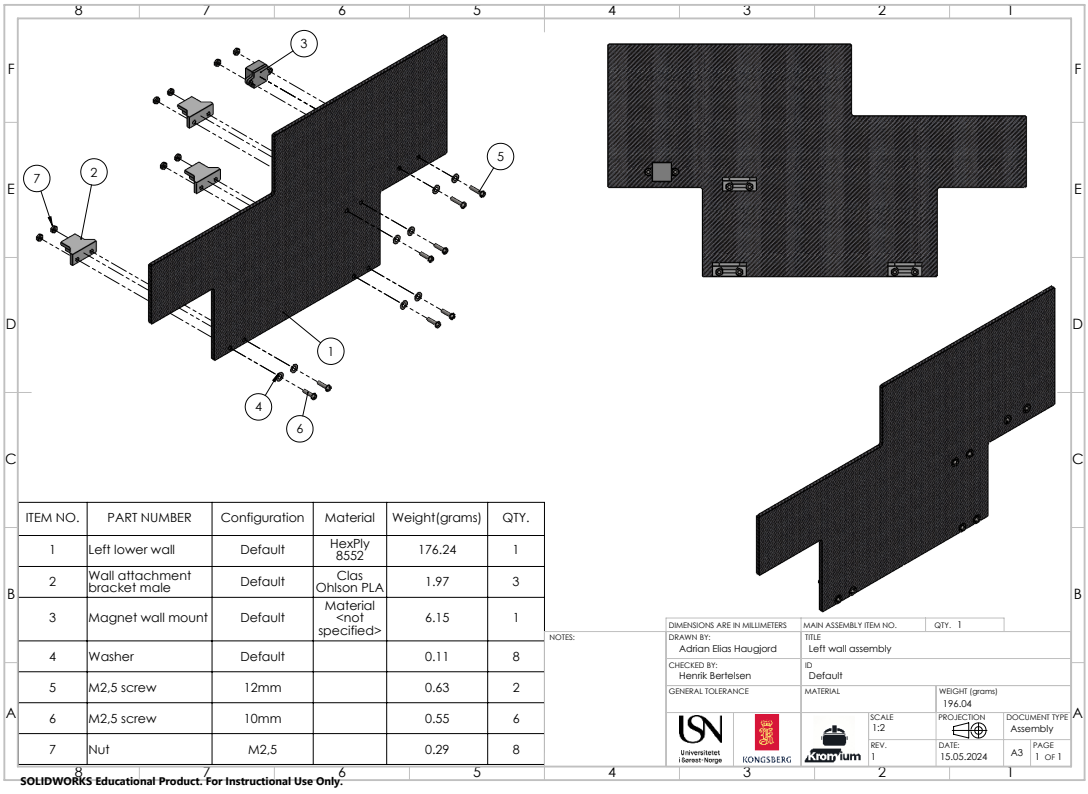
ITEM NO.	PART NUMBER	Configuration	Material	Weight(grams)	QTY.
1	Front wall	Default	HexPly 8552	101.28	1
2	Wall attachment bracket male	Default	Clas Ohlson PLA	1.97	3
3	Magnet wall mount	Default	Material <not specified>	6.15	1
4	Washer	M2,5		0.11	8
5	M2,5 screw	12mm		0.63	2
6	M2,5 screw	10mm		0.55	6
7	Nut	M2,5		0.29	12
8	Shock absorber Kromium	Default			1

NOTES:

DIMENSIONS ARE IN MILLIMETERS	MAIN ASSEMBLY ITEM NO.	QTY. 1	
DRAWN BY: Adrian Elias Haugjord	TITLE: Front wall assembly		
CHECKED BY: Henrik Bertelsen	ID: Default		
GENERAL TOLERANCE	MATERIAL	WEIGHT (grams): 162.41	
Universitetet i Sørøst-Norge / KONGSBERG / Kromium	SCALE: 1:2	PROJECTION	DOCUMENT TYPE: Detail
	REV.: 1	DATE: 16.05.2024	A3 / PAGE: 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.

Figure T.75: Front wall assembly



ITEM NO.	PART NUMBER	Configuration	Material	Weight(grams)	QTY.
1	Left lower wall	Default	HexPly 8552	176.24	1
2	Wall attachment bracket male	Default	Clas Ohlson PLA	1.97	3
3	Magnet wall mount	Default	Material <not specified>	6.15	1
4	Washer	Default		0.11	8
5	M2,5 screw	12mm		0.63	2
6	M2,5 screw	10mm		0.55	6
7	Nut	M2,5		0.29	8

NOTES:

DIMENSIONS ARE IN MILLIMETERS	MAIN ASSEMBLY ITEM NO.	QTY. 1	
DRAWN BY: Adrian Elias Haugjord	TITLE: Left wall assembly		
CHECKED BY: Henrik Bertelsen	ID: Default		
GENERAL TOLERANCE	MATERIAL	WEIGHT (grams): 196.04	
Universitetet i Sørøst-Norge / KONGSBERG / Kromium	SCALE: 1:2	PROJECTION	DOCUMENT TYPE: Assembly
	REV.: 1	DATE: 15.05.2024	A3 / PAGE: 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.

Figure T.76: Left wall assembly

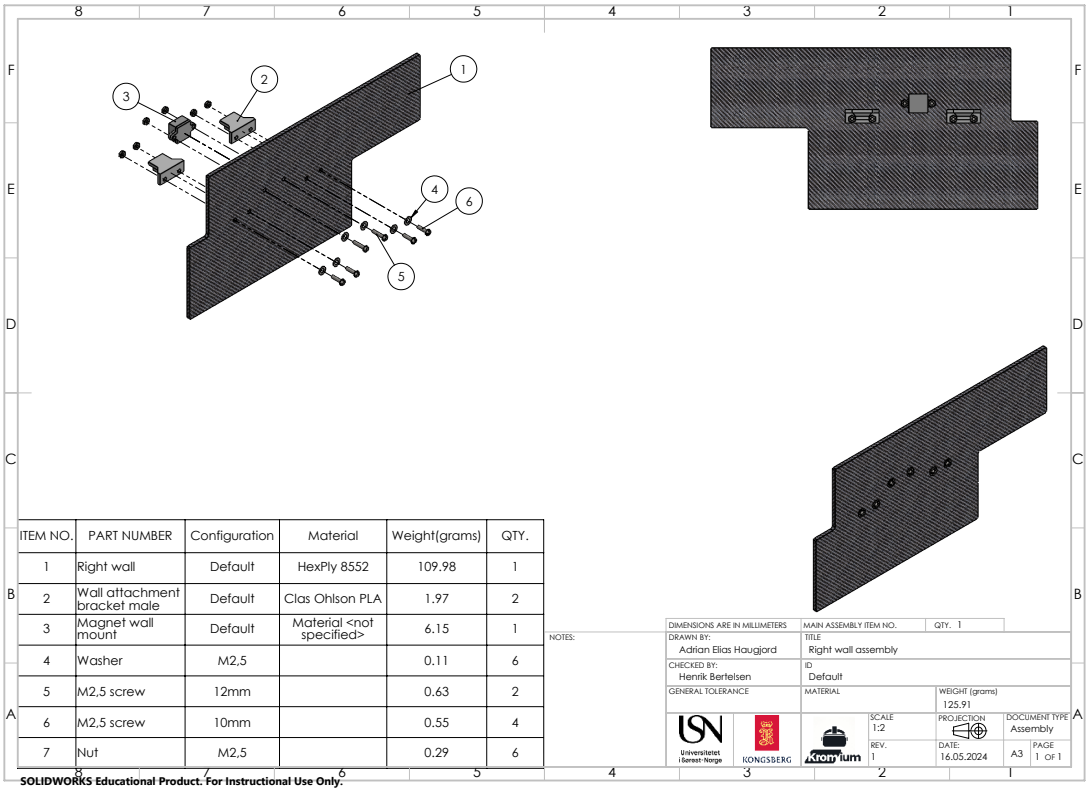


Figure T.77: Right wall assembly

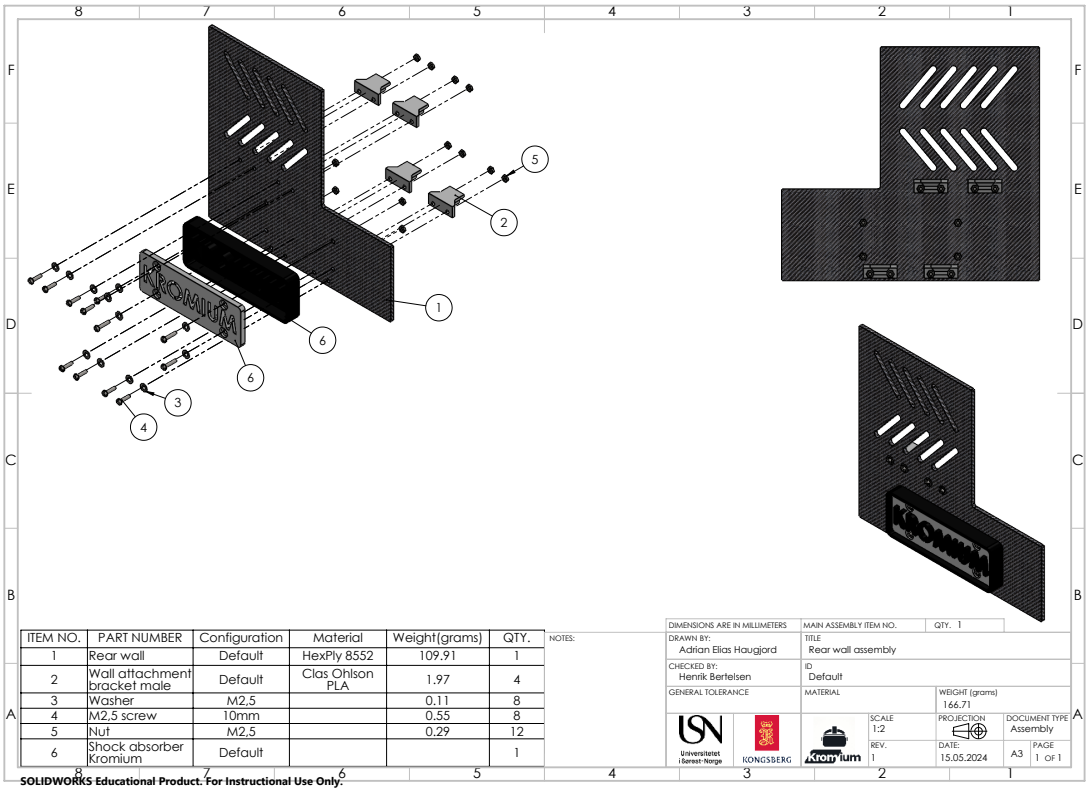


Figure T.78: Rear wall assembly

T.7 Main assembly

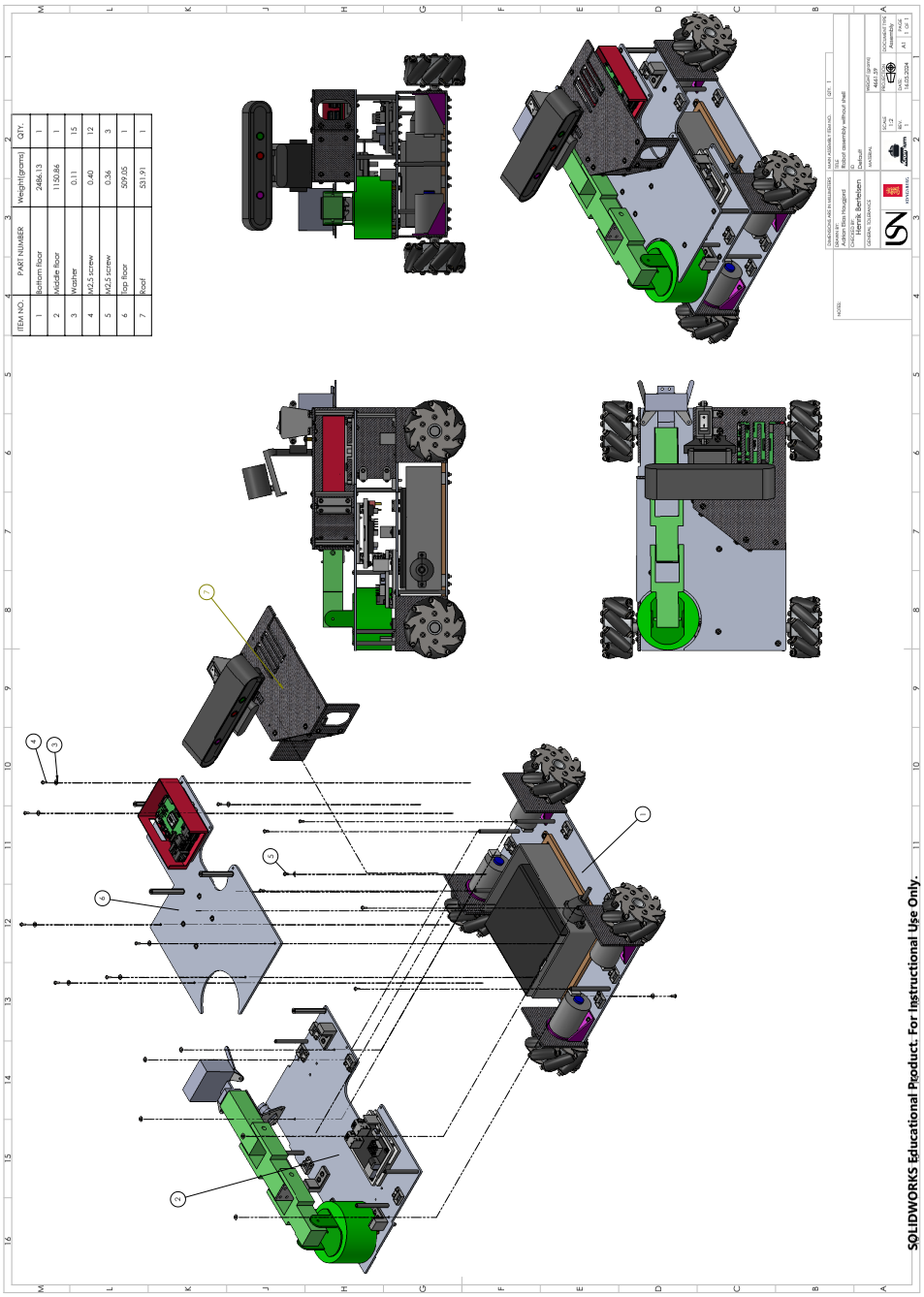
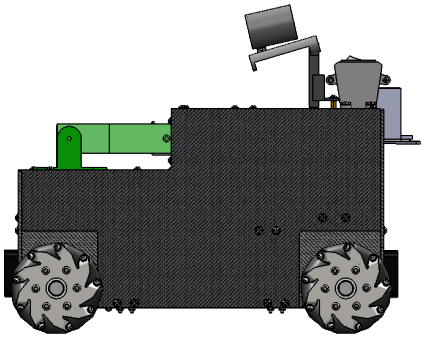
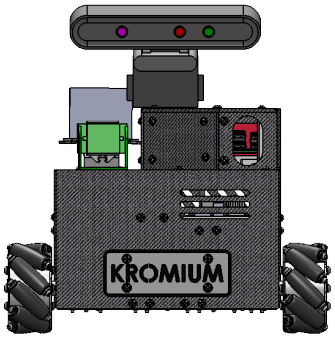
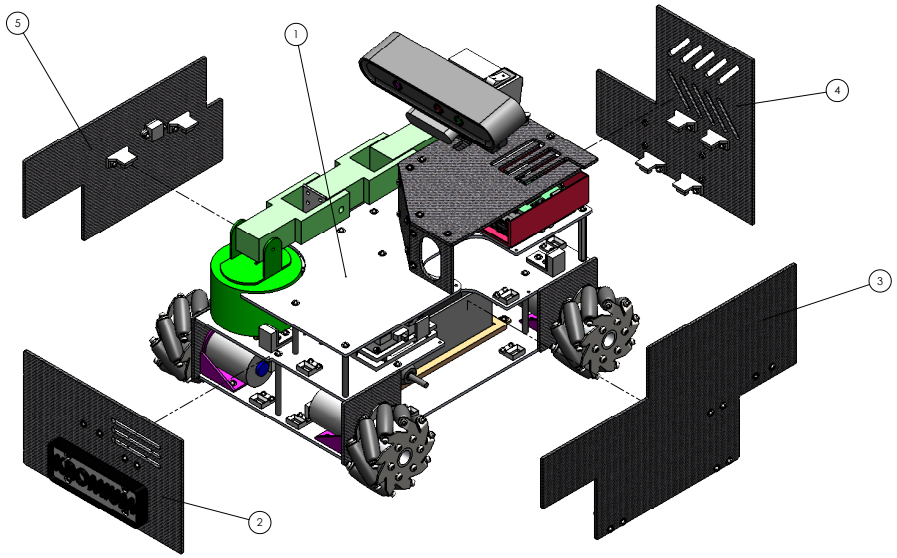


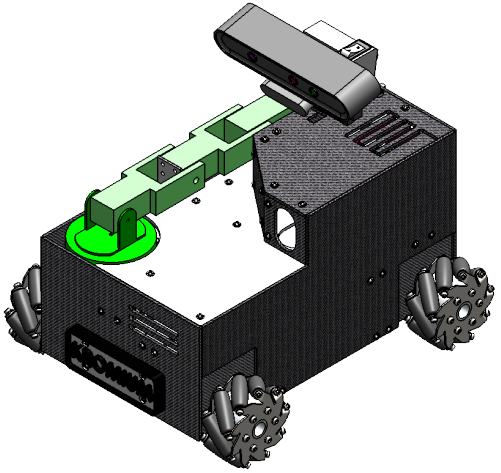
Figure T.79: Robot assembly without walls

ITEM NO.	PART NUMBER	Weight(grams)	QTY.
1	Robot assembly without shell	4661.59	1
2	Front wall	162.41	1
3	Left wall	196.04	1
4	Rear wall	166.71	1
5	Right wall	125.91	1









NOTES:	DIMENSIONS ARE IN MILLIMETERS	MAIN ASSEMBLY ITEM NO.	QTY.
	DRAWN BY: Adrian Elias Haugjord	TITLE: Robot assembly with shell	
	CHECKED BY: Henrik Bertelsen	ID: Default	
	GENERAL TOLERANCE	MATERIAL	WEIGHT (grams): 5312.66
		SCALE: 1:2	PROJECTION
		REV.: 1	DATE: 16.05.2024
		A1	PAGE: 1 OF 2

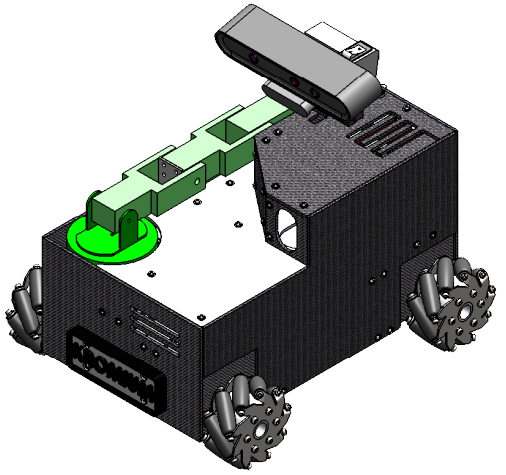
SOLIDWORKS Educational Product. For Instructional Use Only.

Figure T.80: Robot assembly with shell

Full assembly bill of materials

ITEM NO.	PART NUMBER	Configuration	Material	Weight(grams)	Total weight	QTY.
1	Robot car bottom floor		5052-H32	431.58	431.58	1
2	JGB-520 DC motor		Material <not specified>	150.00	600	4
3	MecanumWheel.stp			120	480	4
4	Wheel covers		HexPly 8552	20.21	80.84	4
5	Washer	M3		0.11	17.05	155
6	M2,5 screw	10mm		0.55	59.4	108
7	Nut	M2,5		0.29	33.93	117
8	Battery drawer rails		Clas Ohlson PLA	17.99	17.99	1
9	Battey drawer		Clas Ohlson PLA	122.54	122.54	1
11	Battery cell holder				0	2
110	Pole connector				0	16
111	able lug insulator				0	16
14	Cable lug metal part				0	16
16	M2.5 x10 mm		1060 Alloy	0.16	0.32	2
17	Nut M2.5		1060 Alloy	0.06	0.36	6
18	18650 cell				0	6
25	RS PRO magnetic male connector		Material <not specified>	6.50	6.5	1
26	Female connector holder		Clas Ohlson PLA	6.03	6.03	1
27	Female connector		Material <not specified>	9.40	9.4	1
28	Female connector holder		Clas Ohlson PLA	6.03	6.03	1
29	Sylinder spacer		1060 Alloy	2.04	12.24	6
30	M2,5 screw	6mm		0.40	11.2	28
31	Wall attachment bracket female		Clas Ohlson PLA	0.82	9.84	12
32	Robot car middle floor		5052-H32	408.86	408.86	1
33	Sylinder spacer		1060 Alloy	1.70	8.5	5
34	Hexagon sylinder		1060 Alloy	3.19	12.76	4
35	Base of arm				0	1
36	Servo 1-2 fastener				0	1
37	Simplefied arm double male				0	2
38	Simplefied arm double male				0	1
41	Raspberry pi camera		Material <not specified>	30.00	30	1
42	Golden fastner				0	4
43	Gripper base				0	1
44	Grip base				0	2
45	Grip end				0	1
46	Grip support				0	2
47	MirrorGrip end				0	1
48	Magnet house car front		Clas Ohlson PLA	3.31	9.93	3
49	Magnet				0	6
50	Magnet house car back		Clas Ohlson PLA	1.33	3.99	3
51	M2,5 screw	12mm		0.63	12.6	20
52	USB hub snap fastner bottom		Clas Ohlson PLA	14.54	14.54	1
53	USB hub		Material <not specified>	36.50	36.5	1
54	USB hub snap fastener top		Clas Ohlson PLA	10.49	10.49	1
55	M2 screw	5mm	Plain Carbon Steel	0.29	1.16	4
56	Nut	M2		0.31	3.72	12
57	Robot car top floor		5052-H32	334.89	334.89	1
58	Motor expansion board snap fastener bottom		Clas Ohlson PLA	19.12	19.12	1
59	Motor expansion board snap fastener top		Clas Ohlson PLA	10.89	10.89	1
60	Motor control board		Material <not specified>	46.00	46	1

ITEM NO.	PART NUMBER	Configuration	Material	Weight(grams)	Total weight	QTY.
61	M2 screw	8mm	Plain Carbon Steel	0.36	2.88	8
62	Red and white case		Material <not specified>	13.38	13.38	1
63	Raspberry pi r and w case sideways fastener top		Clas Ohlson PLA	20.85	20.85	1
64	Raspberry pi red and white case sideways fastener bottom		Clas Ohlson PLA	20.86	20.86	1
65	M2,5 screw	5mm		0.36	2.16	6
66	Robot roof house roof		HexPly 8552	81.48	81.48	1
67	Raspberry pi house side wall		HexPly 8552	52.23	52.23	1
68	Robot roof house wall with hole		HexPly 8552	7.23	7.23	1
69	Vegg tak		HexPly 8552	18.69	18.69	1
71	Other bent connector		Clas Ohlson PLA	2.96	2.96	1
72	M2,5 screw	8mm		0.47	6.58	14
73	Base joint				0	1
74	Top joint				0	1
75	Golden fastner				0	2
76	Camera (shit version)				0	1
77	Washer	M2		0.13	0.52	4
78	M2 screw	3mm	Plain Carbon Steel	0.24	0.96	4
79	Small wall behind arm camera		HexPly 8552	18.02	18.02	1
80	L bracket		Clas Ohlson PLA	0.75	1.5	2
81	Power switch			10.45	10.45	1
82	Switch fastener left side		Clas Ohlson PLA	6.85	6.85	1
83	Switch fastener right side		Clas Ohlson PLA	6.00	6	1
84	Front wall		HexPly 8552	101.28	101.28	1
85	Wall attachment bracket male		Clas Ohlson PLA	1.97	23.64	12
86	Magnet fastener wall bottom		Clas Ohlson PLA	0.59	1.77	3
87	Magnet fastener wall top		Clas Ohlson PLA	1.12	3.36	3
88	Shock absorber name				0	2
89	Shock absorber name hollow				0	2
90	Left lower wall		HexPly 8552	176.24	176.24	1
91	Rear wall		HexPly 8552	109.91	109.91	1
92	Right wall		HexPly 8552	109.98	109.98	1
94	RASPBERRY_PI_5				0	1
114	520 DC motor fastener		Clas Ohlson PLA	9.21	36.84	4
113	Arm cam holder fixed angle		Clas Ohlson PLA	6.27	6.27	1
99	Cam cable holder		Clas Ohlson PLA	6.31	6.31	1
102	Battery cell holder seperator big		Oak	19.39	19.39	1
103	Wire short				0	6
104	Wire short				0	1
105	Negative to out				0	1
106	pos to neg				0	1
107	M2.5 x12 mm				0	4
108	Battery assy box		Clas Ohlson PLA	112.56	112.56	1
109	End cover w. mag		Clas Ohlson PLA	12.99	12.99	1



NOTES:

DIMENSIONS ARE IN MILLIMETERS	MAIN ASSEMBLY ITEM NO.	QTY.	
DRAWN BY: Adrian Elias Haugjord	TITLE: Robot assembly with shell		
CHECKED BY: Henrik Bertelsen	ID: Default		
GENERAL TOLERANCE	MATERIAL	WEIGHT (grams): 5312.66	
	SCALE: 1:2	PROJECTION	DOCUMENT TYPE: Assembly
	REV.: 1	DATE: 16.05.2024	A1 PAGE 2 OF 2

SOLIDWORKS Educational Product. For Instructional Use Only.

Figure T.81: Robot assembly bill of materials

T.8 Production drawings

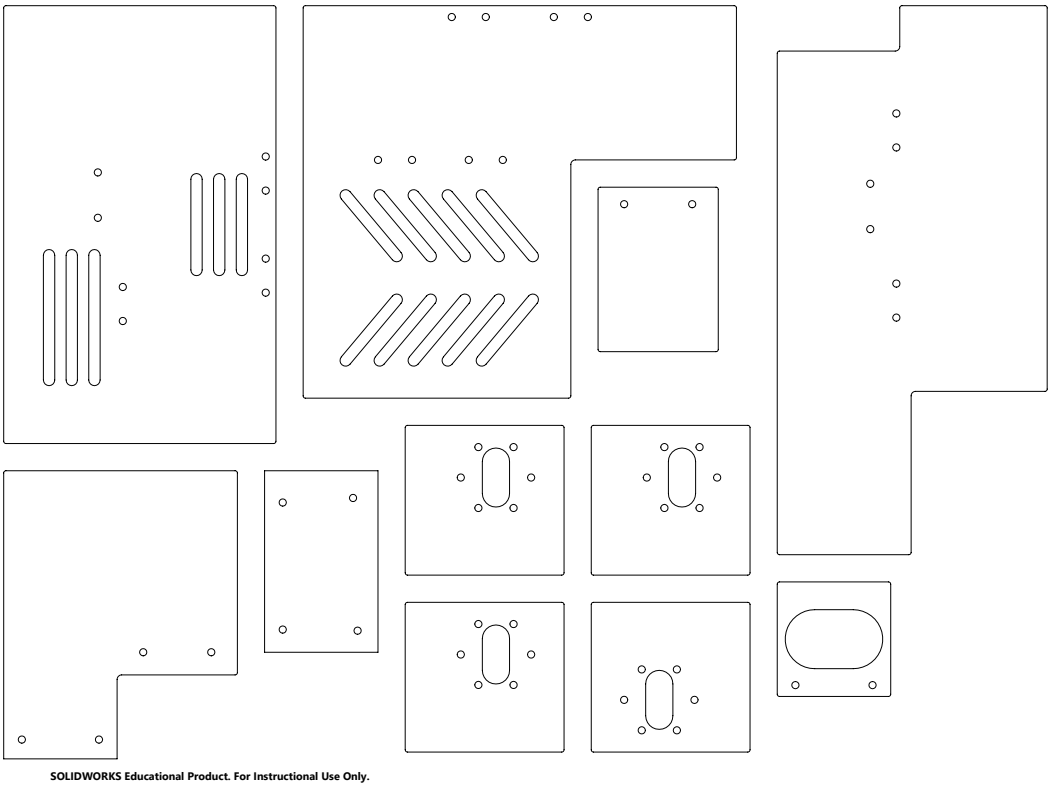


Figure T.82: Parts layout for CNC machining

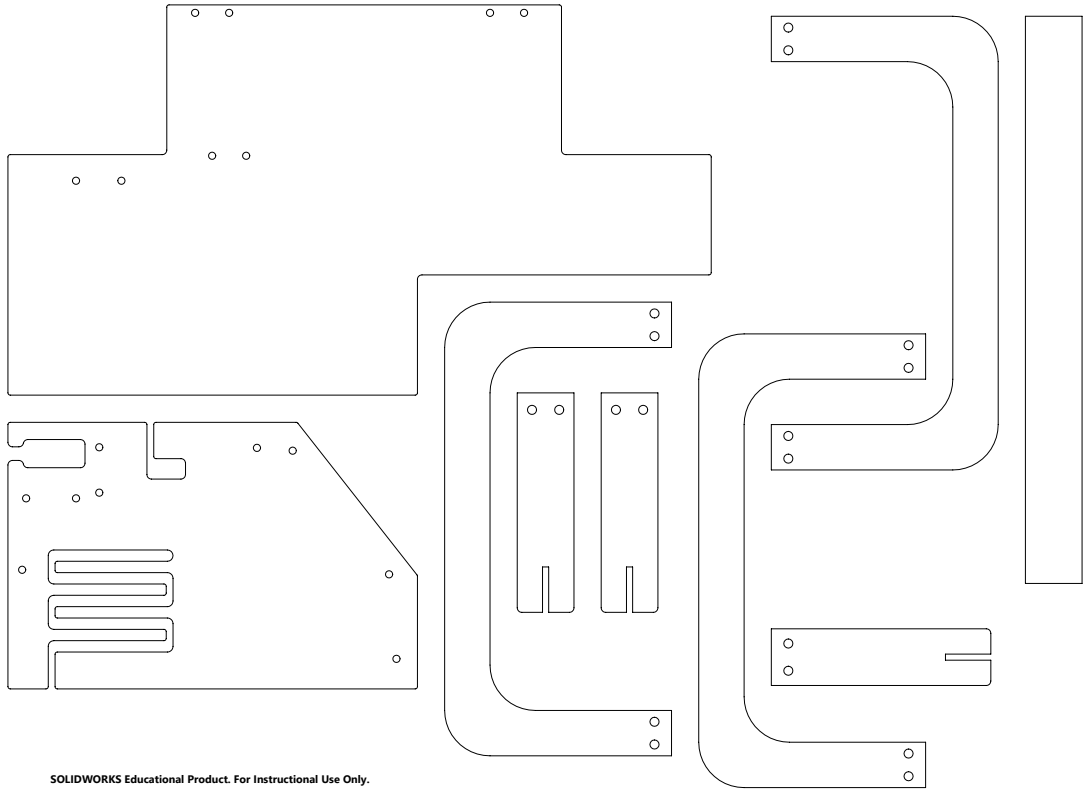


Figure T.83: Parts layout for water jet cutting

U Robot code documentation

robot

Release 2.0.0

Kromium

May 15, 2024

CONTENTS:

1 interfaces **1**
- 1.1 ArmAngles . . . 1
- 1.2 DepthData . . . 1
- 1.3 Message . . . 1
- 1.4 RobotData . . . 1
- 1.5 SwitchCamera . . . 1
- 1.6 VRMode . . . 2
- 1.7 VRDrive . . . 2
- 1.8 VRArm . . . 2
- 1.9 VRScrew . . . 2

2 vr_linker **3**
- 2.1 vr_linker node . . . 3
- 2.2 vr_linker . . . 4
- 2.3 utils . . . 7

3 master **9**
- 3.1 master node . . . 9
- 3.2 modes . . . 10

4 controller **11**
- 4.1 controller node . . . 11
- 4.2 controller . . . 12
- 4.3 robot . . . 16
- 4.4 arm kinematics . . . 20
- 4.5 exceptions . . . 21
- 4.6 enums . . . 22
- 4.7 utils . . . 23
- 4.8 rosmaster . . . 24

5 ai_detection **31**
- 5.1 detection . . . 31
- 5.2 database . . . 32
- 5.3 utils . . . 32

6 depth_data **35**
- 6.1 depth data . . . 35
- 6.2 utils . . . 35

7 logger **37**
- 7.1 logger_node . . . 37

7.2 logger . . . 37
7.3 utils . . . 38

8 robot **39**
8.1 Important information . . . 39
8.2 Prerequisites . . . 40
8.3 Installation . . . 40
8.4 Running . . . 41
8.5 Development . . . 42
8.6 Documentation . . . 42

Python Module Index **45**

Index **47**

CHAPTER

ONE

INTERFACES

Here are all the custom message interfaces used by the ROS 2 environment.

1.1 ArmAngles

uint8 rotation uint8 shoulder uint8 elbow uint8 tilt uint8 wrist uint8 pinch

1.2 DepthData

uint64 time int8[] x int8[] y int8[] z

1.3 Message

string message uint8 level

1.4 RobotData

geometry_msgs/Vector3 accelerometer geometry_msgs/Vector3 gyroscope geometry_msgs/Vector3 magnetometer float32 cms_speed int8 speed int16 battery float32 voltage string mode

1.5 SwitchCamera

string camera bool flip_image bool draw_dot

1.6 VRMode

uint8 mode

1.7 VRDrive

float64 x float64 y int64 speed string drive_mode

1.8 VRArm

float64 x float64 y float64 z float64 wrist float64 strength

1.9 VRScrew

bool clockwise

CHAPTER

TWO

VR_LINKER

This is the node which connects the Meta Quest 3 VR headset to the ROS 2 ecosystem. It hosts a TCP socket server the headset can connect to. Data received here gets sent to the ROS 2 ecosystem.

Subscribed topics

- robot_data (RobotData): The robot data received from the robot.
- arm_angles (ArmAngles): The arm angles received from the robot.
- depth_data (DepthData): The depth data received from the robot.
- vr_mode (VRMode): The VR mode data received from the robot.
- vr_arm (VRArm): The VR arm data received from the robot.
- vr_drive (VRDrive): The VR drive data received from the robot.
- message (Message): Message to be logged including level.

Published topics

- _vr_drive (VRDrive): The VR drive data received from the headset.
- _vr_arm (VRArm): The VR arm data received from the headset.
- _vr_mode (VRMode): The VR mode data received from the headset.
- _vr_screw (VRScrew): The VR screw data received from the headset.
- message (Message): Message to be logged including level.

2.1 vr_linker node

class robot.src.vr_linker.vr_linker.vr_linker_node.**VRLinkerNode**(**args: Any*, ***kwargs: Any*)

listen() → None

Listens for incoming connections and messages.

client_disconnected(*e*) → None

Handles the case when the client disconnects.

Parameters

e – Exception message

cleanup()

Cleans up the node.

robot.src.vr_linker.vr_linker.vr_linker_node.**main**(*args=None*)

2.2 vr_linker

class robot.src.vr_linker.vr_linker.vr_linker.**AnyMessage**

class robot.src.vr_linker.vr_linker.vr_linker.**VRLinker**(*node*)

static _to_json(*data: dict | bytes*) → str | bytes

Converts data to JSON.

Parameters
data – data to convert

Returns
JSON data as dict or bytes

_is_vr_connected() → bool

Checks if the VR headset is connected.

Returns
True if the VR headset is connected

_send(*data: dict*) → None

Sends data over the TCP socket.

Parameters
data – data

_send_compressed(*data: dict*) → None

Sends compressed data over the TCP socket.

Parameters
data – data

static _get_vector_data(*msg*, *key: str*) → list[float]

Gets the x, y, z data from a vector message.

Parameters

- **msg** – message
- **key** – key to get data from

Returns
x, y, z data

static _get_content_any_msg_type(*msg*) → dict

Gets the content of a message of any type.

Parameters
msg – ROS 2 message

Returns
message content as dict

handle_log_backup(*msg*) → None

Receives a message of any type and sends to VR headset as a backup.

Parameters
msg – message

handle_robot_data(*msg*) → None

Receives RobotData messages and sends over socket.

Parameters
msg – RobotData message

handle_arm_angles(*msg*) → None

Receives ArmAngles messages and sends over socket.

Parameters
msg – ArmAngles message

handle_depth_data(*msg*) → None

Receives DepthData messages and sends over socket.

Parameters
msg – DepthData message

process_message() → None

Processes a message sent by the VR headset over TCP socket.

static _is_vr_arm_message(*data: dict*) → bool

Checks if the message is a VRArm message.

Parameters
data – data

Returns
True if the message is a VRArm message

static _is_vr_drive_message(*data: dict*) → bool

Checks if the message is a VRDrive message.

Parameters
data – data

Returns
True if the message is a VRDrive message

static _is_screw_message(*data: dict*) → bool

Checks if the message is a mode message.

Parameters
data – data

Returns
True if the message is a mode message

static _is_mode_message(*data: dict*) → bool

Checks if the message is a mode message.

Parameters
data – data

Returns
True if the message is a mode message

_publish_vr_arm(*data: dict*) → None

Publishes VRArm messages.

Parameters
data – data

_publish_vr_drive(*data: dict*) → None

Publishes VRDrive messages.

Parameters
data – data

_publish_ping(*data: dict*) → None

Publishes ping messages.

Parameters
data – data

_publish_mode(*data: dict*) → None

Publishes mode messages.

Parameters
data – data

_publish_screw(*data: dict*) → None

Publishes screw messages.

Parameters
data – data

_publish_get_depth() → None

Publishes get depth messages.

Parameters
data – data

static _is_drive_mode(*data: dict*) → bool

Checks if the message is a drive mode message.

Parameters
data – data

Returns
True if the message is a drive mode message

static _is_get_depth_message(*data: dict*) → bool

Checks if the message is a get depth message.

Parameters
data – data

Returns
True if the message is a get depth message

_set_drive_mode(*data: dict*) → None

Sets the driving mode.

Parameters
data – data

static _is_ping_message(*data: dict*) → bool

Checks if the message is a ping message.

Parameters
data – data

Returns
True if the message is a ping message

_publish_data(*data: dict*) → None

Publishes VRDrive messages.

Parameters
data – data

2.3 utils

robot.src.vr_linker.vr_linker.utils.**get_config**() → dict

Reads the config file.

Returns
config

robot.src.vr_linker.vr_linker.utils.**get_arm_timeout_seconds**() → float

Returns the arm timeout in seconds.

Returns
timeout in seconds

CHAPTER

THREE

MASTER

This node makes sure that the VR data is received from the headset is valid (in correct mode) and published to the rest of the system.

Subscribed topics

- _vr_drive (VRDrive): The untrusted VR drive data received from the headset.
- _vr_arm (VRArm): The untrusted VR arm data received from the headset.
- _vr_mode (VRMode): The untrusted VR mode data received from the headset.
- _vr_screw (VRScrew): The untrusted VR screw data received from the headset.
- _robot_data (RobotData): The untrusted robot data received from the robot.

Published topics

- vr_drive (VRDrive): The VR drive data received from the headset.
- vr_arm (VRArm): The VR arm data received from the headset.
- vr_mode (VRMode): The VR mode data received from the headset.
- vr_screw (VRScrew): The VR screw data received from the headset.
- robot_data (RobotData): The robot data received from the robot.
- message (Message): Log data to send to logger.

3.1 master node

robot.src.master.master.master_node.**matches_mode_and_not_emergency**(*mode:* Mode)

class robot.src.master.master.master_node.**MasterNode**(**args: Any*, ***kwargs: Any*)

_switch_camera(*mode: str*) → None

Publishes a SwitchCamera message.

Parameters
mode – camera mode

set_mode(*mode:* Mode) → None

Sets the mode.

Parameters
mode – mode

int_to_mode(*mode: int*) → *Mode*

Converts an integer to a Mode.

Parameters
mode – mode

Returns
Mode

handle_robot_data(*msg*) → None

Handles RobotData messages.

Parameters
msg – RobotData message

handle_unsafe_vr_mode(*msg*) → None

Handles VRMode messages.

Parameters
msg – VRMode message

handle_unsafe_vr_drive(*msg*) → None

Handles VRDrive messages.

Parameters
msg – VRDrive message

handle_unsafe_vr_arm(*msg*) → None

Handles VRArm messages.

Parameters
msg – VRArm message

handle_unsafe_vr_screw(*msg*) → None

Handles VRScrew messages.

Parameters
msg – VRScrew message

robot.src.master.master.master_node.**main**(*args=None*)

3.2 modes

class robot.src.master.master.modes.**Mode**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

IDLE = 0

DRIVE = 1

ARM = 2

EMERGENCY = 3

CHAPTER

FOUR

CONTROLLER

This is the only node which interacts with the expansion board (robot). It is responsible for controlling the robot's movement and arm movements.

Subscribed topics

- vr_drive (VRDrive) - The drive data from the VR headset
- vr_arm (VRArm) - The hand data from the VR headset to control the robotic arm
- vr_mode (VRMode) - The mode data from the VR headset
- vr_screw (VRScrew) - The screw data from the VR headset

Published topics

- switch_camera (SwitchCamera) - Switches the camera view depending on the mode
- arm_angles (ArmAngles) - The angles of the arm joints to publish to the digital twin
- _robot_data (RobotData) - Data from the robot such as voltage, speed, gyroscope, etc.
- message (Message) - Message to be logged including level.

4.1 controller node

class robot.src.controller.controller.controller_node.**ControllerNode**(**args: Any*, ***kwargs: Any*)

Interacts with the Expansion board.

get_robot_data() → None

Gets robot data such as voltage, speed, gyroscope, etc. and publishes it.

_switch_to_forward_camera() → None

Switches to the forward camera.

_switch_to_reverse_camera() → None

Switches to the reverse camera.

handle_vr_drive(*msg*) → None

Handles VRDrive messages.

Parameters

msg – VRDrive message

get_arm_angles() → None

Gets the arm angles and publishes them.

robot.src.controller.controller.controller_node.**main**(*args=None*)

4.2 controller

class robot.src.controller.controller.controller.**Controller**(*production: bool = True*)

_convert_coordinates_to_direction(*x: float*, *y: float*) → *Direction*

Converts x, y coordinates to a direction.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns

direction to drive in

static convert_x_to_angle_difference(*x: float*) → int

Converts x to angle difference.

Parameters

x – x

Returns

angle difference

static convert_y_to_angle_difference(*y: float*) → int

Converts y to angle difference.

Parameters

y – y

Returns

angle difference

static convert_pinch_to_angle(*pinch: float*) → int

Converts pinch to angle.

Parameters

pinch – pinch

Returns

angle

_set_last_arm_mode(*mode: int*) → None

Sets the last arm mode.

Parameters

mode – mode

_set_mode_defaults(*mode: int*) → None

Sets the mode defaults.

Parameters

mode – mode

check_last_message_received() → None

Checks if last message was received more than *sleep_mode_after* seconds ago.

If so, the robot will stop and sleep. Will long beep twice to indicate this. Function gets called by rclpy timer with the hertz specified in the config file.

static _speed_out_of_range(*speed: int*) → bool

Checks if the speed is out of range.

Parameters
speed – speed

Returns
True if the speed is out of range

static _convert_value_to_angle(*x: float*) → int

Converts x to angle in degrees.

Parameters
x – x

Returns
angle in degrees

static _convert_to_point(*angle: int*, *y: float*) → tuple[float, float]

Converts angle and y to x and y coordinates.

Parameters

- **angle** – angle
- **y** – y

Returns
x and y coordinates

static _convert_coordinates(*y: float*, *z: float*) → tuple[float]

Converts y and z coordinates from [-1, 1] to cm.

Parameters

- **y** – y
- **z** – z

Returns
y and z coordinates in cm

static _would_collide_with_front(*x: float*, *y: float*, *z: float*) → bool

Checks if the point would collide with the front of the robot.

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **z** – z coordinate

Returns
True if the point would collide with the front of the robot

_is_illegal_point(*x: float*, *y: float*, *z: float*) → bool

Checks if the point is illegal.

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **z** – z coordinate

Returns
True if the point is illegal

handle_vr_arm(*msg*) → None

Handles VRArm messages.

Parameters
msg – VRArm message

handle_vr_screw(*msg*) → None

Handles VRScrew messages.

Parameters
msg – VRScrew message

static _get_reverse_direction(*direction:* Direction) → *Direction*

Converts the direction to reverse mode direction.

Parameters
direction – direction

Returns
reverse mode direction

static _is_within_origin_buffer(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are within the origin buffer.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are within the origin buffer

static _is_in_first_quadrant(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the first quadrant.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the first quadrant

static _is_in_second_quadrant(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the second quadrant.

Parameters

- **x** – x coordinate

- **y** – y coordinate

Returns
True if the x,y coordinates are in the second quadrant

static _is_in_third_quadrant(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the third quadrant.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the third quadrant

static _is_in_fourth_quadrant(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the fourth quadrant.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the fourth quadrant

static _is_right(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the right side.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the right side

static _is_left(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the left side.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the left side

static _is_forward(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the forward side.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns
True if the x,y coordinates are in the forward side

static _is_backward(*x: float*, *y: float*) → bool

Checks if the x, y coordinates are in the backward side.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns

True if the x,y coordinates are in the backward side

_get_precision_direction(*x: float*, *y: float*) → *Direction*

Converts the direction to precision mode direction.

Parameters

direction – direction

Returns

precision mode direction

_set_speed(*speed: int*) → None

Sets the speed of the robot.

Parameters

speed – speed

vr_drive(*x: float*, *y: float*, *speed: int*, *drive_mode: str*) → None

Drives the robot based on ROS message input.

Parameters

- **x** – x VR hand coordinate
- **y** – y VR hand coordinate
- **speed** – speed (length of the xy vector)
- **drive_mode** – drive mode (normal, precision, reverse)

handle_vr_mode(*msg*) → None

Handles VRMode messages.

Parameters

msg – VRMode message

4.3 robot

class robot.src.controller.controller.robot.**Robot**(*production: bool = True*)

get_stored_angles() → dict

_reset_arm() → None

Resets the arm to its default position.

reset() → None

Resets the robot.

get_speed() → int

Returns the speed.

Returns
speed

get_direction(*direction:* Direction) → list[int]

Returns the direction vector.

Parameters
direction – direction

Returns
direction vector

Example

```
>>> from controller.robot import Robot, Direction
>>> robot = Robot()
>>> robot.set_speed(50)
>>> robot.get_direction(Direction.FORWARD)
[50, 50, 50, 50]
```

set_speed(*speed: int*) → None

Sets the speed.

Parameters
speed – speed

drive(*direction:* Direction | *tuple[int]*) → None

Drives the robot in a direction.

Parameters
direction – direction

stop() → None

Stops the robot.

forward() → None

Drives the robot forward.

set_pinch(*angle: int*) → None

Sets the pinch angle.

Parameters
angle – angle

pinch() → None

Pinches the “fingers”.

unpinch() → None

Unpinches the “fingers”.

get_wrist() → int

Returns the wrist angle.

Returns
wrist angle

set_wrist(*angle: int*) → None

Turns the wrist. Angle should be between 0 and 270.

Parameters
angle – angle

reset_wrist() → None

Resets the wrist to its default position.

_screw(*clockwise: bool*, *timeout: int = 1*) → None

Screws the "fingers" in a direction.

Parameters

- **clockwise** – if True, the fingers will move clockwise
- **timeout** – timeout

handle_screw(*clockwise: bool*, *rounds: int*) → None

Screws the "fingers" in a direction.

Parameters

- **clockwise** – if True, the fingers will move clockwise
- **rounds** – number of rounds

get_arm_rotation() → int

Returns the arm rotation angle.

Returns
arm rotation angle

set_arm_rotation(*angle: int*) → None

Sets the arm rotation angle.

Parameters
angle – angle

reset_arm_rotation() → None

Resets the arm rotation to its default position.

get_arm_shoulder() → int

Returns the arm shoulder angle.

Returns
arm shoulder angle

set_arm_shoulder(*angle: int*) → None

Sets the arm shoulder angle.

Parameters
angle – angle

set_arm_rotation_difference(*angle: int*) → None

Sets the arm rotation angle difference.

Parameters
angle – angle

reset_arm_shoulder() → None

Resets the arm shoulder to its default position.

`get_arm_elbow()` → int
Returns the arm elbow angle.

Returns
arm elbow angle

`set_arm_elbow(angle: int)` → None
Sets the arm elbow angle.

Parameters
angle – angle

`reset_arm_elbow()` → None
Resets the arm elbow to its default position.

`get_arm_tilt()` → int
Returns the arm tilt angle.

Returns
arm tilt angle

`set_arm_tilt(angle: int)` → None
Sets the arm tilt angle.

Parameters
angle – angle

`reset_arm_tilt()` → None
Resets the arm tilt to its default position.

`beep()` → None
Beeps the robot for 100ms, used for indicating mode change.

`long_beep()` → None
Beeps the robot for 200ms, used for indicating robot is ready.

`static _estimate_battery_percentage(voltage: float)` → int
Estimates the battery percentage based on the voltage.

Parameters
voltage – battery voltage

Returns
battery percentage

`static _estimate_cms_speed(speed: int)` → float
Estimates the cm/s speed

Parameters
speed – speed in percentage

Returns
cm/s speed

`get_data()` → dict
Returns the robot data.

Returns
robot data

`__del__()` → None
Cleans up the robot.

4.4 arm kinematics

robot.src.controller.controller.arm_kinematics.**calculate_auxilaries**(*partial_point_y: float*, *partial_point_z: float*) → tuple[float]

Calculates auxiliary angles alpha and beta between triangle delta_squared and the two links.

Parameters

- **partial_point_y** – partial point y
- **partial_point_z** – partial point z

Returns
tuple of alpha and beta in radians with the last element being None or np.nan

robot.src.controller.controller.arm_kinematics.**calculate_elbow_up_degrees**(*partial_point_y: float*, *partial_point_z: float*, *phi: float*) → tuple[float]

Calculates the up angles in degrees.

Parameters

- **partial_point_y** – partial point y
- **partial_point_z** – partial point z
- **phi** – angle in radians

Returns
tuple of theta_1, theta_2, theta_3 in degrees

robot.src.controller.controller.arm_kinematics.**calculate_elbow_down_degrees**(*partial_point_y: float*, *partial_point_z: float*, *phi: float*) → tuple[float]

Calculates the down angles in degrees.

Parameters

- **partial_point_y** – partial point y
- **partial_point_z** – partial point z
- **phi** – angle in radians

Returns
tuple of theta_1, theta_2, theta_3 in degrees

robot.src.controller.controller.arm_kinematics.**has_any_nan**(*thetas: tuple[float]*) → bool

Checks if any of the thetas are np.nan.

Parameters
thetas – tuple of thetas in degrees

Returns
True if any of the thetas are np.nan, False otherwise

robot.src.controller.controller.arm_kinematics.**has_out_of_range**(*thetas: tuple[float]*) → bool

Checks if any of the thetas are out of range. The range is -90 to 90 degrees for all joints except the first one which is 0 to 180 degrees.

Parameters
thetas – tuple of thetas in degrees

Returns
True if any of the thetas are out of range, False otherwise

robot.src.controller.controller.arm_kinematics.**get_mapped_phi**(*z: float*) → float

Maps the z value to a phi value in degrees.

Parameters
z – z value in cm

Returns
phi value in radians

robot.src.controller.controller.arm_kinematics.**calculate**(*y: float*, *z: float*, *phi: float = None*) → tuple[float | None]

Calculates the angles of the robotic arm.

Parameters

- **y** – y value in cm
- **z** – z value in cm
- **phi** – angle in degrees

Returns
tuple of theta_1, theta_2, theta_3 in degrees

robot.src.controller.controller.arm_kinematics.**convert_thetas_to_servo_angles**(*thetas: tuple[float]*) → tuple[int]

Converts the thetas to servo angles.

Parameters
thetas – tuple of thetas in degrees

Returns
tuple of servo angles in degrees

4.5 exceptions

exception robot.src.controller.controller.exceptions.**ControllerException**

Generic exception for the controller module.

exception robot.src.controller.controller.exceptions.**NotInProductionMode**

Raised when a method requires production mode to be enabled.

Production mode indicates if the expasion board should be used or not.

4.6 enums

class robot.src.controller.controller.enums.**Direction**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Direction enum.

FORWARD = (1, 1, 1, 1)

BACKWARD = (-1, -1, -1, -1)

STOP = (0, 0, 0, 0)

TURN_LEFT = (0, 0, 1, 1)

TURN_RIGHT = (1, 1, 0, 0)

REVERSE_TURN_LEFT = (-1, -1, 0, 0)

REVERSE_TURN_RIGHT = (0, 0, -1, -1)

LATERAL_LEFT = (-1, 1, 1, -1)

LATERAL_RIGHT = (1, -1, -1, 1)

DIAGONAL_LEFT = (0, 1, 1, 0)

DIAGONAL_RIGHT = (1, 0, 0, 1)

REVERSE_DIAGONAL_LEFT = (0, -1, -1, 0)

REVERSE_DIAGONAL_RIGHT = (-1, 0, 0, -1)

class robot.src.controller.controller.enums.**Preset**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Preset enum.

SPEED = 50

UNPINCH_ANGLE = 45

PINCH_ANGLE = 180

WRIST_ANGLE = 100

ARM_TILT_ANGLE = 90

ARM_ROTATION_ANGLE = 90

ARM_SHOULDER_ANGLE = 90

ARM_ELBOW_ANGLE = 90

class robot.src.controller.controller.enums.**Arm**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Arm enum.

ROTATION = 1

SHOULDER = 2

ELBOW = 3

TILT = 4

WRIST = 5

PINCH = 6

4.7 utils

robot.src.controller.controller.utils.**in_production_mode**(*func*)

Decorator which checks if we are in production mode or not.

If we are in production mode, the function will be executed. If not, the *NotInProductionMode* exception will be raised.

Parameters
func – function to decorate

Returns
wrapper function

Return type
function_wrapper

robot.src.controller.controller.utils.**set_last_message**(*func*)

Decorator which sets the last message time and is_sleeping to False.

Parameters
func – function to decorate

Returns
wrapper function

Return type
function_wrapper

robot.src.controller.controller.utils.**fill_vector_msg**(*msg*, *key: str*, *data: list*)

Fills a std_msgs.msg.Vector3 message for given key.

Parameters

- **msg** – message
- **key** – key to fill for
- **data** – x, y, z data

Returns
message with filled data

robot.src.controller.controller.utils.**get_config**() → dict

Returns the config from a local JSON file.

JSON files do not get compiled, so values can be changed without recompiling the code. Hence the reason for having some options in a JSON file.

Returns
config

robot.src.controller.controller.utils.**get_threshold**() → int

Returns the threshold.

Returns
threshold

robot.src.controller.controller.utils.**get_production**() → bool

Returns if we are in production mode or not.

Production mode means that the robot will interact with the expansion board (true) or not (false).

Returns
production mode

robot.src.controller.controller.utils.**get_data_hertz**() → int

Returns the hertz of which robot data should be sent to the VR headset.

Returns
hertz

robot.src.controller.controller.utils.**get_sleep_mode_hertz**() → int

Returns the hertz of which how often the robot should check.

Returns
hertz

robot.src.controller.controller.utils.**get_sleep_mode_after**() → int

Returns the time after which the robot should go to sleep mode.

Returns
seconds

4.8 rosmaster

class robot.src.controller.controller.rosmaster.**Rosmaster**(*car_type: int = 1*, *com: str = '/dev/expbrd'*, *delay: float = 0.002*, *debug: bool = False*)

This is a class for controlling the robot. Which is a modified version of yahboom.net's code.

__uart_state = 0

__parse_data(*ext_type: int*, *ext_data*) → None

Parses the data.

Parameters

- **ext_type** – the type of the data
- **ext_data** – the data

set_data() → None

Receives the data.

__request_data(*function*, *param: int = 0*) → None

Requests the data.

Parameters

- **function** – the function
- **param** – the parameter

__arm_convert_value(*s_id: int*, *s_angle: int*) → int

Converts the value of the arm.

Parameters

- **s_id** – the id of the arm
- **s_angle** – the angle of the arm

Returns
the value of the arm

static _arm_convert_angle(*s_id: int*, *s_value: int*) → int

Converts the angle of the arm.

Parameters

- **s_id** – the id of the arm
- **s_value** – the value of the arm

Returns
the angle of the arm

static _clamp_motor_value(*value: int*) → int

Limits (clamps) the motor value.

Parameters
value – the value

Returns
the clamped value of the motor between -100 and 100

create_receive_threading() → None

Creates the receive threading.

set_auto_report_state(*enable: bool*, *forever: bool = False*) → None

Sets the auto report state.

Parameters

- **enable** – if True, the auto report state is enabled
- **forever** – if True, the auto report state is permanent

set_beep(*on_time: int*) → None

Sets the beep (buzzer).

Parameters
on_time – the time the beep is on

set_pwm_servo(*servo_id: int*, *angle: int*) → None

Sets the PWM servo.

Parameters

- **servo_id** – the id of the servo between 1 and 4
- **angle** – the angle of the servo between 0 and 180

set_pwm_servo_all(*angle_s1: int*, *angle_s2: int*, *angle_s3: int*, *angle_s4: int*) → None

Sets the PWM signal for all servos. Angle values are between 0 and 180.

Parameters

- **angle_s1** – the angle of the first servo
- **angle_s2** – the angle of the second servo
- **angle_s3** – the angle of the third servo
- **angle_s4** – the angle of the fourth servo

set_colorful_lamps(*led_id: int*, *red: int*, *green: int*, *blue: int*) → None

Sets the colorful lamps. Color values between 0 and 255.

Parameters

- **led_id** – the id of the led
- **red** – the red value
- **green** – the green value
- **blue** – the blue value

set_colorful_effect(*effect: int*, *speed: int = 255*, *parm: int = 255*) → None

Sets the colorful effect.

Parameters

- **effect** – the effect between 0 and 6
- **speed** – the speed between 1 and 10
- **parm** – the parameter between 0 and 6

set_motor(*speed_1: int*, *speed_2: int*, *speed_3: int*, *speed_4: int*) → None

Send PWM pulse to each of the motors. Speeds are between -100 and 100.

Parameters

- **speed_1** – the speed of motor 1
- **speed_2** – the speed of motor 2
- **speed_3** – the speed of motor 3
- **speed_4** – the speed of motor 4

set_car_run(*state: int*, *speed: int*, *adjust: bool = False*) → None

Control the car to move forward, backward, left, right, etc.

Parameters

- **state** – the state of the car between 0 and 7
- **speed** – the speed of the car between -100 and 100
- **adjust** – if True, the gyroscope auxiliary motion direction is activated

set_car_motion(*v_x: float*, *v_y: float*, *v_z: float*) → None

Control the car movement.

Parameters

- **v_x** – the speed of the car in the x direction
- **v_y** – the speed of the car in the y direction
- **v_z** – the speed of the car in the z direction

set_pid_param(*kp: int*, *ki: int*, *kd: int*, *forever: bool = False*) → None

Sets the PID parameters.

Parameters

- **kp** – the proportional gain
- **ki** – the integral gain
- **kd** – the derivative gain
- **forever** – if True, the PID parameters are permanent

set_car_type(*car_type: int*) → None

Sets the car type.

Parameters

car_type – the type of the car

set_uart_servo(*servo_id: int*, *pulse_value: int*, *run_time: int = 500*) → None

Sets the position of one of the servos (on the arm).

Parameters

- **servo_id** – the id of the servo
- **pulse_value** – the position of the servo
- **run_time** – the running time in milliseconds

set_uart_servo_angle(*s_id: int | Arm*, *s_angle: int*, *run_time: int = 500*) → None

Sets the angle of one of the servos (on the arm). Degrees: 1-4, 6:[0, 180], 5:[0, 270]

1 - Rotation of the arm 2 - Shoulder of the arm 3 - Elbow of the arm 4 - Tilt of the arm 5 - Wrist of the arm (rotation) 6 - Fingers of the arm (pinch)

Parameters

- **s_id** – the id of the servo
- **s_angle** – the angle of the servo
- **run_time** – the running time in milliseconds

set_uart_servo_id(*servo_id: int*) → None

Sets the ID of the bus servo.

Parameters

servo_id – the id of the servo

set_uart_servo_torque(*enable: bool*) → None

Sets the torque of the bus servo.

Parameters

enable – if 1, the torque is enabled

set_uart_servo_ctrl_enable(*enable: bool*) → None

Sets the control switch of the manipulator.

Parameters

enable – if True, the control protocol is normally sent

set_uart_servo_angle_array(*angle_s: list[int] = [90, 90, 90, 90, 90, 180]*, *run_time: int = 500*) → None

Sets the angle of all the servos in the arm.

Parameters

- **angle_s** – the angle of the servos
- **run_time** – the running time in milliseconds

set_uart_servo_offset(*servo_id: int*) → int

Sets the offset of the servo.

Parameters

servo_id – the id of the servo

set_akm_default_angle(*angle: int*, *forever: bool = False*) → None

Sets the default angle of the front wheel of the ackerman type (R2) car.

Parameters

- **angle** – the angle between 60 and 120
- **forever** – if True, the angle is permanently saved

set_akm_steering_angle(*angle: int*, *ctrl_car: bool = False*) → None

Controls the steering angle of the ackerman type (R2) car.

Parameters

- **angle** – the angle between -45 and 45
- **ctrl_car** – if True, the steering gear angle and the speed of the left and
- **modified** (*right motors are*)

reset_flash_value() → None

Resets the car flash saved data, restore the factory default value.

reset_car_state() → None

Resets the car status, including parking, lights off, buzzer off.

clear_auto_report_data() → None

Clears the cache data automatically sent by the MCU.

get_akm_default_angle() → int

Reads the default angle of the front wheel of the ackerman type (R2) car.

get_uart_servo_value(*servo_id: int*) → tuple[int]

Reads the position parameters of the bus servo.

Parameters

servo_id – the id of the servo

get_uart_servo_angle(*s_id: int* | Arm) → int

Reads the angle of one of the servos (on the arm).

Parameters

s_id – the id of the servo

get_uart_servo_angle_array() → list[int]

Reads the angles of all the servos (on the arm).

get_accelerometer_data() → tuple[float]

Get the accelerometer triaxial data, return a_x, a_y, a_z.

get_gyroscope_data() → tuple[float]

Get the gyro triaxial data, return g_x, g_y, g_z.

get_magnetometer_data() → tuple[float]

Get the magnetometer triaxial data, return m_x, m_y, m_z.

get_imu_attitude_data(*to_angle: bool = True*) → tuple[float]

Get the board attitude Angle, return yaw, roll, pitch.

Parameters

to_angle – if True, return the Angle; if False, return the radian

get_motion_data() → tuple[float]

Get the car speed, val_vx, val_vy, val_vz.

get_battery_voltage() → float

Get the battery voltage.

get_motor_encoder() → tuple[int]

Obtain data of four-channel motor encoder.

get_motion_pid() → list[int]

Get the motion PID parameters of the dolly and return [kp, ki, kd]

get_car_type_from_machine() → int

Gets the current car type from machine.

get_version() → float

Get the underlying microcontroller version number.

CHAPTER

FIVE

AI_DETECTION

This node takes care of object detection and hosts a Flask server for the VR headset to fetch images from. The node also has access to the two cameras (drive and arm). It switches between these two depending on the current mode.

Subscribed topics

- switch_camera (SwitchCamera): Camera to switch to including information for what to show.

Published topics

- message (Message): Log information.

5.1 detection

class robot.src.ai_detection.ai_detection.detection.**ObjectDetection**(**args: Any*, ***kwargs: Any*)

ObjectDetection class for the object detection application

_switch_camera(*camera:* Camera) → None

Switches the camera to the specified camera.

Parameters
camera – Camera to switch to

handle_switch_camera(*msg*) → None

Handles switch camera message.

Parameters
msg – SwitchCamera message

save_result(*result: mediapipe.tasks.python.vision.ObjectDetectorResult*, *unused_output_image: mediapipe.Image*, *timestamp_ms: int*) → None

Save the detection result to the global variable

Parameters

- **result** (*vision.ObjectDetectorResult*) – The detection result
- **unused_output_image** (*mp.Image*) – The output image
- **timestamp_ms** (*int*) – The timestamp in milliseconds

inference(*class_name: str*, *max_results=5*, *score_threshold=0.8*) → None

Perform the object detection inference

Parameters

- **class_name** – The class name
- **max_results** – The maximum number of results
- **score_threshold** – The score threshold

_detect_objects(*frame*)

Detect objects in the frame

Parameters
frame – The input frame

Returns
The frame with the detected objects

__retreive_info(*object_name: str*) → dict

Retrieve all information for the object from the database

Parameters
object_name – Name of the object

Returns
Dictionary containing all information for the object

Return type
all_info_dict

websocket_handler(*ws*) → None

generate_frame() → str

Generate the frames for the object detection's snapshot.

robot.src.ai_detection.ai_detection.detection.**main**(*args=None*)

5.2 database

class robot.src.ai_detection.ai_detection.database.**Database**

get_item_info(*data: dict*) → dict

Finds the item in the database and returns the information

Parameters
data – dict to search for in the database

Returns
One item from the database or an empty dict

5.3 utils

class robot.src.ai_detection.ai_detection.utils.**Camera**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

DRIVE = '/dev/v4l/by-id/usb-Sonix_Technology_Co.__Ltd._Astra_Pro_HD_Camera-video-index0'

ARM = '/dev/v4l/by-id/usb-Sonix_Technology_Co.__Ltd._USB_2.0_Camera-video-index0'

robot.src.ai_detection.ai_detection.utils.**get_path**(*subpath: str*) → str

Get the path of the file.

Parameters
subpath – Subpath of the file

Returns
Path of the file

robot.src.ai_detection.ai_detection.utils.**get_config**() → dict

Get the configuration.

Returns
config dict

robot.src.ai_detection.ai_detection.utils.**visualize**(*image*, *detection_result*) → numpy.ndarray

Draws bounding boxes on the input image and return it. :param image: The input RGB image. :param detection_result: The list of all "Detection" entities to be visualized.

Returns
Image with bounding boxes. category_name: The category name of the detected object.

CHAPTER

SIX

DEPTH_DATA

This is the only node which interacts with the expansion board (robot). It is responsible for controlling the robot's movement and arm movements.

Subscribed topics

- /camera/depth/points (PointCloud2) - The depth data from the camera
- get_depth (GetDepth) - Enable the transfer of depth data for 1 image

Published topics

- depth_data (DepthData) - The compressed depth data from the camera

6.1 depth data

class robot.src.depth_data.depth_data.depth_data.**DepthDataNode**(**args: Any*, ***kwargs: Any*)

handle_get_depth(*msg*) → None

Enable the get_depth functionality.

Parameters
msg – GetDepth message

handle_depth_points(*msg*) → None

Get the depth data from the camera, compresses it and publishes the data.

Parameters
msg – PointCloud2 message

robot.src.depth_data.depth_data.depth_data.**main**(*args=None*)

6.2 utils

robot.src.depth_data.depth_data.utils.**_get_struct_fmt**(*is_bigendian: bool*, *fields*, *field_names: tuple[str] = None*)

Get struct format string from a list of PointFields.

Parameters

- **is_bigendian** – Whether the data is big-endian.
- **fields** – List of PointFields.

- **field_names** – List of field names to include in the format string.

Returns
Format string for struct.unpack from the given fields.

robot.src.depth_data.depth_data.utils.**read_points**(*cloud*, *field_names: tuple[str] = None*, *skip_nans: bool = False*, *uvs=[]*)

Read points from a L{sensor_msgs.PointCloud2} message.

Parameters

- **cloud** – The point cloud to read from.
- **field_names** – The names of fields to read. If None, read all fields.
- **skip_nans** – If True, then don't return any point with a NaN value.
- **uvs** – If specified, then only return the points at the given coordinates.

Returns
Generator which yields a list of values for each point.

CHAPTER

SEVEN

LOGGER

This node logs data to a .log file for easier debugging.

Subscribed topics

- arm_angles (ArmAngles): The arm angles for the robot which gets sent to the digital twin.
- _vr_dirve (VRDrive): The VR drive data received from the headset.
- _vr_arm (VRArm): The VR arm data received from the headset.
- _vr_mode (VRMode): The VR mode data received from the headset.
- switch_camera (SwitchCamera): The camera to switch to.
- robot_data (RobotData): Robot data (voltage, battery, etc.)
- message (Message): Message to be logged including level.

Published topics

- None

7.1 logger_node

`class robot.src.logger.logger.logger_node.LoggerNode(*args: Any, **kwargs: Any)`

Logs messages from nodes.

`robot.src.logger.logger.logger_node.main(args=None)`

7.2 logger

`robot.src.logger.logger.logger`

alias of <module 'robot.src.logger.logger.logger' from '/home/pi/robot/robot/src/logger/logger/logger.py'>

7.3 utils

class robot.src.logger.logger.utils.**FileFormatter**(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*, *, *defaults=None*)

_format = '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s'

FORMATS = {10: '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s', 20: '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s', 30: '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s', 40: '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s', 50: '%(asctime)s.%(msecs)03d - [%(levelname)s]: %(message)s'}

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

CHAPTER

EIGHT

ROBOT

Kromium's code which controls the robot and communicates with the VR headset.

8.1 Important information

- If you want to run any ROS2 node after making changes to any Python file inside the `robot/src` folder, you need to rebuild (`./colcon`).
- If a new Python package is added to `requirements.txt` you would need to run `./build_image` again.

8.1.1 Test physically (production)

Physically implies using the expansion board.

Make sure that `production` is set to `true` in the config.

```
# robot/
./run_docker_production
```

Enters the container as `root` to access the `/dev/ttyUSB0` (expansion board). This device is also passed into the container.

8.1.2 Not testing physically (not production)

Not using expansion board.

Make sure that `production` is set to `false` in the config.

```
# robot/
./run_docker
```

If you get an error when trying to `./run_all`, it might be because of non-root permissions. You can try to delete the `build`, `install` and `log` folders and run `./colcon` again.

8.1.3 Error: Access already in use when trying to start

Make sure the Meta Quest 3 headset has closed the app, and run `./kill_address`.

8.2 Prerequisites

8.2.1 Install Docker

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.
↪asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https:/
↪/download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
↪compose-plugin -y
```

8.2.2 Give permission to use Docker

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

8.3 Installation

8.3.1 Build the image

```
# robot/ (same directory as the Dockerfile)
./build_image
```

8.3.2 Install MongoDB

```
docker pull mongodb/mongodb-community-server:latest
```

Add data to the database.

8.3.3 Start the MongoDB container

```
docker run --name mongodb --network host -d mongodb/mongodb-community-server:latest
```

8.4 Running

```
# robot/
./run_docker # or ./run_docker_production starts the container
./colcon # builds the ROS2 packages
./run_all # starts all ROS2 nodes
```

First time building will lead to this error.

```
--- stderr: astra_camera
In file included from /usr/include/c++/11/string:40,
                 from /usr/include/c++/11/stdexcept:39,
                 from /usr/include/c++/11/system_error:41,
                 from /usr/include/c++/11/bits/fs_fwd.h:35,
                 from /usr/include/c++/11/filesystem:44,
                 from /robot/src/astra_camera/src/ob_camera_node_factory.cpp:13:
In static member function 'static constexpr std::size_t std::char_traits<char>
↪::length(const char_type*)',
    inlined from 'std::basic_ostream<char, _Traits>& std::operator<<(std::basic_ostream
↪<char, _Traits>&, const char*) [with _Traits = std::char_traits<char>]' at /usr/
↪include/c++/11/ostream:617:44,
    inlined from 'void astra_camera::OBCameraNodeFactory::onDeviceConnected(const␣
↪openni::DeviceInfo*)' at /robot/src/astra_camera/src/ob_camera_node_factory.cpp:118:5:
/usr/include/c++/11/bits/char_traits.h:399:32: warning: 'long unsigned int __builtin_
↪strlen(const char*)' reading 1 or more bytes from a region of size 0 [-Wstringop-
↪overread]
  399 |         return __builtin_strlen(__s);
      |                ~~~~~~~~~~~~~~~~^~~~
---
```

Running `./colcon` again will fix it.

8.5 Development

8.5.1 Start the container

```
# robot/
./run_docker
```

To exit the container use `CTRL + D`.

When the container is running the `robot/src` folder will be shared between the container and your local one. Changes made inside here will be apply to both. After changes has been made, make sure to run `./colcon` or `colcon build` inside the `robot/src` folder. Now you can run using `./run_all` (if the package is included inside that file) or by running:

```
# robot/src
source install/local_setup.bash
ros2 run package_name executable_variable
```

8.5.2 Testing

```
# robot/src
./run_tests
```

Every unittest should output OK.

8.5.3 Create a new ROS2 package

```
# robot/src
ros2 pkg create --build-type ament_python package_name
```

8.6 Documentation

8.6.1 HTML

```
# install sphinx
sudo apt-get install python3-sphinx -y
pip3 install furo --break-system-packages
cd docs/
make clean
make html
```

8.6.2 PDF

```
sudo apt install latexmk texlive-latex-extra
cd docs/
make latexpdf
```

PYTHON MODULE INDEX

r

robot.src.ai_detection.ai_detection.database, 32
robot.src.ai_detection.ai_detection.detection, 31
robot.src.ai_detection.ai_detection.utils, 32
robot.src.controller.controller.arm_kinematics, 20
robot.src.controller.controller.controller, 12
robot.src.controller.controller.controller_node, 11
robot.src.controller.controller.enums, 22
robot.src.controller.controller.exceptions, 21
robot.src.controller.controller.robot, 16
robot.src.controller.controller.rosmaster, 24
robot.src.controller.controller.utils, 23
robot.src.depth_data.depth_data.depth_data, 35
robot.src.depth_data.depth_data.utils, 35
robot.src.logger.logger.logger_node, 37
robot.src.logger.logger.utils, 38
robot.src.master.master.master_node, 9
robot.src.master.master.modes, 10
robot.src.vr_linker.vr_linker.utils, 7
robot.src.vr_linker.vr_linker.vr_linker, 4
robot.src.vr_linker.vr_linker.vr_linker_node, 3

INDEX

Symbols

__arm_convert_value() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 25
__del__() (*robot.src.controller.controller.robot.Robot method*), 19
__parse_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 24
__request_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 24
__retreive_info() (*robot.src.ai_detection.ai_detection.detection.ObjectDetection method*), 32
__uart_state (*robot.src.controller.controller.rosmaster.Rosmaster attribute*), 24
_arm_convert_angle() (*robot.src.controller.controller.rosmaster.Rosmaster static method*), 25
_clamp_motor_value() (*robot.src.controller.controller.rosmaster.Rosmaster static method*), 25
_convert_coordinates() (*robot.src.controller.controller.controller.Controller static method*), 13
_convert_coordinates_to_direction() (*robot.src.controller.controller.controller.Controller method*), 12
_convert_to_point() (*robot.src.controller.controller.controller.Controller static method*), 13
_convert_value_to_angle() (*robot.src.controller.controller.controller.Controller static method*), 13
_detect_objects() (*robot.src.ai_detection.ai_detection.detection.ObjectDetection method*), 32
_estimate_battery_percentage() (*robot.src.controller.controller.robot.Robot static method*), 19
_estimate_cms_speed() (*robot.src.controller.controller.robot.Robot static method*), 19
_format (*robot.src.logger.logger.utils.FileFormatter attribute*), 38
_get_content_any_msg_type() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 4
_get_precision_direction() (*robot.src.controller.controller.controller.Controller method*), 16
_get_reverse_direction() (*robot.src.controller.controller.controller.Controller static method*), 14
_get_struct_fmt() (*in module robot.src.depth_data.depth_data.utils*), 35
_get_vector_data() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 4
_is_backward() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_drive_mode() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 6
_is_forward() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_get_depth_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 6
_is_illegal_point() (*robot.src.controller.controller.controller.Controller method*), 13
_is_in_first_quadrant() (*robot.src.controller.controller.controller.Controller static method*), 14
_is_in_fourth_quadrant() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_in_second_quadrant() (*robot.src.controller.controller.controller.Controller static method*), 14
_is_in_third_quadrant() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_left() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_mode_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 5
_is_ping_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker*

static method), 6
_is_right() (*robot.src.controller.controller.controller.Controller static method*), 15
_is_screw_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 5
_is_vr_arm_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 5
_is_vr_connected() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 4
_is_vr_drive_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 5
_is_within_origin_buffer() (*robot.src.controller.controller.controller.Controller static method*), 14
_publish_data() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_publish_get_depth() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_publish_mode() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_publish_ping() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_publish_screw() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_publish_vr_arm() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 5
_publish_vr_drive() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 5
_reset_arm() (*robot.src.controller.controller.robot.Robot method*), 16
_screw() (*robot.src.controller.controller.robot.Robot method*), 18
_send() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 4
_send_compressed() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 4
_set_drive_mode() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 6
_set_last_arm_mode() (*robot.src.controller.controller.controller.Controller method*), 12
_set_mode_defaults() (*robot.src.controller.controller.controller.Controller method*), 12
_set_speed() (*robot.src.controller.controller.controller.Controller method*), 16
_speed_out_of_range() (*robot.src.controller.controller.controller.Controller static method*), 13
_switch_camera() (*robot.src.ai_detection.ai_detection.detection.ObjectD method*), 31
_switch_camera() (*robot.src.master.master.master_node.MasterNode method*), 9
_switch_to_forward_camera() (*robot.src.controller.controller.controller_node.ControllerNode method*), 11
_switch_to_reverse_camera() (*robot.src.controller.controller.controller_node.ControllerNode method*), 11
_to_json() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker static method*), 4
_would_collide_with_front() (*robot.src.controller.controller.controller.Controller static method*), 13

A

AnyMessage (*class in robot.src.vr_linker.vr_linker.vr_linker*), 4
Arm (*class in robot.src.controller.controller.enums*), 22
ARM (*robot.src.ai_detection.ai_detection.utils.Camera attribute*), 32
ARM (*robot.src.master.master.modes.Mode attribute*), 10
ARM_ELBOW_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22
ARM_ROTATION_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22
ARM_SHOULDER_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22
ARM_TILT_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22

B

BACKWARD (*robot.src.controller.controller.enums.Direction attribute*), 22
beep() (*robot.src.controller.controller.robot.Robot method*), 19

C

calculate() (*in module robot.src.controller.controller.arm_kinematics*), 21
calculate_auxilaries() (*in module robot.src.controller.controller.arm_kinematics*), 20
calculate_elbow_down_degrees() (*in module robot.src.controller.controller.arm_kinematics*), 20
calculate_elbow_up_degrees() (*in module robot.src.controller.controller.arm_kinematics*), 20
Camera (*class in robot.src.ai_detection.ai_detection.utils*), 32

check_last_message_received() (*robot.src.controller.controller.controller.Controller method*), 12
cleanup() (*robot.src.vr_linker.vr_linker.vr_linker_node.VRLinkerNode method*), 3
clear_auto_report_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
client_disconnected() (*robot.src.vr_linker.vr_linker.vr_linker_node.VRLinkerNode method*), 3
Controller (*class in robot.src.controller.controller.controller*), 12
ControllerException, 21
ControllerNode (*class in robot.src.controller.controller.controller_node*), 11
convert_pinch_to_angle() (*robot.src.controller.controller.controller.Controller static method*), 12
convert_thetas_to_servo_angles() (*in module robot.src.controller.controller.arm_kinematics*), 21
convert_x_to_angle_difference() (*robot.src.controller.controller.controller.Controller static method*), 12
convert_y_to_angle_difference() (*robot.src.controller.controller.controller.Controller static method*), 12
create_receive_threading() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 25

D

Database (*class in robot.src.ai_detection.ai_detection.database*), 32
DepthDataNode (*class in robot.src.depth_data.depth_data.depth_data*), 35
DIAGONAL_LEFT (*robot.src.controller.controller.enums.Direction attribute*), 22
DIAGONAL_RIGHT (*robot.src.controller.controller.enums.Direction attribute*), 22
Direction (*class in robot.src.controller.controller.enums*), 22
DRIVE (*robot.src.ai_detection.ai_detection.utils.Camera attribute*), 32
DRIVE (*robot.src.master.master.modes.Mode attribute*), 10
drive() (*robot.src.controller.controller.robot.Robot method*), 17

E

ELBOW (*robot.src.controller.controller.enums.Arm attribute*), 23
EMERGENCY (*robot.src.master.master.modes.Mode attribute*), 10

F

FileFormatter (*class in robot.src.logger.logger.utils*), 38
fill_vector_msg() (*in module robot.src.controller.controller.utils*), 23
format() (*robot.src.logger.logger.utils.FileFormatter method*), 38
FORMATS (*robot.src.logger.logger.utils.FileFormatter attribute*), 38
FORWARD (*robot.src.controller.controller.enums.Direction attribute*), 22
forward() (*robot.src.controller.controller.robot.Robot method*), 17

G

generate_frame() (*robot.src.ai_detection.ai_detection.detection.ObjectD method*), 32
get_accelerometer_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_akm_default_angle() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
get_arm_angles() (*robot.src.controller.controller.controller_node.Contro method*), 11
get_arm_elbow() (*robot.src.controller.controller.robot.Robot method*), 18
get_arm_rotation() (*robot.src.controller.controller.robot.Robot method*), 18
get_arm_shoulder() (*robot.src.controller.controller.robot.Robot method*), 18
get_arm_tilt() (*robot.src.controller.controller.robot.Robot method*), 19
get_arm_timeout_seconds() (*in module robot.src.vr_linker.vr_linker.utils*), 7
get_battery_voltage() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_car_type_from_machine() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_config() (*in module robot.src.ai_detection.ai_detection.utils*), 33
get_config() (*in module robot.src.controller.controller.utils*), 23
get_config() (*in module robot.src.vr_linker.vr_linker.utils*), 7
get_data() (*robot.src.controller.controller.robot.Robot method*), 19

get_data_hertz() (*in module robot.src.controller.controller.utils*), 24
get_direction() (*robot.src.controller.controller.robot.Robot method*), 17
get_gyroscope_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_imu_attitude_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_item_info() (*robot.src.ai_detection.ai_detection.database.Database method*), 32
get_magnetometer_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_mapped_phi() (*in module robot.src.controller.controller.arm_kinematics*), 21
get_motion_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_motion_pid() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_motor_encoder() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_path() (*in module robot.src.ai_detection.ai_detection.utils*), 33
get_production() (*in module robot.src.controller.controller.utils*), 24
get_robot_data() (*robot.src.controller.controller.controller_node.ControllerNode method*), 11
get_sleep_mode_after() (*in module robot.src.controller.controller.utils*), 24
get_sleep_mode_hertz() (*in module robot.src.controller.controller.utils*), 24
get_speed() (*robot.src.controller.controller.robot.Robot method*), 16
get_stored_angles() (*robot.src.controller.controller.robot.Robot method*), 16
get_threshold() (*in module robot.src.controller.controller.utils*), 24
get_uart_servo_angle() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
get_uart_servo_angle_array() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_uart_servo_value() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
get_version() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 29
get_wrist() (*robot.src.controller.controller.robot.Robot method*), 17

H

handle_arm_angles() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 5
handle_depth_data() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 5
handle_depth_points() (*robot.src.depth_data.depth_data.depth_data.DepthDataNode method*), 35
handle_get_depth() (*robot.src.depth_data.depth_data.depth_data.Depth method*), 35
handle_log_backup() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 4
handle_robot_data() (*robot.src.master.master.master_node.MasterNode method*), 10
handle_robot_data() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 4
handle_screw() (*robot.src.controller.controller.robot.Robot method*), 18
handle_switch_camera() (*robot.src.ai_detection.ai_detection.detection.ObjectDetection method*), 31
handle_unsafe_vr_arm() (*robot.src.master.master.master_node.MasterNode method*), 10
handle_unsafe_vr_drive() (*robot.src.master.master.master_node.MasterNode method*), 10
handle_unsafe_vr_mode() (*robot.src.master.master.master_node.MasterNode method*), 10
handle_unsafe_vr_screw() (*robot.src.master.master.master_node.MasterNode method*), 10
handle_vr_arm() (*robot.src.controller.controller.controller.Controller method*), 14
handle_vr_drive() (*robot.src.controller.controller.controller_node.Cont method*), 11
handle_vr_mode() (*robot.src.controller.controller.controller.Controller method*), 16
handle_vr_screw() (*robot.src.controller.controller.controller.Controller method*), 14
has_any_nan() (*in module robot.src.controller.controller.arm_kinematics*), 20
has_out_of_range() (*in module robot.src.controller.controller.arm_kinematics*),

20

I

IDLE (*robot.src.master.master.modes.Mode attribute*), 10
in_production_mode() (*in module robot.src.controller.controller.utils*), 23
inference() (*robot.src.ai_detection.ai_detection.detection.ObjectDetection method*), 31
int_to_mode() (*robot.src.master.master.master_node.MasterNode method*), 9

L

LATERAL_LEFT (*robot.src.controller.controller.enums.Direction attribute*), 22
LATERAL_RIGHT (*robot.src.controller.controller.enums.Direction attribute*), 22
listen() (*robot.src.vr_linker.vr_linker.vr_linker_node.VRLinkerNode method*), 3
logger (*in module robot.src.logger.logger*), 37
LoggerNode (*class in robot.src.logger.logger.logger_node*), 37
long_beep() (*robot.src.controller.controller.robot.Robot method*), 19

M

main() (*in module robot.src.ai_detection.ai_detection.detection*), 32
main() (*in module robot.src.controller.controller.controller_node*), 12
main() (*in module robot.src.depth_data.depth_data.depth_data*), 35
main() (*in module robot.src.logger.logger.logger_node*), 37
main() (*in module robot.src.master.master.master_node*), 10
main() (*in module robot.src.vr_linker.vr_linker.vr_linker_node*), 3
MasterNode (*class in robot.src.master.master.master_node*), 9
matches_mode_and_not_emergency() (*in module robot.src.master.master.master_node*), 9
Mode (*class in robot.src.master.master.modes*), 10
module
    robot.src.ai_detection.ai_detection.database, 32
    robot.src.ai_detection.ai_detection.detection, 31
    robot.src.ai_detection.ai_detection.utils, 32
    robot.src.controller.controller.arm_kinematics, 20
    robot.src.controller.controller.controller, 12
    robot.src.controller.controller.controller_node, 11
    robot.src.controller.controller.enums, 22
    robot.src.controller.controller.exceptions, 21
    robot.src.controller.controller.robot, 16
    robot.src.controller.controller.rosmaster, 24
    robot.src.controller.controller.utils, 23
    robot.src.depth_data.depth_data.depth_data, 35
    robot.src.depth_data.depth_data.utils, 35
    robot.src.logger.logger.logger_node, 37
    robot.src.logger.logger.utils, 38
    robot.src.master.master.master_node, 9
    robot.src.master.master.modes, 10
    robot.src.vr_linker.vr_linker.utils, 7
    robot.src.vr_linker.vr_linker.vr_linker, 4
    robot.src.vr_linker.vr_linker.vr_linker_node, 3

N

NotInProductionMode, 21

O

ObjectDetection (*class in robot.src.ai_detection.ai_detection.detection*), 31

P

PINCH (*robot.src.controller.controller.enums.Arm attribute*), 23
pinch() (*robot.src.controller.controller.robot.Robot method*), 17
PINCH_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22
Preset (*class in robot.src.controller.controller.enums*), 22
process_message() (*robot.src.vr_linker.vr_linker.vr_linker.VRLinker method*), 5

R

read_points() (*in module robot.src.depth_data.depth_data.utils*), 36
reset() (*robot.src.controller.controller.robot.Robot method*), 16
reset_arm_elbow() (*robot.src.controller.controller.robot.Robot method*), 19
reset_arm_rotation() (*robot.src.controller.controller.robot.Robot method*), 18

reset_arm_shoulder() (*robot.src.controller.controller.robot.Robot method*), 18
reset_arm_tilt() (*robot.src.controller.controller.robot.Robot method*), 19
reset_car_state() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
reset_flash_value() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
reset_wrist() (*robot.src.controller.controller.robot.Robot method*), 18
REVERSE_DIAGONAL_LEFT (*robot.src.controller.controller.enums.Direction attribute*), 22
REVERSE_DIAGONAL_RIGHT (*robot.src.controller.controller.enums.Direction attribute*), 22
REVERSE_TURN_LEFT (*robot.src.controller.controller.enums.Direction attribute*), 22
REVERSE_TURN_RIGHT (*robot.src.controller.controller.enums.Direction attribute*), 22
Robot (*class in robot.src.controller.controller.robot*), 16
robot.src.ai_detection.ai_detection.database module, 32
robot.src.ai_detection.ai_detection.detection module, 31
robot.src.ai_detection.ai_detection.utils module, 32
robot.src.controller.controller.arm_kinematics module, 20
robot.src.controller.controller.controller module, 12
robot.src.controller.controller.controller_node module, 11
robot.src.controller.controller.enums module, 22
robot.src.controller.controller.exceptions module, 21
robot.src.controller.controller.robot module, 16
robot.src.controller.controller.rosmaster module, 24
robot.src.controller.controller.utils module, 23
robot.src.depth_data.depth_data.depth_data module, 35
robot.src.depth_data.depth_data.utils module, 35
robot.src.logger.logger.logger_node module, 37
robot.src.logger.logger.utils module, 38
robot.src.master.master.master_node module, 9
robot.src.master.master.modes module, 10
robot.src.vr_linker.vr_linker.utils module, 7
robot.src.vr_linker.vr_linker.vr_linker module, 4
robot.src.vr_linker.vr_linker.vr_linker_node module, 3
Rosmaster (*class in robot.src.controller.controller.rosmaster*), 24
ROTATION (*robot.src.controller.controller.enums.Arm attribute*), 22

S

save_result() (*robot.src.ai_detection.ai_detection.detection.ObjectDetec method*), 31
set_akm_default_angle() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
set_akm_steering_angle() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
set_arm_elbow() (*robot.src.controller.controller.robot.Robot method*), 19
set_arm_rotation() (*robot.src.controller.controller.robot.Robot method*), 18
set_arm_rotation_difference() (*robot.src.controller.controller.robot.Robot method*), 18
set_arm_shoulder() (*robot.src.controller.controller.robot.Robot method*), 18
set_arm_tilt() (*robot.src.controller.controller.robot.Robot method*), 19
set_auto_report_state() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 25
set_beep() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 25
set_car_motion() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_car_run() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_car_type() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_colorful_effect() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_colorful_lamps() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_data() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 24

set_last_message() (*in module robot.src.controller.controller.utils*), 23
set_mode() (*robot.src.master.master.master_node.MasterNode method*), 9
set_motor() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_pid_param() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_pinch() (*robot.src.controller.controller.robot.Robot method*), 17
set_pwm_servo() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 25
set_pwm_servo_all() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 26
set_speed() (*robot.src.controller.controller.robot.Robot method*), 17
set_uart_servo() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_uart_servo_angle() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_uart_servo_angle_array() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
set_uart_servo_ctrl_enable() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_uart_servo_id() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_uart_servo_offset() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 28
set_uart_servo_torque() (*robot.src.controller.controller.rosmaster.Rosmaster method*), 27
set_wrist() (*robot.src.controller.controller.robot.Robot method*), 17
SHOULDER (*robot.src.controller.controller.enums.Arm attribute*), 23
SPEED (*robot.src.controller.controller.enums.Preset attribute*), 22
STOP (*robot.src.controller.controller.enums.Direction attribute*), 22
stop() (*robot.src.controller.controller.robot.Robot method*), 17

T

TILT (*robot.src.controller.controller.enums.Arm attribute*), 23
TURN_LEFT (*robot.src.controller.controller.enums.Direction attribute*), 22
TURN_RIGHT (*robot.src.controller.controller.enums.Direction attribute*), 22

U

unpinch() (*robot.src.controller.controller.robot.Robot method*), 17
UNPINCH_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22

V

visualize() (*in module robot.src.ai_detection.ai_detection.utils*), 33
vr_drive() (*robot.src.controller.controller.controller.Controller method*), 16
VRLinker (*class in robot.src.vr_linker.vr_linker.vr_linker*), 4
VRLinkerNode (*class in robot.src.vr_linker.vr_linker.vr_linker_node*), 3

W

websocket_handler() (*robot.src.ai_detection.ai_detection.detection.ObjectDetection method*), 32
WRIST (*robot.src.controller.controller.enums.Arm attribute*), 23
WRIST_ANGLE (*robot.src.controller.controller.enums.Preset attribute*), 22

V Object-detection code documentation

object-detection

Release 1.0.0

Kromium

Mar 01, 2024

CONTENTS:

1 objdetection_vr_2 **1**

2 object-detection **3**

2.1 Object_Detection on Raspberry pi 5 3

2.2 Pre-trained model directory 3

2.3 Custom-transfer learning 3

2.4 Documentation 4

Python Module Index **5**

Index **7**

CHAPTER

ONE

OBJDETECTION_VR_2

custom_transfer_learning.objdetection_vr_2.**load_labels**(*label_path*)

Loads the labels file. Supports files with or without index numbers.

If the file contains index numbers, then the index number is removed from the label.

Parameters
label_path – path to the labels file.

Returns
A list with the labels.

custom_transfer_learning.objdetection_vr_2.**getCaps**()

custom_transfer_learning.objdetection_vr_2.**draw_boxes**(*frame*, *num_detections*, *boxes*, *classes*, *scores*, *labels*, *threshold=0.4*)

Draw bounding boxes on the frame.

Parameters

- **frame** – the frame to draw on.
- **num_detections** – the number of detections.
- **boxes** – the bounding boxes.
- **classes** – the class of the detected object.
- **scores** – the confidence scores of the detected object.
- **labels** – the labels of the detected object.
- **threshold** – the confidence threshold to use.

custom_transfer_learning.objdetection_vr_2.**getFrame**()

Get a frame from the camera.

Returns
A tuple with a boolean indicating if the frame was captured successfully and the frame.

custom_transfer_learning.objdetection_vr_2.**process_frame_for_detection**(*frame=None*)

Process a single frame for object detection and return the frame.

Parameters
frame – the frame to process.

Returns
The frame with bounding boxes drawn around the detected objects.

custom_transfer_learning.objdetection_vr_2.**snapshot**()

Get a snapshot from the camera when a GET request is received.

Returns

A response with the snapshot.

CHAPTER

TWO

OBJECT-DETECTION

2.1 Object_Detection on Raspberry pi 5

To deploy this on the raspberry pi: 1.Create a virtual environment:

```
python -m venv myenv
source myenv/bin/activate
```

2. download requirements:

```
pip install -r requirements.txt
```

3. go to either pre-trained model or custom-transfer learning directory using cd
4. Run test.py

2.2 Pre-trained model directory

The code uses the COCO dataset and a pre-trained model called mobilenet to detect objects from the dataset. Link for the coco dataset: https://cocodataset.org/#home Link for the Mobilenet tflite model : https://www.kaggle.com/models/iree/ssd-mobilenet-v2 you can also stream the object detection, which the VR headset displays.

2.3 Custom-transfer learning

Here, I followed the tutorial for transfer learning on a custom dataset given my Tensorflow : https://www.tensorflow.org/lite/models/modify/model_maker/object_detection In the directory tflite_models there are several Tensorflow lite models to choose from. The best one for now is the people_Detection_2 one. The datasets I trained on : https://www.kaggle.com/datasets/sbaghbidi/human-faces-object-detection?rvi=1

'Open Images Dataset V7'. Accessed: Feb. 23, 2024. [Online]. Available: https://storage.googleapis.com/openimages/web/visualizer/index.html?type=detection&set=train&c=%2Fm%2F02rdsp

2.4 Documentation

2.4.1 Build HTML

```
# install sphinx
sudo apt-get install python3-sphinx
pip3 install furo sphinxcontrib-jquery --break-system-packages
cd docs/
make clean
make html
```

2.4.2 Make PDF

```
sudo apt install latexmk texlive-latex-extra
cd docs/
make latexpdf
```

PYTHON MODULE INDEX

C

custom_transfer_learning.objdetection_vr_2, 1

INDEX

C

custom_transfer_learning.objdetection_vr_2
    module, 1

D

draw_boxes() (*in module custom_transfer_learning.objdetection_vr_2*), 1

G

getCaps() (*in module custom_transfer_learning.objdetection_vr_2*), 1
getFrame() (*in module custom_transfer_learning.objdetection_vr_2*), 1

L

load_labels() (*in module custom_transfer_learning.objdetection_vr_2*), 1

M

module
    custom_transfer_learning.objdetection_vr_2, 1

P

process_frame_for_detection() (*in module custom_transfer_learning.objdetection_vr_2*), 1

S

snapshot() (*in module custom_transfer_learning.objdetection_vr_2*), 1

W Transfer learning training code documentation

transfer-learning-training

Release 2.0.0

Kromium

May 21, 2024

CONTENTS:

CHAPTER

ONE

MODEL_MAKER_TRAINING

class Model_maker_training.**DataLoader**(*csv_file_path*, *images_dir*)

load_data()

Loads data from the csv file. :param csv_file_path: The file path to the csv file. :param images_dir: The directory that contains images.

Returns: A Dataloader containing the data of training, validation and test set

class Model_maker_training.**ModelTrainer**(*train_data*, *validation_data*)

A class to train the model on the given dataset and evaluate it.

train_data

Training dataset, in the form of Dataloader.

validation_data

Validation dataset, in the form of Dataloader.

eval

The evaluation result.

model

The trained model.

train_model()

Fine tunes the EfficientDet-Lite0 model on the given dataset.

Parameters

- **train_data** – Training dataset, in the form of Dataloader.
- **validation_data** – Validation dataset, in the form of Dataloader.

Returns
A trained model.

evaluate_and_export(*test_data*)

Evaluates the model and exports it to the export directory.

Parameters
test_data – Test dataset, in the form of Dataloader.

class Model_maker_training.**ObjectDetector**(*model*, *model_path*, *classes*)

A class to perform object detection with a given model.

model

The trained model.

model_path
The file path to the trained model.

classes
A list of class labels.

colors
A list of colors for visualization.

interpreter
The TensorFlow Lite interpreter.

preprocess_image(*image_path*, *input_size*)

detect_objects(*image*, *threshold=0.5*)

run_odt_and_draw_results(*image_path*, *interpreter*, *threshold=0.5*)
Run object detection on the input image and draw the results.

Parameters

- **image_path** – The file path to the input image.
- **interpreter** – The TensorFlow Lite interpreter.
- **threshold** – The minimum confidence score for detected objects.

Returns
A NumPy array of the input image with the detection results.

CHAPTER

TWO

TEST_WITH_CAMERA

test_with_camera.**load_labels**(*path*)

Load labels from text file

Parameters
path – Path to text file containing labels

Returns
List of labels

Return type
labels

class test_with_camera.**Interpreter**(*model_path*)

Load TFLite model and allocate tensors

model_path

Path to TFLite model

interpreter

TFLite interpreter

get_interpreter()

Get TFLite interpreter

Returns
TFLite interpreter

Return type
interpreter

get_details() → tuple

Get input and output details

Returns
Input details output_details: Output details height: Height of input tensor width: Width of input tensor

Return type
input_details

class test_with_camera.**Detection**(*frame*, *height*, *width*, *interpreter*, *input_detail*, *output_detail*)

Detect objects in frame using TFLite model

frame

Input frame

height
Frame height

width
Frame width

interpreter
TFLite interpreter

input_detail
Input details

output_detail
Output details

scores
Confidence scores

boxes
Bounding box coordinates

classes
Class indices

num_detections
Number of detections

nms(*boxes*, *scores*, *iou_threshold*)
Non-maximum suppression

Parameters

- **boxes** – Bounding box coordinates
- **scores** – Confidence scores
- **iou_threshold** – IoU threshold

Returns
Indices of boxes to keep

Return type
keep

interpret()
Interpret the frame and make predictions

Parameters

- **frame** – Input frame
- **height** – Frame height
- **width** – Frame width
- **interpreter** – TFLite interpreter
- **input_details** – Input details
- **output_details** – Output details

Returns
Number of detections scores: Confidence scores boxes: Bounding box coordinates classes: Class indices

Return type
num_detections

`classFilter`(*classdata*) → list

Filter classes

Parameters
`classdata` – Class data

Returns
Filtered classes

`YOLOdetect`(*output_data*) → tuple

Detect objects using YOLO

Parameters
`output_data` – Output data

Returns
Bounding box coordinates, class indices, and confidence scores

`make_boxes`(*scores*, *xyxy*)

Draw bounding boxes on frame

Parameters

- **`frame`** – Input frame
- **`scores`** – Confidence scores
- **`xyxy`** – Bounding box coordinates

Returns
Frame with bounding boxes

`make_boxes_2`()

Draw bounding boxes on frame version 2

Returns
Frame with bounding boxes

X VR headset code documentation

Kromium virtual office

1.0

Generated by Doxygen 1.9.1

1 Kromium VR headset 1

1.1 Getting started . . . 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Class Documentation 9

5.1 ArmController Class Reference . . . 9

5.1.1 Detailed Description . . . 11

5.1.2 Member Function Documentation . . . 11

5.1.2.1 CalculateDistances() . . . 11

5.1.2.2 CalculateNormalizedControlValues() . . . 12

5.1.2.3 CalculateSpeed() . . . 12

5.1.2.4 HandleReceivedData() . . . 13

5.1.2.5 OnTriggerEnter() . . . 13

5.1.2.6 OnTriggerExit() . . . 14

5.1.2.7 OnTriggerStay() . . . 14

5.1.2.8 RepeatedlyDistanceCalculation() . . . 14

5.1.2.9 ResetControlValues() . . . 15

5.1.2.10 SendControlValues() . . . 15

5.1.2.11 SendDataToServer() . . . 16

5.1.2.12 Start() . . . 17

5.1.2.13 Update() . . . 17

5.1.2.14 UpdateVisualIndicator() . . . 17

5.1.3 Member Data Documentation . . . 18

5.1.3.1 controlValues . . . 18

5.1.3.2 controlX . . . 18

5.1.3.3 cubeMaterial . . . 18

5.1.3.4 distanceCalculationInterval . . . 18

5.1.3.5 dropdownHandler . . . 19

5.1.3.6 handSkeleton . . . 19

5.1.3.7 insideColor . . . 19

5.1.3.8 isHandDetected . . . 19

5.1.3.9 lastSendTime . . . 19

5.1.3.10 networkManager . . . 20

5.1.3.11 originalColor . . . 20

5.1.3.12 rightHand . . . 20

5.1.3.13 sendInterval . . . 20

5.1.3.14 visualIndicatorTransform . . . 20
5.2 CameraDepthData Class Reference . . . 21
5.2.1 Detailed Description . . . 21
5.2.2 Member Data Documentation . . . 21
5.2.2.1 depthData . . . 21
5.2.2.2 time . . . 21
5.3 DebugDisplay Class Reference . . . 22
5.3.1 Detailed Description . . . 23
5.3.2 Member Function Documentation . . . 23
5.3.2.1 HandleLog() . . . 23
5.3.2.2 OnDisable() . . . 23
5.3.2.3 OnEnable() . . . 24
5.3.2.4 Update() . . . 24
5.3.3 Member Data Documentation . . . 24
5.3.3.1 debugLogs . . . 24
5.3.3.2 debugText . . . 24
5.4 DebugDisplayPro Class Reference . . . 25
5.4.1 Detailed Description . . . 26
5.4.2 Member Function Documentation . . . 26
5.4.2.1 HandleLog() . . . 26
5.4.2.2 OnDisable() . . . 27
5.4.2.3 OnEnable() . . . 27
5.4.2.4 Update() . . . 27
5.4.3 Member Data Documentation . . . 27
5.4.3.1 debugLogs . . . 28
5.4.3.2 debugText . . . 28
5.5 DepthData Class Reference . . . 28
5.5.1 Detailed Description . . . 28
5.5.2 Member Function Documentation . . . 29
5.5.2.1 AddDepthDataPoint() . . . 29
5.5.2.2 ClearPoints() . . . 29
5.5.3 Member Data Documentation . . . 29
5.5.3.1 depthDataPoints . . . 29
5.5.3.2 time . . . 29
5.5.4 Property Documentation . . . 29
5.5.4.1 Count . . . 30
5.5.4.2 Points . . . 30
5.6 DepthDataGenerator Class Reference . . . 30
5.6.1 Detailed Description . . . 31
5.6.2 Member Function Documentation . . . 31
5.6.2.1 GenerateDepthData() . . . 31
5.6.3 Member Data Documentation . . . 31

5.6.3.1 Height . . . 31
5.6.3.2 MaxDepth . . . 32
5.6.3.3 meshCreator . . . 32
5.6.3.4 MinDepth . . . 32
5.6.3.5 Width . . . 32
5.7 DepthDataPoint Class Reference . . . 32
5.7.1 Detailed Description . . . 33
5.7.2 Constructor & Destructor Documentation . . . 33
5.7.2.1 DepthDataPoint() . . . 33
5.7.3 Property Documentation . . . 33
5.7.3.1 X . . . 33
5.7.3.2 Y . . . 33
5.7.3.3 Z . . . 34
5.8 DepthPointCloud Class Reference . . . 34
5.8.1 Detailed Description . . . 35
5.8.2 Member Function Documentation . . . 35
5.8.2.1 DecodeBase64() . . . 35
5.8.2.2 DecompressBrotli() . . . 36
5.8.2.3 OnDepthDataReceived() . . . 36
5.8.2.4 OnDestroy() . . . 36
5.8.2.5 ProcessDepthData() . . . 36
5.8.2.6 RequestDepthData() . . . 37
5.8.2.7 Start() . . . 37
5.8.2.8 Update() . . . 37
5.8.3 Member Data Documentation . . . 37
5.8.3.1 brotlifile . . . 37
5.8.3.2 depthColorGradient . . . 38
5.8.3.3 depthData . . . 38
5.8.3.4 meshCreator . . . 38
5.8.3.5 networkManager . . . 38
5.8.3.6 particles . . . 38
5.8.3.7 particleSystem . . . 38
5.9 DigitalTwinController Class Reference . . . 39
5.9.1 Detailed Description . . . 40
5.9.2 Member Function Documentation . . . 40
5.9.2.1 HandleReceivedRobotInfoData() . . . 40
5.9.2.2 SetTargetRotation() . . . 41
5.9.2.3 SetTargetRotationY() . . . 41
5.9.2.4 Start() . . . 42
5.9.2.5 testFunction() . . . 42
5.9.2.6 Update() . . . 42
5.9.3 Member Data Documentation . . . 43

5.9.3.1 frameCounter . . . 43
5.9.3.2 Link_1 . . . 43
5.9.3.3 Link_2 . . . 43
5.9.3.4 Link_3 . . . 43
5.9.3.5 Link_4 . . . 43
5.9.3.6 Link_5 . . . 44
5.9.3.7 networkManager . . . 44
5.9.3.8 Pinch_left_x . . . 44
5.9.3.9 Pinch_right_y . . . 44
5.9.3.10 rotateTo170 . . . 44
5.10 DriveModeController.DriveMode Class Reference . . . 45
5.10.1 Detailed Description . . . 45
5.10.2 Member Data Documentation . . . 45
5.10.2.1 drive_mode . . . 45
5.11 DriveModeController Class Reference . . . 45
5.11.1 Detailed Description . . . 47
5.11.2 Member Function Documentation . . . 47
5.11.2.1 normalMode() . . . 47
5.11.2.2 precisionMode() . . . 47
5.11.2.3 reverseMode() . . . 47
5.11.2.4 SendDataToServer() . . . 47
5.11.2.5 Start() . . . 48
5.11.2.6 Update() . . . 48
5.11.2.7 updateButtonColor() . . . 48
5.11.2.8 updateDriveModeData() . . . 49
5.11.3 Member Data Documentation . . . 49
5.11.3.1 activeColor . . . 49
5.11.3.2 defaultColor . . . 49
5.11.3.3 networkManager . . . 49
5.11.3.4 normalButton . . . 50
5.11.3.5 precisionButton . . . 50
5.11.3.6 reverseButton . . . 50
5.11.4 Event Documentation . . . 50
5.11.4.1 DriveModeChanged . . . 50
5.12 DropdownHandler Class Reference . . . 51
5.12.1 Detailed Description . . . 52
5.12.2 Member Function Documentation . . . 52
5.12.2.1 DropdownValueChanged() . . . 53
5.12.2.2 GetDropdownValue() . . . 53
5.12.2.3 HandleVoiceCommand() . . . 53
5.12.2.4 SendDataToServer() . . . 53
5.12.2.5 Start() . . . 54

5.12.3 Member Data Documentation . . . 54
5.12.3.1 dropdown . . . 54
5.12.3.2 modeValues . . . 54
5.12.3.3 networkManager . . . 55
5.12.3.4 responsehandler . . . 55
5.12.4 Event Documentation . . . 55
5.12.4.1 OnDropdownValueChanged . . . 55
5.13 HandGestureAndRotation.HandData Class Reference . . . 55
5.13.1 Detailed Description . . . 55
5.13.2 Member Data Documentation . . . 56
5.13.2.1 pinch . . . 56
5.13.2.2 wrist . . . 56
5.14 HandDataTransmission Class Reference . . . 56
5.14.1 Detailed Description . . . 57
5.14.2 Member Function Documentation . . . 57
5.14.2.1 Start() . . . 57
5.14.2.2 Update() . . . 57
5.15 HandDetectionCube Class Reference . . . 58
5.15.1 Detailed Description . . . 59
5.15.2 Member Function Documentation . . . 60
5.15.2.1 CalculateDistances() . . . 60
5.15.2.2 CalculateNormalizedControlValues() . . . 60
5.15.2.3 CalculateSpeed() . . . 61
5.15.2.4 HandleReceivedData() . . . 61
5.15.2.5 OnDestroy() . . . 62
5.15.2.6 OnTriggerEnter() . . . 62
5.15.2.7 OnTriggerExit() . . . 62
5.15.2.8 OnTriggerStay() . . . 63
5.15.2.9 RepeatedlyDistanceCalculation() . . . 63
5.15.2.10 ResetControlValues() . . . 64
5.15.2.11 SendControlValues() . . . 64
5.15.2.12 SendDataToServer() . . . 64
5.15.2.13 Start() . . . 65
5.15.2.14 Update() . . . 65
5.15.2.15 UpdateVisualIndicator() . . . 66
5.15.3 Member Data Documentation . . . 67
5.15.3.1 controlValues . . . 67
5.15.3.2 controlX . . . 67
5.15.3.3 cubeMaterial . . . 67
5.15.3.4 distanceCalculationInterval . . . 68
5.15.3.5 dropdownHandler . . . 68
5.15.3.6 handSkeleton . . . 68

5.15.3.7 insideColor . . . 68
5.15.3.8 isHandDetected . . . 68
5.15.3.9 lastSendTime . . . 69
5.15.3.10 networkManager . . . 69
5.15.3.11 originalColor . . . 69
5.15.3.12 rightHand . . . 69
5.15.3.13 sendInterval . . . 69
5.15.3.14 visualIndicatorTransform . . . 70
5.16 HandGestureAndRotation Class Reference . . . 70
5.16.1 Detailed Description . . . 71
5.16.2 Member Function Documentation . . . 71
5.16.2.1 SendDataToServer() . . . 71
5.16.2.2 Start() . . . 72
5.16.2.3 Update() . . . 72
5.16.3 Member Data Documentation . . . 73
5.16.3.1 hand . . . 73
5.16.3.2 handData . . . 73
5.16.3.3 handSkeleton . . . 73
5.16.3.4 isTesting . . . 74
5.16.3.5 lastSendTime . . . 74
5.16.3.6 networkManager . . . 74
5.16.3.7 sendInterval . . . 74
5.17 HandInteraction Class Reference . . . 75
5.17.1 Detailed Description . . . 76
5.17.2 Member Function Documentation . . . 76
5.17.2.1 HandleReceivedData() . . . 76
5.17.2.2 OnDestroy() . . . 77
5.17.2.3 SendDataToServer() . . . 77
5.17.2.4 Start() . . . 77
5.17.2.5 Update() . . . 78
5.17.3 Member Data Documentation . . . 78
5.17.3.1 hand . . . 78
5.17.3.2 isTesting . . . 78
5.17.3.3 lastSendTime . . . 79
5.17.3.4 networkManager . . . 79
5.17.3.5 sendInterval . . . 79
5.18 HandleReconnectButton Class Reference . . . 79
5.18.1 Detailed Description . . . 80
5.18.2 Member Function Documentation . . . 81
5.18.2.1 OnReconnectButtonClicked() . . . 81
5.18.2.2 Start() . . . 81
5.18.2.3 Update() . . . 81

5.18.3 Member Data Documentation . . . 81
5.18.3.1 networkManager . . . 82
5.18.3.2 reconnectButton . . . 82
5.19 Heap_up_controller Class Reference . . . 82
5.19.1 Detailed Description . . . 84
5.19.2 Member Function Documentation . . . 84
5.19.2.1 HandleConnectionStatusChanged() . . . 84
5.19.2.2 HandleReceivedPingData() . . . 84
5.19.2.3 HandleReceivedRobotInfoData() . . . 85
5.19.2.4 OnDestroy() . . . 86
5.19.2.5 OnDisable() . . . 86
5.19.2.6 OnEnable() . . . 86
5.19.2.7 Start() . . . 86
5.19.2.8 Update() . . . 86
5.19.2.9 update_head_up_canvas_position_and_rotation() . . . 87
5.19.3 Member Data Documentation . . . 87
5.19.3.1 accelerometer . . . 87
5.19.3.2 battery_precentage . . . 87
5.19.3.3 battery_precentage_text . . . 87
5.19.3.4 cms_speed . . . 88
5.19.3.5 conenctionStatus_text . . . 88
5.19.3.6 connectedColor . . . 88
5.19.3.7 disconnectedColor . . . 88
5.19.3.8 gyroscope . . . 88
5.19.3.9 head_up_canvas . . . 88
5.19.3.10 head_up_canvas_distance . . . 89
5.19.3.11 latecy_text . . . 89
5.19.3.12 magnetometer . . . 89
5.19.3.13 mode . . . 89
5.19.3.14 mode_text . . . 89
5.19.3.15 motion . . . 89
5.19.3.16 networkManager . . . 90
5.19.3.17 offset_ovr_camera_rig . . . 90
5.19.3.18 ovr_camera_rig . . . 90
5.19.3.19 robot_view_plane . . . 90
5.19.3.20 speed . . . 90
5.19.3.21 speed_text . . . 90
5.19.3.22 voltage . . . 91
5.19.3.23 voltage_text . . . 91
5.20 HelloWorldScript Class Reference . . . 91
5.20.1 Detailed Description . . . 92
5.20.2 Member Function Documentation . . . 92

5.20.2.1 Start() . . . 92
5.20.3 Member Data Documentation . . . 92
5.20.3.1 myName . . . 93
5.20.3.2 textMeshPro . . . 93
5.21 JsonArmLengthInfo Class Reference . . . 93
5.21.1 Detailed Description . . . 93
5.21.2 Member Data Documentation . . . 93
5.21.2.1 Link_1 . . . 93
5.21.2.2 Link_2 . . . 94
5.21.2.3 Link_3 . . . 94
5.21.2.4 Link_4 . . . 94
5.21.2.5 Link_5 . . . 94
5.21.2.6 pintch . . . 94
5.22 JsonRobotInfo Class Reference . . . 94
5.22.1 Detailed Description . . . 95
5.22.2 Member Data Documentation . . . 95
5.22.2.1 accelerometer . . . 95
5.22.2.2 battery_precentage . . . 95
5.22.2.3 cms_speed . . . 95
5.22.2.4 gyroscope . . . 96
5.22.2.5 magnetometer . . . 96
5.22.2.6 mode . . . 96
5.22.2.7 motion . . . 96
5.22.2.8 speed . . . 96
5.22.2.9 voltage . . . 96
5.23 Logger Class Reference . . . 97
5.23.1 Detailed Description . . . 98
5.23.2 Member Function Documentation . . . 98
5.23.2.1 Awake() . . . 98
5.23.2.2 Log() . . . 98
5.23.3 Member Data Documentation . . . 98
5.23.3.1 logFilePath . . . 99
5.24 MainController Class Reference . . . 99
5.24.1 Detailed Description . . . 100
5.24.2 Member Function Documentation . . . 100
5.24.2.1 HandleDropdownChange() . . . 100
5.24.2.2 HandleEmergencyStop() . . . 101
5.24.2.3 OnDestroy() . . . 101
5.24.2.4 Start() . . . 102
5.24.3 Member Data Documentation . . . 102
5.24.3.1 armScene . . . 102
5.24.3.2 driveScene . . . 102

5.24.3.3 dropdown . . . 102
5.24.3.4 dropdownHandler . . . 103
5.24.3.5 EmergencyStopButton . . . 103
5.24.3.6 modeAudioPlay . . . 103
5.25 MeshCreator Class Reference . . . 103
5.25.1 Detailed Description . . . 104
5.25.2 Member Function Documentation . . . 104
5.25.2.1 clearVolumeBoxes() . . . 104
5.25.2.2 CreateCubeGridTest() . . . 105
5.25.2.3 CreateMeshInBoxVolume() . . . 105
5.25.2.4 NormalizeDepthData() . . . 106
5.25.2.5 Start() . . . 106
5.25.2.6 Update() . . . 107
5.25.3 Member Data Documentation . . . 107
5.25.3.1 depthColorGradient . . . 107
5.25.3.2 volume . . . 107
5.26 MeshReplacement Class Reference . . . 107
5.26.1 Detailed Description . . . 108
5.26.2 Member Function Documentation . . . 108
5.26.2.1 Start() . . . 108
5.26.2.2 Update() . . . 108
5.26.3 Member Data Documentation . . . 109
5.26.3.1 mesh . . . 109
5.26.3.2 ply . . . 109
5.27 ModeAudioPlay Class Reference . . . 109
5.27.1 Detailed Description . . . 110
5.27.2 Member Function Documentation . . . 110
5.27.2.1 PlayArm() . . . 110
5.27.2.2 PlayDrive() . . . 110
5.27.2.3 PlayEmergency() . . . 111
5.27.2.4 PlayIdle() . . . 111
5.27.2.5 PlayScrew() . . . 111
5.27.2.6 PlayUnScrew() . . . 111
5.27.2.7 Start() . . . 111
5.27.2.8 Update() . . . 112
5.27.3 Member Data Documentation . . . 112
5.27.3.1 armSource . . . 112
5.27.3.2 driveSource . . . 112
5.27.3.3 emergencySource . . . 112
5.27.3.4 idleSource . . . 112
5.27.3.5 screwSource . . . 113
5.27.3.6 unScrewSource . . . 113

5.28 DropdownHandler.ModeValues Class Reference . . . 113
5.28.1 Detailed Description . . . 113
5.28.2 Member Data Documentation . . . 113
5.28.2.1 mode . . . 113
5.29 NetworkManager Class Reference . . . 114
5.29.1 Detailed Description . . . 116
5.29.2 Member Function Documentation . . . 116
5.29.2.1 Awake() . . . 117
5.29.2.2 ConnectToServer() . . . 117
5.29.2.3 DecodeBase64() . . . 117
5.29.2.4 DecompressBrotli() . . . 118
5.29.2.5 Disconnect() . . . 118
5.29.2.6 InvokeConnectionStatus() . . . 119
5.29.2.7 IsBrotli() . . . 119
5.29.2.8 OnApplicationQuit() . . . 120
5.29.2.9 OnDestroy() . . . 120
5.29.2.10 ParseArmLengthInfo() . . . 120
5.29.2.11 ParseFloatList() . . . 121
5.29.2.12 ParseJsonRobotInfo() . . . 121
5.29.2.13 PingRobot() . . . 121
5.29.2.14 ProcessDataCoroutine() . . . 122
5.29.2.15 ProcessDepthData() . . . 122
5.29.2.16 ProcessPingData() . . . 123
5.29.2.17 processRecievedData() . . . 123
5.29.2.18 ReceiveData() . . . 124
5.29.2.19 ReceiveDataAsync() . . . 125
5.29.2.20 Reconnect() . . . 125
5.29.2.21 SendData() . . . 125
5.29.2.22 Start() . . . 126
5.29.2.23 Update() . . . 126
5.29.3 Member Data Documentation . . . 127
5.29.3.1 broHeader . . . 127
5.29.3.2 brotliTestFile . . . 127
5.29.3.3 cameraDepthData . . . 127
5.29.3.4 client . . . 127
5.29.3.5 connected . . . 127
5.29.3.6 depthData . . . 128
5.29.3.7 Instance . . . 128
5.29.3.8 isListening . . . 128
5.29.3.9 port . . . 128
5.29.3.10 receivedDataQueue . . . 129
5.29.3.11 receiveThread . . . 129

5.29.3.12 serverIP . . . 129
5.29.3.13 sizeBeforeUpdate . . . 129
5.29.3.14 stream . . . 129
5.29.3.15 udpPort . . . 130
5.29.4 Event Documentation . . . 130
5.29.4.1 onArmLengthDataReceived . . . 130
5.29.4.2 OnConnected . . . 130
5.29.4.3 OnConnectionStatus . . . 130
5.29.4.4 OnDataReceived . . . 130
5.29.4.5 onDepthDataReceived . . . 131
5.29.4.6 onPingDataReceived . . . 131
5.29.4.7 OnRobotInfoDataReceived . . . 131
5.30 ObjectController Class Reference . . . 131
5.30.1 Detailed Description . . . 132
5.30.2 Member Function Documentation . . . 132
5.30.2.1 SendDataToServer() . . . 133
5.30.2.2 Start() . . . 133
5.30.2.3 Update() . . . 133
5.30.3 Member Data Documentation . . . 133
5.30.3.1 networkManager . . . 134
5.31 PingData Class Reference . . . 134
5.31.1 Detailed Description . . . 134
5.31.2 Member Data Documentation . . . 134
5.31.2.1 ping . . . 134
5.31.2.2 type . . . 134
5.32 DepthPointCloud.RequestDepthDataMsg Class Reference . . . 135
5.32.1 Detailed Description . . . 135
5.32.2 Member Data Documentation . . . 135
5.32.2.1 get_depth . . . 135
5.33 Responsehandler Class Reference . . . 135
5.33.1 Detailed Description . . . 137
5.33.2 Member Function Documentation . . . 137
5.33.2.1 HandleResponseTest() . . . 137
5.33.2.2 OnEmergency() . . . 137
5.33.2.3 OnEmergencyVoiceCommandActivated() . . . 137
5.33.2.4 OnStartListening() . . . 138
5.33.2.5 sendScrewCommandToTheTobot() . . . 138
5.33.2.6 Start() . . . 138
5.33.2.7 Update() . . . 139
5.33.3 Member Data Documentation . . . 139
5.33.3.1 index . . . 139
5.33.3.2 modeAudioPlay . . . 139

5.33.3.3 networkManager . . . 139
5.33.3.4 voiceExperience . . . 139
5.33.4 Event Documentation . . . 139
5.33.4.1 OnVoiceCommandReceived . . . 140
5.34 RobotCamStream Class Reference . . . 140
5.34.1 Detailed Description . . . 141
5.34.2 Member Function Documentation . . . 141
5.34.2.1 HandlePrepareCompleted() . . . 141
5.34.2.2 HandleVideoError() . . . 142
5.34.2.3 SetupVideoPlayer() . . . 142
5.34.2.4 Start() . . . 142
5.34.2.5 TryPrepareVideo() . . . 143
5.34.3 Member Data Documentation . . . 143
5.34.3.1 currentRetries . . . 143
5.34.3.2 maxRetries . . . 143
5.34.3.3 retryDelay . . . 143
5.34.3.4 streamUrl . . . 143
5.34.3.5 videoPlayer . . . 144
5.35 ArmController.RobotControlValues Class Reference . . . 144
5.35.1 Detailed Description . . . 144
5.35.2 Member Data Documentation . . . 144
5.35.2.1 speed . . . 144
5.35.2.2 x . . . 145
5.35.2.3 y . . . 145
5.36 HandDetectionCube.RobotControlValues Class Reference . . . 145
5.36.1 Detailed Description . . . 145
5.36.2 Member Data Documentation . . . 145
5.36.2.1 speed . . . 146
5.36.2.2 x . . . 146
5.36.2.3 y . . . 146
5.37 ArmController.RobotControlX Class Reference . . . 146
5.37.1 Detailed Description . . . 146
5.37.2 Member Data Documentation . . . 147
5.37.2.1 pinch . . . 147
5.37.2.2 strength . . . 147
5.37.2.3 x . . . 147
5.37.2.4 y . . . 147
5.37.2.5 z . . . 147
5.38 HandDetectionCube.RobotControlX Class Reference . . . 148
5.38.1 Detailed Description . . . 148
5.38.2 Member Data Documentation . . . 148
5.38.2.1 pinch . . . 148

5.38.2.2 strength . . . 148
5.38.2.3 x . . . 148
5.38.2.4 y . . . 149
5.39 Responsehandler.ScrewCommand Class Reference . . . 149
5.39.1 Detailed Description . . . 149
5.39.2 Member Data Documentation . . . 149
5.39.2.1 screw . . . 149
5.40 SetAttachTransform Class Reference . . . 150
5.40.1 Detailed Description . . . 151
5.40.2 Member Function Documentation . . . 151
5.40.2.1 Start() . . . 151
5.40.3 Member Data Documentation . . . 151
5.40.3.1 handSkeleton . . . 152
5.40.3.2 poke_finger_tip_id . . . 152
5.40.3.3 pokeInteractor . . . 152
5.41 UdpListener Class Reference . . . 152
5.41.1 Detailed Description . . . 153
5.41.2 Member Function Documentation . . . 153
5.41.2.1 Main() . . . 153
5.41.2.2 StartListener() . . . 154
5.41.3 Member Data Documentation . . . 154
5.41.3.1 listenPort . . . 154
5.42 UnityMainThreadDispatcher Class Reference . . . 154
5.42.1 Detailed Description . . . 155
5.42.2 Member Function Documentation . . . 155
5.42.2.1 Awake() . . . 156
5.42.2.2 Enqueue() . . . 156
5.42.2.3 Update() . . . 156
5.42.3 Member Data Documentation . . . 156
5.42.3.1 _executionQueue . . . 156
5.42.4 Property Documentation . . . 157
5.42.4.1 Instance . . . 157
5.43 Vector3Data Struct Reference . . . 157
5.43.1 Detailed Description . . . 157
5.43.2 Constructor & Destructor Documentation . . . 157
5.43.2.1 Vector3Data() . . . 158
5.43.3 Member Data Documentation . . . 158
5.43.3.1 x . . . 158
5.43.3.2 y . . . 158
5.43.3.3 z . . . 158
5.44 VideoStream Class Reference . . . 159
5.44.1 Detailed Description . . . 159

5.44.2 Member Function Documentation . . . 159
5.44.2.1 Start() . . . 160
5.44.2.2 Update() . . . 160
5.45 VideoStreamController Class Reference . . . 160
5.45.1 Detailed Description . . . 161
5.45.2 Member Function Documentation . . . 161
5.45.2.1 Start() . . . 161
5.45.3 Member Data Documentation . . . 161
5.45.3.1 videoPlayer . . . 161
5.46 VideoStreamReceiver Class Reference . . . 162
5.46.1 Detailed Description . . . 162
5.47 WebcamStreamDisplay Class Reference . . . 163
5.47.1 Detailed Description . . . 164
5.47.2 Member Function Documentation . . . 164
5.47.2.1 ApplyTexture() . . . 164
5.47.2.2 FetchSnapshotRoutine() . . . 164
5.47.2.3 OnDestroy() . . . 164
5.47.2.4 Start() . . . 165
5.47.3 Member Data Documentation . . . 165
5.47.3.1 _refreshWait . . . 165
5.47.3.2 _renderer . . . 165
5.47.3.3 refreshRate . . . 165
5.47.3.4 snapshotUrl . . . 165
5.48 WebRTCReceiver Class Reference . . . 166
5.48.1 Detailed Description . . . 167
5.48.2 Member Function Documentation . . . 167
5.48.2.1 OnDestroy() . . . 167
5.48.2.2 OnError() . . . 167
5.48.2.3 OnMessage() . . . 168
5.48.2.4 Start() . . . 168
5.48.2.5 Update() . . . 168
5.48.3 Member Data Documentation . . . 168
5.48.3.1 _webSocket . . . 169
5.48.3.2 index . . . 169
5.48.3.3 networkManager . . . 169
5.48.3.4 rawImage . . . 169
5.48.3.5 texture . . . 169
5.49 WristMarker Class Reference . . . 170
5.49.1 Detailed Description . . . 171
5.49.2 Member Function Documentation . . . 171
5.49.2.1 Start() . . . 171
5.49.2.2 Update() . . . 171

5.49.3 Member Data Documentation . . . 171
5.49.3.1 greenDot . . . 172
5.49.3.2 Hand_MiddleTip . . . 172
5.49.3.3 skeleton . . . 172

6 File Documentation **173**

6.1 Assets/DebugDisplay.cs File Reference . . . 173
6.2 Assets/DebugDisplayPro.cs File Reference . . . 173
6.3 Assets/HandDataTransmission.cs File Reference . . . 173
6.4 Assets/HandInteraction.cs File Reference . . . 173
6.5 Assets/HelloWorldScript.cs File Reference . . . 174
6.6 Assets/ObjectController.cs File Reference . . . 174
6.7 Assets/RobotCamStream.cs File Reference . . . 174
6.8 Assets/Scripts/ArmController/ArmController.cs File Reference . . . 174
6.9 Assets/Scripts/DigitalTwin/DigitalTwinController.cs File Reference . . . 174
6.10 Assets/Scripts/DriveModeController/DriveModeController.cs File Reference . . . 175
6.11 Assets/Scripts/DropdownHandler.cs File Reference . . . 175
6.12 Assets/Scripts/HandDetectionCube.cs File Reference . . . 175
6.13 Assets/Scripts/HandGestureAndRotation.cs File Reference . . . 175
6.14 Assets/Scripts/HandleReconnectButton.cs File Reference . . . 175
6.15 Assets/Scripts/Head up display/Heap_up_controller.cs File Reference . . . 176
6.16 Assets/Scripts/log/Logger.cs File Reference . . . 176
6.17 Assets/Scripts/LogicController/MainController.cs File Reference . . . 176
6.18 Assets/Scripts/NetworkNamager/NetworkManager.cs File Reference . . . 176
6.19 Assets/Scripts/NetworkNamager/UdpListener.cs File Reference . . . 176
6.20 Assets/Scripts/PlotCameraDepthData/DepthDataGenerator.cs File Reference . . . 177
6.21 Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs File Reference . . . 177
6.22 Assets/Scripts/PlotCameraDepthData/MeshCreator.cs File Reference . . . 177
6.23 Assets/Scripts/PlotCameraDepthData/MeshReplacement.cs File Reference . . . 177
6.24 Assets/Scripts/SetAttachTransform.cs File Reference . . . 177
6.25 Assets/Scripts/UnityMainThreadDispatcher.cs File Reference . . . 177
6.26 Assets/Scripts/VideoStreamer/VideoStream.cs File Reference . . . 178
6.27 Assets/Scripts/VideoStreamer/VideoStreamReceiver.cs File Reference . . . 178
6.28 Assets/Scripts/WebRTCReceiver.cs File Reference . . . 178
6.29 Assets/Scripts/WristMarker.cs File Reference . . . 178
6.30 Assets/VideoStreamController.cs File Reference . . . 178
6.31 Assets/Voice Controll/ModeAudioPlay.cs File Reference . . . 178
6.32 Assets/Voice Controll/Responsehandler.cs File Reference . . . 179
6.33 Assets/WebcamStreamDisplay.cs File Reference . . . 179
6.34 obj/Debug/.NETStandard,Version=v2.1.AssemblyAttributes.cs File Reference . . . 179
6.35 README_.md File Reference . . . 179

Index **181**

Chapter 1

Kromium VR headset

Welcome to the Kromium VR Headset repository! This repository contains all the necessary files to build and modify the Kromium VR Headset application.

1.1 Getting started

- Set up development environment and headset
- Get started with Meta Quest 3 development in Unity

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

CameraDepthData . . . 21
DepthData . . . 28
DepthDataGenerator . . . 30
DepthDataPoint . . . 32
DriveModeController.DriveMode . . . 45
HandGestureAndRotation.HandData . . . 55
JsonArmLengthInfo . . . 93
JsonRobotInfo . . . 94
DropdownHandler.ModeValues . . . 113
MonoBehaviour
- ArmController . . . 9
- DebugDisplay . . . 22
- DebugDisplayPro . . . 25
- DepthPointCloud . . . 34
- DigitalTwinController . . . 39
- DriveModeController . . . 45
- DropdownHandler . . . 51
- HandDataTransmission . . . 56
- HandDetectionCube . . . 58
- HandGestureAndRotation . . . 70
- HandInteraction . . . 75
- HandleReconnectButton . . . 79
- Heap_up_controller . . . 82
- HelloWorldScript . . . 91
- Logger . . . 97
- MainController . . . 99
- MeshCreator . . . 103
- MeshReplacement . . . 107
- ModeAudioPlay . . . 109
- NetworkManager . . . 114
- ObjectController . . . 131
- Responsehandler . . . 135
- RobotCamStream . . . 140
- SetAttachTransform . . . 150
- UdpListener . . . 152

UnityMainThreadDispatcher 154
VideoStream 159
VideoStreamController 160
VideoStreamReceiver 162
WebRTCReceiver 166
WebcamStreamDisplay 163
WristMarker 170
PingData 134
DepthPointCloud.RequestDepthDataMsg 135
ArmController.RobotControlValues 144
HandDetectionCube.RobotControlValues 145
ArmController.RobotControlX 146
HandDetectionCube.RobotControlX 148
Responsehandler.ScrewCommand 149
Vector3Data 157

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ArmController
: This script is used to detect the hand position within a cube and send control values to the robot based on the hand position . . . 9

CameraDepthData
: depth camera interface datatype The data is recieved from the robot as this type . . . 21

DebugDisplay
: This script is used to display debug logs on the a VR screen for testing purposes . . . 22

DebugDisplayPro
: This script is used to display debug logs on the a VR screen for testing purposes . . . 25

DepthData . . . 28

DepthDataGenerator . . . 30

DepthDataPoint
: Depth data interface Datatype . . . 32

DepthPointCloud . . . 34

DigitalTwinController
: This class is used to control the digital twin robot . . . 39

DriveModeController.DriveMode . . . 45

DriveModeController
: This class is used to control the drive mode of the robot . . . 45

DropdownHandler
: This script is used to handle the dropdown in the UI . . . 51

HandGestureAndRotation.HandData
: The HandData interface used to send the data to the robot . . . 55

HandDataTransmission
: This script is used for testing porpuses . . . 56

HandDetectionCube
: This script is used to detect the hand position within a cube and send control values to the robot based on the hand position . . . 58

HandGestureAndRotation
: This class is used to get the hand gesture and rotation data and send it to the server . . . 70

HandInteraction
: This script is used to send hand tracking data to the server at regular intervals . . . 75

HandleReconnectButton
: This script is used to handle the reconnect button in the UI . . . 79

Heap_up_controller
This class is used to control the head up display of the robot . . . 82
HelloWorldScript
This script is used to display a simple "Hello World" message on the screen for testing purposes 91
JsonArmLengthInfo . . . 93
JsonRobotInfo
Class JsonRobotInfo Represents the JSON data structure for the robot information . . . 94
Logger
This script is used to log messages to a file . . . 97
MainController
This class is the main controller for handling scene changes and mode changes . . . 99
MeshCreator . . . 103
MeshReplacement . . . 107
ModeAudioPlay . . . 109
DropdownHandler.ModeValues
The ModeValues interface used to send the mode value to the robot . . . 113
NetworkManager
Class NetworkManager Manages network communications for the application, implementing a singleton pattern to ensure only one instance exists . . . 114
ObjectController
This class is used to send data to the server by pressing the space key . . . 131
PingData . . . 134
DepthPointCloud.RequestDepthDataMsg
Interface for requesting depth camera from the robot . . . 135
Responsehandler . . . 135
RobotCamStream
140
ArmController.RobotControlValues
The control values used to control the robot for controlling the robot (car) . . . 144
HandDetectionCube.RobotControlValues
The control values used to control the robot for controlling the robot (car) . . . 145
ArmController.RobotControlX
The control values used to control the robot for controlling the arm . . . 146
HandDetectionCube.RobotControlX
The control values used to control the robot for controlling the arm . . . 148
Responsehandler.ScrewCommand . . . 149
SetAttachTransform
This script is used to set the attach transform of an XR poke interactor to a specific bone in the hand skeleton . . . 150
UdpListener . . . 152
UnityMainThreadDispatcher . . . 154
Vector3Data
Vector data to store tdepth data (x, y, z) . . . 157
VideoStream . . . 159
VideoStreamController
This script is used to play a video stream from a server . . . 160
VideoStreamReceiver . . . 162
WebcamStreamDisplay . . . 163
WebRTCReceiver . . . 166
WristMarker
This class is used to display a green dot that is attached to the Tip of the Middle Finger of the Hand . . . 170

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Assets/DebugDisplay.cs . . . 173
Assets/DebugDisplayPro.cs . . . 173
Assets/HandDataTransmission.cs . . . 173
Assets/HandInteraction.cs . . . 173
Assets/HelloWorldScript.cs . . . 174
Assets/ObjectController.cs . . . 174
Assets/RobotCamStream.cs . . . 174
Assets/VideoStreamController.cs . . . 178
Assets/WebcamStreamDisplay.cs . . . 179
Assets/Scripts/DropdownHandler.cs . . . 175
Assets/Scripts/HandDetectionCube.cs . . . 175
Assets/Scripts/HandGestureAndRotation.cs . . . 175
Assets/Scripts/HandleReconnectButton.cs . . . 175
Assets/Scripts/SetAttachTransform.cs . . . 177
Assets/Scripts/UnityMainThreadDispatcher.cs . . . 177
Assets/Scripts/WebRTCReceiver.cs . . . 178
Assets/Scripts/WristMarker.cs . . . 178
Assets/Scripts/ArmController/ArmController.cs . . . 174
Assets/Scripts/DigitalTwin/DigitalTwinController.cs . . . 174
Assets/Scripts/DriveModeController/DriveModeController.cs . . . 175
Assets/Scripts/Head up display/Heap_up_controller.cs . . . 176
Assets/Scripts/log/Logger.cs . . . 176
Assets/Scripts/LogicController/MainController.cs . . . 176
Assets/Scripts/NetworkNamager/NetworkManager.cs . . . 176
Assets/Scripts/NetworkNamager/UdpListener.cs . . . 176
Assets/Scripts/PlotCameraDepthData/DepthDataGenerator.cs . . . 177
Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs . . . 177
Assets/Scripts/PlotCameraDepthData/MeshCreator.cs . . . 177
Assets/Scripts/PlotCameraDepthData/MeshReplacement.cs . . . 177
Assets/Scripts/VideoStreamer/VideoStream.cs . . . 178
Assets/Scripts/VideoStreamer/VideoStreamReceiver.cs . . . 178
Assets/Voice Controll/ModeAudioPlay.cs . . . 178
Assets/Voice Controll/Responsehandler.cs . . . 179
obj/Debug/.NETStandard,Version=v2.1.AssemblyAttributes.cs . . . 179

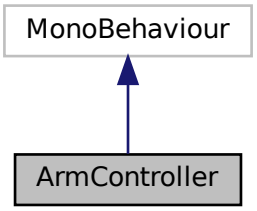
Chapter 5

Class Documentation

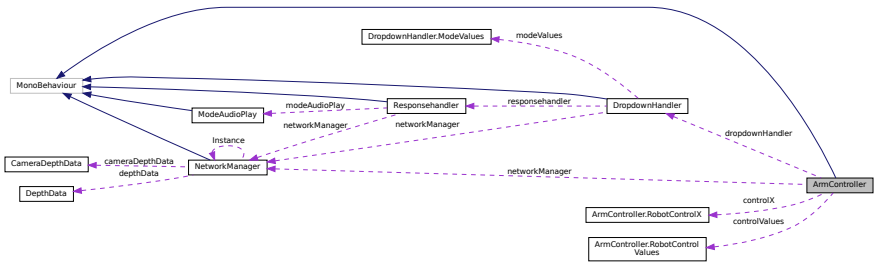
5.1 ArmController Class Reference

This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.

Inheritance diagram for ArmController:



Collaboration diagram for ArmController:



Classes

- class RobotControlValues

  The control values used to control the robot for controlling the robot (car).
- class RobotControlX

  The control values used to control the robot for controlling the arm.

Public Attributes

- DropdownHandler dropdownHandler

  The DropdownHandler component used to get the mode values from the UI(VR).
- OVRSkeleton handSkeleton

  The OVRSkeleton component used to track hand gestures.
- OVRHand rightHand

  The OVRHand component used to track hand gestures.
- float distanceCalculationInterval = 0.5f

  The interval at which to calculate the distances.
- float sendInterval = 0.5f

  The interval at which to send data to the server.
- Color insideColor = new Color(1, 0, 0, 0.2f)

Private Member Functions

- void Start ()

  The script is called before the first frame update and is used to initialize the necessary variables.
- void Update ()

  Update is called once per frame and is used to log the position of the Hand_WristRoot bone in the VR headset.
- void OnTriggerEnter (Collider other)

  This method is called when the hand enters the cube area.
- void OnTriggerExit (Collider other)

  Stop the coroutine when the hand leaves the cube
- void OnTriggerStay (Collider other)
- void ResetControlValues ()

  Reset the control values to stop the robot, this makes the robot stop when the hand leaves the cube
- IEnumerator RepeatedlyDistanceCalculation (Transform handTransform)

  Asynchronous method to calculate the distances repeatedly when the coroutine is started.
- void CalculateDistances (Vector3 handPosition)

  Calculate the distances from the hand to the edges of the cube This is used for debugging and understanding the hand position within the cube
- void CalculateNormalizedControlValues (Vector3 handPosition)

  Calculate the normalized control values based on the hand position within the cube
- void UpdateVisualIndicator (float normalizedX, float normalizedZ, float normalizedY)

  Update the position of the visual indicator within the detection cube according to the normalized X and Z values.
- void SendControlValues (float normalizedX, float normalizedY, float normalizedZ)

  Sending the information to the robot through socket communication.
- int CalculateSpeed (float normalizedX, float normalizedZ)

  Calculate the speed based on the normalized X and Z values
- void HandleReceivedData (string data)

  Handles the recieved data from the server.
- void SendDataToServer (string data)

  Sends the data to the server.

Private Attributes

- RobotControlValues controlValues = new RobotControlValues()
- RobotControlX controlX = new RobotControlX()
- NetworkManager networkManager

  The NetworkManager component used to send data to the server.
- bool isHandDetected = false

  The flag to indicate if the hand is detected within the cube.
- float lastSendTime
- Transform visualIndicatorTransform

  The transform of the visual indicator.
- Material cubeMaterial
- Color originalColor

5.1.1 Detailed Description

This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.

This class is used for both controlling the arm. This class will be refactored to be more modular and to use interfaces which implemented as state machines.

Definition at line 15 of file ArmController.cs.

5.1.2 Member Function Documentation

5.1.2.1 CalculateDistances()

```
void ArmController.CalculateDistances (
            Vector3 handPosition )  [inline], [private]
```

Calculate the distances from the hand to the edges of the cube This is used for debugging and understanding the hand position within the cube

Parameters

handPosition	

Definition at line 206 of file ArmController.cs.

```
207     {
208         float halfScaleX = transform.localScale.x / 2;
209         float halfScaleZ = transform.localScale.z / 2;
210 
211         float distanceToLeftEdge = handPosition.x - (transform.position.x - halfScaleX);
212         float distanceToRightEdge = (transform.position.x + halfScaleX) - handPosition.x;
213         float distanceToFrontEdge = (transform.position.z + halfScaleZ) - handPosition.z;
214         float distanceToBackEdge = handPosition.z - (transform.position.z - halfScaleZ);
215 
216         // Log distances to the console
217         Debug.Log($"Distance to Left Edge: {distanceToLeftEdge}");
218         Debug.Log($"Distance to Right Edge: {distanceToRightEdge}");
219         Debug.Log($"Distance to Front Edge: {distanceToFrontEdge}");
```

```
220         Debug.Log($"Distance to Back Edge: {distanceToBackEdge}");
221     }
```

5.1.2.2 CalculateNormalizedControlValues()

```
void ArmController.CalculateNormalizedControlValues (
            Vector3 handPosition )  [inline], [private]
```

Calculate the normalized control values based on the hand position within the cube

Parameters

handPosition	

Definition at line 227 of file ArmController.cs.

```
228     {
229         // Convert hand position to the cube's local space
230         Vector3 handLocalPosition = transform.InverseTransformPoint(handPosition);
231 
232         // Calculate normalized X within -1 to 1 range
233         float normalizedX = Mathf.Clamp(handLocalPosition.x / (transform.localScale.x / 2), -1, 1);
234 
235         // Calculate normalized Z within 0 to 1 range
236         float normalizedZ = Mathf.Clamp((handLocalPosition.z / (transform.localScale.z / 2) + 1) / 2, 0,
      1);
237 
238         float normalizedY = handLocalPosition.y + 0.50000f;
239         float mappedY;
240 
241         if (normalizedY < 0.3)
242         {
243             mappedY = normalizedY - 0.3f;
244         }
245         else
246         {
247             // Range which comes inside here is 0,3 between and 1 is mapped
248             mappedY = (normalizedY - 0.3f) / (1 - 0.3f);
249         }
250 
251         normalizedY = mappedY;
252         // Debug.Log($"HandLocalPosition: {handLocalPosition.y}");
253         // Debug.Log($"Transform localScale: {transform.localScale.y}");
254         // Debug.Log($"Normalized X: {normalizedX}");
255         // Debug.Log($"Normalized Z: {normalizedZ}");
256         // Debug.Log($"Normalized Y: {normalizedY}");
257 
258         SendControlValues(normalizedX, normalizedY, normalizedZ);
259         // UpdateVisualIndicator(normalizedX, normalizedZ, normalizedY);
260     }
```

5.1.2.3 CalculateSpeed()

```
int ArmController.CalculateSpeed (
            float normalizedX,
            float normalizedZ )  [inline], [private]
```

Calculate the speed based on the normalized X and Z values

Parameters

normalizedX	
normalizedZ	

Returns

Definition at line 367 of file ArmController.cs.

```
368     {
369         float calculatedSpeed = 0;
370         if (normalizedX < 0.3 && normalizedX > -0.3 && normalizedZ < 0.3)
371         {
372             calculatedSpeed = 0;
373         }
374         else if (normalizedX > 0.3 || normalizedX < -0.3)
375         {
376             calculatedSpeed = Math.Abs(100 * normalizedX);
377         }
378         else if (normalizedZ > 0.4)
379         {
380             calculatedSpeed = Math.Abs(100 * normalizedZ);
381         }
382         return (int)calculatedSpeed;
383     }
```

5.1.2.4 HandleReceivedData()

```
void ArmController.HandleReceivedData (
            string data )  [inline], [private]
```

Handles the recieved data from the server.

It is not in use at the moment.

Definition at line 389 of file ArmController.cs.

```
390     {
391         // Process the received data
392         Debug.Log($"ObjectController received data: {data}");
393     }
```

5.1.2.5 OnTriggerEnter()

```
void ArmController.OnTriggerEnter (
            Collider other )  [inline], [private]
```

This method is called when the hand enters the cube area.

It changes the color of the cube and starts the coroutine to calculate the distances repeatedly.

Parameters

other	

Definition at line 128 of file ArmController.cs.

```
129     {
130         // Check if TipBoneEnd is in the cube
131         if (other.CompareTag("TipboneSphere"))
132         {
133             isHandDetected = true;
134             // Debug.Log($"Visualindicator entered cube area.{other.bounds.size}");
135             // Debug.Log($"Other size: {other.bounds.size}");
```

```
136
137             // Change the color of the cube when the hand enters the cube
138             cubeMaterial.color = insideColor;
139
140             // Start coroutine to calculate distances repeatedly
141             int Hand_WristRoot = (int)OVRPlugin.BoneId.Hand_MiddleTip;
142             OVRBone WristBone = handSkeleton.Bones[Hand_WristRoot];
143             StartCoroutine(RepeatedlyDistanceCalculation(WristBone.Transform));
144         }
145     }
```

5.1.2.6 OnTriggerExit()

```
void ArmController.OnTriggerExit (
            Collider other )  [inline], [private]
```

Stop the coroutine when the hand leaves the cube

Parameters

other	

Definition at line 151 of file ArmController.cs.

```
152     {
153         if (other.CompareTag("TipboneSphere"))
154         {
155             isHandDetected = false;
156             // Debug.Log("Hand exited cube area.");
157             // Reset the color of the cube when the hand leaves the cube
158             cubeMaterial.color = originalColor;
159
160             // Stop the coroutine when the hand leaves the Cude
161             StopAllCoroutines();
162
163             // Reset the control values to stop the robot
164             ResetControlValues();
165
166             // Stop the robot when the had leaves the control area(Cube)
167             string json = JsonUtility.ToJson(controlValues);
168             SendDataToServer(json);
169         }
170     }
```

5.1.2.7 OnTriggerStay()

```
void ArmController.OnTriggerStay (
            Collider other )  [inline], [private]
```

Definition at line 172 of file ArmController.cs.

```
173     {
174         // not sure if this is necessary
175     }
```

5.1.2.8 RepeatedlyDistanceCalculation()

```
IEnumerator ArmController.RepeatedlyDistanceCalculation (
            Transform handTransform )  [inline], [private]
```

Asynchronous method to calculate the distances repeatedly when the coroutine is started.

Parameters

handTransform	

Definition at line 191 of file ArmController.cs.

```
192     {
193
194         while (isHandDetected)
195         {
196             CalculateNormalizedControlValues(handTransform.position);
197             yield return new WaitForSeconds(sendInterval);
198         }
199     }
```

5.1.2.9 ResetControlValues()

```
void ArmController.ResetControlValues ( )  [inline], [private]
```

Reset the control values to stop the robot, this makes the robot stop when the hand leaves the cube

Definition at line 180 of file ArmController.cs.

```
181     {
182         controlValues.x = 0;
183         controlValues.y = 0;
184         controlValues.speed = 0;
185     }
```

5.1.2.10 SendControlValues()

```
void ArmController.SendControlValues (
             float normalizedX,
             float normalizedY,
             float normalizedZ )  [inline], [private]
```

Sending the information to the robot through socket communication.

The directions is calculated based on the normalized X and Z values. X goes from -1 (drive left) to 1 (drive right) and Z goes from 0 (stop) to 1 (drive forward). Speed is calculated based in how far the hand is from the edges, the closer the hand is to the edge the faster the robot moves. The max speed is when the hand reaches the edge of the cube.

Parameters

normalizedX	
normalizedZ	
normalizedY	
=speed	

"

Definition at line 302 of file ArmController.cs.

```
303     {
304
305         // int dropdownValue = dropdownHandler.GetDropdownValue();
306         // string data = "";
307         controlX.x = normalizedX;
308         controlX.z = normalizedY;
309         controlX.y = normalizedZ;
310         controlX.pinch = rightHand.GetFingerIsPinching(OVRHand.HandFinger.Index) ? 1 : 0;
311         controlX.strength = rightHand.GetFingerPinchStrength(OVRHand.HandFinger.Index);
312         string data = JsonUtility.ToJson(controlX);
313         // Debug.Log($"Control ARM values: {data}");
314
315         SendDataToServer(data);
316
317         // switch (dropdownValue)
318         // {
319         //     case 0:
320         //         // Idle mode
321         //         break;
322         //     // case 1:
323         //     //     // Drive mode
324         //     //     controlValues.x = normalizedX;
325         //     //     controlValues.y = normalizedZ;
326         //     //     controlValues.speed = speed;
327         //     //     data = JsonUtility.ToJson(controlValues);
328         //     //     Debug.Log($"Drive values: {data}");
329         //     //     Debug.Log($"Speed: {speed}");
330         //     //     break;
331         //     case 2:
332         //         // Arm mode
333         //         controlX.x = normalizedX;
334         //         controlX.z = normalizedY;
335         //         controlX.y = normalizedZ;
336         //         controlX.pinch = rightHand.GetFingerIsPinching(OVRHand.HandFinger.Index) ? 1 : 0;
337         //         controlX.strength = rightHand.GetFingerPinchStrength(OVRHand.HandFinger.Index);
338         //         data = JsonUtility.ToJson(controlX);
339         //         Debug.Log($"Control ARM values: {data}");
340
341         //         break;
342         //     case 3:
343         //         // Emergency stop
344
345         //         break;
346         // }
347
348         // if (!dropdownValue.Equals(0) && (lastSendTime == 0 || Time.time - lastSendTime >
    sendInterval))
349         // {
350         //     SendDataToServer(data);
351         //     lastSendTime = Time.time;
352         //     // Convert the control values to a string
353         // }
354         // else
355         // {
356         //     Debug.Log("No data sent to the robot.");
357         // }
358
359     }
```

5.1.2.11 SendDataToServer()

```
void ArmController.SendDataToServer (
            string data )  [inline], [private]
```

Sends the data to the server.

Definition at line 398 of file ArmController.cs.

```
399     {
400         if (networkManager != null)
401         {
402             networkManager.SendData(data);
403         }
404         else
405         {
406             Debug.Log("NetworkManager component not found.");
407         }
408     }
```

5.1.2.12 Start()

```
void ArmController.Start ( )  [inline], [private]
```

The script is called before the first frame update and is used to initialize the necessary variables.

Definition at line 95 of file ArmController.cs.

```
96     {
97         // Working but we have to install some packages to get access to the logfile in VR headset
98         // It should be easly accessible in the VR headset
99         FindObjectOfType<Logger>().Log("Test log file.");
100         // Finding the visualIndicator child of the cube
101         visualIndicatorTransform = transform.Find("VisualIndicatorHand");
102         // Debug.Log($"Indicator position: {visualIndicatorTransform.localPosition.y}");
103         if (visualIndicatorTransform == null)
104         {
105             Debug.LogError("VisualIndicator child not found!");
106         }
107 
108         // Get the material and the color of the visualIndocatorCube
109         cubeMaterial = GetComponent<Renderer>().material;
110         originalColor = cubeMaterial.color; // Save the original color
111         networkManager = NetworkManager.Instance;
112     }
```

5.1.2.13 Update()

```
void ArmController.Update ( )  [inline], [private]
```

Update is called once per frame and is used to log the position of the Hand_WristRoot bone in the VR headset.

This is used for testing purposes.

Definition at line 119 of file ArmController.cs.

```
120     {
121     }
```

5.1.2.14 UpdateVisualIndicator()

```
void ArmController.UpdateVisualIndicator (
            float normalizedX,
            float normalizedZ,
            float normalizedY )  [inline], [private]
```

Update the position of the visual indicator within the detection cube according to the normalized X and Z values.

This is to give a feedback to the user about the hand position within the cube which indicates the direction and the speed of the robot.

Parameters

normalizedX	
normalizedZ	

Definition at line 268 of file ArmController.cs.

```
269     {
270
271         float unchangedY = visualIndicatorTransform.localPosition.y;
272
273         float scaleX = transform.localScale.x / 2; // Half size of the detection cube in X
274         float scaleZ = transform.localScale.z / 2; // Half size of the detection cube in Z
275         float scaleY = transform.localScale.y + ((1.1f - (-0.3f) - 1) / 2);
276
277         // Map the normalized control values back to the world position within the detection cube
278         float worldX = normalizedX * scaleX;
279         float worldZ = normalizedZ * scaleZ * 2 - scaleZ;
280         float worldY = normalizedY * scaleY;
281
282         worldZ = Mathf.Clamp(worldZ, -scaleZ, scaleZ);
283
284         Debug.Log($"New Position - X: {worldX}");
285         Debug.Log($"New Position -Y: {worldY}");
286         Debug.Log($"New Position -Z: {worldZ}");
287
288         Vector3 newPosition = new Vector3(worldX, worldY, worldZ);
289         visualIndicatorTransform.localPosition = newPosition;
290     }
```

5.1.3 Member Data Documentation

5.1.3.1 controlValues

`RobotControlValues ArmController.controlValues = new RobotControlValues()  [private]`

Definition at line 59 of file ArmController.cs.

5.1.3.2 controlX

`RobotControlX ArmController.controlX = new RobotControlX()  [private]`

Definition at line 60 of file ArmController.cs.

5.1.3.3 cubeMaterial

`Material ArmController.cubeMaterial  [private]`

Definition at line 88 of file ArmController.cs.

5.1.3.4 distanceCalculationInterval

`float ArmController.distanceCalculationInterval = 0.5f`

The interval at which to calculate the distances.

Definition at line 75 of file ArmController.cs.

5.1.3.5 dropdownHandler

```
DropdownHandler ArmController.dropdownHandler
```

The DropdownHandler component used to get the mode values from the UI(VR).

Modes are: Idle, Drive, Arm, and Emergency stop

The mode values (0 , 1, 2, 3) has to match with the ROS2 interface from the robot

Definition at line 48 of file ArmController.cs.

5.1.3.6 handSkeleton

```
OVRSkeleton ArmController.handSkeleton
```

The OVRSkeleton component used to track hand gestures.

Definition at line 53 of file ArmController.cs.

5.1.3.7 insideColor

```
Color ArmController.insideColor = new Color(1, 0, 0, 0.2f)
```

Definition at line 89 of file ArmController.cs.

5.1.3.8 isHandDetected

```
bool ArmController.isHandDetected = false  [private]
```

The flag to indicate if the hand is detected within the cube.

Definition at line 70 of file ArmController.cs.

5.1.3.9 lastSendTime

```
float ArmController.lastSendTime  [private]
```

Definition at line 76 of file ArmController.cs.

5.1.3.10 networkManager

```
NetworkManager ArmController.networkManager  [private]
```

The NetworkManager component used to send data to the server.

Definition at line 65 of file ArmController.cs.

5.1.3.11 originalColor

```
Color ArmController.originalColor  [private]
```

Definition at line 90 of file ArmController.cs.

5.1.3.12 rightHand

```
OVRHand ArmController.rightHand
```

The OVRHand component used to track hand gestures.

Definition at line 58 of file ArmController.cs.

5.1.3.13 sendInterval

```
float ArmController.sendInterval = 0.5f
```

The interval at which to send data to the server.

Definition at line 81 of file ArmController.cs.

5.1.3.14 visualIndicatorTransform

```
Transform ArmController.visualIndicatorTransform  [private]
```

The transform of the visual indicator.

That is a dot which indicates the x and z position of the TipBoneEnd of the users hand.

Definition at line 86 of file ArmController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ArmController/ArmController.cs

5.2 CameraDepthData Class Reference

depth camera interface datatype The data is recieved from the robot as this type

Public Attributes

- int time
- List< Vector3Data > depthData = new List<Vector3Data>()

5.2.1 Detailed Description

depth camera interface datatype The data is recieved from the robot as this type

Definition at line 79 of file NetworkManager.cs.

5.2.2 Member Data Documentation

5.2.2.1 depthData

```
List<Vector3Data> CameraDepthData.depthData = new List<Vector3Data>()
```

Definition at line 83 of file NetworkManager.cs.

5.2.2.2 time

```
int CameraDepthData.time
```

Definition at line 82 of file NetworkManager.cs.

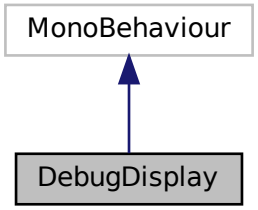
The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

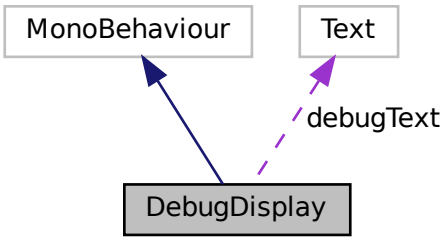
5.3 DebugDisplay Class Reference

This script is used to display debug logs on the a VR screen for testing purposes.

Inheritance diagram for DebugDisplay:



Collaboration diagram for DebugDisplay:



Public Attributes

- Text debugText

Private Member Functions

- void Update ()
- void OnEnable ()
- void OnDisable ()
- void HandleLog (string logString, string stackTrace, LogType type)

Private Attributes

- Dictionary< string, string > debugLogs = new Dictionary<string, string>()

5.3.1 Detailed Description

This script is used to display debug logs on the a VR screen for testing purposes.

This is a simplified version of the DebugDisplayPro script, it is the first version of the script. For documentation on the DebugDisplayPro script, see the DebugDisplayPro class.

Definition at line 11 of file DebugDisplay.cs.

5.3.2 Member Function Documentation

5.3.2.1 HandleLog()

```
void DebugDisplay.HandleLog (
            string logString,
            string stackTrace,
            LogType type )  [inline], [private]
```

Definition at line 34 of file DebugDisplay.cs.

```
35     {
36         if (type == LogType.Log)
37         {
38             string[] splitString = logString.Split(char.Parse(":"));
39             string debugKey = splitString[0];
40             string debugValue = splitString.Length > 1 ? splitString[1] : "";
41 
42             if (debugLogs.ContainsKey(debugKey))
43             {
44                 debugLogs[debugKey] = debugValue;
45             }
46             else
47             {
48                 debugLogs.Add(debugKey, debugValue);
49             }
50 
51         }
52 
53         string debugTextString = "";
54         foreach (KeyValuePair<string, string> log in debugLogs)
55         {
56             if (log.Value == "")
57             {
58                 debugTextString += log.Key + "\n";
59             }
60             else
61             {
62                 debugTextString += log.Key + ": " + log.Value + "\n";
63             }
64         }
65         debugText.text = debugTextString;
66     }
```

5.3.2.2 OnDisable()

```
void DebugDisplay.OnDisable ( )  [inline], [private]
```

Definition at line 29 of file DebugDisplay.cs.

```
30     {
31         Application.logMessageReceived -= HandleLog;
32     }
```

5.3.2.3 OnEnable()

```
void DebugDisplay.OnEnable ( )  [inline], [private]
```

Definition at line 24 of file DebugDisplay.cs.

```
25     {
26         Application.logMessageReceived += HandleLog;
27     }
```

5.3.2.4 Update()

```
void DebugDisplay.Update ( )  [inline], [private]
```

Definition at line 17 of file DebugDisplay.cs.

```
18     {
19         Debug.Log("time:" + Time.time);
20         // Debug.Log(gameObject.name);
21     }
```

5.3.3 Member Data Documentation

5.3.3.1 debugLogs

```
Dictionary<string, string> DebugDisplay.debugLogs = new Dictionary<string, string>()  [private]
```

Definition at line 14 of file DebugDisplay.cs.

5.3.3.2 debugText

```
Text DebugDisplay.debugText
```

Definition at line 15 of file DebugDisplay.cs.

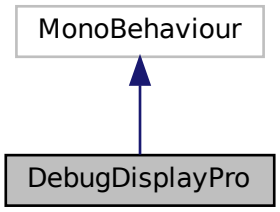
The documentation for this class was generated from the following file:

- Assets/DebugDisplay.cs

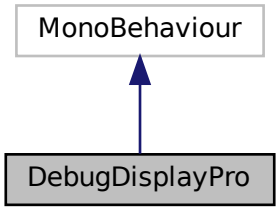
5.4 DebugDisplayPro Class Reference

This script is used to display debug logs on the a VR screen for testing purposes.

Inheritance diagram for DebugDisplayPro:



Collaboration diagram for DebugDisplayPro:



Public Attributes

- TMP_Text debugText

  The TextMeshPro component used to display the debug logs.

Private Member Functions

- void Update ()
- void OnEnable ()

  This method is called when the script is enabled and is used to subscribe to the log message event.
- void OnDisable ()

  This method is called when the script is disabled and is used to unsubscribe from the log message event.
- void HandleLog (string logString, string stackTrace, LogType type)

  This method is used to handle the log messages and store them in the debugLogs dictionary.

Private Attributes

- Dictionary< string, string > debugLogs = new Dictionary<string, string>()

  A dictionary to store the debug logs.

5.4.1 Detailed Description

This script is used to display debug logs on the a VR screen for testing purposes.

It is used to display the logs in a more readable format inside the VR environment.

Definition at line 11 of file DebugDisplayPro.cs.

5.4.2 Member Function Documentation

5.4.2.1 HandleLog()

```
void DebugDisplayPro.HandleLog (
            string logString,
            string stackTrace,
            LogType type )  [inline], [private]
```

This method is used to handle the log messages and store them in the debugLogs dictionary.

It also updates the debugText component with the latest logs.

Parameters

logString	
stackTrace	
type	

Definition at line 60 of file DebugDisplayPro.cs.

```
61     {
62
63         if (debugLogs.Count > 30)
64         {
65             debugLogs.Clear();
66         }
67         if (type == LogType.Log)
68         {
69             string[] splitString = logString.Split(char.Parse(":"));
70             string debugKey = splitString[0];
71             string debugValue = splitString.Length > 1 ? splitString[1] : "";
72
73             // if (debugKey.ToLower().Contains("voice") || debugKey.ToLower().Contains("log"))
74             // {
75             //     return;
76             // }
77             if (debugLogs.ContainsKey(debugKey))
78             {
79                 debugLogs[debugKey] = debugValue;
80             }
81             else
82             {
```

```
83                  debugLogs.Add(debugKey, debugValue);
84              }
85          }
86
87          string debugTextString = "";
88          foreach (KeyValuePair<string, string> log in debugLogs)
89          {
90              if (log.Value == "")
91              {
92                  debugTextString += log.Key + "\n";
93              }
94              else
95              {
96                  debugTextString += log.Key + ": " + log.Value + "\n";
97              }
98          }
99          debugText.text = debugTextString;
100     }
```

5.4.2.2 OnDisable()

```
void DebugDisplayPro.OnDisable ( )  [inline], [private]
```

This method is called when the script is disabled and is used to unsubscribe from the log message event.

Definition at line 48 of file DebugDisplayPro.cs.

```
49      {
50          Application.logMessageReceived -= HandleLog;
51      }
```

5.4.2.3 OnEnable()

```
void DebugDisplayPro.OnEnable ( )  [inline], [private]
```

This method is called when the script is enabled and is used to subscribe to the log message event.

Definition at line 40 of file DebugDisplayPro.cs.

```
41      {
42          Application.logMessageReceived += HandleLog;
43      }
```

5.4.2.4 Update()

```
void DebugDisplayPro.Update ( )  [inline], [private]
```

Definition at line 31 of file DebugDisplayPro.cs.

```
32      {
33          // Debug.Log("time:" + Time.time);
34          // Debug.Log(gameObject.name);
35      }
```

5.4.3 Member Data Documentation

5.4.3.1 debugLogs

```
Dictionary<string, string> DebugDisplayPro.debugLogs = new Dictionary<string, string>()  [private]
```

A dictionary to store the debug logs.

Definition at line 17 of file DebugDisplayPro.cs.

5.4.3.2 debugText

```
TMP_Text DebugDisplayPro.debugText
```

The TextMeshPro component used to display the debug logs.

Definition at line 22 of file DebugDisplayPro.cs.

The documentation for this class was generated from the following file:

- Assets/DebugDisplayPro.cs

5.5 DepthData Class Reference

Public Member Functions

- void AddDepthDataPoint (float x, float y, float z)
- void ClearPoints ()

Public Attributes

- int time

Properties

- IEnumerable< DepthDataPoint > Points [get]
- int Count [get]

Private Attributes

- List< DepthDataPoint > depthDataPoints = new List<DepthDataPoint>()

5.5.1 Detailed Description

Definition at line 111 of file NetworkManager.cs.

5.5.2 Member Function Documentation

5.5.2.1 AddDepthDataPoint()

```
void DepthData.AddDepthDataPoint (
            float x,
            float y,
            float z )  [inline]
```

Definition at line 116 of file NetworkManager.cs.

```
117     {
118         depthDataPoints.Add(new DepthDataPoint(x, y, z));
119     }
```

5.5.2.2 ClearPoints()

```
void DepthData.ClearPoints ( )  [inline]
```

Definition at line 124 of file NetworkManager.cs.

```
125     {
126         depthDataPoints.Clear();
127     }
```

5.5.3 Member Data Documentation

5.5.3.1 depthDataPoints

```
List<DepthDataPoint> DepthData.depthDataPoints = new List<DepthDataPoint>()  [private]
```

Definition at line 114 of file NetworkManager.cs.

5.5.3.2 time

```
int DepthData.time
```

Definition at line 113 of file NetworkManager.cs.

5.5.4 Property Documentation

5.5.4.1 Count

```
int DepthData.Count  [get]
```

Definition at line 123 of file NetworkManager.cs.

5.5.4.2 Points

```
IEnumerable<DepthDataPoint> DepthData.Points  [get]
```

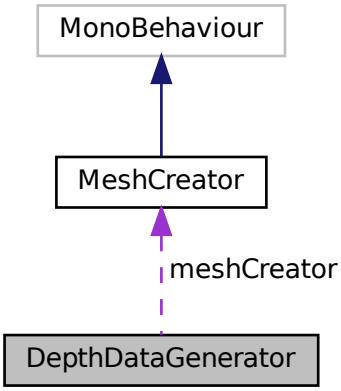
Definition at line 121 of file NetworkManager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

5.6 DepthDataGenerator Class Reference

Collaboration diagram for DepthDataGenerator:



Static Public Member Functions

- static CameraDepthData GenerateDepthData ()

Private Attributes

- MeshCreator meshCreator

Static Private Attributes

- const int Width = 640
- const int Height = 480
- const float MaxDepth = 5.0f
- const float MinDepth = 0.5f

5.6.1 Detailed Description

Definition at line 4 of file DepthDataGenerator.cs.

5.6.2 Member Function Documentation

5.6.2.1 GenerateDepthData()

```
static CameraDepthData DepthDataGenerator.GenerateDepthData ( )  [inline], [static]
```

Definition at line 13 of file DepthDataGenerator.cs.

```
14     {
15         var depthData = new CameraDepthData
16         {
17             time = Environment.TickCount // Use system tick count as a timestamp
18         };
19 
20         for (int y = 0; y < Height; y++)
21         {
22             for (int x = 0; x < Width; x++)
23             {
24                 // Normalize the x, y positions to simulate a real-world field of view
25                 float realX = (x - Width / 2) * (6.4f / Width);  // Center x around 0 and scale to
       real-world meters
26                 float realY = (y - Height / 2) * (4.8f / Height); // Center y around 0 and scale to
       real-world meters
27 
28                 // Generate a random depth between MinDepth and MaxDepth
29                 float depth = MinDepth + (float)(new Random().NextDouble() * (MaxDepth - MinDepth));
30 
31                 // Add the generated point to the dataset
32                 depthData.depthData.Add(new Vector3Data(realX, realY, depth));
33             }
34         }
35 
36         return depthData;
37     }
```

5.6.3 Member Data Documentation

5.6.3.1 Height

```
const int DepthDataGenerator.Height = 480  [static], [private]
```

Definition at line 7 of file DepthDataGenerator.cs.

5.6.3.2 MaxDepth

```
const float DepthDataGenerator.MaxDepth = 5.0f  [static], [private]
```

Definition at line 8 of file DepthDataGenerator.cs.

5.6.3.3 meshCreator

```
MeshCreator DepthDataGenerator.meshCreator  [private]
```

Definition at line 11 of file DepthDataGenerator.cs.

5.6.3.4 MinDepth

```
const float DepthDataGenerator.MinDepth = 0.5f  [static], [private]
```

Definition at line 9 of file DepthDataGenerator.cs.

5.6.3.5 Width

```
const int DepthDataGenerator.Width = 640  [static], [private]
```

Definition at line 6 of file DepthDataGenerator.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/PlotCameraDepthData/DepthDataGenerator.cs

5.7 DepthDataPoint Class Reference

Depth data interface Datatype

Public Member Functions

- DepthDataPoint (float x, float y, float z)

Properties

- float X `[get, set]`
- float Y `[get, set]`
- float Z `[get, set]`

5.7.1 Detailed Description

Depth data interface Datatype

Definition at line 96 of file NetworkManager.cs.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 DepthDataPoint()

```
DepthDataPoint.DepthDataPoint (
            float x,
            float y,
            float z )  [inline]
```

Definition at line 102 of file NetworkManager.cs.

```
103     {
104         X = x;
105         Y = y;
106         Z = z;
107     }
```

5.7.3 Property Documentation

5.7.3.1 X

```
float DepthDataPoint.X  [get], [set]
```

Definition at line 98 of file NetworkManager.cs.

```
98 { get; set; }
```

5.7.3.2 Y

```
float DepthDataPoint.Y  [get], [set]
```

Definition at line 99 of file NetworkManager.cs.

```
99 { get; set; }
```

5.7.3.3 Z

```
float DepthDataPoint.Z  [get], [set]
```

Definition at line 100 of file NetworkManager.cs.

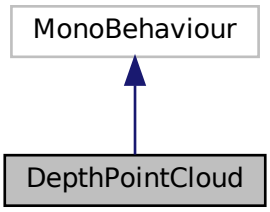
```
100 { get; set; }
```

The documentation for this class was generated from the following file:

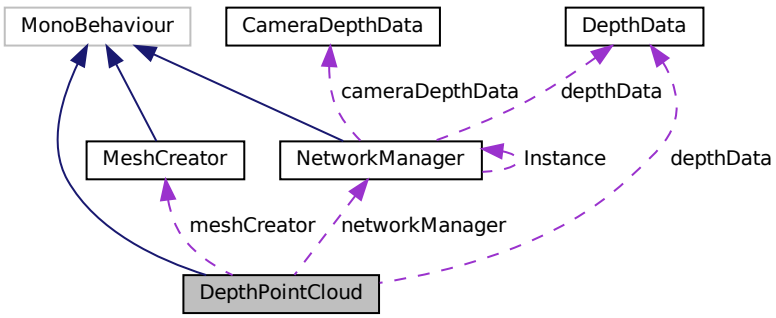
- Assets/Scripts/NetworkNamager/NetworkManager.cs

5.8 DepthPointCloud Class Reference

Inheritance diagram for DepthPointCloud:



Collaboration diagram for DepthPointCloud:



Classes

- class RequestDepthDataMsg

  Interface for requesting depth camera from the robot

Public Member Functions

- byte[ ] DecompressBrotli (byte[ ] compressedData)
- byte[ ] DecodeBase64 (string base64EncodedData)
- void RequestDepthData ()

  This method is used to request depth data from the server.

Public Attributes

- ParticleSystem particleSystem
- Gradient depthColorGradient
- TextAsset brotlifile
- MeshCreator meshCreator

Private Member Functions

- void ProcessDepthData (OVRSimpleJSON.JSONNode jsonData)
- void Start ()
- void Update ()
- void OnDepthDataReceived (DepthData depthData)
- void OnDestroy ()

  This method is used

Private Attributes

- NetworkManager networkManager
- List< ParticleSystem.Particle > particles = new List<ParticleSystem.Particle>()
- DepthData depthData = new DepthData()

5.8.1 Detailed Description

Definition at line 10 of file DepthPointCloud.cs.

5.8.2 Member Function Documentation

5.8.2.1 DecodeBase64()

```
byte [] DepthPointCloud.DecodeBase64 (
            string base64EncodedData )  [inline]
```

Definition at line 35 of file DepthPointCloud.cs.

```
36     {
37         try
38         {
39             byte[] data = Convert.FromBase64String(base64EncodedData);
40             return data;
41         }
42         catch (FormatException ex)
43         {
44             Debug.LogError("Base64 string is not in a valid format: " + ex.Message);
45             return null;
46         }
47     }
```

5.8.2.2 DecompressBrotli()

```
byte [] DepthPointCloud.DecompressBrotli (
            byte[] compressedData )  [inline]
```

Definition at line 23 of file DepthPointCloud.cs.

```
24     {
25         using (var inputStream = new MemoryStream(compressedData))
26         using (var outputStream = new MemoryStream())
27         using (var brotliStream = new BrotliStream(inputStream, CompressionMode.Decompress))
28         {
29             brotliStream.CopyToAsync(outputStream);
30             return outputStream.ToArray();
31         }
32
33     }
```

5.8.2.3 OnDepthDataReceived()

```
void DepthPointCloud.OnDepthDataReceived (
            DepthData depthData )  [inline], [private]
```

Definition at line 82 of file DepthPointCloud.cs.

```
83     {
84         meshCreator.CreateCubeGridTest(depthData);
85         // Processing depth data
86         foreach (DepthDataPoint point in depthData.Points)
87         {
88             Debug.Log($"X, Y, Z: {point.X}, {point.Y}, {point.Z}");
89             Debug.Log($"Time for depthData: {depthData.time}");
90         }
91     }
```

5.8.2.4 OnDestroy()

```
void DepthPointCloud.OnDestroy ( )  [inline], [private]
```

This method is used

Definition at line 116 of file DepthPointCloud.cs.

```
117     {
118         networkManager.onDepthDataReceived -= OnDepthDataReceived;
119     }
```

5.8.2.5 ProcessDepthData()

```
void DepthPointCloud.ProcessDepthData (
            OVRSimpleJSON.JSONNode jsonData )  [inline], [private]
```

Definition at line 49 of file DepthPointCloud.cs.

```
50     {
51         depthData.ClearPoints();
52         depthData.time = jsonData["time"].AsInt;
53
54         for (var i = 0; i < jsonData["data"][0].Count; i++)
55         {
56             depthData.AddDepthDataPoint(jsonData["data"][0][i].AsInt, jsonData["data"][1][i].AsInt,
       jsonData["data"][2][i].AsInt);
57         }
58     }
```

5.8.2.6 RequestDepthData()

```
void DepthPointCloud.RequestDepthData ( )  [inline]
```

This method is used to request depth data from the server.

When it is sent, the robot will respond with the depth data only once.

Definition at line 106 of file DepthPointCloud.cs.

```
107     {
108         RequestDepthDataMsg msg = new RequestDepthDataMsg();
109         msg.get_depth = true;
110         networkManager.SendData(JsonUtility.ToJson(msg));
111     }
```

5.8.2.7 Start()

```
void DepthPointCloud.Start ( )  [inline], [private]
```

Definition at line 59 of file DepthPointCloud.cs.

```
60     {
61         networkManager = NetworkManager.Instance;
62         networkManager.onDepthDataReceived += OnDepthDataReceived;
63     }
```

5.8.2.8 Update()

```
void DepthPointCloud.Update ( )  [inline], [private]
```

Definition at line 65 of file DepthPointCloud.cs.

```
66     {
67         // if (Time.frameCount % 100 == 0)
68         // {
69         //     string receivedData = brotlifile.text;
70         //     byte[] compressedData = DecodeBase64(receivedData);
71         //     byte[] decompressedData = DecompressBrotli(compressedData);
72         //     if (compressedData != null && decompressedData != null)
73         //     {
74         //         receivedData = Encoding.UTF8.GetString(decompressedData);
75         //     }
76         //     var jsonData = OVRSimpleJSON.JSON.Parse(receivedData);
77         //     ProcessDepthData(jsonData);
78         //     OnDepthDataReceived(depthData);
79         // }
80     }
```

5.8.3 Member Data Documentation

5.8.3.1 brotlifile

```
TextAsset DepthPointCloud.brotlifile
```

Definition at line 21 of file DepthPointCloud.cs.

5.8.3.2 depthColorGradient

```
Gradient DepthPointCloud.depthColorGradient
```

Definition at line 15 of file DepthPointCloud.cs.

5.8.3.3 depthData

```
DepthData DepthPointCloud.depthData = new DepthData()  [private]
```

Definition at line 20 of file DepthPointCloud.cs.

5.8.3.4 meshCreator

```
MeshCreator DepthPointCloud.meshCreator
```

Definition at line 22 of file DepthPointCloud.cs.

5.8.3.5 networkManager

```
NetworkManager DepthPointCloud.networkManager  [private]
```

Definition at line 13 of file DepthPointCloud.cs.

5.8.3.6 particles

```
List<ParticleSystem.Particle> DepthPointCloud.particles = new List<ParticleSystem.Particle>()
[private]
```

Definition at line 16 of file DepthPointCloud.cs.

5.8.3.7 particleSystem

```
ParticleSystem DepthPointCloud.particleSystem
```

Definition at line 14 of file DepthPointCloud.cs.

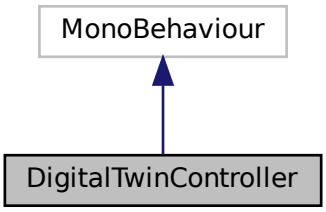
The documentation for this class was generated from the following file:

- Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs

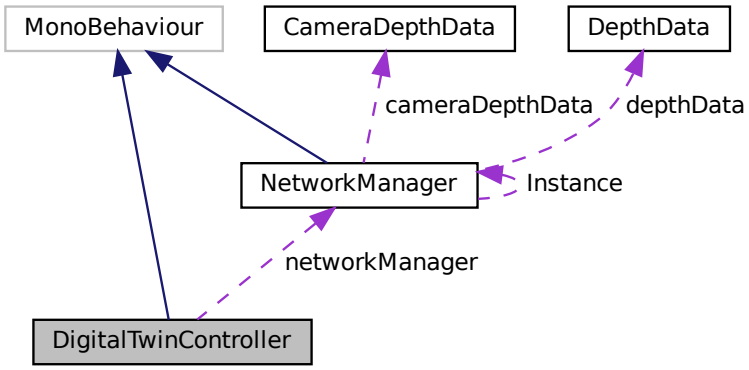
5.9 DigitalTwinController Class Reference

This class is used to control the digital twin robot.

Inheritance diagram for DigitalTwinController:



Collaboration diagram for DigitalTwinController:



Public Attributes

- ArticulationBody Link_1

  The Articulation Link 1 of the robot.
- ArticulationBody Link_2

  The Articulation Link 2 of the robot.
- ArticulationBody Link_3

  The Articulation Link 3 of the robot.
- ArticulationBody Link_4

  The Articulation Link 4 of the robot.
- ArticulationBody Link_5

The Articulation Link 5 of the robot.

- GameObject Pinch_left_x

  The Articulation pinch hand left of the robot.

- GameObject Pinch_right_y

  The Articulation pinch hand right of the robot.

Private Member Functions

- void Start ()
- void HandleReceivedRobotInfoData (JsonArmLengthInfo armInfo)

  This method is used to handle the received robot info data.

- void Update ()
- void testFunction ()
- void SetTargetRotation (ArticulationBody body, float targetAngle)

  This method is used to set the target rotation of an Articulation Body.

- void SetTargetRotationY (ArticulationBody body, float targetAngle)

  This method is used to set the target rotation of an Articulation Body.

Private Attributes

- int frameCounter = 0
- bool rotateTo170 = true
- NetworkManager networkManager

5.9.1 Detailed Description

This class is used to control the digital twin robot.

It updated the position of the digital twin robot based on the received data from the physical robot arm.

Definition at line 18 of file DigitalTwinController.cs.

5.9.2 Member Function Documentation

5.9.2.1 HandleReceivedRobotInfoData()

```
void DigitalTwinController.HandleReceivedRobotInfoData (
            JsonArmLengthInfo armInfo )  [inline], [private]
```

This method is used to handle the received robot info data.

Parameters

robotInfo	The robot info data

Definition at line 75 of file DigitalTwinController.cs.

```
76     {
77         SetTargetRotation(Link_1, -armInfo.Link_1);
78         SetTargetRotation(Link_2, armInfo.Link_2);
79         SetTargetRotation(Link_3, armInfo.Link_3);
80         SetTargetRotation(Link_4, armInfo.Link_4);
81         SetTargetRotationY(Link_5, armInfo.Link_5);
82         // SetTargetRotationY(Link_5, armInfo.Link_5);
83     }
```

5.9.2.2 SetTargetRotation()

```
void DigitalTwinController.SetTargetRotation (
            ArticulationBody body,
            float targetAngle )  [inline], [private]
```

This method is used to set the target rotation of an Articulation Body.

Definition at line 142 of file DigitalTwinController.cs.

```
143     {
144         // Get the current xDrive from the Articulation Body
145         ArticulationDrive drive = body.xDrive;
146 
147         // Set the new target position (degrees)
148         drive.target = targetAngle;
149 
150         // Apply the modified drive back to the Articulation Body
151         body.xDrive = drive;
152     }
```

5.9.2.3 SetTargetRotationY()

```
void DigitalTwinController.SetTargetRotationY (
            ArticulationBody body,
            float targetAngle )  [inline], [private]
```

This method is used to set the target rotation of an Articulation Body.

Parameters

body	
targetAngle	

Definition at line 159 of file DigitalTwinController.cs.

```
160     {
161         // Get the current xDrive from the Articulation Body
162         ArticulationDrive drive = body.yDrive;
163 
164         // Set the new target position (degrees)
165         drive.target = targetAngle;
166 
167         // Apply the modified drive back to the Articulation Body
168         body.yDrive = drive;
169     }
```

5.9.2.4 Start()

```
void DigitalTwinController.Start ( )  [inline], [private]
```

Definition at line 61 of file DigitalTwinController.cs.

```
62     {
63         networkManager = NetworkManager.Instance;
64         networkManager.onArmLengthDataReceived += HandleReceivedRobotInfoData;
65 
66         // Link_1.transform.localRotation = Quaternion.Euler(0, 0, -90);
67     }
```

5.9.2.5 testFunction()

```
void DigitalTwinController.testFunction ( )  [inline], [private]
```

Definition at line 90 of file DigitalTwinController.cs.

```
91     {
92         // frameCounter++;  // Increment frame counter each frame
93 
94         // if (frameCounter >= 500)  // Check if 10 frames have passed
95         // {
96         //     if (rotateTo170)
97         //     {
98         //         SetTargetRotation(Link_2, 180);
99         //         // SetTargetRotation(Link_1, 180);
100         //         // Rotate to 170 degrees on the Y-axis
101         //         // SetTargetRotation(Link_1, 170);
102         //         // SetTargetRotation(Link_2, 53);
103         //         for (int i = 90; i >= 0; i -= 5)
104         //         {
105         //             SetTargetRotation(Link_3, i);
106         //         }
107         //         SetTargetRotation(Link_4, 90);
108         //         // SetTargetRotation(Link_3, 90);
109         //         // SetTargetRotation(Link_3, 90);
110         //         // Link_2.transform.localRotation = Quaternion.Euler(8, -90, -90);
111         //         // Link_3.transform.localRotation = Quaternion.Euler(0, 240, 0);
112         //         // Link_4.transform.localRotation = Quaternion.Euler(0, -140, 0);
113         //         SetTargetRotationY(Link_5, 180);
114         //         rotateTo170 = false;
115         //     }
116         //     else
117         //     {
118         //         // Rotate back to 0 degrees on the Y-axis
119         //         // SetTargetRotation(Link_1, 0);
120         //         // SetTargetRotation(Link_2, -55);
121         //         // SetTargetRotation(Link_2, 0);
122         //         // // SetTargetRotation(Link_1, 0);
123         //         // for (int i = 0; i <= 180; i += 5)
124         //         // {
125         //         //     SetTargetRotation(Link_3, i);
126         //         // }
127         //         // SetTargetRotation(Link_4, 180);
128         //         // // Link_2.transform.localRotation = Quaternion.Euler(-25, -90, -90);
129         //         // // Link_3.transform.localRotation = Quaternion.Euler(0, 180, 0);
130         //         // // Link_4.transform.localRotation = Quaternion.Euler(0, -199, 0);
131         //         SetTargetRotationY(Link_5, 0);
132         //         rotateTo170 = true;
133         //     }
134 
135         //     frameCounter = 0;  // Reset frame counter
136         // }
137     }
```

5.9.2.6 Update()

```
void DigitalTwinController.Update ( )  [inline], [private]
```

Definition at line 85 of file DigitalTwinController.cs.

```
86     {
87 
88     }
```

5.9.3 Member Data Documentation

5.9.3.1 frameCounter

```
int DigitalTwinController.frameCounter = 0  [private]
```

Definition at line 20 of file DigitalTwinController.cs.

5.9.3.2 Link_1

```
ArticulationBody DigitalTwinController.Link_1
```

The Articulation Link 1 of the robot.

Definition at line 28 of file DigitalTwinController.cs.

5.9.3.3 Link_2

```
ArticulationBody DigitalTwinController.Link_2
```

The Articulation Link 2 of the robot.

Definition at line 33 of file DigitalTwinController.cs.

5.9.3.4 Link_3

```
ArticulationBody DigitalTwinController.Link_3
```

The Articulation Link 3 of the robot.

Definition at line 38 of file DigitalTwinController.cs.

5.9.3.5 Link_4

```
ArticulationBody DigitalTwinController.Link_4
```

The Articulation Link 4 of the robot.

Definition at line 43 of file DigitalTwinController.cs.

5.9.3.6 Link_5

```
ArticulationBody DigitalTwinController.Link_5
```

The Articulation Link 5 of the robot.

Definition at line 48 of file DigitalTwinController.cs.

5.9.3.7 networkManager

```
NetworkManager DigitalTwinController.networkManager  [private]
```

Definition at line 23 of file DigitalTwinController.cs.

5.9.3.8 Pinch_left_x

```
GameObject DigitalTwinController.Pinch_left_x
```

The Articulation pinch hand left of the robot.

Definition at line 53 of file DigitalTwinController.cs.

5.9.3.9 Pinch_right_y

```
GameObject DigitalTwinController.Pinch_right_y
```

The Articulation pinch hand right of the robot.

Definition at line 58 of file DigitalTwinController.cs.

5.9.3.10 rotateTo170

```
bool DigitalTwinController.rotateTo170 = true  [private]
```

Definition at line 21 of file DigitalTwinController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/DigitalTwin/DigitalTwinController.cs

5.10 DriveModeController.DriveMode Class Reference

Public Attributes

- string drive_mode

5.10.1 Detailed Description

Definition at line 24 of file DriveModeController.cs.

5.10.2 Member Data Documentation

5.10.2.1 drive_mode

```
string DriveModeController.DriveMode.drive_mode
```

Definition at line 26 of file DriveModeController.cs.

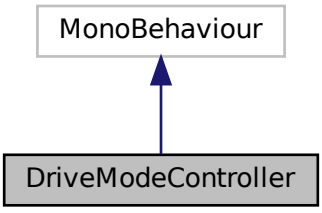
The documentation for this class was generated from the following file:

- Assets/Scripts/DriveModeController/DriveModeController.cs

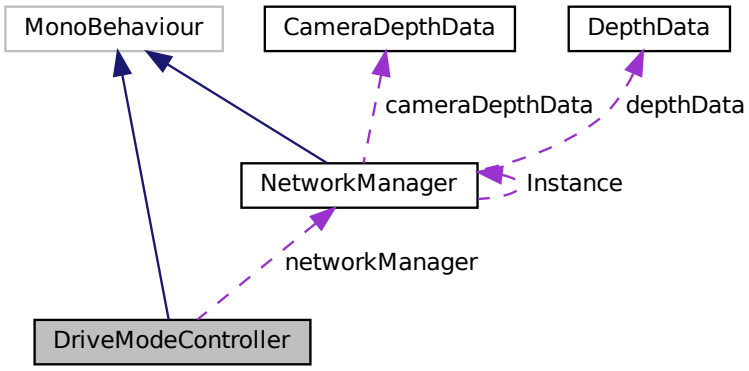
5.11 DriveModeController Class Reference

This class is used to control the drive mode of the robot.

Inheritance diagram for DriveModeController:



Collaboration diagram for DriveModeController:



Classes

- class DriveMode

Public Member Functions

- void precisionMode ()
- void normalMode ()
- void reverseMode ()

Public Attributes

- Button precisionButton
- Button normalButton
- Button reverseButton
- Color activeColor
- Color defaultColor

Events

- Action< string > DriveModeChanged

Private Member Functions

- void Start ()
- void Update ()
- void updateDriveModeData (string mode)

  This method is used to update the drive mode data.
- void updateButtonColor (Color precisionColor, Color normalColor, Color reverseColor)

  This method is used to update the color of the buttons.
- void SendDataToServer (string data)

  This method is used to send the mode value to the server.

Private Attributes

- NetworkManager networkManager

5.11.1 Detailed Description

This class is used to control the drive mode of the robot.

It allows the user to change the drive mode of the robot.

Definition at line 12 of file DriveModeController.cs.

5.11.2 Member Function Documentation

5.11.2.1 normalMode()

```
void DriveModeController.normalMode ( )  [inline]
```

Definition at line 46 of file DriveModeController.cs.

```
47     {
48         updateDriveModeData("normal");
49         updateButtonColor(defaultColor, activeColor, defaultColor);
50     }
```

5.11.2.2 precisionMode()

```
void DriveModeController.precisionMode ( )  [inline]
```

Definition at line 40 of file DriveModeController.cs.

```
41     {
42         updateDriveModeData("precision");
43         updateButtonColor(activeColor, defaultColor, defaultColor);
44     }
```

5.11.2.3 reverseMode()

```
void DriveModeController.reverseMode ( )  [inline]
```

Definition at line 52 of file DriveModeController.cs.

```
53     {
54         updateDriveModeData("reverse");
55         updateButtonColor(defaultColor, defaultColor, activeColor);
56     }
```

5.11.2.4 SendDataToServer()

```
void DriveModeController.SendDataToServer (
            string data )  [inline], [private]
```

This method is used to send the mode value to the server.

Parameters

data	

Definition at line 92 of file DriveModeController.cs.

```
93     {
94         if (networkManager != null)
95         {
96             networkManager.SendData(data);
97         }
98         else
99         {
100             Debug.Log("NetworkManager component not found.");
101         }
102     }
```

5.11.2.5 Start()

```
void DriveModeController.Start ( )  [inline], [private]
```

Definition at line 29 of file DriveModeController.cs.

```
30     {
31         networkManager = NetworkManager.Instance;
32     }
```

5.11.2.6 Update()

```
void DriveModeController.Update ( )  [inline], [private]
```

Definition at line 35 of file DriveModeController.cs.

```
36     {
37
38     }
```

5.11.2.7 updateButtonColor()

```
void DriveModeController.updateButtonColor (
            Color precisionColor,
            Color normalColor,
            Color reverseColor )  [inline], [private]
```

This method is used to update the color of the buttons.

Parameters

precisionColor	
normalColor	
reverseColor	

Definition at line 76 of file DriveModeController.cs.

```
77     {
78         // precisionButton.GetComponent<Image>().color = precisionColor;
79         // normalButton.GetComponent<Image>().color = normalColor;
80         // reverseButton.GetComponent<Image>().color = reverseColor;
81 
82         precisionButton.image.color = precisionColor;
83         normalButton.image.color = normalColor;
84         reverseButton.image.color = reverseColor;
85 
86     }
```

5.11.2.8 updateDriveModeData()

```
void DriveModeController.updateDriveModeData (
            string mode )  [inline], [private]
```

This method is used to update the drive mode data.

Definition at line 61 of file DriveModeController.cs.

```
62     {
63         DriveMode driveMode = new DriveMode();
64         driveMode.drive_mode = mode;
65         SendDataToServer(JsonUtility.ToJson(driveMode));
66         DriveModeChanged?.Invoke(mode);
67     }
```

5.11.3 Member Data Documentation

5.11.3.1 activeColor

```
Color DriveModeController.activeColor
```

Definition at line 22 of file DriveModeController.cs.

5.11.3.2 defaultColor

```
Color DriveModeController.defaultColor
```

Definition at line 23 of file DriveModeController.cs.

5.11.3.3 networkManager

```
NetworkManager DriveModeController.networkManager  [private]
```

Definition at line 14 of file DriveModeController.cs.

5.11.3.4 normalButton

```
Button DriveModeController.normalButton
```

Definition at line 19 of file DriveModeController.cs.

5.11.3.5 precisionButton

```
Button DriveModeController.precisionButton
```

Definition at line 18 of file DriveModeController.cs.

5.11.3.6 reverseButton

```
Button DriveModeController.reverseButton
```

Definition at line 20 of file DriveModeController.cs.

5.11.4 Event Documentation

5.11.4.1 DriveModeChanged

```
Action<string> DriveModeController.DriveModeChanged
```

Definition at line 16 of file DriveModeController.cs.

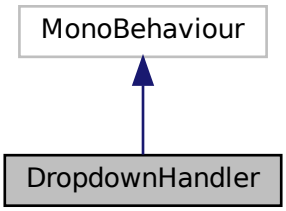
The documentation for this class was generated from the following file:

- Assets/Scripts/DriveModeController/DriveModeController.cs

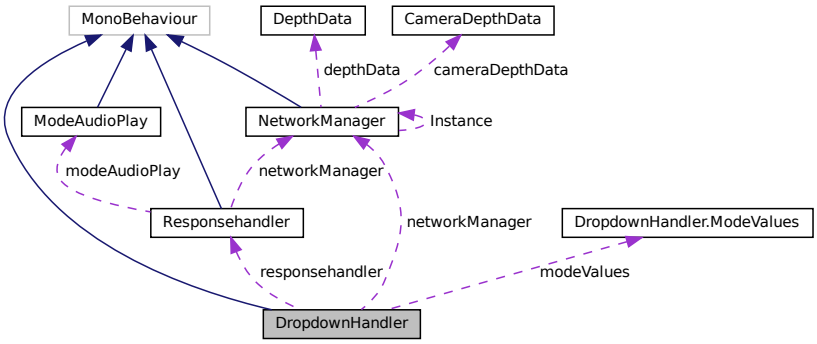
5.12 DropdownHandler Class Reference

This script is used to handle the dropdown in the UI.

Inheritance diagram for DropdownHandler:



Collaboration diagram for DropdownHandler:



Classes

- class ModeValues

  The ModeValues interface used to send the mode value to the robot.

Public Member Functions

- int GetDropdownValue ()

  Get method for the dropdown value.

Public Attributes

- TMP_Dropdown dropdown
- Responsehandler responsehandler

Events

- Action< int > OnDropdownValueChanged

Private Member Functions

- void Start ()

  Initialization of the script by finding the NetworkManager component in the scene and subscribing to the onValue↩ Changed event of the dropdown.
- void HandleVoiceCommand (int mode)
- void DropdownValueChanged (TMP_Dropdown change)

  This method is called when the dropdown value is changed.
- void SendDataToServer (string data)

  This method is used to send the mode value to the server.

Private Attributes

- NetworkManager networkManager
- ModeValues modeValues = new ModeValues()

5.12.1 Detailed Description

This script is used to handle the dropdown in the UI.

When the dropdown value is changed, it sends the new value to the server. The Dropdown is used to controll the mode of the robot.

- Supported modes at the moment:
  - Idle
  - Drive
  - Arm
  - Emergency Stop

Definition at line 18 of file DropdownHandler.cs.

5.12.2 Member Function Documentation

5.12.2.1 DropdownValueChanged()

```
void DropdownHandler.DropdownValueChanged (
            TMP_Dropdown change )  [inline], [private]
```

This method is called when the dropdown value is changed.

It sends the new value to the server.

Definition at line 63 of file DropdownHandler.cs.

```
64     {
65         Debug.Log($"New Dropdown Value Selected: {change.value}");
66         // Perform your action here
67         // For example, if (change.value == 0) { // Do something }
68         modeValues.mode = change.value;
69         SendDataToServer(JsonUtility.ToJson(modeValues));
70         OnDropdownValueChanged?.Invoke(change.value);
71     }
```

5.12.2.2 GetDropdownValue()

```
int DropdownHandler.GetDropdownValue ( )  [inline]
```

Get method for the dropdown value.

Returns

Dropdown.value

Definition at line 93 of file DropdownHandler.cs.

```
94     {
95         return dropdown.value;
96     }
```

5.12.2.3 HandleVoiceCommand()

```
void DropdownHandler.HandleVoiceCommand (
            int mode )  [inline], [private]
```

Definition at line 54 of file DropdownHandler.cs.

```
55     {
56         Debug.Log($"Voice Command Received: {mode}");
57         dropdown.value = mode;
58     }
```

5.12.2.4 SendDataToServer()

```
void DropdownHandler.SendDataToServer (
            string data )  [inline], [private]
```

This method is used to send the mode value to the server.

Parameters

data	

Definition at line 77 of file DropdownHandler.cs.

```
78     {
79         if (networkManager != null)
80         {
81             networkManager.SendData(data);
82         }
83         else
84         {
85             Debug.Log("NetworkManager component not found.");
86         }
87     }
```

5.12.2.5 Start()

```
void DropdownHandler.Start ( )  [inline], [private]
```

Initialization of the script by finding the NetworkManager component in the scene and subscribing to the onValue↩ Changed event of the dropdown.

Definition at line 40 of file DropdownHandler.cs.

```
41     {
42
43         // Ensure the Dropdown is assigned
44         if (dropdown != null)
45         {
46             // Subscribe to the onValueChanged event
47             dropdown.onValueChanged.AddListener(delegate { DropdownValueChanged(dropdown); });
48         }
49         networkManager = NetworkManager.Instance;
50         responsehandler.OnVoiceCommandReceived += HandleVoiceCommand;
51     }
```

5.12.3 Member Data Documentation

5.12.3.1 dropdown

```
TMP_Dropdown DropdownHandler.dropdown
```

Definition at line 30 of file DropdownHandler.cs.

5.12.3.2 modeValues

```
ModeValues DropdownHandler.modeValues = new ModeValues()  [private]
```

Definition at line 32 of file DropdownHandler.cs.

5.12.3.3 networkManager

```
NetworkManager DropdownHandler.networkManager  [private]
```

Definition at line 31 of file DropdownHandler.cs.

5.12.3.4 responsehandler

```
Responsehandler DropdownHandler.responsehandler
```

Definition at line 34 of file DropdownHandler.cs.

5.12.4 Event Documentation

5.12.4.1 OnDropdownValueChanged

```
Action<int> DropdownHandler.OnDropdownValueChanged
```

Definition at line 35 of file DropdownHandler.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/DropdownHandler.cs

5.13 HandGestureAndRotation.HandData Class Reference

The HandData interface used to send the data to the robot.

Public Attributes

- int pinch
- int wrist

5.13.1 Detailed Description

The HandData interface used to send the data to the robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 39 of file HandGestureAndRotation.cs.

5.13.2 Member Data Documentation

5.13.2.1 pinch

`int HandGestureAndRotation.HandData.pinch`

Definition at line 41 of file HandGestureAndRotation.cs.

5.13.2.2 wrist

`int HandGestureAndRotation.HandData.wrist`

Definition at line 42 of file HandGestureAndRotation.cs.

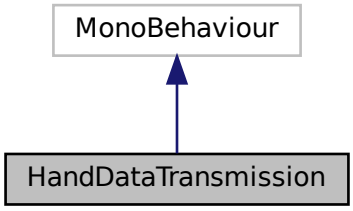
The documentation for this class was generated from the following file:

- Assets/Scripts/HandGestureAndRotation.cs

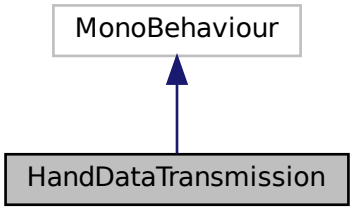
5.14 HandDataTransmission Class Reference

This script is used for testing porpuses.

Inheritance diagram for HandDataTransmission:



Collaboration diagram for HandDataTransmission:



Private Member Functions

- void Start ()
- void Update ()

5.14.1 Detailed Description

This script is used for testing porpuses.

Definition at line 8 of file HandDataTransmission.cs.

5.14.2 Member Function Documentation

5.14.2.1 Start()

```
void HandDataTransmission.Start ( )  [inline], [private]
```

Definition at line 11 of file HandDataTransmission.cs.

```
12     {
13
14     }
```

5.14.2.2 Update()

```
void HandDataTransmission.Update ( )  [inline], [private]
```

Definition at line 17 of file HandDataTransmission.cs.

```
18     {
19
20     }
```

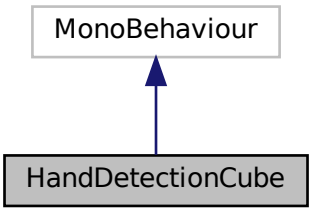
The documentation for this class was generated from the following file:

- Assets/HandDataTransmission.cs

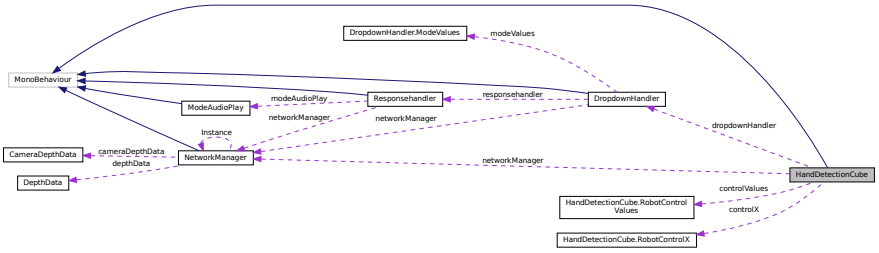
5.15 HandDetectionCube Class Reference

This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.

Inheritance diagram for HandDetectionCube:



Collaboration diagram for HandDetectionCube:



Classes

- class RobotControlValues

  The control values used to control the robot for controlling the robot (car).
- class RobotControlX

  The control values used to control the robot for controlling the arm.

Public Attributes

- DropdownHandler dropdownHandler

  The DropdownHandler component used to get the mode values from the UI(VR).
- OVRSkeleton handSkeleton

  The OVRSkeleton component used to track hand gestures.
- OVRHand rightHand

  The OVRHand component used to track hand gestures.
- float distanceCalculationInterval = 0.5f

  The interval at which to calculate the distances.
- float sendInterval = 0.5f

  The interval at which to send data to the server.
- Color insideColor = new Color(1, 0, 0, 0.1f)

Private Member Functions

- void Start ()

  The script is called before the first frame update and is used to initialize the necessary variables.
- void Update ()

  Update is called once per frame and is used to log the position of the Hand_WristRoot bone in the VR headset.
- void OnTriggerEnter (Collider other)

  This method is called when the hand enters the cube area.
- void OnTriggerExit (Collider other)

  Stop the coroutine when the hand leaves the cube
- void OnTriggerStay (Collider other)
- void ResetControlValues ()

  Reset the control values to stop the robot, this makes the robot stop when the hand leaves the cube
- IEnumerator RepeatedlyDistanceCalculation (Transform handTransform)

  Asynchronous method to calculate the distances repeatedly when the coroutine is started.
- void CalculateDistances (Vector3 handPosition)

  Calculate the distances from the hand to the edges of the cube This is used for debugging and understanding the hand position within the cube
- void CalculateNormalizedControlValues (Vector3 handPosition)

  Calculate the normalized control values based on the hand position within the cube
- void UpdateVisualIndicator (float normalizedX, float normalizedZ)

  Update the position of the visual indicator within the detection cube according to the normalized X and Z values.
- void SendControlValues (float normalizedX, float normalizedZ, float normalizedY, int speed=50)

  Sending the information to the robot through socket communication.
- int CalculateSpeed (float normalizedX, float normalizedZ)

  Calculate the speed based on the normalized X and Z values
- void HandleReceivedData (string data)

  Handles the recieved data from the server.
- void SendDataToServer (string data)

  Sends the data to the server.
- void OnDestroy ()

Private Attributes

- RobotControlValues controlValues = new RobotControlValues()
- RobotControlX controlX = new RobotControlX()
- NetworkManager networkManager

  The NetworkManager component used to send data to the server.
- bool isHandDetected = false

  The flag to indicate if the hand is detected within the cube.
- float lastSendTime
- Transform visualIndicatorTransform

  The transform of the visual indicator.
- Material cubeMaterial
- Color originalColor

5.15.1 Detailed Description

This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.

This class is used for both controlling the robot and the arm. This class will be refactored to be more modular and to use interfaces which implemented as state machines.

Definition at line 15 of file HandDetectionCube.cs.

5.15.2 Member Function Documentation

5.15.2.1 CalculateDistances()

```
void HandDetectionCube.CalculateDistances (
            Vector3 handPosition )  [inline], [private]
```

Calculate the distances from the hand to the edges of the cube This is used for debugging and understanding the hand position within the cube

Parameters

handPosition	

Definition at line 215 of file HandDetectionCube.cs.

```
216     {
217         float halfScaleX = transform.localScale.x / 2;
218         float halfScaleZ = transform.localScale.z / 2;
219 
220         float distanceToLeftEdge = handPosition.x - (transform.position.x - halfScaleX);
221         float distanceToRightEdge = (transform.position.x + halfScaleX) - handPosition.x;
222         float distanceToFrontEdge = (transform.position.z + halfScaleZ) - handPosition.z;
223         float distanceToBackEdge = handPosition.z - (transform.position.z - halfScaleZ);
224 
225         // Log distances to the console
226         Debug.Log($"Distance to Left Edge: {distanceToLeftEdge}");
227         Debug.Log($"Distance to Right Edge: {distanceToRightEdge}");
228         Debug.Log($"Distance to Front Edge: {distanceToFrontEdge}");
229         Debug.Log($"Distance to Back Edge: {distanceToBackEdge}");
230     }
```

5.15.2.2 CalculateNormalizedControlValues()

```
void HandDetectionCube.CalculateNormalizedControlValues (
            Vector3 handPosition )  [inline], [private]
```

Calculate the normalized control values based on the hand position within the cube

Parameters

handPosition	

Definition at line 236 of file HandDetectionCube.cs.

```
237     {
238         // Convert hand position to the cube's local space
239         Vector3 handLocalPosition = transform.InverseTransformPoint(handPosition);
240 
241         // Calculate normalized X within -1 to 1 range
242         float normalizedX = Mathf.Clamp(handLocalPosition.x / (transform.localScale.x / 2), -1, 1);
243 
244         // Calculate normalized Z within 0 to 1 range
245         float normalizedZ = Mathf.Clamp((handLocalPosition.z / (transform.localScale.z / 2) + 1) / 2, 0,
      1);
246 
247         // Calculate normalized Y within 0 to 1 range
248         float scaleY = transform.localScale.y;
249 
```

```
250         // Calculate the normalized Y value within 0 to 1 range
251         float normalizedY = (handLocalPosition.y + scaleY / 2) / scaleY;
252         normalizedY = Mathf.Clamp(normalizedY, 0, 1);
253 
254         // Map the normalized Y value to a range of -1 to 1
255         float mappedY = (normalizedY - 0.3f) * 3.0f;
256 
257         // Debug.Log($"Normalized X: {normalizedX}");
258         // Debug.Log($"Normalized Z: {normalizedZ}");
259         // Debug.Log($"Normalized Y: {mappedY}");
260 
261         // Calculate speed, send control values to the robot, and update the visual indicator
262         int speed = CalculateSpeed(normalizedX, normalizedZ);
263         SendControlValues(normalizedX, normalizedZ, mappedY, speed); // Modify to calculate tresholds
      for X and Z?
264         // UpdateVisualIndicator(normalizedX, normalizedZ);
265     }
```

5.15.2.3 CalculateSpeed()

```
int HandDetectionCube.CalculateSpeed (
            float normalizedX,
            float normalizedZ )  [inline], [private]
```

Calculate the speed based on the normalized X and Z values

Parameters

normalizedX	
normalizedZ	

Returns

Definition at line 324 of file HandDetectionCube.cs.

```
325     {
326         float calculatedSpeed = 0;
327         if (normalizedX < 0.3 && normalizedX > -0.3 && normalizedZ < 0.3)
328         {
329             calculatedSpeed = 0;
330         }
331         else if (normalizedX > 0.3 || normalizedX < -0.3)
332         {
333             calculatedSpeed = Math.Abs(100 * normalizedX);
334         }
335         else if (normalizedZ > 0.4)
336         {
337             calculatedSpeed = Math.Abs(100 * normalizedZ);
338         }
339         return (int)calculatedSpeed;
340     }
```

5.15.2.4 HandleReceivedData()

```
void HandDetectionCube.HandleReceivedData (
            string data )  [inline], [private]
```

Handles the recieved data from the server.

It is not in use at the moment.

Definition at line 346 of file HandDetectionCube.cs.

```
347     {
348         // Process the received data
349         Debug.Log($"ObjectController received data: {data}");
350     }
```

5.15.2.5 OnDestroy()

```
void HandDetectionCube.OnDestroy ( )  [inline], [private]
```

Definition at line 367 of file HandDetectionCube.cs.

```
368     {
369         StopAllCoroutines();
370         networkManager.OnDataReceived -= HandleReceivedData;
371     }
```

5.15.2.6 OnTriggerEnter()

```
void HandDetectionCube.OnTriggerEnter (
            Collider other )  [inline], [private]
```

This method is called when the hand enters the cube area.

It changes the color of the cube and starts the coroutine to calculate the distances repeatedly.

Parameters

other	

Definition at line 137 of file HandDetectionCube.cs.

```
138     {
139         // Check if TipBoneEnd is in the cube
140         if (other.CompareTag("TipboneSphere"))
141         {
142             isHandDetected = true;
143             // Debug.Log($"Visualindicator entered cube area.{other.bounds.size}");
144             // Debug.Log($"Other size: {other.bounds.size}");
145 
146             // Change the color of the cube when the hand enters the cube
147             cubeMaterial.color = insideColor;
148 
149             // Start coroutine to calculate distances repeatedly
150             int Hand_WristRoot = (int)OVRPlugin.BoneId.Hand_MiddleTip;
151             OVRBone WristBone = handSkeleton.Bones[Hand_WristRoot];
152             StartCoroutine(RepeatedlyDistanceCalculation(WristBone.Transform));
153         }
154     }
```

5.15.2.7 OnTriggerExit()

```
void HandDetectionCube.OnTriggerExit (
            Collider other )  [inline], [private]
```

Stop the coroutine when the hand leaves the cube

Parameters

other	

Definition at line 160 of file HandDetectionCube.cs.

```
161     {
162         if (other.CompareTag("TipboneSphere"))
163         {
164             isHandDetected = false;
165             // Debug.Log("Hand exited cube area.");
166             // Reset the color of the cube when the hand leaves the cube
167             cubeMaterial.color = originalColor;
168 
169             // Stop the coroutine when the hand leaves the Cude
170             StopAllCoroutines();
171 
172             // Reset the control values to stop the robot
173             ResetControlValues();
174 
175             // Stop the robot when the had leaves the control area(Cube)
176             string json = JsonUtility.ToJson(controlValues);
177             SendDataToServer(json);
178         }
179     }
```

5.15.2.8 OnTriggerStay()

```
void HandDetectionCube.OnTriggerStay (
             Collider other )  [inline], [private]
```

Definition at line 181 of file HandDetectionCube.cs.

```
182     {
183         // not sure if this is necessary
184     }
```

5.15.2.9 RepeatedlyDistanceCalculation()

```
IEnumerator HandDetectionCube.RepeatedlyDistanceCalculation (
             Transform handTransform )  [inline], [private]
```

Asynchronous method to calculate the distances repeatedly when the coroutine is started.

Parameters

handTransform	

Definition at line 200 of file HandDetectionCube.cs.

```
201     {
202 
203         while (isHandDetected)
204         {
205             CalculateNormalizedControlValues(handTransform.position);
206             yield return new WaitForSeconds(sendInterval);
207         }
208     }
```

5.15.2.10 ResetControlValues()

```
void HandDetectionCube.ResetControlValues ( )  [inline], [private]
```

Reset the control values to stop the robot, this makes the robot stop when the hand leaves the cube

Definition at line 189 of file HandDetectionCube.cs.

```
190     {
191         controlValues.x = 0;
192         controlValues.y = 0;
193         controlValues.speed = 0;
194     }
```

5.15.2.11 SendControlValues()

```
void HandDetectionCube.SendControlValues (
            float normalizedX,
            float normalizedZ,
            float normalizedY,
            int speed = 50 )  [inline], [private]
```

Sending the information to the robot through socket communication.

The directions is calculated based on the normalized X and Z values. X goes from -1 (drive left) to 1 (drive right) and Z goes from 0 (stop) to 1 (drive forward). Speed is calculated based in how far the hand is from the edges, the closer the hand is to the edge the faster the robot moves. The max speed is when the hand reaches the edge of the cube.

Parameters

normalizedX	
normalizedZ	
normalizedY	
=speed	

"

Definition at line 305 of file HandDetectionCube.cs.

```
306     {
307
308         // int dropdownValue = dropdownHandler.GetDropdownValue();
309         // string data = "";
310         controlValues.x = normalizedX;
311         controlValues.y = normalizedZ;
312         controlValues.speed = speed;
313         string data = JsonUtility.ToJson(controlValues);
314         SendDataToServer(data);
315
316     }
```

5.15.2.12 SendDataToServer()

```
void HandDetectionCube.SendDataToServer (
            string data )  [inline], [private]
```

Sends the data to the server.

Definition at line 355 of file HandDetectionCube.cs.

```
356     {
357         if (networkManager != null)
358         {
359             networkManager.SendData(data);
360         }
361         else
362         {
363             Debug.Log("NetworkManager component not found.");
364         }
365     }
```

5.15.2.13 Start()

```
void HandDetectionCube.Start ( )  [inline], [private]
```

The script is called before the first frame update and is used to initialize the necessary variables.

Definition at line 94 of file HandDetectionCube.cs.

```
95     {
96         // Working but we have to install some packages to get access to the logfile in VR headset
97         // It should be easly accessible in the VR headset
98         FindObjectOfType<Logger>().Log("Test log file.");
99         // Finding the visualIndicator child of the cube
100         visualIndicatorTransform = transform.Find("visualIndicator");
101         // Debug.Log($"Indicator position: {visualIndicatorTransform.localPosition.y}");
102         if (visualIndicatorTransform == null)
103         {
104             Debug.LogError("VisualIndicator child not found!");
105         }
106 
107         // Get the material and the color of the visualIndocatorCube
108         cubeMaterial = GetComponent<Renderer>().material;
109         originalColor = cubeMaterial.color; // Save the original color
110         networkManager = NetworkManager.Instance;
111     }
```

5.15.2.14 Update()

```
void HandDetectionCube.Update ( )  [inline], [private]
```

Update is called once per frame and is used to log the position of the Hand_WristRoot bone in the VR headset.

This is used for testing purposes.

Definition at line 118 of file HandDetectionCube.cs.

```
119     {
120         // int Hand_WristRoot = (int)OVRPlugin.BoneId.Hand_WristRoot;
121 
122         // // Loging to the VR LOG screen in the VR headset
123         // OVRBone WristBone = handSkeleton.Bones[Hand_WristRoot];
124         // Debug.Log($"Hand_WristRoot bone ID: {Hand_WristRoot}");
125         // Debug.Log($"Hand_WristRoot bone position: {WristBone.Transform.position}");
126 
127         // Debug.Log($"CurrentNumBones: {handSkeleton.GetCurrentNumBones()}");
128         // Debug.Log($"Right Hand start: {handSkeleton.GetCurrentStartBoneId()}");
129         // Debug.Log($"Right Hanh end: {handSkeleton.GetCurrentEndBoneId()}");
130     }
```

5.15.2.15 UpdateVisualIndicator()

```
void HandDetectionCube.UpdateVisualIndicator (
            float normalizedX,
            float normalizedZ )  [inline], [private]
```

Update the position of the visual indicator within the detection cube according to the normalized X and Z values.

This is to give a feedback to the user about the hand position within the cube which indicates the direction and the speed of the robot.

Parameters

normalizedX	
normalizedZ	

Definition at line 273 of file HandDetectionCube.cs.

```
274     {
275
276         float unchangedY = visualIndicatorTransform.localPosition.y;
277
278         float scaleX = transform.localScale.x / 2; // Half size of the detection cube in X
279         float scaleZ = transform.localScale.z / 2; // Half size of the detection cube in Z
280
281         // Map the normalized control values back to the world position within the detection cube
282         float worldX = normalizedX * scaleX;
283         float worldZ = normalizedZ * scaleZ * 2 - scaleZ;
284
285         worldZ = Mathf.Clamp(worldZ, -scaleZ, scaleZ);
286
287         // Debug.Log($"New Position - X: {worldX}");
288         // Debug.Log($"New Position -Y: {unchangedY}");
289         // Debug.Log($"New Position -Z: {worldZ}");
290
291         Vector3 newPosition = new Vector3(worldX, unchangedY, worldZ);
292         visualIndicatorTransform.localPosition = newPosition;
293     }
```

5.15.3 Member Data Documentation

5.15.3.1 controlValues

```
RobotControlValues HandDetectionCube.controlValues = new RobotControlValues()  [private]
```

Definition at line 58 of file HandDetectionCube.cs.

5.15.3.2 controlX

```
RobotControlX HandDetectionCube.controlX = new RobotControlX()  [private]
```

Definition at line 59 of file HandDetectionCube.cs.

5.15.3.3 cubeMaterial

```
Material HandDetectionCube.cubeMaterial  [private]
```

Definition at line 87 of file HandDetectionCube.cs.

5.15.3.4 distanceCalculationInterval

```
float HandDetectionCube.distanceCalculationInterval = 0.5f
```

The interval at which to calculate the distances.

Definition at line 74 of file HandDetectionCube.cs.

5.15.3.5 dropdownHandler

```
DropdownHandler HandDetectionCube.dropdownHandler
```

The DropdownHandler component used to get the mode values from the UI(VR).

Modes are: Idle, Drive, Arm, and Emergency stop

The mode values (0 , 1, 2, 3) has to match with the ROS2 interface from the robot

Definition at line 47 of file HandDetectionCube.cs.

5.15.3.6 handSkeleton

```
OVRSkeleton HandDetectionCube.handSkeleton
```

The OVRSkeleton component used to track hand gestures.

Definition at line 52 of file HandDetectionCube.cs.

5.15.3.7 insideColor

```
Color HandDetectionCube.insideColor = new Color(1, 0, 0, 0.1f)
```

Definition at line 88 of file HandDetectionCube.cs.

5.15.3.8 isHandDetected

```
bool HandDetectionCube.isHandDetected = false  [private]
```

The flag to indicate if the hand is detected within the cube.

Definition at line 69 of file HandDetectionCube.cs.

5.15.3.9 lastSendTime

```
float HandDetectionCube.lastSendTime  [private]
```

Definition at line 75 of file HandDetectionCube.cs.

5.15.3.10 networkManager

```
NetworkManager HandDetectionCube.networkManager  [private]
```

The NetworkManager component used to send data to the server.

Definition at line 64 of file HandDetectionCube.cs.

5.15.3.11 originalColor

```
Color HandDetectionCube.originalColor  [private]
```

Definition at line 89 of file HandDetectionCube.cs.

5.15.3.12 rightHand

```
OVRHand HandDetectionCube.rightHand
```

The OVRHand component used to track hand gestures.

Definition at line 57 of file HandDetectionCube.cs.

5.15.3.13 sendInterval

```
float HandDetectionCube.sendInterval = 0.5f
```

The interval at which to send data to the server.

Definition at line 80 of file HandDetectionCube.cs.

5.15.3.14 visualIndicatorTransform

```
Transform HandDetectionCube.visualIndicatorTransform  [private]
```

The transform of the visual indicator.

That is a dot which indicates the x and z position of the TipBoneEnd of the users hand.

Definition at line 85 of file HandDetectionCube.cs.

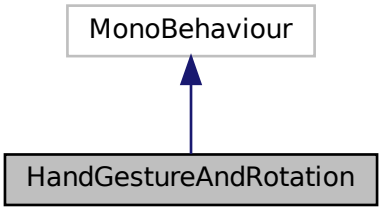
The documentation for this class was generated from the following file:

- Assets/Scripts/HandDetectionCube.cs

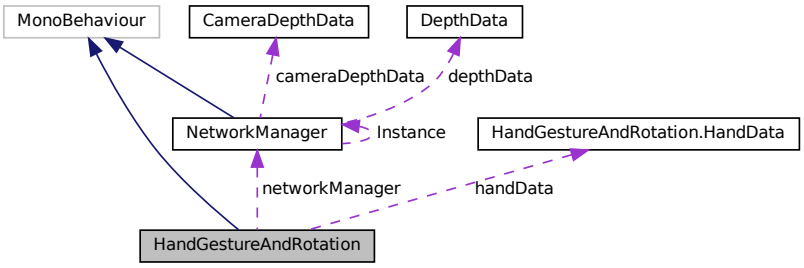
5.16 HandGestureAndRotation Class Reference

This class is used to get the hand gesture and rotation data and send it to the server.

Inheritance diagram for HandGestureAndRotation:



Collaboration diagram for HandGestureAndRotation:



Classes

- class HandData

  The HandData interface used to send the data to the robot.

Public Attributes

- OVRSkeleton handSkeleton

  The OVRSkeleton component used to track hand gestures.
- OVRHand hand

  The OVRHand component used to track hand gestures.
- float sendInterval = 0.3f

  The interval at which to send data to the server.
- bool isTesting = false

  Flag to indicate if the script is running in testing mode.

Private Member Functions

- void Start ()

  Initilization of the script by finding the NetworkManager component in the scene.
- void Update ()

  Update is called once per frame and is used to get the hand gesture and rotation data and send it to the server.
- void SendDataToServer (string data)

  Sends the data to the server.

Private Attributes

- NetworkManager networkManager

  The NetworkManager component used to send data to the server.
- float lastSendTime
- HandData handData = new HandData()

5.16.1 Detailed Description

This class is used to get the hand gesture and rotation data and send it to the server.

It is used for testing purposes to control the robot arm with hand gestures.

This class does not work as expected and needs to be fixed

Definition at line 8 of file HandGestureAndRotation.cs.

5.16.2 Member Function Documentation

5.16.2.1 SendDataToServer()

```
void HandGestureAndRotation.SendDataToServer (
            string data )  [inline], [private]
```

Sends the data to the server.

Parameters

data	The data to send.

Definition at line 124 of file HandGestureAndRotation.cs.

```
125     {
126         if (networkManager != null)
127         {
128             networkManager.SendData(data);
129         }
130         else
131         {
132             Debug.Log("NetworkManager component not found.");
133         }
134     }
```

5.16.2.2 Start()

`void HandGestureAndRotation.Start ( )  [inline], [private]`

Initilization of the script by finding the NetworkManager component in the scene.

Definition at line 50 of file HandGestureAndRotation.cs.

```
51     {
52         if (!isTesting) return;
53         networkManager = NetworkManager.Instance;
54 
55     }
```

5.16.2.3 Update()

`void HandGestureAndRotation.Update ( )  [inline], [private]`

Update is called once per frame and is used to get the hand gesture and rotation data and send it to the server.

Returns if isTesting is false.

Definition at line 61 of file HandGestureAndRotation.cs.

```
62     {
63         if (!isTesting) return;
64         if (handSkeleton == null || hand == null)
65         {
66             Debug.LogWarning("HandSkeleton or OVRHand reference is missing.");
67             return;
68         }
69 
70         // Ensure the skeleton and hand are fully tracked
71         if (!handSkeleton.IsDataValid || !handSkeleton.IsDataHighConfidence || !hand.IsTracked)
72         {
73             return;
74         }
75 
76         // Accessing the MiddleTip bone
77         int handMiddleTipIndex = (int)OVRPlugin.BoneId.Hand_MiddleTip;
78         if (handMiddleTipIndex < 0 || handMiddleTipIndex >= handSkeleton.Bones.Count)
79         {
80             Debug.LogWarning("Hand_MiddleTip index is out of range.");
81             return;
82         }
83         OVRBone middleTipBone = handSkeleton.Bones[handMiddleTipIndex];
84 
85         // Check for pinch
86         bool isPinching = hand.GetFingerIsPinching(OVRHand.HandFinger.Index);
87 
```

```
88          // Output the pinch status and rotation of the MiddleTip bone
89          Debug.Log($"Is Pinching: {isPinching}");
90          Debug.Log($"MiddleTip Bone Rotation: {middleTipBone.Transform.rotation}");
91
92          handData.pinch = isPinching ? 1 : 0;
93
94          Quaternion handRootRotation =
        handSkeleton.Bones[(int)OVRPlugin.BoneId.Hand_WristRoot].Transform.rotation;
95
96          // To display or use this rotation:
97          // Convert to Euler angles for easier understanding or display
98          Vector3 handRootEuler = handRootRotation.eulerAngles;
99
100          // Optionally convert to radians
101          Vector3 handRootRadians = new Vector3(handRootEuler.x * Mathf.Deg2Rad, handRootEuler.y *
        Mathf.Deg2Rad, handRootEuler.z * Mathf.Deg2Rad);
102
103          Debug.Log($"Hand Root Rotation (Euler): {handRootEuler}");
104          Debug.Log($"Hand Root Rotation (Radians): {handRootRadians}");
105
106          // Convert the handData object to a JSON string
107          string data = JsonUtility.ToJson(handData);
108
109          // Send the data to the server
110          if (Time.time - lastSendTime > 0.1f)
111          {
112              SendDataToServer(data);
113              lastSendTime = Time.time;
114          }
115
116          // Optionally perform actions based on pinch status and bone rotation
117          // For example, you could trigger an event or control an object in your scene
118      }
```

5.16.3 Member Data Documentation

5.16.3.1 hand

```
OVRHand HandGestureAndRotation.hand
```

The OVRHand component used to track hand gestures.

Definition at line 18 of file HandGestureAndRotation.cs.

5.16.3.2 handData

```
HandData HandGestureAndRotation.handData = new HandData()  [private]
```

Definition at line 45 of file HandGestureAndRotation.cs.

5.16.3.3 handSkeleton

```
OVRSkeleton HandGestureAndRotation.handSkeleton
```

The OVRSkeleton component used to track hand gestures.

Definition at line 13 of file HandGestureAndRotation.cs.

5.16.3.4 isTesting

```
bool HandGestureAndRotation.isTesting = false
```

Flag to indicate if the script is running in testing mode.

Definition at line 34 of file HandGestureAndRotation.cs.

5.16.3.5 lastSendTime

```
float HandGestureAndRotation.lastSendTime  [private]
```

Definition at line 24 of file HandGestureAndRotation.cs.

5.16.3.6 networkManager

```
NetworkManager HandGestureAndRotation.networkManager  [private]
```

The NetworkManager component used to send data to the server.

Definition at line 23 of file HandGestureAndRotation.cs.

5.16.3.7 sendInterval

```
float HandGestureAndRotation.sendInterval = 0.3f
```

The interval at which to send data to the server.

Definition at line 29 of file HandGestureAndRotation.cs.

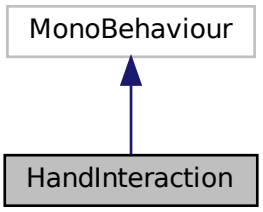
The documentation for this class was generated from the following file:

- Assets/Scripts/HandGestureAndRotation.cs

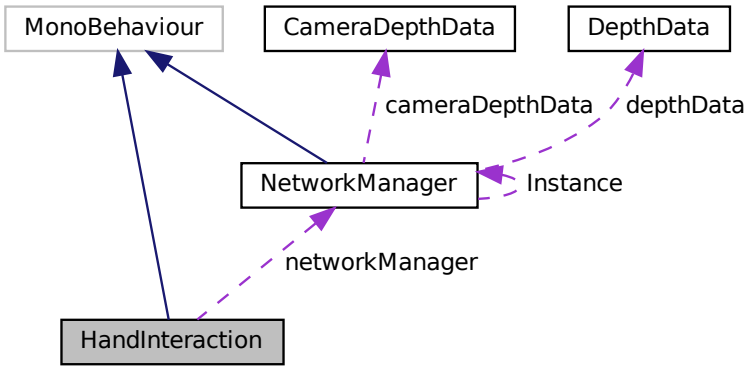
5.17 HandInteraction Class Reference

This script is used to send hand tracking data to the server at regular intervals.

Inheritance diagram for HandInteraction:



Collaboration diagram for HandInteraction:



Public Attributes

- float sendInterval = 0.1f

  The interval at which to send data to the server.
- bool isTesting = false

  Flag to indicate if the script is running in testing mode.

Private Member Functions

- void Start ()

  Start is called before the first frame update and is used to initialize the script.
- void Update ()

  Update is called once per frame and is used to send hand tracking data to the server at regular intervals.
- void HandleReceivedData (string data)

  Handles the received data from the server.
- void SendDataToServer (string data)

  Sends the data to the server.
- void OnDestroy ()

Private Attributes

- OVRHand hand

  The OVRHand component used to track hand gestures.
- float lastSendTime

  The time of the last data send.
- NetworkManager networkManager

  The NetworkManager component used to send data to the server.

5.17.1 Detailed Description

This script is used to send hand tracking data to the server at regular intervals.

It is used for testing purposes.

Definition at line 11 of file HandInteraction.cs.

5.17.2 Member Function Documentation

5.17.2.1 HandleReceivedData()

```
void HandInteraction.HandleReceivedData (
            string data )  [inline], [private]
```

Handles the received data from the server.

Parameters

data	The received data.

Definition at line 88 of file HandInteraction.cs.

```
89     {
90         // Process the received data
91         Debug.Log($"ObjectController received data: {data}");
```

```
92     }
```

5.17.2.2 OnDestroy()

```
void HandInteraction.OnDestroy ( )  [inline], [private]
```

Definition at line 110 of file HandInteraction.cs.

```
111     {
112         // It's important to unsubscribe when the GameObject is destroyed
113         NetworkManager.Instance.OnDataReceived -= HandleReceivedData;
114     }
```

5.17.2.3 SendDataToServer()

```
void HandInteraction.SendDataToServer (
            string data )  [inline], [private]
```

Sends the data to the server.

Parameters

data	The data to send.

Definition at line 98 of file HandInteraction.cs.

```
99     {
100         if (networkManager != null)
101         {
102             networkManager.SendData(data);
103         }
104         else
105         {
106             Debug.Log("NetworkManager component not found.");
107         }
108     }
```

5.17.2.4 Start()

```
void HandInteraction.Start ( )  [inline], [private]
```

Start is called before the first frame update and is used to initialize the script.

Returns is isTesting is false.

Definition at line 43 of file HandInteraction.cs.

```
44     {
45         if (!isTesting) return;
46
47         hand = GetComponent<OVRHand>();
48         Debug.Log("Hand Interaction Script is running!");
49         // Find the NetworkManager component on this GameObject
50         networkManager = NetworkManager.Instance;
51         NetworkManager.Instance.OnDataReceived += HandleReceivedData;
52     }
```

5.17.2.5 Update()

```
void HandInteraction.Update ( )  [inline], [private]
```

Update is called once per frame and is used to send hand tracking data to the server at regular intervals.

Returns if isTesting is false.Hand Tracking

Definition at line 58 of file HandInteraction.cs.

```
59     {
60         if (!isTesting) return;
61 
62         bool isIndexFingerPinching = hand.GetFingerIsPinching(OVRHand.HandFinger.Index);
63         float indexFingerPinchStrength = hand.GetFingerPinchStrength(OVRHand.HandFinger.Index);
64         OVRHand.TrackingConfidence confidence = hand.GetFingerConfidence(OVRHand.HandFinger.Index);
65         Debug.Log($"Index finger pinching: {isIndexFingerPinching}");
66         Debug.Log($"Strength: {indexFingerPinchStrength}");
67         Debug.Log($"Confidence: {confidence}");
71         Vector3 handPosition = hand.transform.position;
72         Debug.Log($"Hand Position: {handPosition}");
73         // Send data at intervals
74         if (Time.time - lastSendTime > sendInterval)
75         {
76             string data = $"{handPosition}";
77             SendDataToServer(data);
78             SendDataToServer($"{handPosition.y}".ToString());
79             lastSendTime = Time.time;
80         }
81 
82     }
```

5.17.3 Member Data Documentation

5.17.3.1 hand

```
OVRHand HandInteraction.hand  [private]
```

The OVRHand component used to track hand gestures.

Definition at line 17 of file HandInteraction.cs.

5.17.3.2 isTesting

```
bool HandInteraction.isTesting = false
```

Flag to indicate if the script is running in testing mode.

Definition at line 37 of file HandInteraction.cs.

5.17.3.3 lastSendTime

```
float HandInteraction.lastSendTime  [private]
```

The time of the last data send.

Definition at line 22 of file HandInteraction.cs.

5.17.3.4 networkManager

```
NetworkManager HandInteraction.networkManager  [private]
```

The NetworkManager component used to send data to the server.

Definition at line 32 of file HandInteraction.cs.

5.17.3.5 sendInterval

```
float HandInteraction.sendInterval = 0.1f
```

The interval at which to send data to the server.

Definition at line 27 of file HandInteraction.cs.

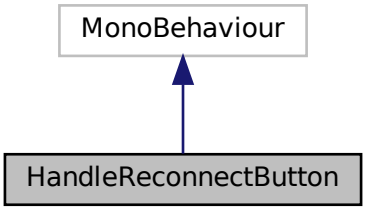
The documentation for this class was generated from the following file:

- Assets/HandInteraction.cs

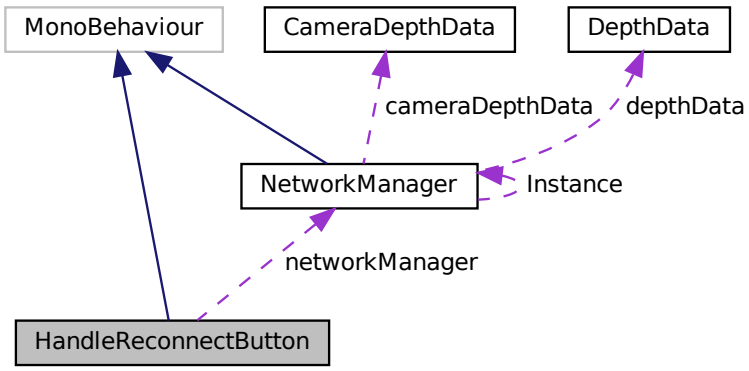
5.18 HandleReconnectButton Class Reference

This script is used to handle the reconnect button in the UI.

Inheritance diagram for HandleReconnectButton:



Collaboration diagram for HandleReconnectButton:



Public Member Functions

- void OnReconnectButtonClicked ()

  This method is called when the reconnect button is clicked.

Public Attributes

- GameObject reconnectButton

  The GameObject used to display the reconnect button in the UI.

Private Member Functions

- void Start ()

  Initializes the script by finding the NetworkManager component in the scene.
- void Update ()

Private Attributes

- NetworkManager networkManager

  The NetworkManager component used to handle the network connection.

5.18.1 Detailed Description

This script is used to handle the reconnect button in the UI.

When the button is clicked, it calls the Reconnect method of the NetworkManager to disconnect and reconnect to the server. It is used for testing purposes.

This script is used for testing purposes.

Definition at line 10 of file HandleReconnectButton.cs.

5.18.2 Member Function Documentation

5.18.2.1 OnReconnectButtonClicked()

```
void HandleReconnectButton.OnReconnectButtonClicked ( )  [inline]
```

This method is called when the reconnect button is clicked.

It calls the Reconnect method of the NetworkManager to disconnect and reconnect to the server.

Definition at line 35 of file HandleReconnectButton.cs.

```
36     {
37         if (networkManager != null)
38         {
39             networkManager.Reconnect();
40         }
41         else
42         {
43             Debug.Log("NetworkManager component not found.");
44         }
45     }
```

5.18.2.2 Start()

```
void HandleReconnectButton.Start ( )  [inline], [private]
```

Initializes the script by finding the NetworkManager component in the scene.

Definition at line 27 of file HandleReconnectButton.cs.

```
28     {
29         networkManager = NetworkManager.Instance;
30     }
```

5.18.2.3 Update()

```
void HandleReconnectButton.Update ( )  [inline], [private]
```

Definition at line 49 of file HandleReconnectButton.cs.

```
50     {
51     }
```

5.18.3 Member Data Documentation

5.18.3.1 networkManager

```
NetworkManager HandleReconnectButton.networkManager  [private]
```

The NetworkManager component used to handle the network connection.

Definition at line 22 of file HandleReconnectButton.cs.

5.18.3.2 reconnectButton

```
GameObject HandleReconnectButton.reconnectButton
```

The GameObject used to display the reconnect button in the UI.

A rectangular button is used for this purpose.

Definition at line 17 of file HandleReconnectButton.cs.

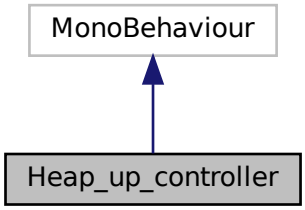
The documentation for this class was generated from the following file:

- Assets/Scripts/HandleReconnectButton.cs

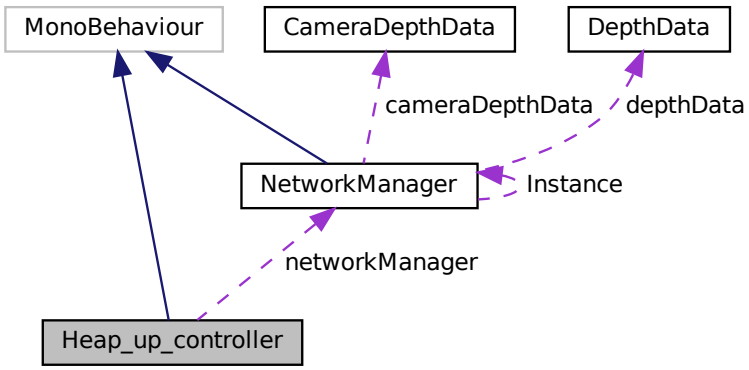
5.19 Heap_up_controller Class Reference

This class is used to control the head up display of the robot.

Inheritance diagram for Heap_up_controller:



Collaboration diagram for Heap_up_controller:



Public Attributes

- Canvas head_up_canvas

  The heap up canvas game object that hostes the heap up display
- Plane robot_view_plane

  Robot view plane(Robot camera monitor)
- OVRCameraRig ovr_camera_rig

  The OVR camera rig that is used to get the position of the camera
- TMP_Text speed_text
- TMP_Text voltage_text
- TMP_Text battery_precentage_text
- TMP_Text mode_text
- TMP_Text latecy_text
- TMP_Text conenctionStatus_text

Private Member Functions

- void update_head_up_canvas_position_and_rotation ()

  Updates the position and rotation of the head up canvas to match the position and rotation of the camera This allows the user to allways see the head up display in front of them
- void Start ()
- void HandleConnectionStatusChanged (bool connected)
- void HandleReceivedRobotInfoData (JsonRobotInfo info)

  This method is used to handle the received robot info data.
- void HandleReceivedPingData (float ping)

  This method is used to handle the received ping data.
- void Update ()
- void OnEnable ()
- void OnDisable ()
- void OnDestroy ()

Private Attributes

- float head_up_canvas_distance = 4f
- Vector3 offset_ovr_camera_rig = new Vector3(0, 0, 0)
- NetworkManager networkManager
- Color connectedColor
- Color disconnectedColor
- List< float > accelerometer
- List< float > gyroscope
- List< float > magnetometer
- List< float > motion
- float speed = 0
- float voltage
- int battery_precentage
- string mode
- float cms_speed

5.19.1 Detailed Description

This class is used to control the head up display of the robot.

It allows the user to see the robot info on the head up display.

Definition at line 12 of file Heap_up_controller.cs.

5.19.2 Member Function Documentation

5.19.2.1 HandleConnectionStatusChanged()

```
void Heap_up_controller.HandleConnectionStatusChanged (
            bool connected )  [inline], [private]
```

Definition at line 94 of file Heap_up_controller.cs.

```
95     {
96
97         conenctionStatus_text.text = connected ? "Connected" : "Disconnected";
98         conenctionStatus_text.color = connected ? connectedColor : disconnectedColor;
99
100     }
```

5.19.2.2 HandleReceivedPingData()

```
void Heap_up_controller.HandleReceivedPingData (
            float ping )  [inline], [private]
```

This method is used to handle the received ping data.

It updates the latency on the head up display.

Parameters

ping	

Definition at line 155 of file Heap_up_controller.cs.

```
156     {
157         latecy_text.text = ping.ToString();
158         Debug.Log("LatencyPing: " + ping);
159     }
```

5.19.2.3 HandleReceivedRobotInfoData()

```
void Heap_up_controller.HandleReceivedRobotInfoData (
            JsonRobotInfo info )  [inline], [private]
```

This method is used to handle the received robot info data.

It updates the robot info on the head up display if the data has changed.

Parameters

info	

Definition at line 108 of file Heap_up_controller.cs.

```
109     {
110         if (accelerometer != info.accelerometer)
111         {
112             accelerometer = info.accelerometer;
113         }
114         if (gyroscope != info.gyroscope)
115         {
116             gyroscope = info.gyroscope;
117         }
118         if (magnetometer != info.magnetometer)
119         {
120             magnetometer = info.magnetometer;
121         }
122         if (motion != info.motion)
123         {
124             motion = info.motion;
125         }
126         if (speed != info.speed)
127         {
128             speed = info.cms_speed;
129             speed_text.text = cms_speed.ToString();
130         }
131         if (voltage != info.voltage)
132         {
133             voltage = info.voltage;
134             voltage_text.text = voltage.ToString();
135         }
136         if (battery_precentage != info.battery_precentage)
137         {
138             battery_precentage = info.battery_precentage;
139             battery_precentage_text.text = battery_precentage.ToString();
140         }
141         if (mode != info.mode)
142         {
143             mode = info.mode;
144             mode_text.text = mode;
145         }
146 
147     }
```

5.19.2.4 OnDestroy()

```
void Heap_up_controller.OnDestroy ( )  [inline], [private]
```

Definition at line 176 of file Heap_up_controller.cs.

```
177     {
178         networkManager.OnRobotInfoDataReceived -= HandleReceivedRobotInfoData;
179     }
```

5.19.2.5 OnDisable()

```
void Heap_up_controller.OnDisable ( )  [inline], [private]
```

Definition at line 171 of file Heap_up_controller.cs.

```
172     {
173
174     }
```

5.19.2.6 OnEnable()

```
void Heap_up_controller.OnEnable ( )  [inline], [private]
```

Definition at line 166 of file Heap_up_controller.cs.

```
167     {
168
169     }
```

5.19.2.7 Start()

```
void Heap_up_controller.Start ( )  [inline], [private]
```

Definition at line 83 of file Heap_up_controller.cs.

```
84     {
85         networkManager = NetworkManager.Instance;
86         networkManager.OnRobotInfoDataReceived += HandleReceivedRobotInfoData;
87         networkManager.onPingDataReceived += HandleReceivedPingData;
88         networkManager.OnConnectionStatus += HandleConnectionStatusChanged;
89         disconnectedColor = conenctionStatus_text.color;
90         connectedColor = latecy_text.color;
91     }
```

5.19.2.8 Update()

```
void Heap_up_controller.Update ( )  [inline], [private]
```

Definition at line 161 of file Heap_up_controller.cs.

```
162     {
163         update_head_up_canvas_position_and_rotation();
164     }
```

5.19.2.9 update_head_up_canvas_position_and_rotation()

```
void Heap_up_controller.update_head_up_canvas_position_and_rotation ( )  [inline], [private]
```

Updates the position and rotation of the head up canvas to match the position and rotation of the camera This allows the user to allways see the head up display in front of them

Definition at line 62 of file Heap_up_controller.cs.

```
63     {
64         // Move the head up canvas to the position of the camera
65         Vector3 newPosition = ovr_camera_rig.centerEyeAnchor.position +
       ovr_camera_rig.centerEyeAnchor.forward * head_up_canvas_distance + offset_ovr_camera_rig;
66
67
68         newPosition.y = head_up_canvas.transform.position.y;
69         newPosition.z = head_up_canvas.transform.position.z;
70
71         head_up_canvas.transform.position = newPosition;
72         head_up_canvas.transform.LookAt(ovr_camera_rig.centerEyeAnchor);
73
74         head_up_canvas.transform.Rotate(0, 180f, 0);
75
76         //Upright relativ to the camera (Users head)
77
78         // The rotation on the X and Z axes but allows it to rotate freely around the Y axis
79         // head_up_canvas.transform.rotation = Quaternion.Euler(0,
       head_up_canvas.transform.rotation.eulerAngles.y, 0);
80     }
```

5.19.3 Member Data Documentation

5.19.3.1 accelerometer

```
List<float> Heap_up_controller.accelerometer  [private]
```

Definition at line 48 of file Heap_up_controller.cs.

5.19.3.2 battery_precentage

```
int Heap_up_controller.battery_precentage  [private]
```

Definition at line 54 of file Heap_up_controller.cs.

5.19.3.3 battery_precentage_text

```
TMP_Text Heap_up_controller.battery_precentage_text
```

Definition at line 39 of file Heap_up_controller.cs.

5.19.3.4 cms_speed

```
float Heap_up_controller.cms_speed  [private]
```

Definition at line 56 of file Heap_up_controller.cs.

5.19.3.5 conenctionStatus_text

```
TMP_Text Heap_up_controller.conenctionStatus_text
```

Definition at line 43 of file Heap_up_controller.cs.

5.19.3.6 connectedColor

```
Color Heap_up_controller.connectedColor  [private]
```

Definition at line 44 of file Heap_up_controller.cs.

5.19.3.7 disconnectedColor

```
Color Heap_up_controller.disconnectedColor  [private]
```

Definition at line 45 of file Heap_up_controller.cs.

5.19.3.8 gyroscope

```
List<float> Heap_up_controller.gyroscope  [private]
```

Definition at line 49 of file Heap_up_controller.cs.

5.19.3.9 head_up_canvas

```
Canvas Heap_up_controller.head_up_canvas
```

The heap up canvas game object that hostes the heap up display

Definition at line 18 of file Heap_up_controller.cs.

5.19.3.10 head_up_canvas_distance

```
float Heap_up_controller.head_up_canvas_distance = 4f  [private]
```

Definition at line 29 of file Heap_up_controller.cs.

5.19.3.11 latecy_text

```
TMP_Text Heap_up_controller.latecy_text
```

Definition at line 41 of file Heap_up_controller.cs.

5.19.3.12 magnetometer

```
List<float> Heap_up_controller.magnetometer  [private]
```

Definition at line 50 of file Heap_up_controller.cs.

5.19.3.13 mode

```
string Heap_up_controller.mode  [private]
```

Definition at line 55 of file Heap_up_controller.cs.

5.19.3.14 mode_text

```
TMP_Text Heap_up_controller.mode_text
```

Definition at line 40 of file Heap_up_controller.cs.

5.19.3.15 motion

```
List<float> Heap_up_controller.motion  [private]
```

Definition at line 51 of file Heap_up_controller.cs.

5.19.3.16 networkManager

```
NetworkManager Heap_up_controller.networkManager  [private]
```

Definition at line 34 of file Heap_up_controller.cs.

5.19.3.17 offset_ovr_camera_rig

```
Vector3 Heap_up_controller.offset_ovr_camera_rig = new Vector3(0, 0, 0)  [private]
```

Definition at line 32 of file Heap_up_controller.cs.

5.19.3.18 ovr_camera_rig

```
OVRCameraRig Heap_up_controller.ovr_camera_rig
```

The OVR camera rig that is used to get the position of the camera

Definition at line 26 of file Heap_up_controller.cs.

5.19.3.19 robot_view_plane

```
Plane Heap_up_controller.robot_view_plane
```

Robot view plane(Robot camera monitor)

Definition at line 22 of file Heap_up_controller.cs.

5.19.3.20 speed

```
float Heap_up_controller.speed = 0  [private]
```

Definition at line 52 of file Heap_up_controller.cs.

5.19.3.21 speed_text

```
TMP_Text Heap_up_controller.speed_text
```

Definition at line 37 of file Heap_up_controller.cs.

5.19.3.22 voltage

```
float Heap_up_controller.voltage  [private]
```

Definition at line 53 of file Heap_up_controller.cs.

5.19.3.23 voltage_text

```
TMP_Text Heap_up_controller.voltage_text
```

Definition at line 38 of file Heap_up_controller.cs.

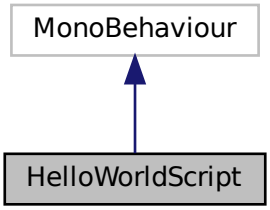
The documentation for this class was generated from the following file:

- Assets/Scripts/Head up display/Heap_up_controller.cs

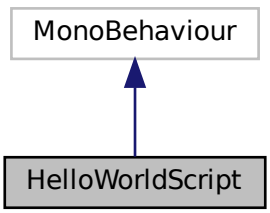
5.20 HelloWorldScript Class Reference

This script is used to display a simple "Hello World" message on the screen for testing purposes.

Inheritance diagram for HelloWorldScript:



Collaboration diagram for HelloWorldScript:



Public Attributes

- string myName = "Kromium Kromiumsen"

Private Member Functions

- void Start ()

  Start is called before the first frame update and is used to initialize the script.

Private Attributes

- TextMeshProUGUI textMeshPro

  The TextMeshPro component used to display the message.

5.20.1 Detailed Description

This script is used to display a simple "Hello World" message on the screen for testing purposes.

Definition at line 9 of file HelloWorldScript.cs.

5.20.2 Member Function Documentation

5.20.2.1 Start()

```
void HelloWorldScript.Start ( )  [inline], [private]
```

Start is called before the first frame update and is used to initialize the script.

It sets the text of the TextMeshPro component to a simple "Hello World" message. It also logs a message to the console.

Definition at line 23 of file HelloWorldScript.cs.

```
24     {
25         Debug.Log("Hello from the other side!");
26         textMeshPro = GetComponent<TextMeshProUGUI>();
27         textMeshPro.text = $"We are {myName}";
28     }
```

5.20.3 Member Data Documentation

5.20.3.1 myName

```
string HelloWorldScript.myName = "Kromium Kromiumsen"
```

Definition at line 11 of file HelloWorldScript.cs.

5.20.3.2 textMeshPro

```
TextMeshProUGUI HelloWorldScript.textMeshPro  [private]
```

The TextMeshPro component used to display the message.

Definition at line 16 of file HelloWorldScript.cs.

The documentation for this class was generated from the following file:

- Assets/HelloWorldScript.cs

5.21 JsonArmLengthInfo Class Reference

Public Attributes

- int Link_1
- int Link_2
- int Link_3
- int Link_4
- int Link_5
- int pintch

5.21.1 Detailed Description

Definition at line 44 of file NetworkManager.cs.

5.21.2 Member Data Documentation

5.21.2.1 Link_1

```
int JsonArmLengthInfo.Link_1
```

Definition at line 46 of file NetworkManager.cs.

5.21.2.2 Link_2

```
int JsonArmLengthInfo.Link_2
```

Definition at line 47 of file NetworkManager.cs.

5.21.2.3 Link_3

```
int JsonArmLengthInfo.Link_3
```

Definition at line 48 of file NetworkManager.cs.

5.21.2.4 Link_4

```
int JsonArmLengthInfo.Link_4
```

Definition at line 50 of file NetworkManager.cs.

5.21.2.5 Link_5

```
int JsonArmLengthInfo.Link_5
```

Definition at line 51 of file NetworkManager.cs.

5.21.2.6 pintch

```
int JsonArmLengthInfo.pintch
```

Definition at line 52 of file NetworkManager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

5.22 JsonRobotInfo Class Reference

Class `JsonRobotInfo` Represents the JSON data structure for the robot information.

Public Attributes

- List< float > accelerometer
- List< float > gyroscope
- List< float > magnetometer
- List< float > motion
- float speed = 88
- float voltage
- int battery_precentage
- string mode
- float cms_speed

5.22.1 Detailed Description

Class `JsonRobotInfo` Represents the JSON data structure for the robot information.

Definition at line 29 of file NetworkManager.cs.

5.22.2 Member Data Documentation

5.22.2.1 accelerometer

```
List<float> JsonRobotInfo.accelerometer
```

Definition at line 32 of file NetworkManager.cs.

5.22.2.2 battery_precentage

```
int JsonRobotInfo.battery_precentage
```

Definition at line 38 of file NetworkManager.cs.

5.22.2.3 cms_speed

```
float JsonRobotInfo.cms_speed
```

Definition at line 40 of file NetworkManager.cs.

5.22.2.4 gyroscope

```
List<float> JsonRobotInfo.gyroscope
```

Definition at line 33 of file NetworkManager.cs.

5.22.2.5 magnetometer

```
List<float> JsonRobotInfo.magnetometer
```

Definition at line 34 of file NetworkManager.cs.

5.22.2.6 mode

```
string JsonRobotInfo.mode
```

Definition at line 39 of file NetworkManager.cs.

5.22.2.7 motion

```
List<float> JsonRobotInfo.motion
```

Definition at line 35 of file NetworkManager.cs.

5.22.2.8 speed

```
float JsonRobotInfo.speed = 88
```

Definition at line 36 of file NetworkManager.cs.

5.22.2.9 voltage

```
float JsonRobotInfo.voltage
```

Definition at line 37 of file NetworkManager.cs.

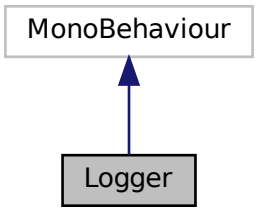
The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

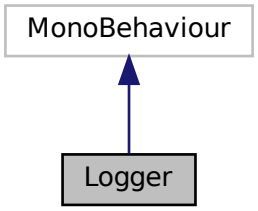
5.23 Logger Class Reference

This script is used to log messages to a file.

Inheritance diagram for Logger:



Collaboration diagram for Logger:



Public Member Functions

- void Log (string message)

  This method is used to log messages to the log file and the Unity Console.

Private Member Functions

- void Awake ()

  Awake is called when the script instance is being loaded.

Private Attributes

- string logFilePath

5.23.1 Detailed Description

This script is used to log messages to a file.

It is used for testing purposes.

Definition at line 7 of file Logger.cs.

5.23.2 Member Function Documentation

5.23.2.1 Awake()

```
void Logger.Awake ( )  [inline], [private]
```

Awake is called when the script instance is being loaded.

It gets the path to the log file and clears it at the start of the session.

Definition at line 14 of file Logger.cs.

```
15     {
16         // Set the log file path to the persistent data path
17         logFilePath = Path.Combine(Application.persistentDataPath, "gameLog.txt");
18 
19         // // Clear the log file at the start of the session.
20         File.WriteAllText(logFilePath, string.Empty);
21     }
```

5.23.2.2 Log()

```
void Logger.Log (
            string message )  [inline]
```

This method is used to log messages to the log file and the Unity Console.

Parameters

message	

Definition at line 27 of file Logger.cs.

```
28     {
29         // string logMessage = System.DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") + ": " + message;
30         // File.AppendAllText(logFilePath, logMessage + "\n");
31 
32         // // For testing purposes, also log the message to the Unity Console
33         // Debug.Log(message);
34     }
```

5.23.3 Member Data Documentation

5.23.3.1 logFilePath

```
string Logger.logFilePath  [private]
```

Definition at line 9 of file Logger.cs.

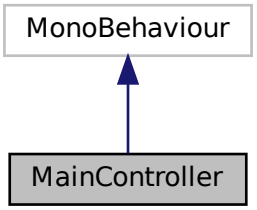
The documentation for this class was generated from the following file:

- Assets/Scripts/log/Logger.cs

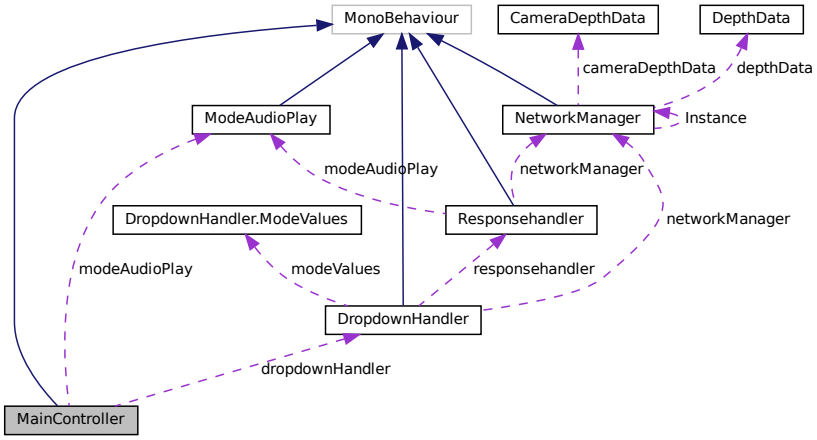
5.24 MainController Class Reference

This class is the main controller for handling scene changes and mode changes.

Inheritance diagram for MainController:



Collaboration diagram for MainController:



Public Attributes

- DropdownHandler dropdownHandler
- GameObject driveScene
- GameObject armScene
- TMP_Dropdown dropdown
- ModeAudioPlay modeAudioPlay
- XRPokeFollowAffordance EmergencyStopButton

Private Member Functions

- void Start ()

  Start is called before the first frame update, it is used to initialize the controller, subscribe to relevant events, and deactivate scenes.
- void HandleEmergencyStop (bool emergency)

  This method is used to handle the emergency stop event.
- void OnDestroy ()
- void HandleDropdownChange (int newValue)

  This method is used to enabling/disabling scenes based on the selected mode.

5.24.1 Detailed Description

This class is the main controller for handling scene changes and mode changes.

It enables and disables scenes based on the selected mode dynamically.

Definition at line 12 of file MainController.cs.

5.24.2 Member Function Documentation

5.24.2.1 HandleDropdownChange()

```
void MainController.HandleDropdownChange (
            int newValue )  [inline], [private]
```

This method is used to enabling/disabling scenes based on the selected mode.

Parameters

newValue	

Definition at line 75 of file MainController.cs.

```
76     {
77         Debug.Log($"Dropdown value changed to: {newValue}");
78         // Modify the switch statement to activate/deactivate scenes
79         switch (newValue) // Changed to use newValue directly
80         {
81             case 0:
```

```
82                  // Idle mode
83                  driveScene.SetActive(false);
84                  armScene.SetActive(false);
85                  modeAudioPlay.PlayIdle();
86                  break;
87              case 1:
88                  // Drive mode
89                  driveScene.SetActive(true);
90                  armScene.SetActive(false);
91                  modeAudioPlay.PlayDrive();
92                  break;
93              case 2:
94                  // Arm mode
95                  driveScene.SetActive(false);
96                  armScene.SetActive(true);
97                  modeAudioPlay.PlayArm();
98                  break;
99              case 3:
100                 // Emergency Stop mode
101                 driveScene.SetActive(false);
102                 armScene.SetActive(false);
103                 modeAudioPlay.PlayEmergency();
104                 break;
105             default:
106                 // Default case
107                 break;
108         }
109     }
```

5.24.2.2 HandleEmergencyStop()

```
void MainController.HandleEmergencyStop (
             bool emergency )  [inline], [private]
```

This method is used to handle the emergency stop event.

Parameters

emergency	

Definition at line 51 of file MainController.cs.

```
52     {
53         Debug.Log($"Emergency Stop: {emergency}");
54         // Modify the switch statement to activate/deactivate scenes
55         if (emergency)
56         {
57             driveScene.SetActive(false);
58             armScene.SetActive(false);
59             dropdown.value = 3;
60         }
61     }
```

5.24.2.3 OnDestroy()

```
void MainController.OnDestroy ( )  [inline], [private]
```

Definition at line 63 of file MainController.cs.

```
64     {
65         if (dropdownHandler != null)
66         {
67             dropdownHandler.OnDropdownValueChanged -= HandleDropdownChange;
68         }
69     }
```

5.24.2.4 Start()

```
void MainController.Start ( )  [inline], [private]
```

Start is called before the first frame update, it is used to initialize the controller, subscribe to relevant events, and deactivate scenes.

Definition at line 26 of file MainController.cs.

```
27     {
28         if (dropdownHandler != null)
29         {
30             dropdownHandler.OnDropdownValueChanged += HandleDropdownChange;
31             EmergencyStopButton.OnEmergency += HandleEmergencyStop;
32         }
33 
34         // Deactivate both scenes at the start
35         if (driveScene != null)
36         {
37             driveScene.SetActive(false);
38         }
39 
40         if (armScene != null)
41         {
42             armScene.SetActive(false);
43         }
44 
45     }
```

5.24.3 Member Data Documentation

5.24.3.1 armScene

```
GameObject MainController.armScene
```

Definition at line 16 of file MainController.cs.

5.24.3.2 driveScene

```
GameObject MainController.driveScene
```

Definition at line 15 of file MainController.cs.

5.24.3.3 dropdown

```
TMP_Dropdown MainController.dropdown
```

Definition at line 18 of file MainController.cs.

5.24.3.4 dropdownHandler

```
DropdownHandler MainController.dropdownHandler
```

Definition at line 14 of file MainController.cs.

5.24.3.5 EmergencyStopButton

```
XRPokeFollowAffordance MainController.EmergencyStopButton
```

Definition at line 21 of file MainController.cs.

5.24.3.6 modeAudioPlay

```
ModeAudioPlay MainController.modeAudioPlay
```

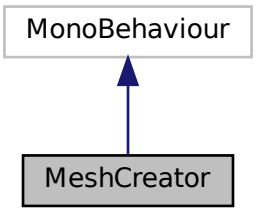
Definition at line 20 of file MainController.cs.

The documentation for this class was generated from the following file:

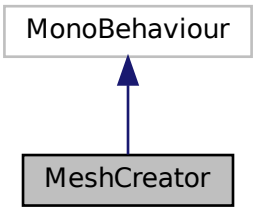
- Assets/Scripts/LogicController/MainController.cs

5.25 MeshCreator Class Reference

Inheritance diagram for MeshCreator:



Collaboration diagram for MeshCreator:



Public Member Functions

- void CreateCubeGridTest (DepthData depthData)

Public Attributes

- Volume volume
- Gradient depthColorGradient

Private Member Functions

- void Start ()
- void Update ()
- void clearVolumeBoxes ()
- void CreateMeshInBoxVolume (DepthData depthData)
- DepthData NormalizeDepthData (DepthData depthData)

5.25.1 Detailed Description

Definition at line 8 of file MeshCreator.cs.

5.25.2 Member Function Documentation

5.25.2.1 clearVolumeBoxes()

```
void MeshCreator.clearVolumeBoxes ( )  [inline], [private]
```

Definition at line 63 of file MeshCreator.cs.

```
64     {
65         foreach (Transform child in volume.transform)
66         {
67             GameObject.Destroy(child.gameObject);
68         }
69     }
```

5.25.2.2 CreateCubeGridTest()

```
void MeshCreator.CreateCubeGridTest (
            DepthData depthData )  [inline]
```

Definition at line 46 of file MeshCreator.cs.

```
47     {
48         clearVolumeBoxes();
49         depthData = NormalizeDepthData(depthData); // Normalize the depth data
50         float localScale = 0.03578f;
51         // Create a cube grid
52 
53         foreach (DepthDataPoint point in depthData.Points)
54         {
55             GameObject quad = GameObject.CreatePrimitive(PrimitiveType.Quad);
56             quad.transform.parent = volume.transform;
57             quad.transform.localPosition = new Vector3(point.X, point.Y, point.Z);
58             quad.transform.localScale = new Vector3(localScale, localScale, 1);  // Note that the z-scale
    is irrelevant for quads.
59             quad.GetComponent<Renderer>().material.color = depthColorGradient.Evaluate(point.Z);
60         }
61     }
```

5.25.2.3 CreateMeshInBoxVolume()

```
void MeshCreator.CreateMeshInBoxVolume (
            DepthData depthData )  [inline], [private]
```

Definition at line 71 of file MeshCreator.cs.

```
72     {
73         depthData = NormalizeDepthData(depthData); // Normalize the depth data
74         Mesh mesh = new Mesh();
75 
76         // Create vertices array
77         Vector3[] vertices = new Vector3[depthData.Count];
78         int[] triangles = new int[(depthData.Count - 2) * 3]; // Only valid if you have at least 3 points
79 
80         // Fill vertices array
81         int i = 0;
82         foreach (DepthDataPoint point in depthData.Points)
83         {
84             vertices[i++] = new Vector3(point.X, point.Y, point.Z);
85         }
86 
87         // Assuming you want to create a mesh from vertices like a connected series of triangles
88         if (depthData.Count >= 3)
89         {
90             for (int j = 0; j < depthData.Count - 2; j++)
91             {
92                 triangles[j * 3] = 0;
93                 triangles[j * 3 + 1] = j + 1;
94                 triangles[j * 3 + 2] = j + 2;
95             }
96         }
97 
98         // Set the mesh vertices and triangles
99         mesh.vertices = vertices;
100         mesh.triangles = triangles;
101         mesh.RecalculateNormals(); // To make sure the mesh is rendered correctly
102 
103         // Assign the mesh to a MeshFilter component
104         MeshFilter meshFilter = GetComponent<MeshFilter>();
105         meshFilter.mesh = mesh;
106     }
```

5.25.2.4 NormalizeDepthData()

```
DepthData MeshCreator.NormalizeDepthData (
            DepthData depthData )  [inline], [private]
```

Definition at line 108 of file MeshCreator.cs.

```
109     {
110         // Optional: Define min and max depth for color normalization
111         float minX = float.MaxValue;
112         float maxX = float.MinValue;
113         float minY = float.MaxValue;
114         float maxY = float.MinValue;
115         float minZ = float.MaxValue;
116         float maxZ = float.MinValue;
117 
118         // Find the minimum and maximum values for each axis
119         foreach (DepthDataPoint point in depthData.Points)
120         {
121             minX = Mathf.Min(minX, point.X);
122             maxX = Mathf.Max(maxX, point.X);
123             minY = Mathf.Min(minY, point.Y);
124             maxY = Mathf.Max(maxY, point.Y);
125             minZ = Mathf.Min(minZ, point.Z);
126             maxZ = Mathf.Max(maxZ, point.Z);
127         }
128 
129         DepthData normalizedDepthData = new DepthData();
130 
131         // Volume scales
132         float scaleX = 1f;
133         float scaleY = 1f;
134         float scaleZ = 1f;
135 
136         // Normalize the depth data to the volume's scales
137         foreach (DepthDataPoint point in depthData.Points)
138         {
139             float normalizedX = Mathf.InverseLerp(minX, maxX, point.X) * scaleX - scaleX / 2;
140             float normalizedY = Mathf.InverseLerp(minY, maxY, point.Y) * scaleY - scaleY / 2;
141             float normalizedZ = Mathf.InverseLerp(minZ, maxZ, point.Z) * scaleZ - scaleZ / 2;
142             normalizedDepthData.AddDepthDataPoint(normalizedX, normalizedY, normalizedZ);
143         }
144 
145         return normalizedDepthData;
146     }
```

5.25.2.5 Start()

```
void MeshCreator.Start ( )  [inline], [private]
```

Definition at line 15 of file MeshCreator.cs.

```
16     {
17         depthColorGradient = new Gradient();
18 
19         // Create gradient keys
20         GradientColorKey[] colorKeys = new GradientColorKey[2];
21         colorKeys[0].color = Color.blue;
22         colorKeys[0].time = 0.0f; // Start of the gradient
23         colorKeys[1].color = Color.red;
24         colorKeys[1].time = 1.0f; // End of the gradient
25 
26         // Create alpha keys
27         GradientAlphaKey[] alphaKeys = new GradientAlphaKey[2];
28         alphaKeys[0].alpha = 1.0f;
29         alphaKeys[0].time = 0.0f;
30         alphaKeys[1].alpha = 1.0f;
31         alphaKeys[1].time = 1.0f;
32 
33         // Assign keys to the gradient
34         depthColorGradient.SetKeys(colorKeys, alphaKeys);
35     }
```

5.25.2.6 Update()

```
void MeshCreator.Update ( )  [inline], [private]
```

Definition at line 38 of file MeshCreator.cs.

```
39     {
40         if (Time.frameCount % 1000 == 0)
41         {
42             clearVolumeBoxes();
43         }
44     }
```

5.25.3 Member Data Documentation

5.25.3.1 depthColorGradient

```
Gradient MeshCreator.depthColorGradient
```

Definition at line 13 of file MeshCreator.cs.

5.25.3.2 volume

```
Volume MeshCreator.volume
```

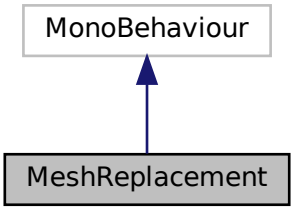
Definition at line 12 of file MeshCreator.cs.

The documentation for this class was generated from the following file:

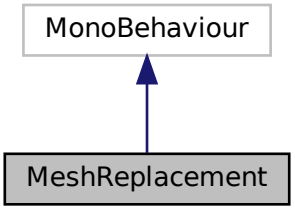
- Assets/Scripts/PlotCameraDepthData/MeshCreator.cs

5.26 MeshReplacement Class Reference

Inheritance diagram for MeshReplacement:



Collaboration diagram for MeshReplacement:



Public Attributes

- GameObject mesh
- GameObject ply

Private Member Functions

- void Start ()
- void Update ()

5.26.1 Detailed Description

Definition at line 6 of file MeshReplacement.cs.

5.26.2 Member Function Documentation

5.26.2.1 Start()

```
void MeshReplacement.Start ( )  [inline], [private]
```

Definition at line 13 of file MeshReplacement.cs.

```
14     {
15     }
```

5.26.2.2 Update()

```
void MeshReplacement.Update ( )  [inline], [private]
```

Definition at line 18 of file MeshReplacement.cs.

```
19     {
20
21     }
```

5.26.3 Member Data Documentation

5.26.3.1 mesh

```
GameObject MeshReplacement.mesh
```

Definition at line 9 of file MeshReplacement.cs.

5.26.3.2 ply

```
GameObject MeshReplacement.ply
```

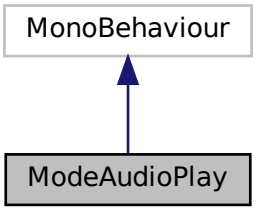
Definition at line 10 of file MeshReplacement.cs.

The documentation for this class was generated from the following file:

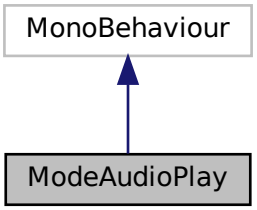
- Assets/Scripts/PlotCameraDepthData/MeshReplacement.cs

5.27 ModeAudioPlay Class Reference

Inheritance diagram for ModeAudioPlay:



Collaboration diagram for ModeAudioPlay:



Public Member Functions

- void PlayIdle ()
- void PlayDrive ()
- void PlayArm ()
- void PlayEmergency ()
- void PlayScrew ()
- void PlayUnScrew ()

Public Attributes

- AudioSource idleSource
- AudioSource driveSource
- AudioSource armSource
- AudioSource emergencySource
- AudioSource screwSource
- AudioSource unScrewSource

Private Member Functions

- void Start ()
- void Update ()

5.27.1 Detailed Description

Definition at line 5 of file ModeAudioPlay.cs.

5.27.2 Member Function Documentation

5.27.2.1 PlayArm()

```
void ModeAudioPlay.PlayArm ( )  [inline]
```

Definition at line 27 of file ModeAudioPlay.cs.

```
28     {
29         armSource.Play();
30     }
```

5.27.2.2 PlayDrive()

```
void ModeAudioPlay.PlayDrive ( )  [inline]
```

Definition at line 22 of file ModeAudioPlay.cs.

```
23     {
24         driveSource.Play();
25     }
```

5.27.2.3 PlayEmergency()

```
void ModeAudioPlay.PlayEmergency ( )  [inline]
```

Definition at line 32 of file ModeAudioPlay.cs.

```
33     {
34         emergencySource.Play();
35     }
```

5.27.2.4 PlayIdle()

```
void ModeAudioPlay.PlayIdle ( )  [inline]
```

Definition at line 17 of file ModeAudioPlay.cs.

```
18     {
19         idleSource.Play();
20     }
```

5.27.2.5 PlayScrew()

```
void ModeAudioPlay.PlayScrew ( )  [inline]
```

Definition at line 37 of file ModeAudioPlay.cs.

```
38     {
39         screwSource.Play();
40     }
```

5.27.2.6 PlayUnScrew()

```
void ModeAudioPlay.PlayUnScrew ( )  [inline]
```

Definition at line 42 of file ModeAudioPlay.cs.

```
43     {
44         unScrewSource.Play();
45     }
```

5.27.2.7 Start()

```
void ModeAudioPlay.Start ( )  [inline], [private]
```

Definition at line 47 of file ModeAudioPlay.cs.

```
48     {
49
50     }
```

5.27.2.8 Update()

```
void ModeAudioPlay.Update ( )  [inline], [private]
```

Definition at line 53 of file ModeAudioPlay.cs.

```
54     {
55
56     }
```

5.27.3 Member Data Documentation

5.27.3.1 armSource

```
AudioSource ModeAudioPlay.armSource
```

Definition at line 11 of file ModeAudioPlay.cs.

5.27.3.2 driveSource

```
AudioSource ModeAudioPlay.driveSource
```

Definition at line 10 of file ModeAudioPlay.cs.

5.27.3.3 emergencySource

```
AudioSource ModeAudioPlay.emergencySource
```

Definition at line 12 of file ModeAudioPlay.cs.

5.27.3.4 idleSource

```
AudioSource ModeAudioPlay.idleSource
```

Definition at line 9 of file ModeAudioPlay.cs.

5.27.3.5 screwSource

```
AudioSource ModeAudioPlay.screwSource
```

Definition at line 13 of file ModeAudioPlay.cs.

5.27.3.6 unScrewSource

```
AudioSource ModeAudioPlay.unScrewSource
```

Definition at line 14 of file ModeAudioPlay.cs.

The documentation for this class was generated from the following file:

- Assets/Voice Controll/ModeAudioPlay.cs

5.28 DropdownHandler.ModeValues Class Reference

The ModeValues interface used to send the mode value to the robot.

Public Attributes

- int mode

5.28.1 Detailed Description

The ModeValues interface used to send the mode value to the robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 25 of file DropdownHandler.cs.

5.28.2 Member Data Documentation

5.28.2.1 mode

```
int DropdownHandler.ModeValues.mode
```

Definition at line 27 of file DropdownHandler.cs.

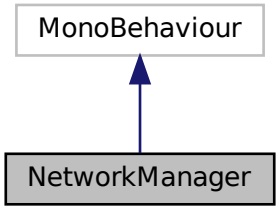
The documentation for this class was generated from the following file:

- Assets/Scripts/DropdownHandler.cs

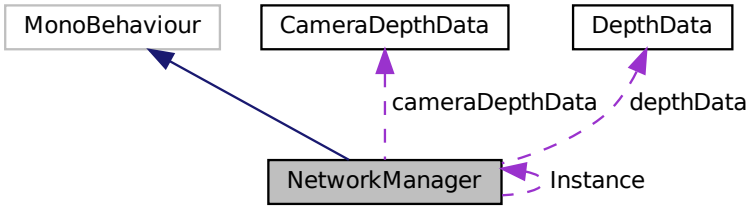
5.29 NetworkManager Class Reference

Class `NetworkManager` Manages network communications for the application, implementing a singleton pattern to ensure only one instance exists.

Inheritance diagram for NetworkManager:



Collaboration diagram for NetworkManager:



Public Member Functions

- bool SendData (string data)

  Sends data to the server.
- byte[ ] DecompressBrotli (byte[ ] compressedData)

  Decodes a base64 string to a byte array.
- byte[ ] DecodeBase64 (string base64EncodedData)

  Decodes a base64 string to a byte array.
- void Reconnect ()

  Reconnects to the server, closing the existing connection if any.

Public Attributes

- TextAsset brotliTestFile
- string serverIP = "192.168.1.6"

  The server IP address.

Static Public Attributes

- static NetworkManager Instance

  Singleton instance of the NetworkManager class.

Events

- Action< string > OnDataReceived

  Event to handle received data.
- Action OnConnected

  Emit event when connected to the server.
- Action< bool > OnConnectionStatus

  Event connection status
- Action< JsonRobotInfo > OnRobotInfoDataReceived
- Action< DepthData > onDepthDataReceived
- Action< JsonArmLengthInfo > onArmLengthDataReceived
- Action< float > onPingDataReceived

Private Member Functions

- void Awake ()

  Awake is called when the script instance is being loaded.
- void Start ()

  Start is called before the first frame update.
- IEnumerator InvokeConnectionStatus ()

  Coroutine for invoking event for connection status
- IEnumerator PingRobot ()

  Coroutine for pinging the robot each second
- void ConnectToServer ()

  Connects to the server.
- void ReceiveData ()
- async Task ReceiveDataAsync ()
- void OnApplicationQuit ()

  Called when the application quits.
- void Update ()
- void processRecievedData (OVRSimpleJSON.JSONNode jsonData)
- void ProcessPingData (OVRSimpleJSON.JSONNode jsonData)

  process ping data
- bool IsBrotli (string data)

  Checks if the data is compressed using Brotli.
- IEnumerator ProcessDataCoroutine (List< string > batchData)

  For testing
- JsonRobotInfo ParseJsonRobotInfo (OVRSimpleJSON.JSONNode jsonNode)

  Parse the depth data from the JSON node
- JsonArmLengthInfo ParseArmLengthInfo (OVRSimpleJSON.JSONNode jsonNode)
- List< float > ParseFloatList (OVRSimpleJSON.JSONNode node)
- void ProcessDepthData (OVRSimpleJSON.JSONNode jsonData)

  Parse the depth data from the JSON node
- void OnDestroy ()

  Called when the object is destroyed.
- void Disconnect ()

  Disconnects from the server.

Private Attributes

- ConcurrentQueue< string > receivedDataQueue = new ConcurrentQueue<string>()

  Queue to store received data.
- TcpClient client

  The TCP client for the network connection.
- NetworkStream stream

  The network stream for reading and writing data to the server.
- int port = 8080

  The port number for the server.
- int udpPort = 12000

  UDP port
- Thread receiveThread

  The receive thread for processing incoming data.
- bool isListening = false

  Flag to indicate if the client is listening for incoming data.
- bool connected = false
- CameraDepthData cameraDepthData = new CameraDepthData()

  The camera depth data.
- DepthData depthData = new DepthData()

  Camera Depth data
- int sizeBeforeUpdate = 10000

  Size before updating camera depth data
- byte[ ] broHeader = new byte[ ] { (byte)'b', (byte)'r', (byte)'o' }

  Receives data from the server.

5.29.1 Detailed Description

Class `NetworkManager` Manages network communications for the application, implementing a singleton pattern to ensure only one instance exists.

This class handles the creation of a TCP client, manages connections, and processes incoming data asynchronously.

Definition at line 135 of file NetworkManager.cs.

5.29.2 Member Function Documentation

5.29.2.1 Awake()

```
void NetworkManager.Awake ( )  [inline], [private]
```

Awake is called when the script instance is being loaded.

Definition at line 241 of file NetworkManager.cs.

```
242     {
243         if (depthData == null)
244         {
245             depthData = new DepthData();
246         }
247         if (Instance == null)
248         {
249             Instance = this;
250             DontDestroyOnLoad(gameObject); // Keep the instance alive across scenes
251         }
252         else if (Instance != this)
253         {
254             Destroy(gameObject); // Ensures only one instance exists
255         }
256     }
```

5.29.2.2 ConnectToServer()

```
void NetworkManager.ConnectToServer ( )  [inline], [private]
```

Connects to the server.

Starts also the receiving thread to process incoming data.

Definition at line 307 of file NetworkManager.cs.

```
308     {
309         try
310         {
311             client = new TcpClient(serverIP, port);
312             if (client.Connected)
313             {
314                 stream = client.GetStream();
315                 isListening = true;
316 
317                 // Start the receiving thread
318                 receiveThread = new Thread(new ThreadStart(ReceiveData));
319                 // receiveThread = new Thread(new ThreadStart(ReceiveDataAsync));
320                 receiveThread.IsBackground = true;
321                 receiveThread.Start();
322 
323                 // await ReceiveDataAsync();
324                 Debug.Log("Connected to server.");
325                 OnConnected?.Invoke();
326             }
327         }
328         catch (Exception e)
329         {
330             Debug.LogError("Error connecting to server: " + e.Message);
331         }
332     }
```

5.29.2.3 DecodeBase64()

```
byte [] NetworkManager.DecodeBase64 (
            string base64EncodedData )  [inline]
```

Decodes a base64 string to a byte array.

Parameters

base64EncodedData	

Returns

Definition at line 581 of file NetworkManager.cs.

```
582     {
583         try
584         {
585             byte[] data = Convert.FromBase64String(base64EncodedData);
586             return data;
587         }
588         catch (FormatException ex)
589         {
590             Debug.Log($"Base64 string is not in a valid format: {ex.Message}");
591             return null;
592         }
593     }
```

5.29.2.4 DecompressBrotli()

```
byte [] NetworkManager.DecompressBrotli (
            byte[] compressedData )  [inline]
```

Decodes a base64 string to a byte array.

Parameters

compressedData	

Returns

Definition at line 564 of file NetworkManager.cs.

```
565     {
566         using (var inputStream = new MemoryStream(compressedData))
567         using (var outputStream = new MemoryStream())
568         using (var brotliStream = new BrotliStream(inputStream, CompressionMode.Decompress))
569         {
570             brotliStream.CopyToAsync(outputStream);
571             return outputStream.ToArray();
572         }
573
574     }
```

5.29.2.5 Disconnect()

```
void NetworkManager.Disconnect ( )  [inline], [private]
```

Disconnects from the server.

Definition at line 759 of file NetworkManager.cs.

```
760     {
761         isListening = false;
762 
763         // Wait for the receive thread to finish, if it's running
764         if (receiveThread != null && receiveThread.IsAlive)
765         {
766             receiveThread.Join();
767         }
768 
769         // Close the client connection and the stream
770         if (client != null)
771         {
772             if (stream != null)
773             {
774                 stream.Close();
775                 stream = null;
776             }
777             client.Close();
778             client = null;
779         }
780 
781         Debug.Log("Disconnected from server.");
782     }
```

5.29.2.6 InvokeConnectionStatus()

```
IEnumerator NetworkManager.InvokeConnectionStatus ( )  [inline], [private]
```

Coroutine for invoking event for connection status

Definition at line 278 of file NetworkManager.cs.

```
279     {
280         while (true)
281         {
282             OnConnectionStatus?.Invoke(connected);
283             connected = false;
284             yield return new WaitForSeconds(5);
285         }
286     }
```

5.29.2.7 IsBrotli()

```
bool NetworkManager.IsBrotli (
            string data )  [inline], [private]
```

Checks if the data is compressed using Brotli.

Parameters

data	

Returns

Definition at line 554 of file NetworkManager.cs.

```
555     {
556         return data.Contains("brotli");
557     }
```

5.29.2.8 OnApplicationQuit()

```
void NetworkManager.OnApplicationQuit ( )  [inline], [private]
```

Called when the application quits.

Closes the client connection and the receive thread.

Definition at line 440 of file NetworkManager.cs.

```
441     {
442         isListening = false;
443         if (receiveThread != null && receiveThread.IsAlive)
444         {
445             receiveThread.Join();
446         }
447 
448         if (client != null)
449         {
450             client.Close();
451         }
452     }
```

5.29.2.9 OnDestroy()

```
void NetworkManager.OnDestroy ( )  [inline], [private]
```

Called when the object is destroyed.

Closes the client connection as well.

Definition at line 738 of file NetworkManager.cs.

```
739     {
740         if (client != null)
741             client.Close();
742     }
```

5.29.2.10 ParseArmLengthInfo()

```
JsonArmLengthInfo NetworkManager.ParseArmLengthInfo (
            OVRSimpleJSON.JSONNode jsonNode )  [inline], [private]
```

Definition at line 689 of file NetworkManager.cs.

```
690     {
691         return new JsonArmLengthInfo
692         {
693             Link_1 = jsonNode["rotation"].AsInt,
694             Link_2 = jsonNode["shoulder"].AsInt,
695             Link_3 = jsonNode["elbow"].AsInt,
696             Link_4 = jsonNode["tilt"].AsInt,
697             Link_5 = jsonNode["wrist"].AsInt,
698             pintch = jsonNode["pinch"].AsInt
699         };
700     }
```

5.29.2.11 ParseFloatList()

```
List<float> NetworkManager.ParseFloatList (
            OVRSimpleJSON.JSONNode node )  [inline], [private]
```

Definition at line 703 of file NetworkManager.cs.

```
704     {
705         List<float> list = new List<float>();
706         if (node.IsArray)
707         {
708             foreach (OVRSimpleJSON.JSONNode n in node.AsArray)
709             {
710                 list.Add(n.AsFloat);
711             }
712         }
713         else
714         {
715             Debug.LogError("Node is not an array");
716         }
717         return list;
718     }
```

5.29.2.12 ParseJsonRobotInfo()

```
JsonRobotInfo NetworkManager.ParseJsonRobotInfo (
            OVRSimpleJSON.JSONNode jsonNode )  [inline], [private]
```

Parse the depth data from the JSON node

Parameters

jsonNode	

Returns

Definition at line 673 of file NetworkManager.cs.

```
674     {
675         return new JsonRobotInfo
676         {
677             accelerometer = ParseFloatList(jsonNode["accelerometer"]),
678             gyroscope = ParseFloatList(jsonNode["gyroscope"]),
679             magnetometer = ParseFloatList(jsonNode["magnetometer"]),
680             motion = ParseFloatList(jsonNode["motion"]),
681             speed = jsonNode["speed"].AsFloat,
682             voltage = jsonNode["voltage"].AsFloat,
683             battery_precentage = jsonNode["battery"],
684             mode = jsonNode["mode"],
685             cms_speed = jsonNode["cms_speed"].AsFloat
686         };
687     }
```

5.29.2.13 PingRobot()

```
IEnumerator NetworkManager.PingRobot ( )  [inline], [private]
```

Coroutine for pinging the robot each second

Definition at line 291 of file NetworkManager.cs.

```
292     {
293         while (true)
294         {
295             PingData pingData = new PingData();
296             pingData.ping = Time.realtimeSinceStartup;
297             SendData(JsonUtility.ToJson(pingData));
298 
299             yield return new WaitForSeconds(1);
300         }
301     }
```

5.29.2.14 ProcessDataCoroutine()

```
IEnumerator NetworkManager.ProcessDataCoroutine (
            List< string > batchData )  [inline], [private]
```

For testing

Definition at line 620 of file NetworkManager.cs.

```
621     {
622         foreach (string recievedData in batchData)
623         {
624             try
625             {
626                 var jsonData = OVRSimpleJSON.JSON.Parse(recievedData);
627                 if (jsonData != null)
628                 {
629                     if (jsonData.HasKey("type"))
630                     {
631 
632                         if (jsonData["type"] == "log")
633                         {
634 
635                         }
636                         else if (jsonData["type"] == "depth")
637                         {
638                             // CameraDepthData depthData = ParseCameraDepthData(jsonData);
639                             // ParseCameraDepthData(jsonData);
640 
641                             // if (cameraDepthData.depthData.Count > sizeBeforeUpdate)
642                             // {
643                             //     // onDepthDataReceived?.Invoke(cameraDepthData);
644                             //     cameraDepthData.depthData.Clear();
645                             // }
646                         }
647                         else if (jsonData["type"] == "robot_data")
648                         {
649                             JsonRobotInfo info = ParseJsonRobotInfo(jsonData);
650                             OnRobotInfoDataReceived?.Invoke(info);
651                         }
652                     }
653                 }
654                 else
655                 {
656                     Debug.LogError("Error parsing JSON: " + recievedData);
657                 }
658             }
659             catch (Exception e)
660             {
661                 Debug.LogError($"Error parsing JSON: {e.Message}");
662             }
663         }
664         yield return null;
665 
666     }
```

5.29.2.15 ProcessDepthData()

```
void NetworkManager.ProcessDepthData (
            OVRSimpleJSON.JSONNode jsonData )  [inline], [private]
```

Parse the depth data from the JSON node

Parameters

jsonNode	

Definition at line 724 of file NetworkManager.cs.

```
725     {
726         depthData.ClearPoints();
727         depthData.time = jsonData["time"].AsInt;
728         Debug.Log($"ProcessDepthData res time: {depthData.time}");
729 
730         for (var i = 0; i < jsonData["data"][0].Count; i++)
731         {
732             depthData.AddDepthDataPoint(jsonData["data"][0][i].AsInt, jsonData["data"][1][i].AsInt,
      jsonData["data"][2][i].AsInt);
733         }
734     }
```

5.29.2.16 ProcessPingData()

```
void NetworkManager.ProcessPingData (
            OVRSimpleJSON.JSONNode jsonData )  [inline], [private]
```

process ping data

Parameters

jsonData	

Definition at line 541 of file NetworkManager.cs.

```
542     {
543         float latency = Mathf.Round((Time.realtimeSinceStartup - jsonData["ping"].AsFloat) * 100000) /
      100;
544         Debug.Log($"LatencyNetwork: {latency}");
545         onPingDataReceived?.Invoke(latency);
546         connected = true;
547     }
```

5.29.2.17 processRecievedData()

```
void NetworkManager.processRecievedData (
            OVRSimpleJSON.JSONNode jsonData )  [inline], [private]
```

Definition at line 482 of file NetworkManager.cs.

```
483     {
484         if (!jsonData.HasKey("type"))
485         {
486             Debug.Log($"No type key found in the JSON data: a{jsonData}");
487             return;
488         }
489         // switch (jsonData["type"].ToString())
490         // {
491         //     case "log":
492         //         // TODO process log data
493         //         break;
494         //     case "robot_data":
495         //         JsonRobotInfo info = ParseJsonRobotInfo(jsonData);
496         //         OnRobotInfoDataReceived?.Invoke(info);
497         //         break;
```

```
498         //     case "depth":
499         //         ProcessDepthData(jsonData);
500         //         onDepthDataReceived?.Invoke(depthData);
501         //         break;
502         //     default:
503         //         Debug.Log($"Unknown data type: {jsonData["type"].ToString()}");
504         //         break;
505         // }
506 
507         if (jsonData["type"] == "log")
508         {
509             Debug.Log($"Log from robot: {jsonData}");
510             // Logger.Log(jsonData);  // Store the loging data in a file
511         }
512         else if (jsonData["type"] == "ping")
513         {
514             ProcessPingData(jsonData);
515         }
516         else if (jsonData["type"] == "robot_data")
517         {
518             JsonRobotInfo info = ParseJsonRobotInfo(jsonData);
519             OnRobotInfoDataReceived?.Invoke(info);
520         }
521         else if (jsonData["type"] == "depth")
522         {
523             ProcessDepthData(jsonData);
524             Debug.Log($"Pricess res Depth data: {jsonData}");
525             onDepthDataReceived?.Invoke(depthData);
526         }
527         else if (jsonData["type"] == "arm_angles")
528         {
529             JsonArmLengthInfo armLengthInfo = ParseArmLengthInfo(jsonData);
530             onArmLengthDataReceived?.Invoke(armLengthInfo);
531             Debug.Log($"Arm angels data: {jsonData}");
532         }
533     }
```

5.29.2.18 ReceiveData()

```
void NetworkManager.ReceiveData ( )  [inline], [private]
```

Definition at line 367 of file NetworkManager.cs.

```
368     {
369         byte[] buffer = new byte[4096];
370         int byteLength;
371         string dataReceived = string.Empty;
372 
373         while (isListening && client != null && client.Connected)
374         {
375             try
376             {
377                 if (stream.DataAvailable)
378                 {
379                     byteLength = stream.Read(buffer, 0, buffer.Length);
380                     if (buffer.Take(3).SequenceEqual(broHeader))
381                     {
382                         byte[] compressedData = DecompressBrotli(buffer.Skip(3).ToArray());  // Skip the
    first 3 bytes which are the header
383                         if (compressedData != null)
384                         {
385                             dataReceived = Encoding.UTF8.GetString(compressedData);  // Decode the
    entire decompressed data
386                         }
387                     }
388                     else
389                     {
390                         dataReceived = Encoding.ASCII.GetString(buffer, 0, byteLength);
391                     }
392                     receivedDataQueue.Enqueue(dataReceived);
393                 }
394             }
395             catch (Exception e)
396             {
397                 Debug.LogError($"Error receiving data: A{e.Message}");
398                 isListening = false;
399             }
400         }
401     }
```

5.29.2.19 ReceiveDataAsync()

```
async Task NetworkManager.ReceiveDataAsync ( )  [inline], [private]
```

Definition at line 403 of file NetworkManager.cs.

```
404     {
405         byte[] buffer = new byte[8192]; // Increased buffer size
406 
407         try
408         {
409             while (isListening && client != null && client.Connected)
410             {
411                 if (stream.DataAvailable)
412                 {
413 
414                     int byteLength = await stream.ReadAsync(buffer, 0, buffer.Length);
415                     if (byteLength > 0)
416                     {
417                         string dataReceived = Encoding.ASCII.GetString(buffer, 0, byteLength);
418                         lock (receivedDataQueue)
419                         {
420                             receivedDataQueue.Enqueue(dataReceived);
421                         }
422                         // Debug.Log($"Received data: {dataReceived}");
423                     }
424                     else
425                     {
426                     }
427                 }
428                 await Task.Delay(10); // Yield to maintain responsiveness, adjust timing as necessary
429             }
430         }
431         catch (Exception e)
432         {
433             Debug.LogError($"Error receiving data asynchronously: {e.Message}");
434             isListening = false;
435         }
436     }
```

5.29.2.20 Reconnect()

```
void NetworkManager.Reconnect ( )  [inline]
```

Reconnects to the server, closing the existing connection if any.

Definition at line 747 of file NetworkManager.cs.

```
748     {
749         // First, disconnect existing connection if any
750         Disconnect();
751 
752         // Then, attempt to reconnect
753         ConnectToServer();
754     }
```

5.29.2.21 SendData()

```
bool NetworkManager.SendData (
            string data )  [inline]
```

Sends data to the server.

Parameters

data	

Returns

Definition at line 340 of file NetworkManager.cs.

```
341     {
342         if (stream == null)
343         {
344             Debug.LogError("Network stream is not available.");
345             return false;
346         }
347 
348         try
349         {
350             byte[] bytes = Encoding.ASCII.GetBytes(data);
351             stream.Write(bytes, 0, bytes.Length);
352             Debug.Log($"Sent: {data}");
353             return true;
354         }
355         catch (Exception e)
356         {
357             Debug.LogError($"Error sending data: {e.Message}");
358             return false;
359         }
360     }
```

5.29.2.22 Start()

```
void NetworkManager.Start ( )  [inline], [private]
```

Start is called before the first frame update.

Connects to the server when the application starts.

Definition at line 262 of file NetworkManager.cs.

```
263     {
264         // ConnectToServer();
265         Debug.Log($"Network manager is a live: {true}");
266         // if (depthData == null)
267         // {
268         //     depthData = new DepthData();
269         // }
270         StartCoroutine(PingRobot());
271         StartCoroutine(InvokeConnectionStatus());
272     }
```

5.29.2.23 Update()

```
void NetworkManager.Update ( )  [inline], [private]
```

Definition at line 457 of file NetworkManager.cs.

```
458     {
459         // Process all pending messages
460         while (receivedDataQueue.TryDequeue(out string receivedData))
461         {
462             try
463             {
464                 var jsonData = OVRSimpleJSON.JSON.Parse(receivedData);
465                 if (jsonData != null)
466                 {
467                     processRecievedData(jsonData);
468                 }
469                 else
470                 {
471                     Debug.Log($"Error parsing JSON: {receivedData}");
472                 }
473             }
474             catch (Exception e)
475             {
476                 Debug.LogError($"Error parsing JSON: {e.Message}");
477             }
478         }
479     }
```

5.29.3 Member Data Documentation

5.29.3.1 broHeader

```
byte [] NetworkManager.broHeader = new byte[] { (byte)'b', (byte)'r', (byte)'o' }  [private]
```

Receives data from the server.

This method runs on a separate thread.

Definition at line 366 of file NetworkManager.cs.

5.29.3.2 brotliTestFile

```
TextAsset NetworkManager.brotliTestFile
```

Definition at line 138 of file NetworkManager.cs.

5.29.3.3 cameraDepthData

```
CameraDepthData NetworkManager.cameraDepthData = new CameraDepthData()  [private]
```

The camera depth data.

Definition at line 225 of file NetworkManager.cs.

5.29.3.4 client

```
TcpClient NetworkManager.client  [private]
```

The TCP client for the network connection.

Definition at line 160 of file NetworkManager.cs.

5.29.3.5 connected

```
bool NetworkManager.connected = false  [private]
```

Definition at line 212 of file NetworkManager.cs.

5.29.3.6 depthData

```
DepthData NetworkManager.depthData = new DepthData()  [private]
```

Camera Depth data

Definition at line 230 of file NetworkManager.cs.

5.29.3.7 Instance

```
NetworkManager NetworkManager.Instance  [static]
```

Singleton instance of the NetworkManager class.

```
NetworkManager = NetworkManager.Instance;
```

Definition at line 145 of file NetworkManager.cs.

5.29.3.8 isListening

```
bool NetworkManager.isListening = false  [private]
```

Flag to indicate if the client is listening for incoming data.

true

if the client is listening; otherwise,

false

Definition at line 193 of file NetworkManager.cs.

5.29.3.9 port

```
int NetworkManager.port = 8080  [private]
```

The port number for the server.

Definition at line 177 of file NetworkManager.cs.

5.29.3.10 receivedDataQueue

```
ConcurrentQueue<string> NetworkManager.receivedDataQueue = new ConcurrentQueue<string>()
[private]
```

Queue to store received data.

The received data queue.

Definition at line 155 of file NetworkManager.cs.

5.29.3.11 receiveThread

```
Thread NetworkManager.receiveThread  [private]
```

The receive thread for processing incoming data.

Definition at line 187 of file NetworkManager.cs.

5.29.3.12 serverIP

```
string NetworkManager.serverIP = "192.168.1.6"
```

The server IP address.

Definition at line 171 of file NetworkManager.cs.

5.29.3.13 sizeBeforeUpdate

```
int NetworkManager.sizeBeforeUpdate = 10000  [private]
```

Size before updating camera depth data

Definition at line 236 of file NetworkManager.cs.

5.29.3.14 stream

```
NetworkStream NetworkManager.stream  [private]
```

The network stream for reading and writing data to the server.

Definition at line 165 of file NetworkManager.cs.

5.29.3.15 udpPort

```
int NetworkManager.udpPort = 12000  [private]
```

UDP port

Definition at line 182 of file NetworkManager.cs.

5.29.4 Event Documentation

5.29.4.1 onArmLengthDataReceived

```
Action<JsonArmLengthInfo> NetworkManager.onArmLengthDataReceived
```

Definition at line 218 of file NetworkManager.cs.

5.29.4.2 OnConnected

```
Action NetworkManager.OnConnected
```

Emit event when connected to the server.

Definition at line 205 of file NetworkManager.cs.

5.29.4.3 OnConnectionStatus

```
Action<bool> NetworkManager.OnConnectionStatus
```

Event connection status

Definition at line 210 of file NetworkManager.cs.

5.29.4.4 OnDataReceived

```
Action<string> NetworkManager.OnDataReceived
```

Event to handle received data.

```
NetworkManager.OnDataReceived += OnDataReceivedHandler;
```

Definition at line 199 of file NetworkManager.cs.

5.29.4.5 onDepthDataReceived

```
Action<DepthData> NetworkManager.onDepthDataReceived
```

Definition at line 216 of file NetworkManager.cs.

5.29.4.6 onPingDataReceived

```
Action<float> NetworkManager.onPingDataReceived
```

Definition at line 221 of file NetworkManager.cs.

5.29.4.7 OnRobotInfoDataReceived

```
Action<JsonRobotInfo> NetworkManager.OnRobotInfoDataReceived
```

Definition at line 214 of file NetworkManager.cs.

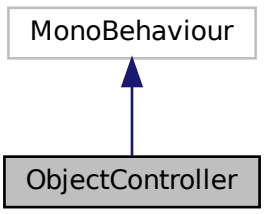
The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

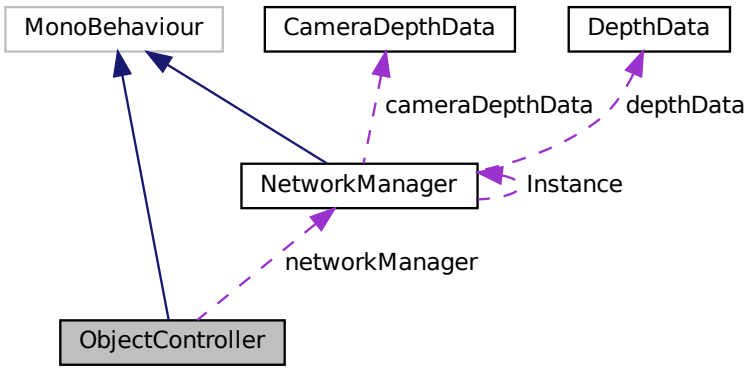
5.30 ObjectController Class Reference

This class is used to send data to the server by pressing the space key.

Inheritance diagram for ObjectController:



Collaboration diagram for ObjectController:



Private Member Functions

- void Start ()

  Initialization of the script by finding the NetworkManager component in the scene.
- void Update ()

  Update is called once per frame and is used to send data to the server when the space key is pressed.
- void SendDataToServer (string data)

  This method is used to send data to the server.

Private Attributes

- NetworkManager networkManager

  The NetworkManager component used to send data to the server.

5.30.1 Detailed Description

This class is used to send data to the server by pressing the space key.

It is used only for testing the communication with the server.

This class is used for testing purposes.

Definition at line 8 of file ObjectController.cs.

5.30.2 Member Function Documentation

5.30.2.1 SendDataToServer()

```
void ObjectController.SendDataToServer (
            string data )  [inline], [private]
```

This method is used to send data to the server.

Definition at line 40 of file ObjectController.cs.

```
41     {
42         if (networkManager != null)
43         {
44             networkManager.SendData(data);
45         }
46         else
47         {
48             Debug.LogError("NetworkManager component not found.");
49         }
50 
51 
52     }
```

5.30.2.2 Start()

```
void ObjectController.Start ( )  [inline], [private]
```

Initialization of the script by finding the NetworkManager component in the scene.

Definition at line 19 of file ObjectController.cs.

```
20     {
21         // Get the singelton instance of the NetworkManager
22         networkManager = NetworkManager.Instance;
23     }
```

5.30.2.3 Update()

```
void ObjectController.Update ( )  [inline], [private]
```

Update is called once per frame and is used to send data to the server when the space key is pressed.

Definition at line 28 of file ObjectController.cs.

```
29     {
30         if (Input.GetKeyDown(KeyCode.Space))
31         {
32             SendDataToServer("Space key pressed!");
33         }
34         // SendDataToServer("hello world!");
35     }
```

5.30.3 Member Data Documentation

5.30.3.1 networkManager

```
NetworkManager ObjectController.networkManager  [private]
```

The NetworkManager component used to send data to the server.

Definition at line 14 of file ObjectController.cs.

The documentation for this class was generated from the following file:

- Assets/ObjectController.cs

5.31 PingData Class Reference

Public Attributes

- string type = "ping"
- float ping

5.31.1 Detailed Description

Definition at line 87 of file NetworkManager.cs.

5.31.2 Member Data Documentation

5.31.2.1 ping

```
float PingData.ping
```

Definition at line 90 of file NetworkManager.cs.

5.31.2.2 type

```
string PingData.type = "ping"
```

Definition at line 89 of file NetworkManager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/NetworkNamager/NetworkManager.cs

5.32 DepthPointCloud.RequestDepthDataMsg Class Reference

Interface for requesting depth camera from the robot

Public Attributes

- bool get_depth

5.32.1 Detailed Description

Interface for requesting depth camera from the robot

Definition at line 97 of file DepthPointCloud.cs.

5.32.2 Member Data Documentation

5.32.2.1 get_depth

```
bool DepthPointCloud.RequestDepthDataMsg.get_depth
```

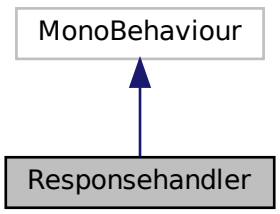
Definition at line 99 of file DepthPointCloud.cs.

The documentation for this class was generated from the following file:

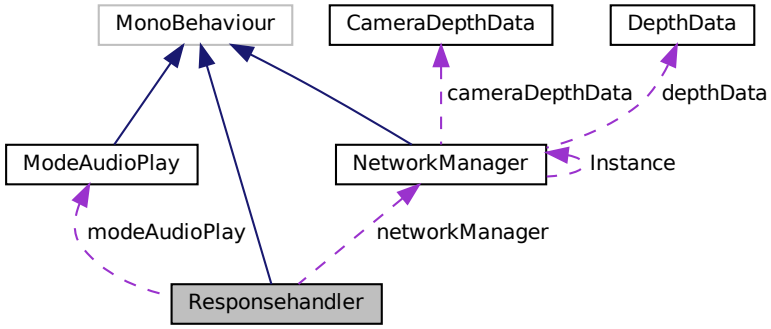
- Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs

5.33 Responsehandler Class Reference

Inheritance diagram for Responsehandler:



Collaboration diagram for Responsehandler:



Classes

- class ScrewCommand

Public Member Functions

- void HandleResponseTest (WitResponseNode response)
- void OnStartListening ()
- void OnEmergencyVoiceCommandActivated (string[] em)
- void OnEmergency (string emergency)

Public Attributes

- AppVoiceExperience voiceExperience
- ModeAudioPlay modeAudioPlay

Events

- Action< int > OnVoiceCommandReceived

Private Member Functions

- void Start ()
- void sendScrewCommandToTheTobot (string screw)
- void Update ()

Private Attributes

- NetworkManager networkManager
- int index = 0

5.33.1 Detailed Description

Definition at line 9 of file Responsehandler.cs.

5.33.2 Member Function Documentation

5.33.2.1 HandleResponseTest()

```
void Responsehandler.HandleResponseTest (
            WitResponseNode response )  [inline]
```

Definition at line 35 of file Responsehandler.cs.

```
36     {
37         // string value = response["entities"]["intent"][0]["value"];
38 
39         string value = response["entities"];
40         Debug.Log($"ResponseKromium: {response}");
41         Debug.Log($"Value Response: {value}");
42         Debug.Log($"Got Response from voice command: {true}");
43         string command = "Voice command" + index++;
44         // OnVoiceCommandReceived?.Invoke(command);
45     }
```

5.33.2.2 OnEmergency()

```
void Responsehandler.OnEmergency (
            string emergency )  [inline]
```

Definition at line 97 of file Responsehandler.cs.

```
98     {
99         Debug.Log($"Vocie controll Listening Emergency: {emergency}  +{index++}");
100     }
```

5.33.2.3 OnEmergencyVoiceCommandActivated()

```
void Responsehandler.OnEmergencyVoiceCommandActivated (
            string[] em )  [inline]
```

Definition at line 52 of file Responsehandler.cs.

```
53     {
54         Debug.Log($"Emerg index: {index++}");
55         Debug.Log($"Vocie controll Listening Emergency STRING NAME: {em[0]}");
56 
57         int mode = -1;
58         switch (em[0].ToLower())
59         {
60             case "idle":
61                 Debug.Log($"Vocie controll Listening Emergency: {em[0]}");
62                 mode = 0;
63                 break;
64             case "drive":
65                 mode = 1;
66                 Debug.Log($"Vocie controll Listening Emergency: {em[0]}");
67                 break;
```

```
68              case "arm":
69                  mode = 2;
70                  break;
71              case "emergency":
72                  mode = 3;
73                  Debug.Log($"Vocie controll Listening Emergency: {em[0]}");
74                  break;
75              case "screw":
76                  sendScrewCommandToTheTobot("right");
77                  modeAudioPlay.PlayScrew();
78                  break;
79              case "unscrew":
80                  sendScrewCommandToTheTobot("left");
81                  modeAudioPlay.PlayUnScrew();
82                  break;
83          }
84
85          if (mode != -1)
86          {
87              OnVoiceCommandReceived?.Invoke(mode);
88          }
89      }
```

5.33.2.4 OnStartListening()

```
void Responsehandler.OnStartListening ( )  [inline]
```

Definition at line 47 of file Responsehandler.cs.

```
48      {
49          Debug.Log($"Vocie controll Listening: {index++}");
50      }
```

5.33.2.5 sendScrewCommandToTheTobot()

```
void Responsehandler.sendScrewCommandToTheTobot (
            string screw )  [inline], [private]
```

Definition at line 90 of file Responsehandler.cs.

```
91      {
92          ScrewCommand screwCommand = new ScrewCommand();
93          screwCommand.screw = screw;
94          networkManager.SendData(JsonUtility.ToJson(screwCommand));
95
96      }
```

5.33.2.6 Start()

```
void Responsehandler.Start ( )  [inline], [private]
```

Definition at line 17 of file Responsehandler.cs.

```
18      {
19          // voiceExperience.OnVoiceCommandReceived += HandleResponseTest;
20          // voiceExperience.OnStartListening += OnStartListening;
21          // voiceExperience.OnEmergencyVoiceCommandActivated += OnEmergencyVoiceCommandActivated;
22          // voiceExperience.OnEmergency += OnEmergency;
23          networkManager = NetworkManager.Instance;
24      }
```

5.33.2.7 Update()

```
void Responsehandler.Update ( )  [inline], [private]
```

Definition at line 106 of file Responsehandler.cs.

```
107     {
108         if (!voiceExperience.Active)
109         {
110             voiceExperience.Activate();
111         }
112     }
```

5.33.3 Member Data Documentation

5.33.3.1 index

```
int Responsehandler.index = 0  [private]
```

Definition at line 31 of file Responsehandler.cs.

5.33.3.2 modeAudioPlay

```
ModeAudioPlay Responsehandler.modeAudioPlay
```

Definition at line 15 of file Responsehandler.cs.

5.33.3.3 networkManager

```
NetworkManager Responsehandler.networkManager  [private]
```

Definition at line 12 of file Responsehandler.cs.

5.33.3.4 voiceExperience

```
AppVoiceExperience Responsehandler.voiceExperience
```

Definition at line 11 of file Responsehandler.cs.

5.33.4 Event Documentation

5.33.4.1 OnVoiceCommandReceived

```
Action<int> Responsehandler.OnVoiceCommandReceived
```

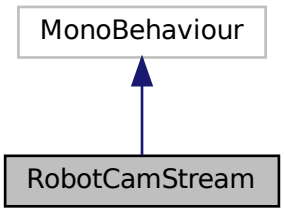
Definition at line 33 of file Responsehandler.cs.

The documentation for this class was generated from the following file:

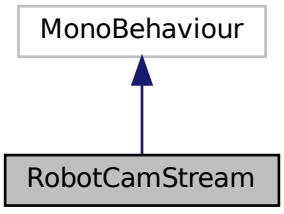
- Assets/Voice Controll/Responsehandler.cs

5.34 RobotCamStream Class Reference

Inheritance diagram for RobotCamStream:



Collaboration diagram for RobotCamStream:



Public Attributes

- string streamUrl = "http://192.168.1.5:8000/stream.m3u8"
- int maxRetries = 3
- float retryDelay = 2f

Private Member Functions

- void Start ()

  Start is called before the first frame and gets the VideoPlayer component and sets up the video player.
- void SetupVideoPlayer ()

  Sets up the video player with the stream URL and other settings.
- void TryPrepareVideo ()

  Tries to prepare the video for playback.
- void HandlePrepareCompleted (VideoPlayer source)

  Handles the prepare completed event of the video player.
- void HandleVideoError (VideoPlayer source, string message)

  Handles the video error event of the video player.

Private Attributes

- VideoPlayer videoPlayer
- int currentRetries = 0

5.34.1 Detailed Description

This script is used to stream video from a URL and handle errors and retries. It does not work! There is a problem with the format that it receives from the server.

Definition at line 14 of file RobotCamStream.cs.

5.34.2 Member Function Documentation

5.34.2.1 HandlePrepareCompleted()

```
void RobotCamStream.HandlePrepareCompleted (
            VideoPlayer source )  [inline], [private]
```

Handles the prepare completed event of the video player.

Parameters

source	The video player source.

Definition at line 64 of file RobotCamStream.cs.

```
65     {
66         videoPlayer.Play();
67     }
```

5.34.2.2 HandleVideoError()

```
void RobotCamStream.HandleVideoError (
            VideoPlayer source,
            string message )  [inline], [private]
```

Handles the video error event of the video player.

Parameters

source	The video player source.

Definition at line 73 of file RobotCamStream.cs.

```
74     {
75         Debug.LogError("Video Player Error: " + message);
76         currentRetries++;
77 
78         if (currentRetries < maxRetries)
79         {
80             Debug.Log($"Retry {currentRetries}/{maxRetries} in {retryDelay} seconds.");
81             Invoke(nameof(TryPrepareVideo), retryDelay); // Wait for retryDelay seconds before retrying
82         }
83         else
84         {
85             Debug.LogError("Max retries reached. Stopping attempts to play video.");
86         }
87     }
```

5.34.2.3 SetupVideoPlayer()

```
void RobotCamStream.SetupVideoPlayer ( )  [inline], [private]
```

Sets up the video player with the stream URL and other settings.

Definition at line 35 of file RobotCamStream.cs.

```
36     {
37         videoPlayer.url = streamUrl;
38         videoPlayer.playOnAwake = false;
39         videoPlayer.renderMode = VideoRenderMode.MaterialOverride;
40         videoPlayer.targetMaterialRenderer = GetComponent<Renderer>();
41         videoPlayer.audioOutputMode = VideoAudioOutputMode.None;
42         videoPlayer.isLooping = false; // Change to false to prevent auto-retry on reaching the end
43         videoPlayer.errorReceived += HandleVideoError;
44         videoPlayer.prepareCompleted += HandlePrepareCompleted;
45 
46         TryPrepareVideo();
47     }
```

5.34.2.4 Start()

```
void RobotCamStream.Start ( )  [inline], [private]
```

Start is called before the first frame and gets the VideoPlayer component and sets up the video player.

Definition at line 26 of file RobotCamStream.cs.

```
27     {
28         videoPlayer = GetComponent<VideoPlayer>();
29         SetupVideoPlayer();
30     }
```

5.34.2.5 TryPrepareVideo()

```
void RobotCamStream.TryPrepareVideo ( )  [inline], [private]
```

Tries to prepare the video for playback.

Definition at line 52 of file RobotCamStream.cs.

```
53     {
54         if (currentRetries < maxRetries)
55         {
56             videoPlayer.Prepare();
57         }
58     }
```

5.34.3 Member Data Documentation

5.34.3.1 currentRetries

```
int RobotCamStream.currentRetries = 0  [private]
```

Definition at line 21 of file RobotCamStream.cs.

5.34.3.2 maxRetries

```
int RobotCamStream.maxRetries = 3
```

Definition at line 17 of file RobotCamStream.cs.

5.34.3.3 retryDelay

```
float RobotCamStream.retryDelay = 2f
```

Definition at line 18 of file RobotCamStream.cs.

5.34.3.4 streamUrl

```
string RobotCamStream.streamUrl = "http://192.168.1.5:8000/stream.m3u8"
```

Definition at line 16 of file RobotCamStream.cs.

5.34.3.5 videoPlayer

```
VideoPlayer RobotCamStream.videoPlayer  [private]
```

Definition at line 20 of file RobotCamStream.cs.

The documentation for this class was generated from the following file:

- Assets/RobotCamStream.cs

5.35 ArmController.RobotControlValues Class Reference

The control values used to control the robot for controlling the robot (car).

Public Attributes

- float x
- float y
- int speed

5.35.1 Detailed Description

The control values used to control the robot for controlling the robot (car).

The is the interface between the VR and the Robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 22 of file ArmController.cs.

5.35.2 Member Data Documentation

5.35.2.1 speed

```
int ArmController.RobotControlValues.speed
```

Definition at line 26 of file ArmController.cs.

5.35.2.2 x

```
float ArmController.RobotControlValues.x
```

Definition at line 24 of file ArmController.cs.

5.35.2.3 y

```
float ArmController.RobotControlValues.y
```

Definition at line 25 of file ArmController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ArmController/ArmController.cs

5.36 HandDetectionCube.RobotControlValues Class Reference

The control values used to control the robot for controlling the robot (car).

Public Attributes

- float x
- float y
- int speed

5.36.1 Detailed Description

The control values used to control the robot for controlling the robot (car).

The is the interface between the VR and the Robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 22 of file HandDetectionCube.cs.

5.36.2 Member Data Documentation

5.36.2.1 speed

```
int HandDetectionCube.RobotControlValues.speed
```

Definition at line 26 of file HandDetectionCube.cs.

5.36.2.2 x

```
float HandDetectionCube.RobotControlValues.x
```

Definition at line 24 of file HandDetectionCube.cs.

5.36.2.3 y

```
float HandDetectionCube.RobotControlValues.y
```

Definition at line 25 of file HandDetectionCube.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/HandDetectionCube.cs

5.37 ArmController.RobotControlX Class Reference

The control values used to control the robot for controlling the arm.

Public Attributes

- float x
- float y
- float z
- int pinch
- float strength

5.37.1 Detailed Description

The control values used to control the robot for controlling the arm.

The is the interface between the VR and the Robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 34 of file ArmController.cs.

5.37.2 Member Data Documentation

5.37.2.1 pinch

```
int ArmController.RobotControlX.pinch
```

Definition at line 39 of file ArmController.cs.

5.37.2.2 strength

```
float ArmController.RobotControlX.strength
```

Definition at line 40 of file ArmController.cs.

5.37.2.3 x

```
float ArmController.RobotControlX.x
```

Definition at line 36 of file ArmController.cs.

5.37.2.4 y

```
float ArmController.RobotControlX.y
```

Definition at line 37 of file ArmController.cs.

5.37.2.5 z

```
float ArmController.RobotControlX.z
```

Definition at line 38 of file ArmController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ArmController/ArmController.cs

5.38 HandDetectionCube.RobotControlX Class Reference

The control values used to control the robot for controlling the arm.

Public Attributes

- float x
- float y
- int pinch
- float strength

5.38.1 Detailed Description

The control values used to control the robot for controlling the arm.

The is the interface between the VR and the Robot.

This interface has to match with the ROS2 interface from the robot

Definition at line 34 of file HandDetectionCube.cs.

5.38.2 Member Data Documentation

5.38.2.1 pinch

```
int HandDetectionCube.RobotControlX.pinch
```

Definition at line 38 of file HandDetectionCube.cs.

5.38.2.2 strength

```
float HandDetectionCube.RobotControlX.strength
```

Definition at line 39 of file HandDetectionCube.cs.

5.38.2.3 x

```
float HandDetectionCube.RobotControlX.x
```

Definition at line 36 of file HandDetectionCube.cs.

5.38.2.4 y

```
float HandDetectionCube.RobotControlX.y
```

Definition at line 37 of file HandDetectionCube.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/HandDetectionCube.cs

5.39 Responsehandler.ScrewCommand Class Reference

Public Attributes

- string screw

5.39.1 Detailed Description

Definition at line 27 of file Responsehandler.cs.

5.39.2 Member Data Documentation

5.39.2.1 screw

```
string Responsehandler.ScrewCommand.screw
```

Definition at line 29 of file Responsehandler.cs.

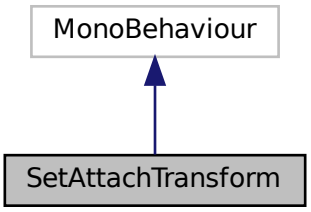
The documentation for this class was generated from the following file:

- Assets/Voice Controll/Responsehandler.cs

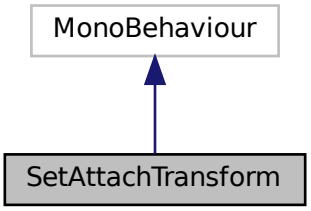
5.40 SetAttachTransform Class Reference

This script is used to set the attach transform of an XR poke interactor to a specific bone in the hand skeleton.

Inheritance diagram for SetAttachTransform:



Collaboration diagram for SetAttachTransform:



Public Attributes

- XRBaseInteractor pokeInteractor

  The XRBaseInteractor component used to interact with the UI Menu.
- OVRSkeleton handSkeleton

  The OVRSkeleton component used to get the bone data.

Private Member Functions

- void Start ()

  Start is called before the first frame update and is used to set the attach transform of the poke interactor to the middle finger tip bone.

Private Attributes

- int poke_finger_tip_id = 20

  The index of the finger tip bone in the OVRSkeleton.Bones array.

5.40.1 Detailed Description

This script is used to set the attach transform of an XR poke interactor to a specific bone in the hand skeleton.

The middle finger tip bone is used as the attach transform and it is used for poking interactions, such as poking buttons. This is used for interaction with the UI Menu in the VR environment.

Definition at line 9 of file SetAttachTransform.cs.

5.40.2 Member Function Documentation

5.40.2.1 Start()

```
void SetAttachTransform.Start ( )  [inline], [private]
```

Start is called before the first frame update and is used to set the attach transform of the poke interactor to the middle finger tip bone.

Definition at line 29 of file SetAttachTransform.cs.

```
30     {
31         if (handSkeleton != null && pokeInteractor != null)
32         {
33             // Attempt to find the desired bone in the skeleton
34             OVRBone bone = handSkeleton.Bones[poke_finger_tip_id];
35             if (bone != null)
36             {
37                 // If the bone is found, set the interactor's attach transform to the bone's transform
38                 pokeInteractor.attachTransform = bone.Transform;
39             }
40             else
41             {
42                 Debug.LogError("Desired bone not found in the skeleton.");
43             }
44         }
45         else
46         {
47             Debug.LogError("HandSkeleton or Interactor is not assigned.");
48         }
49     }
```

5.40.3 Member Data Documentation

5.40.3.1 handSkeleton

```
OVRSkeleton SetAttachTransform.handSkeleton
```

The OVRSkeleton component used to get the bone data.

Definition at line 19 of file SetAttachTransform.cs.

5.40.3.2 poke_finger_tip_id

```
int SetAttachTransform.poke_finger_tip_id = 20  [private]
```

The index of the finger tip bone in the OVRSkeleton.Bones array.

Definition at line 24 of file SetAttachTransform.cs.

5.40.3.3 pokeInteractor

```
XRBaseInteractor SetAttachTransform.pokeInteractor
```

The XRBaseInteractor component used to interact with the UI Menu.

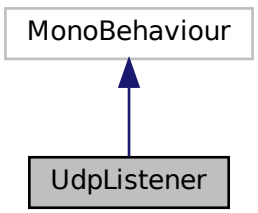
Definition at line 14 of file SetAttachTransform.cs.

The documentation for this class was generated from the following file:

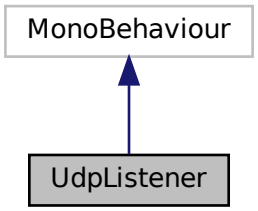
- Assets/Scripts/SetAttachTransform.cs

5.41 UdpListener Class Reference

Inheritance diagram for UdpListener:



Collaboration diagram for UdpListener:



Static Public Member Functions

- static void Main ()

Static Public Attributes

- const int listenPort = 12000

Static Private Member Functions

- static void StartListener ()

5.41.1 Detailed Description

Definition at line 8 of file UdpListener.cs.

5.41.2 Member Function Documentation

5.41.2.1 Main()

```
static void UdpListener.Main ( )  [inline], [static]
```

Definition at line 40 of file UdpListener.cs.

```
41     {
42         Task.Run(() => StartListener());
43         Console.WriteLine("Listening on port 12000. Press a key to quit.");
44         Console.ReadKey();
45     }
```

5.41.2.2 StartListener()

```
static void UdpListener.StartListener ( )  [inline], [static], [private]
```

Definition at line 12 of file UdpListener.cs.

```
13     {
14         UdpClient listener = new UdpClient(listenPort);
15         IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, listenPort);
16 
17         try
18         {
19             while (true)
20             {
21                 Console.WriteLine("Waiting for broadcast");
22                 byte[] bytes = listener.Receive(ref groupEP);
23 
24                 Console.WriteLine($"Received broadcast from {groupEP} :");
25                 Console.WriteLine($"Data: {Encoding.UTF8.GetString(bytes, 0, bytes.Length)}");
26 
27                 // Process the data as needed
28             }
29         }
30         catch (SocketException e)
31         {
32             Console.WriteLine(e);
33         }
34         finally
35         {
36             listener.Close();
37         }
38     }
```

5.41.3 Member Data Documentation

5.41.3.1 listenPort

```
const int UdpListener.listenPort = 12000  [static]
```

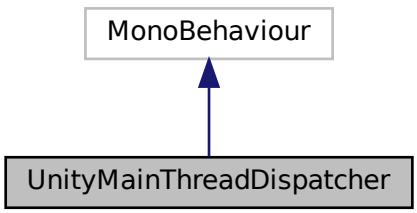
Definition at line 10 of file UdpListener.cs.

The documentation for this class was generated from the following file:

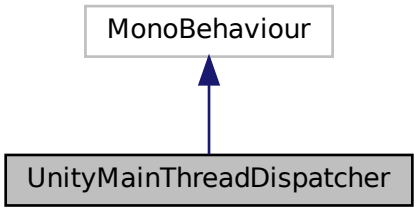
- Assets/Scripts/NetworkNamager/UdpListener.cs

5.42 UnityMainThreadDispatcher Class Reference

Inheritance diagram for UnityMainThreadDispatcher:



Collaboration diagram for UnityMainThreadDispatcher:



Public Member Functions

- void Enqueue (Action action)

Properties

- static UnityMainThreadDispatcher Instance `[get, private set]`

Private Member Functions

- void Awake ()
- void Update ()

Static Private Attributes

- static readonly Queue< Action > _executionQueue = new Queue<Action>()

5.42.1 Detailed Description

Definition at line 5 of file UnityMainThreadDispatcher.cs.

5.42.2 Member Function Documentation

5.42.2.1 Awake()

```
void UnityMainThreadDispatcher.Awake ( )  [inline], [private]
```

Definition at line 11 of file UnityMainThreadDispatcher.cs.

```
12     {
13         if (Instance == null)
14         {
15             Instance = this;
16             DontDestroyOnLoad(this.gameObject);
17         }
18         else
19         {
20             Destroy(gameObject);
21         }
22     }
```

5.42.2.2 Enqueue()

```
void UnityMainThreadDispatcher.Enqueue (
             Action action )  [inline]
```

Definition at line 35 of file UnityMainThreadDispatcher.cs.

```
36     {
37         lock (_executionQueue)
38         {
39             _executionQueue.Enqueue(action);
40         }
41     }
```

5.42.2.3 Update()

```
void UnityMainThreadDispatcher.Update ( )  [inline], [private]
```

Definition at line 24 of file UnityMainThreadDispatcher.cs.

```
25     {
26         lock (_executionQueue)
27         {
28             while (_executionQueue.Count > 0)
29             {
30                 _executionQueue.Dequeue().Invoke();
31             }
32         }
33     }
```

5.42.3 Member Data Documentation

5.42.3.1 _executionQueue

```
readonly Queue<Action> UnityMainThreadDispatcher._executionQueue = new Queue<Action>()  [static],
[private]
```

Definition at line 7 of file UnityMainThreadDispatcher.cs.

5.42.4 Property Documentation

5.42.4.1 Instance

```
UnityMainThreadDispatcher UnityMainThreadDispatcher.Instance  [static], [get], [private set]
```

Definition at line 9 of file UnityMainThreadDispatcher.cs.

```
9 { get; private set; }
```

The documentation for this class was generated from the following file:

- Assets/Scripts/UnityMainThreadDispatcher.cs

5.43 Vector3Data Struct Reference

Vector data to store tdepth data (x, y, z)

Public Member Functions

- Vector3Data (float x, float y, float z)

Public Attributes

- float x
- float y
- float z

5.43.1 Detailed Description

Vector data to store tdepth data (x, y, z)

Definition at line 60 of file NetworkManager.cs.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 Vector3Data()

```
Vector3Data.Vector3Data (
            float x,
            float y,
            float z )  [inline]
```

Definition at line 66 of file NetworkManager.cs.

```
67     {
68         this.x = x;
69         this.y = y;
70         this.z = z;
71     }
```

5.43.3 Member Data Documentation

5.43.3.1 x

```
float Vector3Data.x
```

Definition at line 62 of file NetworkManager.cs.

5.43.3.2 y

```
float Vector3Data.y
```

Definition at line 63 of file NetworkManager.cs.

5.43.3.3 z

```
float Vector3Data.z
```

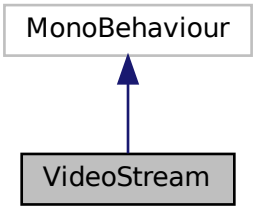
Definition at line 64 of file NetworkManager.cs.

The documentation for this struct was generated from the following file:

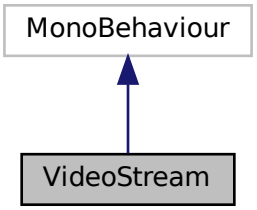
- Assets/Scripts/NetworkNamager/NetworkManager.cs

5.44 VideoStream Class Reference

Inheritance diagram for VideoStream:



Collaboration diagram for VideoStream:



Private Member Functions

- void Start ()
- void Update ()

5.44.1 Detailed Description

Definition at line 5 of file VideoStream.cs.

5.44.2 Member Function Documentation

5.44.2.1 Start()

```
void VideoStream.Start ( )  [inline], [private]
```

Definition at line 8 of file VideoStream.cs.

```
9     {
10
11     }
```

5.44.2.2 Update()

```
void VideoStream.Update ( )  [inline], [private]
```

Definition at line 14 of file VideoStream.cs.

```
15     {
16
17     }
```

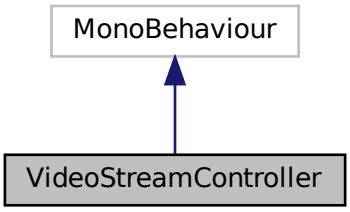
The documentation for this class was generated from the following file:

- Assets/Scripts/VideoStreamer/VideoStream.cs

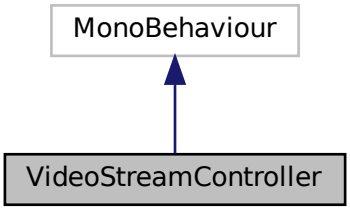
5.45 VideoStreamController Class Reference

This script is used to play a video stream from a server.

Inheritance diagram for VideoStreamController:



Collaboration diagram for VideoStreamController:



Private Member Functions

- void Start ()

Private Attributes

- VideoPlayer videoPlayer

5.45.1 Detailed Description

This script is used to play a video stream from a server.

Used for testing purposes. It does not work with HSL as intended.

Definition at line 10 of file VideoStreamController.cs.

5.45.2 Member Function Documentation

5.45.2.1 Start()

```
void VideoStreamController.Start ( )  [inline], [private]
```

Definition at line 14 of file VideoStreamController.cs.

```
15     {
16         videoPlayer = GetComponent<VideoPlayer>();
17         videoPlayer.url = "http://192.168.0.21:5000/stream/stream.m3u8";
18         videoPlayer.Play();
19     }
```

5.45.3 Member Data Documentation

5.45.3.1 videoPlayer

```
VideoPlayer VideoStreamController.videoPlayer  [private]
```

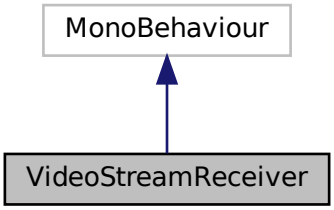
Definition at line 12 of file VideoStreamController.cs.

The documentation for this class was generated from the following file:

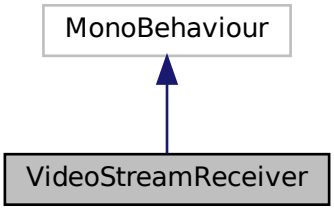
- Assets/VideoStreamController.cs

5.46 VideoStreamReceiver Class Reference

Inheritance diagram for VideoStreamReceiver:



Collaboration diagram for VideoStreamReceiver:



5.46.1 Detailed Description

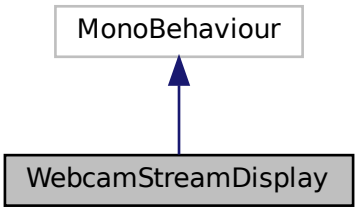
Definition at line 4 of file VideoStreamReceiver.cs.

The documentation for this class was generated from the following file:

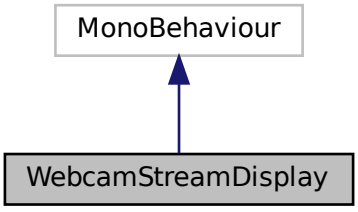
- Assets/Scripts/VideoStreamer/VideoStreamReceiver.cs

5.47 WebcamStreamDisplay Class Reference

Inheritance diagram for WebcamStreamDisplay:



Collaboration diagram for WebcamStreamDisplay:



Public Attributes

- string snapshotUrl = "http://192.168.1.11:5000/snapshot"
- float refreshRate = 0.1f

Private Member Functions

- void Start ()
- IEnumerator FetchSnapshotRoutine ()
- void ApplyTexture (Texture2D texture)
- void OnDestroy ()

Private Attributes

- Renderer _renderer
- WaitForSeconds _refreshWait

5.47.1 Detailed Description

Definition at line 8 of file WebcamStreamDisplay.cs.

5.47.2 Member Function Documentation

5.47.2.1 ApplyTexture()

```
void WebcamStreamDisplay.ApplyTexture (
            Texture2D texture )  [inline], [private]
```

Definition at line 50 of file WebcamStreamDisplay.cs.

```
51     {
52         _renderer.material.mainTexture = texture;
53     }
```

5.47.2.2 FetchSnapshotRoutine()

```
IEnumerator WebcamStreamDisplay.FetchSnapshotRoutine ( )  [inline], [private]
```

Definition at line 23 of file WebcamStreamDisplay.cs.

```
24     {
25         while (true)
26         {
27             // Start a new asynchronous request
28             UnityWebRequest www = UnityWebRequestTexture.GetTexture(snapshotUrl);
29             yield return www.SendWebRequest();  // Asynchronously wait for the web request to complete
30 
31             if (www.result == UnityWebRequest.Result.Success)
32             {
33                 // Asynchronously get the texture content
34                 Texture2D texture = DownloadHandlerTexture.GetContent(www);
35                 if (texture != null)
36                 {
37                     ApplyTexture(texture);
38                 }
39             }
40             else
41             {
42                 // Debug.LogError("Error fetching snapshot: " + www.error);
43             }
44 
45             www.Dispose(); // Clean up the web request
46             yield return _refreshWait; // Wait before making the next request
47         }
48     }
```

5.47.2.3 OnDestroy()

```
void WebcamStreamDisplay.OnDestroy ( )  [inline], [private]
```

Definition at line 55 of file WebcamStreamDisplay.cs.

```
56     {
57         StopAllCoroutines();
58     }
```

5.47.2.4 Start()

```
void WebcamStreamDisplay.Start ( )  [inline], [private]
```

Definition at line 16 of file WebcamStreamDisplay.cs.

```
17     {
18         _renderer = GetComponent<Renderer>();
19         _refreshWait = new WaitForSeconds(refreshRate);
20         StartCoroutine(FetchSnapshotRoutine());
21     }
```

5.47.3 Member Data Documentation

5.47.3.1 _refreshWait

```
WaitForSeconds WebcamStreamDisplay._refreshWait  [private]
```

Definition at line 14 of file WebcamStreamDisplay.cs.

5.47.3.2 _renderer

```
Renderer WebcamStreamDisplay._renderer  [private]
```

Definition at line 13 of file WebcamStreamDisplay.cs.

5.47.3.3 refreshRate

```
float WebcamStreamDisplay.refreshRate = 0.1f
```

Definition at line 11 of file WebcamStreamDisplay.cs.

5.47.3.4 snapshotUrl

```
string WebcamStreamDisplay.snapshotUrl = "http://192.168.1.11:5000/snapshot"
```

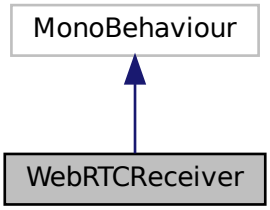
Definition at line 10 of file WebcamStreamDisplay.cs.

The documentation for this class was generated from the following file:

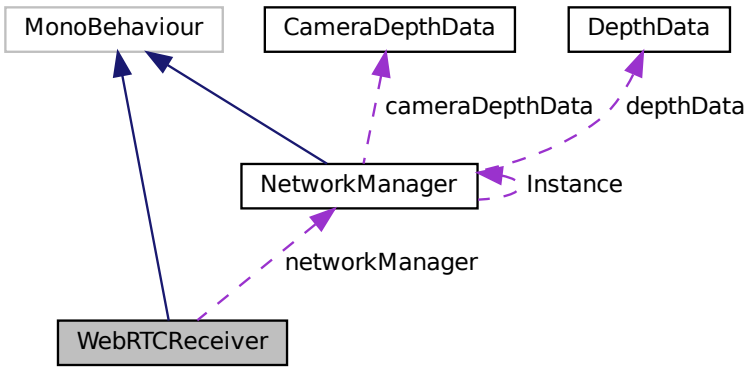
- Assets/WebcamStreamDisplay.cs

5.48 WebRTCReceiver Class Reference

Inheritance diagram for WebRTCReceiver:



Collaboration diagram for WebRTCReceiver:



Public Attributes

- RawImage rawImage

Private Member Functions

- void Start ()
- void Update ()
- void OnMessage (object sender, MessageEventArgs e)
- void OnError (object sender, ErrorEventArgs e)
- void OnDestroy ()

Private Attributes

- WebSocket _webSocket
- NetworkManager networkManager
- Texture2D texture
- int index = 0

5.48.1 Detailed Description

Definition at line 7 of file WebRTCReceiver.cs.

5.48.2 Member Function Documentation

5.48.2.1 OnDestroy()

```
void WebRTCReceiver.OnDestroy ( )  [inline], [private]
```

Definition at line 78 of file WebRTCReceiver.cs.

```
79     {
80         if (_webSocket != null)
81         {
82             _webSocket.OnMessage -= OnMessage;
83             _webSocket.OnError -= OnError;
84             _webSocket.Close();
85             _webSocket = null;
86         }
87     }
```

5.48.2.2 OnError()

```
void WebRTCReceiver.OnError (
            object sender,
            ErrorEventArgs e )  [inline], [private]
```

Definition at line 73 of file WebRTCReceiver.cs.

```
74     {
75         Debug.LogError("Camera: WebSocket Error" + e.Message);
76     }
```

5.48.2.3 OnMessage()

```
void WebRTCReceiver.OnMessage (
            object sender,
            MessageEventArgs e )  [inline], [private]
```

Definition at line 54 of file WebRTCReceiver.cs.

```
55     {
56         Debug.Log($"Camera: Received data {index++}");
57         byte[] bytes = System.Convert.FromBase64String(e.Data);
58 
59         UnityMainThreadDispatcher.Instance.Enqueue(() =>
60         {
61             if (texture.LoadImage(bytes))
62             {
63                 texture.Apply();
64                 rawImage.texture = texture;
65                 Debug.Log("Camera: Texture applied successfully");
66             }
67             else
68             {
69                 Debug.LogError("Camera: Failed to load texture from received data");
70             }
71         });
72     }
```

5.48.2.4 Start()

```
void WebRTCReceiver.Start ( )  [inline], [private]
```

Definition at line 13 of file WebRTCReceiver.cs.

```
14     {
15         networkManager = NetworkManager.Instance;
16         texture = new Texture2D(27, 15, TextureFormat.RGB24, false); // Adjust format as necessary
17         networkManager.OnConnected += () =>
18         {
19             _webSocket = new WebSocket("ws://192.168.1.6:5000");
20             _webSocket.OnMessage += OnMessage;
21             _webSocket.OnError += OnError;
22             _webSocket.Connect();
23         };
24 
25     }
```

5.48.2.5 Update()

```
void WebRTCReceiver.Update ( )  [inline], [private]
```

Definition at line 27 of file WebRTCReceiver.cs.

```
28     {
29 
30     }
```

5.48.3 Member Data Documentation

5.48.3.1 _webSocket

```
WebSocket WebRTCReceiver._webSocket  [private]
```

Definition at line 9 of file WebRTCReceiver.cs.

5.48.3.2 index

```
int WebRTCReceiver.index = 0  [private]
```

Definition at line 32 of file WebRTCReceiver.cs.

5.48.3.3 networkManager

```
NetworkManager WebRTCReceiver.networkManager  [private]
```

Definition at line 10 of file WebRTCReceiver.cs.

5.48.3.4 rawImage

```
RawImage WebRTCReceiver.rawImage
```

Definition at line 11 of file WebRTCReceiver.cs.

5.48.3.5 texture

```
Texture2D WebRTCReceiver.texture  [private]
```

Definition at line 12 of file WebRTCReceiver.cs.

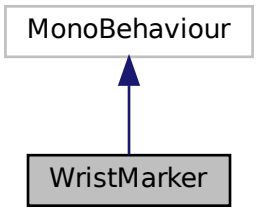
The documentation for this class was generated from the following file:

- Assets/Scripts/WebRTCReceiver.cs

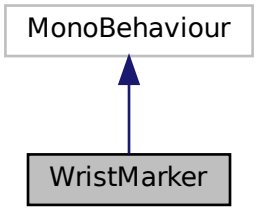
5.49 WristMarker Class Reference

This class is used to display a green dot that is attached to the Tip of the Middle Finger of the Hand.

Inheritance diagram for WristMarker:



Collaboration diagram for WristMarker:



Public Attributes

- OVRSkeleton skeleton

  The OVRSkeleton component used to get the bone data.
- GameObject greenDot

  The GameObject used to display the green dot.

Private Member Functions

- void Start ()
- void Update ()

  Update is called once per frame and is used to update the position of the green dot to the wrist bone's position.

Private Attributes

- int Hand_MiddleTip = (int)OVRPlugin.BoneId.Hand_MiddleTip

  The index of the Hand_MiddleTip bone in the OVRSkeleton.Bones array.

5.49.1 Detailed Description

This class is used to display a green dot that is attached to the Tip of the Middle Finger of the Hand.

It displays the green dot in the VR environment with 2 cm offset from the wrist bone.

Definition at line 7 of file WristMarker.cs.

5.49.2 Member Function Documentation

5.49.2.1 Start()

```
void WristMarker.Start ( )  [inline], [private]
```

Definition at line 23 of file WristMarker.cs.

```
24     {
25     }
```

5.49.2.2 Update()

```
void WristMarker.Update ( )  [inline], [private]
```

Update is called once per frame and is used to update the position of the green dot to the wrist bone's position.

Definition at line 30 of file WristMarker.cs.

```
31     {
32         if (skeleton == null || greenDot == null)
33             return;
34 
35         // Debug.Log("Time.captureFramerate: " + Time.captureFramerate);
36         OVRBone WristBone = skeleton.Bones[Hand_MiddleTip];
37         if (WristBone != null)
38         {
39             // Update the green dot's position to the wrist bone's position
40             // Assuming an offset of 0.02 meters (2 cm) above the wrist bone
41             greenDot.transform.position = WristBone.Transform.position + Vector3.up * 0.01f;
42 
43             Debug.Log($"Green dot position: {greenDot.transform.position}");
44         }
45 
46     }
```

5.49.3 Member Data Documentation

5.49.3.1 greenDot

```
GameObject WristMarker.greenDot
```

The GameObject used to display the green dot.

A sphere is used to display the green dot.

Definition at line 17 of file WristMarker.cs.

5.49.3.2 Hand_MiddleTip

```
int WristMarker.Hand_MiddleTip = (int)OVRPlugin.BoneId.Hand_MiddleTip  [private]
```

The index of the Hand_MiddleTip bone in the OVRSkeleton.Bones array.

Definition at line 22 of file WristMarker.cs.

5.49.3.3 skeleton

```
OVRSkeleton WristMarker.skeleton
```

The OVRSkeleton component used to get the bone data.

Definition at line 12 of file WristMarker.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/WristMarker.cs

Chapter 6

File Documentation

6.1 Assets/DebugDisplay.cs File Reference

Classes

- class DebugDisplay

  This script is used to display debug logs on the a VR screen for testing purposes.

6.2 Assets/DebugDisplayPro.cs File Reference

Classes

- class DebugDisplayPro

  This script is used to display debug logs on the a VR screen for testing purposes.

6.3 Assets/HandDataTransmission.cs File Reference

Classes

- class HandDataTransmission

  This script is used for testing porpuses.

6.4 Assets/HandInteraction.cs File Reference

Classes

- class HandInteraction

  This script is used to send hand tracking data to the server at regular intervals.

6.5 Assets/HelloWorldScript.cs File Reference

Classes

- class HelloWorldScript

  This script is used to display a simple "Hello World" message on the screen for testing purposes.

6.6 Assets/ObjectController.cs File Reference

Classes

- class ObjectController

  This class is used to send data to the server by pressing the space key.

6.7 Assets/RobotCamStream.cs File Reference

Classes

- class RobotCamStream

6.8 Assets/Scripts/ArmController/ArmController.cs File Reference

Classes

- class ArmController

  This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.
- class ArmController.RobotControlValues

  The control values used to control the robot for controlling the robot (car).
- class ArmController.RobotControlX

  The control values used to control the robot for controlling the arm.

6.9 Assets/Scripts/DigitalTwin/DigitalTwinController.cs File Reference

Classes

- class DigitalTwinController

  This class is used to control the digital twin robot.

6.10 Assets/Scripts/DriveModeController/DriveModeController.cs File Reference

Classes

- class DriveModeController

  This class is used to control the drive mode of the robot.
- class DriveModeController.DriveMode

6.11 Assets/Scripts/DropdownHandler.cs File Reference

Classes

- class DropdownHandler

  This script is used to handle the dropdown in the UI.
- class DropdownHandler.ModeValues

  The ModeValues interface used to send the mode value to the robot.

6.12 Assets/Scripts/HandDetectionCube.cs File Reference

Classes

- class HandDetectionCube

  This script is used to detect the hand position within a cube and send control values to the robot based on the hand position.
- class HandDetectionCube.RobotControlValues

  The control values used to control the robot for controlling the robot (car).
- class HandDetectionCube.RobotControlX

  The control values used to control the robot for controlling the arm.

6.13 Assets/Scripts/HandGestureAndRotation.cs File Reference

Classes

- class HandGestureAndRotation

  This class is used to get the hand gesture and rotation data and send it to the server.
- class HandGestureAndRotation.HandData

  The HandData interface used to send the data to the robot.

6.14 Assets/Scripts/HandleReconnectButton.cs File Reference

Classes

- class HandleReconnectButton

  This script is used to handle the reconnect button in the UI.

6.15 Assets/Scripts/Head up display/Heap_up_controller.cs File Reference

Classes

- class Heap_up_controller

  This class is used to control the head up display of the robot.

6.16 Assets/Scripts/log/Logger.cs File Reference

Classes

- class Logger

  This script is used to log messages to a file.

6.17 Assets/Scripts/LogicController/MainController.cs File Reference

Classes

- class MainController

  This class is the main controller for handling scene changes and mode changes.

6.18 Assets/Scripts/NetworkNamager/NetworkManager.cs File Reference

Classes

- class JsonRobotInfo

  Class `JsonRobotInfo` Represents the JSON data structure for the robot information.
- class JsonArmLengthInfo
- struct Vector3Data

  Vector data to store tdepth data (x, y, z)
- class CameraDepthData

  depth camera interface datatype The data is recieved from the robot as this type
- class PingData
- class DepthDataPoint

  Depth data interface Datatype
- class DepthData
- class NetworkManager

  Class `NetworkManager` Manages network communications for the application, implementing a singleton pattern to ensure only one instance exists.

6.19 Assets/Scripts/NetworkNamager/UdpListener.cs File Reference

Classes

- class UdpListener

6.20 Assets/Scripts/PlotCameraDepthData/DepthDataGenerator.cs File Reference

Classes

- class DepthDataGenerator

6.21 Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs File Reference

Classes

- class DepthPointCloud
- class DepthPointCloud.RequestDepthDataMsg

  Interface for requesting depth camera from the robot

6.22 Assets/Scripts/PlotCameraDepthData/MeshCreator.cs File Reference

Classes

- class MeshCreator

6.23 Assets/Scripts/PlotCameraDepthData/MeshReplacement.cs File Reference

Classes

- class MeshReplacement

6.24 Assets/Scripts/SetAttachTransform.cs File Reference

Classes

- class SetAttachTransform

  This script is used to set the attach transform of an XR poke interactor to a specific bone in the hand skeleton.

6.25 Assets/Scripts/UnityMainThreadDispatcher.cs File Reference

Classes

- class UnityMainThreadDispatcher

6.26 Assets/Scripts/VideoStreamer/VideoStream.cs File Reference

Classes

- class VideoStream

6.27 Assets/Scripts/VideoStreamer/VideoStreamReceiver.cs File Reference

Classes

- class VideoStreamReceiver

6.28 Assets/Scripts/WebRTCReceiver.cs File Reference

Classes

- class WebRTCReceiver

6.29 Assets/Scripts/WristMarker.cs File Reference

Classes

- class WristMarker

  This class is used to display a green dot that is attached to the Tip of the Middle Finger of the Hand.

6.30 Assets/VideoStreamController.cs File Reference

Classes

- class VideoStreamController

  This script is used to play a video stream from a server.

6.31 Assets/Voice Controll/ModeAudioPlay.cs File Reference

Classes

- class ModeAudioPlay

6.32 Assets/Voice Controll/Responsehandler.cs File Reference

Classes

- class Responsehandler
- class Responsehandler.ScrewCommand

6.33 Assets/WebcamStreamDisplay.cs File Reference

Classes

- class WebcamStreamDisplay

6.34 obj/Debug/.NETStandard,Version=v2.1.AssemblyAttributes.cs File Reference

6.35 README_.md File Reference

Index

_executionQueue
    UnityMainThreadDispatcher, 156
_refreshWait
    WebcamStreamDisplay, 165
_renderer
    WebcamStreamDisplay, 165
_webSocket
    WebRTCReceiver, 168

accelerometer
    Heap_up_controller, 87
    JsonRobotInfo, 95
activeColor
    DriveModeController, 49
AddDepthDataPoint
    DepthData, 29
ApplyTexture
    WebcamStreamDisplay, 164
ArmController, 9
    CalculateDistances, 11
    CalculateNormalizedControlValues, 12
    CalculateSpeed, 12
    controlValues, 18
    controlX, 18
    cubeMaterial, 18
    distanceCalculationInterval, 18
    dropdownHandler, 18
    HandleReceivedData, 13
    handSkeleton, 19
    insideColor, 19
    isHandDetected, 19
    lastSendTime, 19
    networkManager, 19
    OnTriggerEnter, 13
    OnTriggerExit, 14
    OnTriggerStay, 14
    originalColor, 20
    RepeatedlyDistanceCalculation, 14
    ResetControlValues, 15
    rightHand, 20
    SendControlValues, 15
    SendDataToServer, 16
    sendInterval, 20
    Start, 16
    Update, 17
    UpdateVisualIndicator, 17
    visualIndicatorTransform, 20
ArmController.RobotControlValues, 144
    speed, 144
    x, 144
    y, 145
ArmController.RobotControlX, 146
    pinch, 147
    strength, 147
    x, 147
    y, 147
    z, 147
armScene
    MainController, 102
armSource
    ModeAudioPlay, 112
Assets/DebugDisplay.cs, 173
Assets/DebugDisplayPro.cs, 173
Assets/HandDataTransmission.cs, 173
Assets/HandInteraction.cs, 173
Assets/HelloWorldScript.cs, 174
Assets/ObjectController.cs, 174
Assets/RobotCamStream.cs, 174
Assets/Scripts/ArmController/ArmController.cs, 174
Assets/Scripts/DigitalTwin/DigitalTwinController.cs, 174
Assets/Scripts/DriveModeController/DriveModeController.cs, 175
Assets/Scripts/DropdownHandler.cs, 175
Assets/Scripts/HandDetectionCube.cs, 175
Assets/Scripts/HandGestureAndRotation.cs, 175
Assets/Scripts/HandleReconnectButton.cs, 175
Assets/Scripts/Head up display/Heap_up_controller.cs, 176
Assets/Scripts/log/Logger.cs, 176
Assets/Scripts/LogicController/MainController.cs, 176
Assets/Scripts/NetworkNamager/NetworkManager.cs, 176
Assets/Scripts/NetworkNamager/UdpListener.cs, 176
Assets/Scripts/PlotCameraDepthData/DepthDataGenerator.cs, 177
Assets/Scripts/PlotCameraDepthData/DepthPointCloud.cs, 177
Assets/Scripts/PlotCameraDepthData/MeshCreator.cs, 177
Assets/Scripts/PlotCameraDepthData/MeshReplacement.cs, 177
Assets/Scripts/SetAttachTransform.cs, 177
Assets/Scripts/UnityMainThreadDispatcher.cs, 177
Assets/Scripts/VideoStreamer/VideoStream.cs, 178
Assets/Scripts/VideoStreamer/VideoStreamReceiver.cs, 178
Assets/Scripts/WebRTCReceiver.cs, 178
Assets/Scripts/WristMarker.cs, 178
Assets/VideoStreamController.cs, 178

Assets/Voice Controll/ModeAudioPlay.cs, 178
Assets/Voice Controll/Responsehandler.cs, 179
Assets/WebcamStreamDisplay.cs, 179
Awake
    Logger, 98
    NetworkManager, 116
    UnityMainThreadDispatcher, 155

battery_precentage
    Heap_up_controller, 87
    JsonRobotInfo, 95
battery_precentage_text
    Heap_up_controller, 87
broHeader
    NetworkManager, 127
brotlifile
    DepthPointCloud, 37
brotliTestFile
    NetworkManager, 127

CalculateDistances
    ArmController, 11
    HandDetectionCube, 60
CalculateNormalizedControlValues
    ArmController, 12
    HandDetectionCube, 60
CalculateSpeed
    ArmController, 12
    HandDetectionCube, 61
CameraDepthData, 21
    depthData, 21
    time, 21
cameraDepthData
    NetworkManager, 127
ClearPoints
    DepthData, 29
clearVolumeBoxes
    MeshCreator, 104
client
    NetworkManager, 127
cms_speed
    Heap_up_controller, 87
    JsonRobotInfo, 95
conenctionStatus_text
    Heap_up_controller, 88
connected
    NetworkManager, 127
connectedColor
    Heap_up_controller, 88
ConnectToServer
    NetworkManager, 117
controlValues
    ArmController, 18
    HandDetectionCube, 67
controlX
    ArmController, 18
    HandDetectionCube, 67
Count
    DepthData, 29
CreateCubeGridTest
    MeshCreator, 104
CreateMeshInBoxVolume
    MeshCreator, 105
cubeMaterial
    ArmController, 18
    HandDetectionCube, 67
currentRetries
    RobotCamStream, 143

DebugDisplay, 22
    debugLogs, 24
    debugText, 24
    HandleLog, 23
    OnDisable, 23
    OnEnable, 23
    Update, 24
DebugDisplayPro, 25
    debugLogs, 27
    debugText, 28
    HandleLog, 26
    OnDisable, 27
    OnEnable, 27
    Update, 27
debugLogs
    DebugDisplay, 24
    DebugDisplayPro, 27
debugText
    DebugDisplay, 24
    DebugDisplayPro, 28
DecodeBase64
    DepthPointCloud, 35
    NetworkManager, 117
DecompressBrotli
    DepthPointCloud, 35
    NetworkManager, 118
defaultColor
    DriveModeController, 49
depthColorGradient
    DepthPointCloud, 37
    MeshCreator, 107
DepthData, 28
    AddDepthDataPoint, 29
    ClearPoints, 29
    Count, 29
    depthDataPoints, 29
    Points, 30
    time, 29
depthData
    CameraDepthData, 21
    DepthPointCloud, 38
    NetworkManager, 127
DepthDataGenerator, 30
    GenerateDepthData, 31
    Height, 31
    MaxDepth, 31
    meshCreator, 32
    MinDepth, 32
    Width, 32

DepthDataPoint, 32
DepthDataPoint, 33
X, 33
Y, 33
Z, 33
depthDataPoints
DepthData, 29
DepthPointCloud, 34
brotlifile, 37
DecodeBase64, 35
DecompressBrotli, 35
depthColorGradient, 37
depthData, 38
meshCreator, 38
networkManager, 38
OnDepthDataReceived, 36
OnDestroy, 36
particles, 38
particleSystem, 38
ProcessDepthData, 36
RequestDepthData, 36
Start, 37
Update, 37
DepthPointCloud.RequestDepthDataMsg, 135
get_depth, 135
DigitalTwinController, 39
frameCounter, 43
HandleReceivedRobotInfoData, 40
Link_1, 43
Link_2, 43
Link_3, 43
Link_4, 43
Link_5, 43
networkManager, 44
Pinch_left_x, 44
Pinch_right_y, 44
rotateTo170, 44
SetTargetRotation, 41
SetTargetRotationY, 41
Start, 41
testFunction, 42
Update, 42
Disconnect
NetworkManager, 118
disconnectedColor
Heap_up_controller, 88
distanceCalculationInterval
ArmController, 18
HandDetectionCube, 67
drive_mode
DriveModeController.DriveMode, 45
DriveModeChanged
DriveModeController, 50
DriveModeController, 45
activeColor, 49
defaultColor, 49
DriveModeChanged, 50
networkManager, 49
normalButton, 49
normalMode, 47
precisionButton, 50
precisionMode, 47
reverseButton, 50
reverseMode, 47
SendDataToServer, 47
Start, 48
Update, 48
updateButtonColor, 48
updateDriveModeData, 49
DriveModeController.DriveMode, 45
drive_mode, 45
driveScene
MainController, 102
driveSource
ModeAudioPlay, 112
dropdown
DropdownHandler, 54
MainController, 102
DropdownHandler, 51
dropdown, 54
DropdownValueChanged, 52
GetDropdownValue, 53
HandleVoiceCommand, 53
modeValues, 54
networkManager, 54
OnDropdownValueChanged, 55
responsehandler, 55
SendDataToServer, 53
Start, 54
dropdownHandler
ArmController, 18
HandDetectionCube, 68
MainController, 102
DropdownHandler.ModeValues, 113
mode, 113
DropdownValueChanged
DropdownHandler, 52

emergencySource
ModeAudioPlay, 112
EmergencyStopButton
MainController, 103
Enqueue
UnityMainThreadDispatcher, 156

FetchSnapshotRoutine
WebcamStreamDisplay, 164
frameCounter
DigitalTwinController, 43

GenerateDepthData
DepthDataGenerator, 31
get_depth
DepthPointCloud.RequestDepthDataMsg, 135
GetDropdownValue
DropdownHandler, 53
greenDot

WristMarker, 171
gyroscope
Heap_up_controller, 88
JsonRobotInfo, 95

hand
HandGestureAndRotation, 73
HandInteraction, 78
Hand_MiddleTip
WristMarker, 172
handData
HandGestureAndRotation, 73
HandDataTransmission, 56
Start, 57
Update, 57
HandDetectionCube, 58
CalculateDistances, 60
CalculateNormalizedControlValues, 60
CalculateSpeed, 61
controlValues, 67
controlX, 67
cubeMaterial, 67
distanceCalculationInterval, 67
dropdownHandler, 68
HandleReceivedData, 61
handSkeleton, 68
insideColor, 68
isHandDetected, 68
lastSendTime, 68
networkManager, 69
OnDestroy, 62
OnTriggerEnter, 62
OnTriggerExit, 62
OnTriggerStay, 63
originalColor, 69
RepeatedlyDistanceCalculation, 63
ResetControlValues, 63
rightHand, 69
SendControlValues, 64
SendDataToServer, 64
sendInterval, 69
Start, 65
Update, 65
UpdateVisualIndicator, 65
visualIndicatorTransform, 69
HandDetectionCube.RobotControlValues, 145
speed, 145
x, 146
y, 146
HandDetectionCube.RobotControlX, 148
pinch, 148
strength, 148
x, 148
y, 148
HandGestureAndRotation, 70
hand, 73
handData, 73
handSkeleton, 73
isTesting, 73
lastSendTime, 74
networkManager, 74
SendDataToServer, 71
sendInterval, 74
Start, 72
Update, 72
HandGestureAndRotation.HandData, 55
pinch, 56
wrist, 56
HandInteraction, 75
hand, 78
HandleReceivedData, 76
isTesting, 78
lastSendTime, 78
networkManager, 79
OnDestroy, 77
SendDataToServer, 77
sendInterval, 79
Start, 77
Update, 77
HandleConnectionStatusChanged
Heap_up_controller, 84
HandleDropdownChange
MainController, 100
HandleEmergencyStop
MainController, 101
HandleLog
DebugDisplay, 23
DebugDisplayPro, 26
HandlePrepareCompleted
RobotCamStream, 141
HandleReceivedData
ArmController, 13
HandDetectionCube, 61
HandInteraction, 76
HandleReceivedPingData
Heap_up_controller, 84
HandleReceivedRobotInfoData
DigitalTwinController, 40
Heap_up_controller, 85
HandleReconnectButton, 79
networkManager, 81
OnReconnectButtonClicked, 81
reconnectButton, 82
Start, 81
Update, 81
HandleResponseTest
Responsehandler, 137
HandleVideoError
RobotCamStream, 141
HandleVoiceCommand
DropdownHandler, 53
handSkeleton
ArmController, 19
HandDetectionCube, 68
HandGestureAndRotation, 73
SetAttachTransform, 151
head_up_canvas

Heap_up_controller, 88
head_up_canvas_distance
Heap_up_controller, 88
Heap_up_controller, 82
accelerometer, 87
battery_precentage, 87
battery_precentage_text, 87
cms_speed, 87
conenctionStatus_text, 88
connectedColor, 88
disconnectedColor, 88
gyroscope, 88
HandleConnectionStatusChanged, 84
HandleReceivedPingData, 84
HandleReceivedRobotInfoData, 85
head_up_canvas, 88
head_up_canvas_distance, 88
latecy_text, 89
magnetometer, 89
mode, 89
mode_text, 89
motion, 89
networkManager, 89
offset_ovr_camera_rig, 90
OnDestroy, 85
OnDisable, 86
OnEnable, 86
ovr_camera_rig, 90
robot_view_plane, 90
speed, 90
speed_text, 90
Start, 86
Update, 86
update_head_up_canvas_position_and_rotation, 86
voltage, 90
voltage_text, 91
Height
DepthDataGenerator, 31
HelloWorldScript, 91
myName, 92
Start, 92
textMeshPro, 93

idleSource
ModeAudioPlay, 112
index
Responsehandler, 139
WebRTCReceiver, 169
insideColor
ArmController, 19
HandDetectionCube, 68
Instance
NetworkManager, 128
UnityMainThreadDispatcher, 157
InvokeConnectionStatus
NetworkManager, 119
IsBrotli
NetworkManager, 119

isHandDetected
ArmController, 19
HandDetectionCube, 68
isListening
NetworkManager, 128
isTesting
HandGestureAndRotation, 73
HandInteraction, 78

JsonArmLengthInfo, 93
Link_1, 93
Link_2, 93
Link_3, 94
Link_4, 94
Link_5, 94
pintch, 94
JsonRobotInfo, 94
accelerometer, 95
battery_precentage, 95
cms_speed, 95
gyroscope, 95
magnetometer, 96
mode, 96
motion, 96
speed, 96
voltage, 96

lastSendTime
ArmController, 19
HandDetectionCube, 68
HandGestureAndRotation, 74
HandInteraction, 78
latecy_text
Heap_up_controller, 89
Link_1
DigitalTwinController, 43
JsonArmLengthInfo, 93
Link_2
DigitalTwinController, 43
JsonArmLengthInfo, 93
Link_3
DigitalTwinController, 43
JsonArmLengthInfo, 94
Link_4
DigitalTwinController, 43
JsonArmLengthInfo, 94
Link_5
DigitalTwinController, 43
JsonArmLengthInfo, 94
listenPort
UdpListener, 154
Log
Logger, 98
logFilePath
Logger, 98
Logger, 97
Awake, 98
Log, 98
logFilePath, 98

magnetometer
Heap_up_controller, 89
JsonRobotInfo, 96
Main
UdpListener, 153
MainController, 99
armScene, 102
driveScene, 102
dropdown, 102
dropdownHandler, 102
EmergencyStopButton, 103
HandleDropdownChange, 100
HandleEmergencyStop, 101
modeAudioPlay, 103
OnDestroy, 101
Start, 101
MaxDepth
DepthDataGenerator, 31
maxRetries
RobotCamStream, 143
mesh
MeshReplacement, 109
MeshCreator, 103
clearVolumeBoxes, 104
CreateCubeGridTest, 104
CreateMeshInBoxVolume, 105
depthColorGradient, 107
NormalizeDepthData, 105
Start, 106
Update, 106
volume, 107
meshCreator
DepthDataGenerator, 32
DepthPointCloud, 38
MeshReplacement, 107
mesh, 109
ply, 109
Start, 108
Update, 108
MinDepth
DepthDataGenerator, 32
mode
DropdownHandler.ModeValues, 113
Heap_up_controller, 89
JsonRobotInfo, 96
mode_text
Heap_up_controller, 89
ModeAudioPlay, 109
armSource, 112
driveSource, 112
emergencySource, 112
idleSource, 112
PlayArm, 110
PlayDrive, 110
PlayEmergency, 110
PlayIdle, 111
PlayScrew, 111
PlayUnScrew, 111
screwSource, 112
Start, 111
unScrewSource, 113
Update, 111
modeAudioPlay
MainController, 103
Responsehandler, 139
modeValues
DropdownHandler, 54
motion
Heap_up_controller, 89
JsonRobotInfo, 96
myName
HelloWorldScript, 92

NetworkManager, 114
Awake, 116
broHeader, 127
brotliTestFile, 127
cameraDepthData, 127
client, 127
connected, 127
ConnectToServer, 117
DecodeBase64, 117
DecompressBrotli, 118
depthData, 127
Disconnect, 118
Instance, 128
InvokeConnectionStatus, 119
IsBrotli, 119
isListening, 128
OnApplicationQuit, 120
onArmLengthDataReceived, 130
OnConnected, 130
OnConnectionStatus, 130
OnDataReceived, 130
onDepthDataReceived, 130
OnDestroy, 120
onPingDataReceived, 131
OnRobotInfoDataReceived, 131
ParseArmLengthInfo, 120
ParseFloatList, 120
ParseJsonRobotInfo, 121
PingRobot, 121
port, 128
ProcessDataCoroutine, 122
ProcessDepthData, 122
ProcessPingData, 123
processRecievedData, 123
ReceiveData, 124
ReceiveDataAsync, 124
receivedDataQueue, 128
receiveThread, 129
Reconnect, 125
SendData, 125
serverIP, 129
sizeBeforeUpdate, 129
Start, 126
stream, 129

udpPort, 129
Update, 126
networkManager
ArmController, 19
DepthPointCloud, 38
DigitalTwinController, 44
DriveModeController, 49
DropdownHandler, 54
HandDetectionCube, 69
HandGestureAndRotation, 74
HandInteraction, 79
HandleReconnectButton, 81
Heap_up_controller, 89
ObjectController, 133
Responsehandler, 139
WebRTCReceiver, 169
normalButton
DriveModeController, 49
NormalizeDepthData
MeshCreator, 105
normalMode
DriveModeController, 47

obj/Debug/.NETStandard,Version=v2.1.AssemblyAttributes.cs, 179
ObjectController, 131
networkManager, 133
SendDataToServer, 132
Start, 133
Update, 133
offset_ovr_camera_rig
Heap_up_controller, 90
OnApplicationQuit
NetworkManager, 120
onArmLengthDataReceived
NetworkManager, 130
OnConnected
NetworkManager, 130
OnConnectionStatus
NetworkManager, 130
OnDataReceived
NetworkManager, 130
OnDepthDataReceived
DepthPointCloud, 36
onDepthDataReceived
NetworkManager, 130
OnDestroy
DepthPointCloud, 36
HandDetectionCube, 62
HandInteraction, 77
Heap_up_controller, 85
MainController, 101
NetworkManager, 120
WebcamStreamDisplay, 164
WebRTCReceiver, 167
OnDisable
DebugDisplay, 23
DebugDisplayPro, 27
Heap_up_controller, 86
OnDropdownValueChanged
DropdownHandler, 55
OnEmergency
Responsehandler, 137
OnEmergencyVoiceCommandActivated
Responsehandler, 137
OnEnable
DebugDisplay, 23
DebugDisplayPro, 27
Heap_up_controller, 86
OnError
WebRTCReceiver, 167
OnMessage
WebRTCReceiver, 167
onPingDataReceived
NetworkManager, 131
OnReconnectButtonClicked
HandleReconnectButton, 81
OnRobotInfoDataReceived
NetworkManager, 131
OnStartListening
Responsehandler, 138
OnTriggerEnter
ArmController, 13
HandDetectionCube, 62
OnTriggerExit
ArmController, 14
HandDetectionCube, 62
OnTriggerStay
ArmController, 14
HandDetectionCube, 63
OnVoiceCommandReceived
Responsehandler, 139
originalColor
ArmController, 20
HandDetectionCube, 69
ovr_camera_rig
Heap_up_controller, 90

ParseArmLengthInfo
NetworkManager, 120
ParseFloatList
NetworkManager, 120
ParseJsonRobotInfo
NetworkManager, 121
particles
DepthPointCloud, 38
particleSystem
DepthPointCloud, 38
pinch
ArmController.RobotControlX, 147
HandDetectionCube.RobotControlX, 148
HandGestureAndRotation.HandData, 56
Pinch_left_x
DigitalTwinController, 44
Pinch_right_y
DigitalTwinController, 44
ping
PingData, 134

PingData, 134
    ping, 134
    type, 134
PingRobot
    NetworkManager, 121
pintch
    JsonArmLengthInfo, 94
PlayArm
    ModeAudioPlay, 110
PlayDrive
    ModeAudioPlay, 110
PlayEmergency
    ModeAudioPlay, 110
PlayIdle
    ModeAudioPlay, 111
PlayScrew
    ModeAudioPlay, 111
PlayUnScrew
    ModeAudioPlay, 111
ply
    MeshReplacement, 109
Points
    DepthData, 30
poke_finger_tip_id
    SetAttachTransform, 152
pokeInteractor
    SetAttachTransform, 152
port
    NetworkManager, 128
precisionButton
    DriveModeController, 50
precisionMode
    DriveModeController, 47
ProcessDataCoroutine
    NetworkManager, 122
ProcessDepthData
    DepthPointCloud, 36
    NetworkManager, 122
ProcessPingData
    NetworkManager, 123
processRecievedData
    NetworkManager, 123

rawImage
    WebRTCReceiver, 169
README_.md, 179
ReceiveData
    NetworkManager, 124
ReceiveDataAsync
    NetworkManager, 124
receivedDataQueue
    NetworkManager, 128
receiveThread
    NetworkManager, 129
Reconnect
    NetworkManager, 125
reconnectButton
    HandleReconnectButton, 82
refreshRate
    WebcamStreamDisplay, 165
RepeatedlyDistanceCalculation
    ArmController, 14
    HandDetectionCube, 63
RequestDepthData
    DepthPointCloud, 36
ResetControlValues
    ArmController, 15
    HandDetectionCube, 63
Responsehandler, 135
    HandleResponseTest, 137
    index, 139
    modeAudioPlay, 139
    networkManager, 139
    OnEmergency, 137
    OnEmergencyVoiceCommandActivated, 137
    OnStartListening, 138
    OnVoiceCommandReceived, 139
    sendScrewCommandToTheTobot, 138
    Start, 138
    Update, 138
    voiceExperience, 139
responsehandler
    DropdownHandler, 55
Responsehandler.ScrewCommand, 149
    screw, 149
retryDelay
    RobotCamStream, 143
reverseButton
    DriveModeController, 50
reverseMode
    DriveModeController, 47
rightHand
    ArmController, 20
    HandDetectionCube, 69
robot_view_plane
    Heap_up_controller, 90
RobotCamStream, 140
    currentRetries, 143
    HandlePrepareCompleted, 141
    HandleVideoError, 141
    maxRetries, 143
    retryDelay, 143
    SetupVideoPlayer, 142
    Start, 142
    streamUrl, 143
    TryPrepareVideo, 142
    videoPlayer, 143
rotateTo170
    DigitalTwinController, 44

screw
    Responsehandler.ScrewCommand, 149
screwSource
    ModeAudioPlay, 112
SendControlValues
    ArmController, 15
    HandDetectionCube, 64
SendData

NetworkManager, 125
SendDataToServer
ArmController, 16
DriveModeController, 47
DropdownHandler, 53
HandDetectionCube, 64
HandGestureAndRotation, 71
HandInteraction, 77
ObjectController, 132
sendInterval
ArmController, 20
HandDetectionCube, 69
HandGestureAndRotation, 74
HandInteraction, 79
sendScrewCommandToTheTobot
Responsehandler, 138
serverIP
NetworkManager, 129
SetAttachTransform, 150
handSkeleton, 151
poke_finger_tip_id, 152
pokeInteractor, 152
Start, 151
SetTargetRotation
DigitalTwinController, 41
SetTargetRotationY
DigitalTwinController, 41
SetupVideoPlayer
RobotCamStream, 142
sizeBeforeUpdate
NetworkManager, 129
skeleton
WristMarker, 172
snapshotUrl
WebcamStreamDisplay, 165
speed
ArmController.RobotControlValues, 144
HandDetectionCube.RobotControlValues, 145
Heap_up_controller, 90
JsonRobotInfo, 96
speed_text
Heap_up_controller, 90
Start
ArmController, 16
DepthPointCloud, 37
DigitalTwinController, 41
DriveModeController, 48
DropdownHandler, 54
HandDataTransmission, 57
HandDetectionCube, 65
HandGestureAndRotation, 72
HandInteraction, 77
HandleReconnectButton, 81
Heap_up_controller, 86
HelloWorldScript, 92
MainController, 101
MeshCreator, 106
MeshReplacement, 108
ModeAudioPlay, 111
NetworkManager, 126
ObjectController, 133
Responsehandler, 138
RobotCamStream, 142
SetAttachTransform, 151
VideoStream, 159
VideoStreamController, 161
WebcamStreamDisplay, 164
WebRTCReceiver, 168
WristMarker, 171
StartListener
UdpListener, 153
stream
NetworkManager, 129
streamUrl
RobotCamStream, 143
strength
ArmController.RobotControlX, 147
HandDetectionCube.RobotControlX, 148

testFunction
DigitalTwinController, 42
textMeshPro
HelloWorldScript, 93
texture
WebRTCReceiver, 169
time
CameraDepthData, 21
DepthData, 29
TryPrepareVideo
RobotCamStream, 142
type
PingData, 134

UdpListener, 152
listenPort, 154
Main, 153
StartListener, 153
udpPort
NetworkManager, 129
UnityMainThreadDispatcher, 154
_executionQueue, 156
Awake, 155
Enqueue, 156
Instance, 157
Update, 156
unScrewSource
ModeAudioPlay, 113
Update
ArmController, 17
DebugDisplay, 24
DebugDisplayPro, 27
DepthPointCloud, 37
DigitalTwinController, 42
DriveModeController, 48
HandDataTransmission, 57
HandDetectionCube, 65
HandGestureAndRotation, 72

HandInteraction, 77
HandleReconnectButton, 81
Heap_up_controller, 86
MeshCreator, 106
MeshReplacement, 108
ModeAudioPlay, 111
NetworkManager, 126
ObjectController, 133
Responsehandler, 138
UnityMainThreadDispatcher, 156
VideoStream, 160
WebRTCReceiver, 168
WristMarker, 171
update_head_up_canvas_position_and_rotation
Heap_up_controller, 86
updateButtonColor
DriveModeController, 48
updateDriveModeData
DriveModeController, 49
UpdateVisualIndicator
ArmController, 17
HandDetectionCube, 65

Vector3Data, 157
Vector3Data, 157
x, 158
y, 158
z, 158
videoPlayer
RobotCamStream, 143
VideoStreamController, 161
VideoStream, 159
Start, 159
Update, 160
VideoStreamController, 160
Start, 161
videoPlayer, 161
VideoStreamReceiver, 162
visualIndicatorTransform
ArmController, 20
HandDetectionCube, 69
voiceExperience
Responsehandler, 139
voltage
Heap_up_controller, 90
JsonRobotInfo, 96
voltage_text
Heap_up_controller, 91
volume
MeshCreator, 107

WebcamStreamDisplay, 163
_refreshWait, 165
_renderer, 165
ApplyTexture, 164
FetchSnapshotRoutine, 164
OnDestroy, 164
refreshRate, 165
snapshotUrl, 165
Start, 164
WebRTCReceiver, 166
_webSocket, 168
index, 169
networkManager, 169
OnDestroy, 167
OnError, 167
OnMessage, 167
rawImage, 169
Start, 168
texture, 169
Update, 168
Width
DepthDataGenerator, 32
wrist
HandGestureAndRotation.HandData, 56
WristMarker, 170
greenDot, 171
Hand_MiddleTip, 172
skeleton, 172
Start, 171
Update, 171

X
DepthDataPoint, 33
x
ArmController.RobotControlValues, 144
ArmController.RobotControlX, 147
HandDetectionCube.RobotControlValues, 146
HandDetectionCube.RobotControlX, 148
Vector3Data, 158

Y
DepthDataPoint, 33
y
ArmController.RobotControlValues, 145
ArmController.RobotControlX, 147
HandDetectionCube.RobotControlValues, 146
HandDetectionCube.RobotControlX, 148
Vector3Data, 158

Z
DepthDataPoint, 33
z
ArmController.RobotControlX, 147
Vector3Data, 158

Y Working hours

Name/ week	Technical																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	Total
Shahin	0	26	24	25	29	32	27	25	48	52	27	0	1	26	40	32	48	53	46	71	13	0	0	0	644
Aditi	0	8	21	22	30	33	18	29	39	43	19	0	18	33	49	47	41	47	38	58	18	0	0	0	606
Oscar	2	18	22	6	31	37	19	22	40	44	26	0	1	37	44	39	41	41	40	71	20	0	0	0	598
Adrian	0	6	20	10	15	20	15	19	19	32	22	3	13	28	49	51	57	53	39	85	17	0	0	0	570
Henrik	0	0	20	9	29	28	23	26	24	40	23	0	1	36	43	41	49	51	45	78	18	0	0	0	580
Total	**2**	**56**	**106**	**72**	**133**	**150**	**101**	**120**	**170**	**210**	**116**	**3**	**34**	**159**	**224**	**210**	**236**	**245**	**208**	**362**	**86**	**0**	**0**	**0**	**2997**

Name / week	Administrative																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	Total
Shahin	11	15	8	0	10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
Aditi	9	17	4	2	0	1	2	2	1	0	2	0	0	3	1	2	0	2	0	0	0	0	0	0	47
Oscar	7	6	6	3	3	0	0	2	0	11	2	0	0	1	0	0	0	0	0	0	0	0	0	0	38
Adrian	8	20	5	1	15	6	3	2	13	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80
Henrik	8	22	6	0	3	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47
Total	**41**	**80**	**27**	**6**	**30**	**12**	**5**	**6**	**23**	**20**	**4**	**0**	**0**	**4**	**1**	**2**	**0**	**2**	**0**	**0**	**0**	**0**	**0**	**0**	**259**

Name / week	Technical & administrative																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	Total
Shahin	11	41	32	25	39	37	27	25	48	52	27	0	1	26	40	32	48	53	46	71	13	0	0	0	692
Aditi	9	24	25	24	30	34	20	31	40	43	21	0	18	36	50	49	41	49	38	58	18	0	0	0	653
Oscar	9	24	28	9	34	37	19	23	40	55	27	0	1	37	44	39	41	41	40	71	20	0	0	0	636
Adrian	8	26	24	11	29	26	18	21	32	41	22	3	13	28	49	51	57	53	39	85	17	0	0	0	649
Henrik	8	22	25	9	32	28	23	26	33	40	23	0	1	36	43	41	49	51	45	78	18	0	0	0	627
Total	**43**	**136**	**133**	**77**	**163**	**161**	**106**	**125**	**193**	**230**	**119**	**3**	**34**	**163**	**225**	**212**	**236**	**247**	**208**	**362**	**86**	**0**	**0**	**0**	**3256**

This page was intentionally left blank.