HYDRO PLANT

Bachelor's thesis

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Kongsberg

Acknowledgements

Thank you to Hydroplant Technologies AS for giving us the project and to the staff and fellow students at the University of South-Eastern Norway for their help and guidance.

We would also like to thank Hivemind for supplying us with the LaTeX template as a starting point [1].

Abstract

This project focuses on moving a lettuce from one location to another and detecting the type of lettuce. This is a research project for Hydroplant Technologies, who wants to develop their own in-house solution and a starting point that they can develop further. For this project, we chose to develop an articulated robotic arm that can detect where a lettuce is in a 3 dimensional room. It shall identify the lettuce, detect its type, and then pick it up and move it to a designated area.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 7, 9–11, 29, 43, 62, 63, 65, 68, 69, 125, 126, 149, 152–154, 156, 159, 193, 445, 446, 451, 452, 455, 457, 459, 461, 467, 502

API Application Programming Interface. 10, 126, 475, 488

BOM Bill of Materials. 119

CAD Computer-Aided Design. 103, 172, 176, 270, 273

Central Raspberry Pi 5. 64, 65, 137, 140, 142, 143

Core Arduino R4 WiFi. 64, 65, 137–140, 142, 143

CSS Cascading Style Sheet. 128

CV Computer Vision. 7, 41, 43, 126, 149, 152–154, 156, 454, 490

DOF Degrees Of Freedom. 83

EMC Electromagnetic Compatibility. 122

EMI Electromagnetic Interference. 122

ESP32 Espressif32. 149

FBD Free Body Diagram. 16, 108, 109

FDM Fused Deposition Modeling. 103, 272

FEM Finite Element Method. 173, 191

FOS Factor of Safety. 281

FOV Field Of View. 151

FR4 Fire-Retardant 4. 118

GPU Graphical Processing Unit. 457

GUI Graphical User Interface. 126, 484

HMI Human Machine Interface. 6, 16, 29, 125–128, 130, 131, 193

HPT Hydroplant Technologies. 36, 38, 80, 89, 90

HSV Hue, Saturation, Value. 490

HTML Hyper Text Markup Language. 128

HTTP HyperText Transfer Protocol. 29, 125, 126, 488

HTTPS HyperText Transfer Protocol Secure. 66

IC Integrated Circuit. 122

IP Internet Protocol. 126

ISP Image Signal Processor. 64

JS JavaScript. 128

JSON JavaScript Object Notation. 10, 488

LAN Local Area Network. 29

MDF Medium-density Fiberboard. 83, 92, 165

ML Machine Learning. 9, 125, 152, 445, 459

MVC Model View Controller. 16, 128, 129

PCB Printed Circuit Board. 31, 65, 118, 119

PLA Polylactic Acid. 103, 161, 175, 191, 272

QoS Quality of Service. 69

REST Representational State Transfer. 488

RGB Red, Green, Blue. 490

ROS2 Robot Operating System 2. 64, 69, 125

RPM Revolutions per minute. 138

RTOS Real-Time Operating System. 136

SolidWorks SolidWorks®. 172, 173, 176, 179, 181, 191

STL STL. 173, 180

the University University of South-Eastern Norway. 36, 59, 189

TPU Thermoplastic Polyurethane. 102, 103, 161, 175, 176, 191, 272

WAF Web Application Framework. 29

WAN Wide Area Network. 29

WCGI Web Server Gateway Interface. 32

Glossary

BLDC A brushless DC motor is an electric motor that uses magnets and coils to translate electrical energy to mechanical energy.. 113

CC BY 4.0 Open source license which stands for "Creative Commons Attribution 4.0". It allows you to share and adapt the material as long as attribution is provided. 451, 452

DDS The Object Management Groups Data Distribution Service is a standardised middleware protocol for real-time, scalable publish/subscribe communication, featuring configurable Quality of Service policies for reliability, latency, and resource management. 69

Docker An open-source platform for containerizing applications. Docker packages software and its dependencies into lightweight, portable containers that can run consistently across different environments, ensuring isolation and simplified deployment. 69, 72

Doxygen Software for generating code documentation. 125

ESP32 Simple and low-cost system on a chip manufactured by Espressif Systems. 149

ESP32-CAM The ESP32-CAM is a small embedded device containing the ESP32 SoC (System on a Chip) and an integrated camera sensor. 29, 135, 149–151, 476

Fine tuning Fine tuning is the process of taking existing AI models and improving their accuracy within certain domains. This is achieved by using a dataset, which is a collection of inputs and their expected outputs. 451

Finite Element Method blabla. 181

Flask Flask is a Web Application Framework (WAF) written in Python. We use it to to host the HMI which is a web application hosted on either a Local Area Network (LAN) or Wide Area Network (WAN), depending on the customers use case. It was also used for some earlier iterations of the HTTP networking between ESP32-CAM and Central.. 32, 128

Google Colab Google Colab is an online service which allows you to run code on dedicated and powerful hardware. It is especially useful for training AI models. 451

HMI The user-facing part of a system that allows a human operator to monitor, control, or interact with the system via a user interface. 63

Hugging Face An open source platform where the machine learning community collaborates on AI models, datasets and applications. 446, 455

JQuery JavaScript library for interfacing with HTML elements in a webapp. 128

Kanban A visual workflow method for managing tasks and optimising flow. Work items, often called tasks or in our case backlog-items, are represented on a board with columns (To Do, In Progress, Needs Review, and Done). Teams pull new tasks only when they have capacity, respecting Work-In-Progress. Kanban focuses on continuous delivery, transparency.. 44–46

limit switch A limit switch is a sensor used to detect the physical limits of a motors movement. It triggers once the maximum point of the motors movement range has been met. It then provides a signal to stop further motion, thus preventing mechanical overrun or damage.. 65, 68

Macbook Air M1 A laptop released by Apple in 2020. It has been used as a test rig for AI model benchmarking in our project.. 457

micro-ROS micro-ROS is a lightweight version of ROS2 designed to run on microcontrollers with limited resources. It enables small embedded devices to participate as nodes in a ROS2 system by communicating over standard ROS2 protocols. . 65, 72

microstep A small fraction of a full step in a stepper motor (for example, dividing a 1.8ř full step into 16 equal parts), used to achieve smoother motion and finer position control. 138

MQTT MQTT (Message Queuing Telemetry Transport) is a publishsubscribe messaging protocol designed for efficient communication between devices over networks with limited bandwidth or high latency. It uses a central broker to route messages between publishers and subscribers based on topic names.. 65, 66, 68, 69, 72, 140, 142, 143

node An isolated ROS 2 process responsible for a specific task. Nodes communicate with one another via topics (asynchronous publish/subscribe), services (synchronous request/response), and actions (preemptible, longrunning goals with feedback).. 69

OpenCV Open source computer vision library. 154, 496–498

PlantCV Open source computer vision library based on OpenCV for plant specific tasks. 154

Product Backlog An ordered list of all tasks needed to progress a product towards completion. The list is prioritised and reviewed regularly to ensure that the most important work is done first. The Development Team pulls items from the backlog into each sprint. The backlog is dynamic and evolves as new requirements and tasks emerge and priorities change.. 45, 46

Python Python is a programming language renowned for its easy of use, and is often used for scientific purposes like Artificial Intelligence. 29, 32, 128

Roboflow A service which hosts a collection of fine-tuned open source AI models and datasets online. 451

ROS2 Robot Operating System 2 - An open source framework for developing and deploying robotic applications, providing libraries and tools for building modular systems with inter-process communication, hardware abstraction, and various high-level functionalities.. 57, 59, 68, 69, 72, 137

Rotary Encoder A rotary encoder is a device that reports the position or motion of a shaft.. 65

SCRUM An agile framework for managing work in small teams. Scrum breaks projects into short, fixed-length cycles called sprints. Key roles include a Product Owner (who sets priorities), a Scrum Master (who helps the team follow Scrum), and the Development Team (who build the product). Scrum uses simple artifactslike the Product Backlog (a prioritised list of work, often called tasks or backlog items)and regular eventssuch as Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospectiveto plan, track, and improve the teams work continuously.. 45

topic In publish/subscribe messaging, a topic is a named channel where the publishers their data too, and which subscribers receive them. This decouples the senders from the receivers, allowing for data to be sent asynchronously without being directly connected.. 65, 69

UART UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol used for serial data exchange between two devices. It transmits data asynchronously, meaning it does not require a shared clock signal, and is commonly used for communication between microcontrollers, sensors, and other embedded components.. 65

utf-8 Widely used character encoding format for text strings. 131

via For PCBs a **via** is a connection between the layers of the PCB.. 118, 122

Visual Studio Code An open-source code editor used for writing and debugging code in multiple languages. 48

WebSocket A communications protocol built for the web. Initial handshake is done over HTTP, followed by low-level TCP. 129

Werkzeug A simple WCGI library that contains utility functions which are useful for web server applications. is built on top of this library. 130

Zephyr Real Time Operating System made by the Linux Foundation. 136

1 Introduction

Our bachelor's thesis is given by Hydroplant Technologies AS. Our task is to develop a versatile robot system that autonomously harvests and processes several types of vegetables.
Food production around the world has to increase considerably to meet the growth of population. Climate change needs us to rethink sustainability around how we produce eatables. Hydroplant's goal is to find cost effective solutions in a environment-friendly way.

1.1 Hydroplant Technologies AS

DAB | *SME*

Hydroplant Technologies started out as a student project provided by USN, and founded AS in may 2024.
Their goal is to innovate and automate the vertical farming setup. The company is developing autonomous harvest solutions, from seed to fully packed goods, see figure 1 for a simplified view of the process and figure 2 for a more detailed overview of the process. The goal with this technology is to create systems that are cost-effective while also ensuring that the systems are highly optimized and effective.

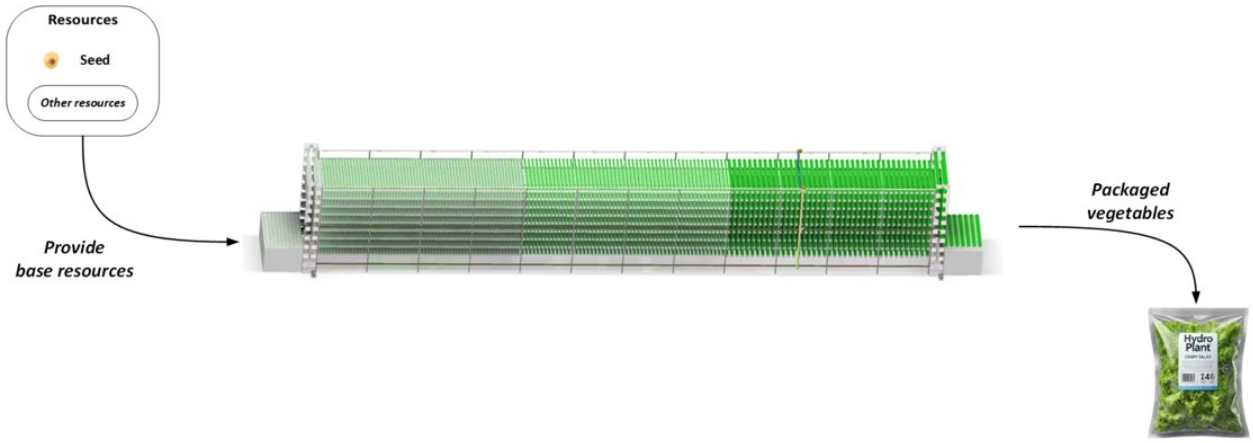


Figure 1: Hydroplant Technologies - From seed to plant

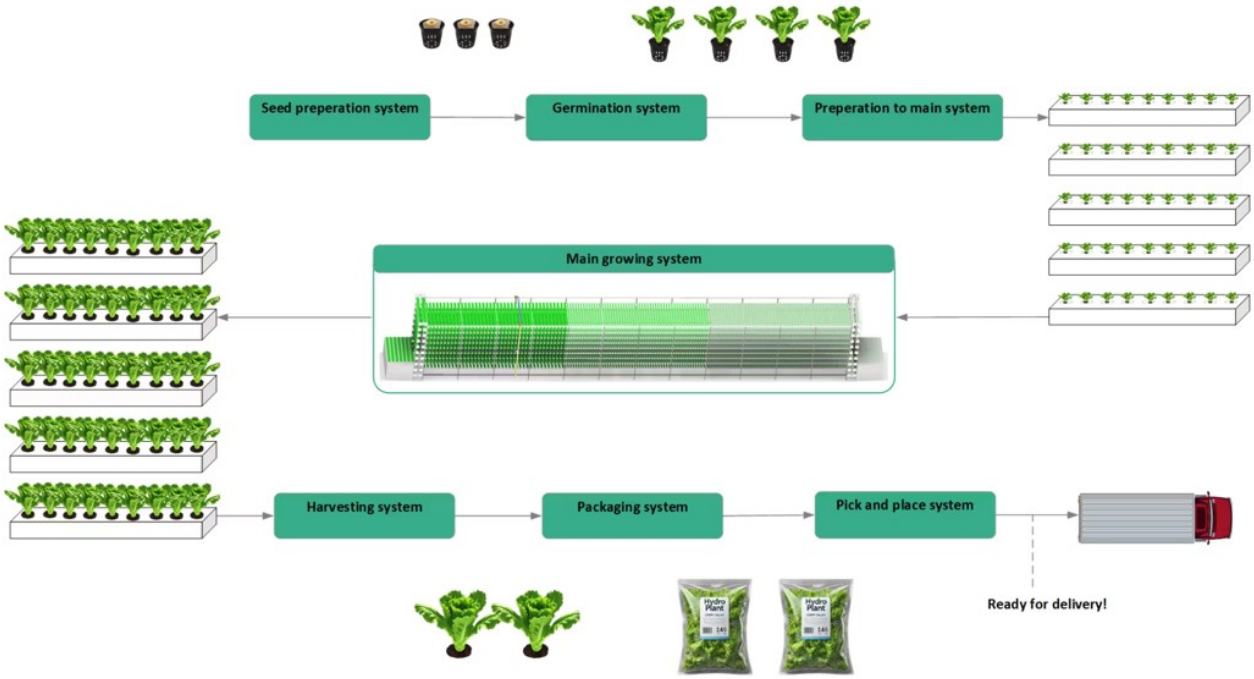


Figure 2: Hydroplant Technologies overview

Figure 3: Leafy greens in Hydroplant Technology system

This is a small test run at the USN Kongsberg campus.

1.2 Project description

SME | *JCDH*

Hydroplant Technologies AS wants our help to develop a versatile system for the automatic harvesting and processing of salads. In their overall system, we will only focus on the harvesting part. From seed to fully grown salads, they will enter our system where they will be:

1. Recognized by AI technology.
2. Handled with care and attention so that we do not harm the salad.
3. Picked up and transferred from their growing station.
4. Positioned in a specific and defined place to be ready for next step in Hydroplant's system.

The existing autonomous solutions in today's market are expensive, and Norwegian farmers do not have the economy that is required to upgrade. There are also negative effects in manual farming: it is high-cost and there are hygiene-related risks. Bacteria, viruses and parasites can contaminate the products from poor hand hygiene by those who harvest vegetables.
Hydroplant Technologies wants to remove manual work from harvesting systems, to ensure better hygiene and to increase the durability of vegetables. In addition, their goal is to enlarge vegetable cultivation, and to make it more efficient. For an overview of Hydroplantts system, and where Leafy Automation will be operating see Figure 4.

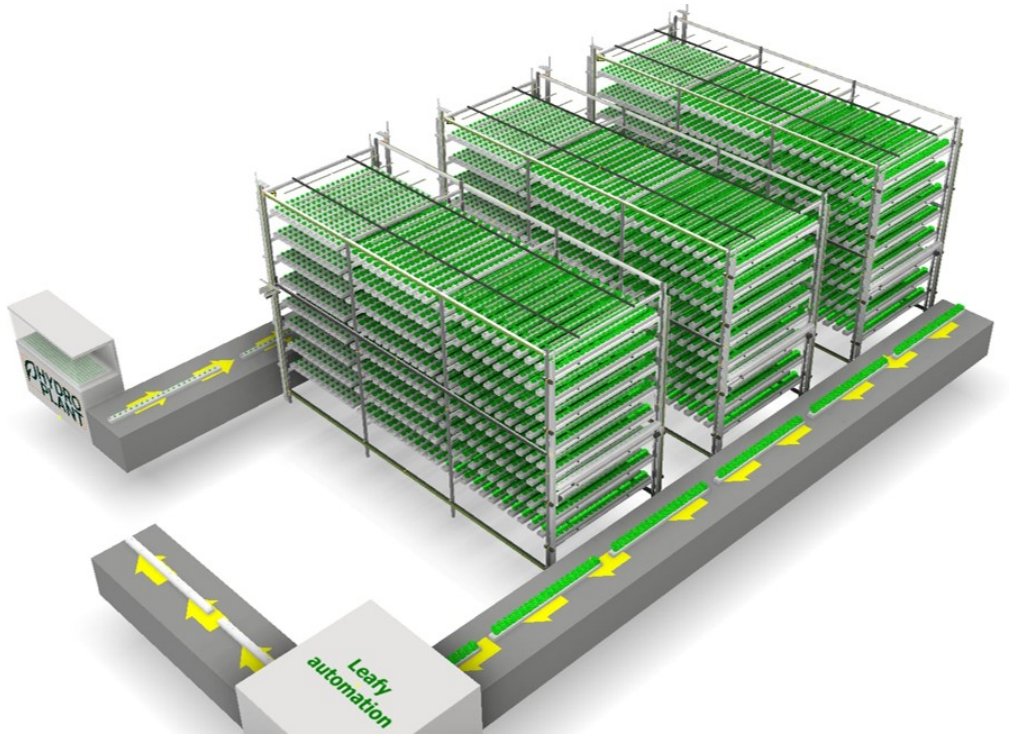


Figure 4: Leafy Automation Systems position in the Hydroplant Technologies ecosystem

1.3 Budget and expenses

BMR | *SME*

Keeping the costs at a minimum is a key requirement for this bachelor project (see section 7). The given budget is 30 000,- NOK, in addition the bachelor group has received some hardware from HPT and has access to resources at the University. An overview of the project expenses can be seen in appendix I.

1.4 Group members

The bachelor group is a multidisciplinary group consisting of 1 electronic, 2 software and 3 mechanical engineer students. They are all presented in tab.1. Working in several disciplines can pose a challenge but is also a great learning possibility for the group members.

1.4.1 Authors

The author and proofreader for each section and subsection is identified by their initials to the right of the heading (the initials can be seen in (tab. 1). The first initials belong to the author/-s which is responsible for the main content. The other initials identify the proofreader who has looked over the text to ensure a good flow and coherence as well as correcting spelling mistakes. If there are more authors that have been working on a section together, their initials will be on the left side of the line. Some sections will only have an author and no proofreader due to time constraints.

	***Name***	**Sunniva Myrvang Eineteig**
	Initials	SME
	Discipline	Mechanical engineer - Product development
	Role(s)	External contact & Instagram
	***Name***	**Beatrix Møller Rimestad**
	Initials	BMR
	Discipline	Mechanical engineer - Product development
	Role(s)	Internal contact, time wizard & Overall structural integrity
	***Name***	**Daniels Aleksandrs Blomnieks**
	Initials	DAB
	Discipline	Mechanical engineer - Product development
	Role(s)	Risk-analysis, interface between parts & LaTeX
	***Name***	**Jim Christian Dale Haukvik**
	Initials	JCDH
	Discipline	Computer engineer - Cyber physical systems
	Role(s)	Artificial Intelligence / Computer Vision, Camera, Human Machine Interface, Website & ClickUp
	***Name***	**Elin Gravningen**
	Initials	EG
	Discipline	Computer engineer - Cyber physical systems
	Role(s)	System Architecture, Robotics & Project Fascilitator
	***Name***	**Vetle Myhre Nilsen**
	Initials	VMN
	Discipline	Electrical engineer - Cybernetics
	Role(s)	Electronics & LaTeX

Table 1: Group members

2 Stakeholders

BMR |

When developing a new product, it is crucial to identify the different stakeholders. Analysing their interest and influence towards the product will help in how they should be handled and/or included in the development process.

For this project the main stakeholder is Hydroplant Technologies AS, as the client they both have high interest and high influence over the product and must be consulted regularly. Their input will help in defining the requirements needed for the product. An iterative approach in combination with close collaboration with the client can ensure that the product evolves in the wanted direction. A design review halfway through the project period provided useful feedback (see section 12).

Another stakeholder to regard is the possible customer of HPT. These are the ones who will actually use our subsystem in their production line, and will have valuable information about what aspects around the harvesting that are important to consider. A company visit gave the project group valuable input (see section 1.4.1).

A high influence stakeholder group is the ones responsible for the national regulations and laws that deal with machines, electricity and food safety. This project will focus mostly on the initial development and not the final production, still the needs of these stakeholders will be taken into consideration when making the requirements for the product. Especially in the development of the gripper, that will have direct contact with the edibles, the regulations for food safe materials is an crucial aspect (see section C). For future work it will be important to consider what is required to attain different certifications necessary to put the product on the market.

3 Company Visit: O. Espedal Handelsgartneri AS

SME | *JCDH*

As part of our collaboration with Hydroplant AS we conducted a field visit to Osmund Espedal Handelsgartneri AS to gain practical insights into modern horticultural practices. The visit provided essential contextual understanding that has significantly informed the design and functionality of our robotic system.

Figure 5: O. Espedal Handelsgartneri AS visit

O. Espedal Handelsgartneri AS is a family-owned horticultural enterprise located in Lier, Norway. The company has been in continuous operation since 1914 and family-owned since 1917. Over the past two decades, the business has specialized exclusively in the cultivation of Crispi lettuce and cucumbers, which are supplied primarily to Bama Gruppen AS.

Relevance to Our Project

Our robotic system is designed to autonomously harvest various types of leafy greens, one of which is Crispi lettuce. Observing the full production cycle at O. Espedal, from seed to harvest, was both instructive and valuable. The visit offered a detailed view into the operational logistics, technical setup, and workflow efficiency of a modern horticultural facility.

Notably, O. Espedal employs both traditional soil-based agriculture and hydroponic cultivation techniques. While Hydroplant AS currently uses only hydroponics, the exposure to both cultivation methods helped broaden our understanding and ensured that our robot design is flexible enough to accommodate potential future shifts in production strategy.

Figure 6: Lettuce study

Key Observations and Design Implications

During the visit, we closely studied the harvesting process, which remains fully manual at O. Espedal. The growing beds automatically advance the plants to the end of a working table, where human workers harvest each lettuce head by hand. Workers remove roots and damaged leaves before placing the products into packaging crates.

This process revealed several critical insights:

- **Plant Tilting During Movement:** As the growing beds advanced, the lettuce heads were observed to tilt due to a lack of structural support. This informed our decision to incorporate a rotational capability into our robotic gripper, enabling it to adapt to tilted or irregularly oriented lettuces during harvesting.

Figure 7: Lettuce with roots

- **Manual Removal of Damaged Leaves:** Damaged or unmarketable leaves were manually discarded into separate bins. Based on this, we decided to include a disposal zone within the robot's working area for damaged plants and waste material.

Figure 8: Lettuce ready for pick-up, with disposal bins

- **Overlapping Foliage:** In many cases, lettuce leaves overlapped, making it difficult to distinguish between individual heads. To address this, we drafted a plan for the implementation of a Computer Vision (CV) system to identify and separate individual lettuce heads, as well as to distinguish between healthy and damaged foliage.

Figure 9: Lettuce heaven

- **Need for Interchangeable Grippers:** Another important takeaway from the visit was the necessity for the robotic gripper to be interchangeable. While the current focus is on harvesting lettuce, future applications may involve other vegetables or entirely different tasks, such as sowing seeds. Since seeds are much smaller and require a different handling mechanism, the ability to swap out the end-effector will enable the robot to be used for a wider range of agricultural operations.

Figure 10: Pelleted seeds

- **Plant Fragility and Transportation:** The farmer informed us that the plants are very fragile and do not tolerate movement or handling well. Their grow bed transportation system is slow and gentle, and works well primarily because the individual plants are supported and held in place by the surrounding plants, which helps keep them stable

until they reach the harvesting line. Even so, many are damaged to the point where they need to be discarded. For this reason, the farmer advised that it would be much safer to retain the plant transportation system as it is and instead position multiple robots alongside it. This would cause significantly less stress and damage to the plants than, for example, removing the grow beds from the growing area and transporting them to the robot on a conveyor belt.

Conclusion

The visit to O. Espedal Handelsgartneri AS provided real-world context that has greatly influenced the development of our harvesting robot. The insights gained have translated into specific design decisions, including a gripper with rotational flexibility, a designated disposal area, an AI/Computer Vision-based sorting and detection system, and a modular, interchangeable gripper interface that allows for adaptability to future agricultural tasks.

Figure 11: Crispi-smile

4 Project Methodology

In this chapter, we describe how our multidisciplinary team structured our work, the processes and tools we used.

4.1 Leadership model

EG |

We adopted a flat leadership model in which all six team members shared equal responsibility for planning, decision-making, and deliverables. Overseeing tasks such as leading meetings, taking minutes, supervising interfacing, and contributing to the risk analysis were considered shared responsibilities among all the group members. The goal of this approach is to encourage creativity, collaboration and shared ownership.

4.2 Structure

EG |

Figure 12: Team departments by discipline

We created three departments to reflect each of the three disciplines of the group, where the electronics department is supported by the other two disciplines. See Figure 12. Under this flat leadership structure, the group as a whole were in charge of overseeing project development and ensuring that deadlines are met. We used a Kanban board to keep track of workflow, but internally, each department were encouraged to use their own preferred methods to ensure. In practice this meant that design and development decisions, and tasks, were defined on the group level, but how the the methods in which the tasks were completed was at the departments' prerogative.

4.3 Project model

EG |

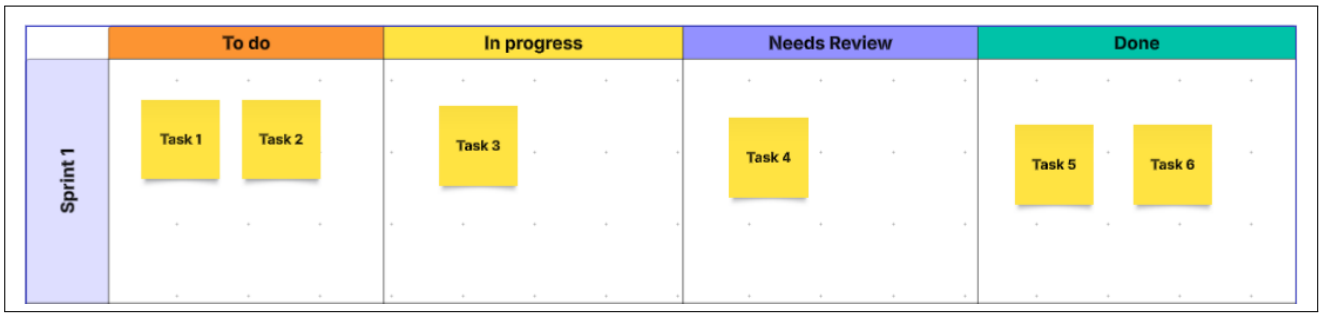


Figure 13: Illustrating our kanban board.

The group's aim from the start was to use agile project development principles. Though we started with the intention of basing delopment on SCRUM methodology, this did not prove to be a good fit. We therefore moved over to Kanban, yet kept some of the key features from SCRUM that worked well and aided our work. We found that developing a product Product Backlog, doing weekly sprint planning sessions, daily stand-ups, and sprint retrospectives worked well.

Our task items followed a set structure, as seen in Figure 14:

- A short and easy to understand title
- A short description.
- Acceptance criteria and verification/validation testing.
- An owner, once claimed during a sprint.

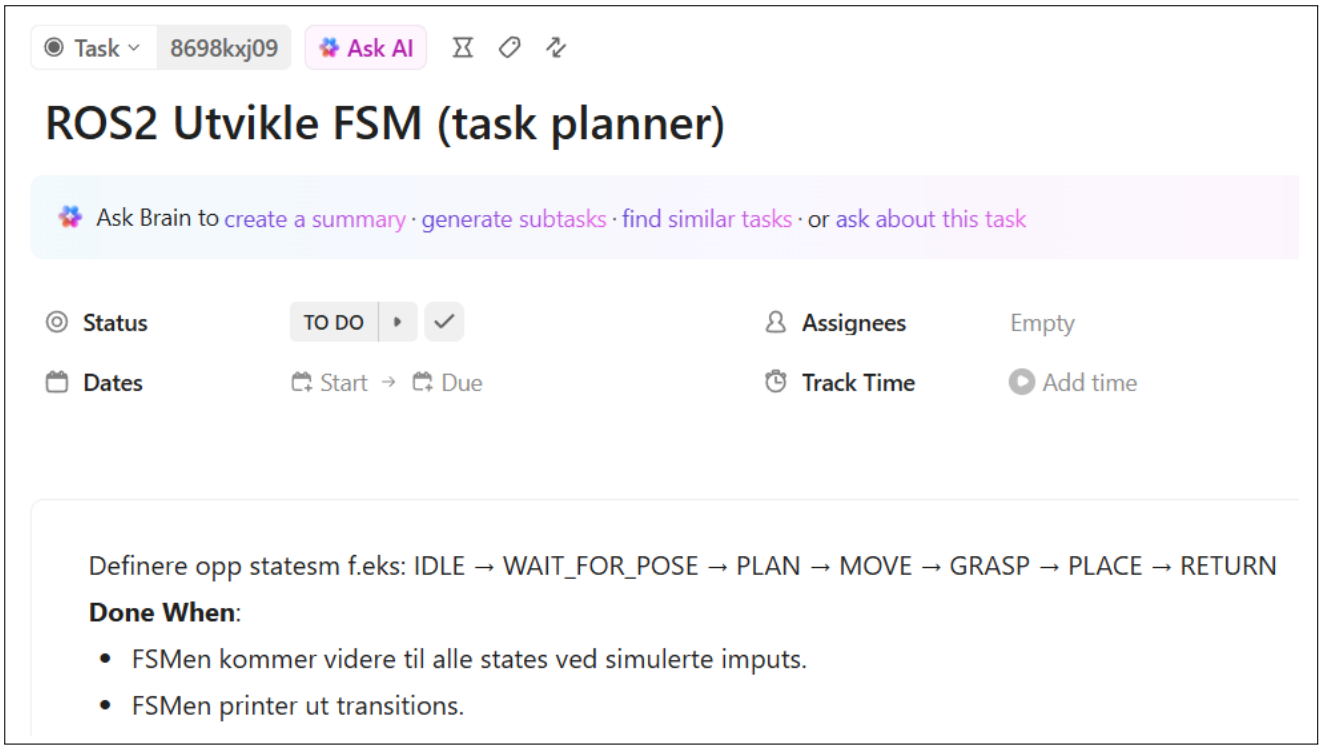


Figure 14: Example of a kanban task/ product backlog item

Our Kanban summaries can be found in Appendix B.7.

Other key features we used in our work flow:

- **Stand-up meetings**. Short, max 15 minute where we inform everyone about what we're planning to do that day, along with any hindrances we're facing or support we need from other members of the group.
- **Sprint Planning** Once a week we look at the product and pull items from it and place it onto our sprint plan. The sprint plan consists of the work we're planning to get done the coming week. Product Backlog
- Supervisor meetings. We held regular meetings with our supervisors where we received constructive feedback and discussed ideas based on our progress the past week.
- Sprint Retrospectives. These were held following sprint completions to evaluate our process and optimise our workflow.

4.4 Work shops

EG |

A number of workshops and educational excursions were held:

- SCRUM workshop before project commencement.5
- Workshop with Hydroplant Technologies 31/01/2025
- A visit to Espedal Gartneri where we learned about aggricultural farming from a farmer's perspective.1.4.1

4.5 Office and remote work days

EG |

Before Easter Exams we held regular office days Wednesdays through Friday. The team maintained core working hours from 9.00-15.00, except for one member who attended office office hours 9.00-12.00, with some flexibility. After this point and through to completion, we all worked core hours 09.00-15.00 including a regular home office day on Tuesdays each week. Tuesdays were considered "writing days".

Out project method resources can be found in the appendix in Section 3

Please refer to Appendix B.2 for earlier work written on project model.

4.6 Software used

JCDH | *SME*

Below is a list of software used in the development of our project.

4.6.1 Shared software

JCDH, EG |

The following table outlines the software used by all members of our team. These are important pieces which facilitate communication and project management.

Artificial Intelligence
Some group members have used AI tools to support our workflow. This includes AI-assisted text formatting to improve readability, without altering technical content. Copilot has been

used in Visual Studio Code for auto-completion and to explore different code strategies.

Software	Description	Version	Href
ClickUp	Project and task tracking system	Latest	https://clickup.com
Clocklify	Time tracking software	Latest	https://clockify.me
Overleaf	Software for editing LaTeX documents (used for the thesis itself)	Latest	https://overleaf.com
Office 365	Microsoft software such as Word, Excel, Powerpoint, OneDrive and Teams	Latest	https://office.com

Figure 15: List of software used in the project (common software).

4.6.2 Computer engineering

JCDH | *SME*

Software	Description	Version	Href
Python	Programming language used for development.	Latest	https://www.python.org
Poetry	Dependency management and packaging tool for Python.	Latest	https://python-poetry.org
Doxygen	Tool for generating documentation from source code.	Latest	https://www.doxygen.nl
VSCode	Source code editor with extensive support for development.	Latest	https://code.visualstudio.com
draw.io	Diagram creation software.	Latest	https://draw.io
Postman	HTTP API testing software.	Latest	https://postman.com
Platform.IO	The use of PlatformIO allows us to prototype quickly and change the development board in the future without changing software.	Latest	https://platformio.org
Google Colab	Jupyter notebook service used for training AI models.	Latest	https://colab.google

Figure 16: List of software used in the project (computer engineering).

4.6.3 Mechanical engineering

JCDH | *SME*

Software	Description	Version	Href
SolidWorks	CAD modelling software for 2D and 3D workflows. Also used for analysis.	Latest	https://solidworks.com

Figure 17: List of software used in the project (mechanical engineering).

4.6.4 Electrical engineering

JCDH | *SME*

Software	Description	Version	Href
Altium Designer	PCB design software.	Latest	Altium Designer
LTspice	Software for creating and simulation electronic circuits.	Latest	Analog Devices LTSpice

Figure 18: List of software used in the project (electrical engineering).

4.7 Other work

JCDH | *SME*

Please refer to Appendix B.1 for earlier project work, and Appendix B.5 for work on design and website.

5 Requirements

5.1 Introduction

JCDH | *SME*

Requirements are an important part of any project. They must be well defined, which means that they are quantifiable using methods such as testing, validation and verification. As part of the development of the requirements, stakeholders naturally provide input. It has been our intention to follow the general recommendations defined in the ISO 29148 standard when developing these requirements [6]. Please refer to Appendix A for earlier work done on requirements.

5.2 User stories

JCDH | *SME*

Before defining each requirement in detail, it is useful to create user stories to get an idea of how a user will actually use the system. Figure 19 outlines the user stories we have defined, which is divided into an "User story ID", a "Short title" and the "User Story" itself.

User story ID	Short title	User Story
US-01	Autonomous harvesting	As a farmer, I want the harvesting system to harvest plants autonomously so that I can reduce the need for manual labor.
US-02	Plant recognition	As a farmer, I want the harvesting system to analyse plants so that it can optimise harvesting for different plant types and handling needs.
US-03	Optimal handling	As a farmer, I want the system to optimise handling of each plant, so that I can deliver produce meeting the uniformity and quality expectations of my customers.
US-04	Safety and efficiency	As a farmer, I want the harvesting system to operate efficiently, and safely to minimise risk of damages to personnel, equipment or plants.
US-05	Modularity and expansion	As a future developer, I want the system to be easy to expand or modify, so I can add new features or update existing parts without disrupting the rest of the system.
US-06	Scalability	As a farmer, I want the system able to scale up to larger or more complex growing areas.

Figure 19: User Stories

5.3 Requirements in detail

JCDH | *SME*

Our requirements follow these defined guidelines:

- All requirements shall reside within a specific US (User Story) ID, which is related to context of the user story.
- All principal requirements shall have a Use Case ID.
- All principal requirements will have a named Use Case.

- All requirements shall have a Requirement ID defined in the format *REQ-xyz-abcd*.

- All requirements shall have a Requirement description.

- All requirements are given a priority of either A, B or C in order of importance.

Figure 20, 21, 22, 23, 24 and 25 outlines all requirements.

User Story ID: US-01
User story short title: Autonomous harvesting

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-001	**System requirements**	REQ-001-0001	The system shall be able to move the combined weight of the arm, gripper and payload.	A
		REQ-001-0002	Cable management - avoid damage to cables in operation	A
		REQ-001-0003	The project shall keep monetary costs as low as possible by relying primarily on respources provided by Hydroplant Technologies and the University of South-Eastern Norway.	A
		REQ-001-0004	The Leafy Automation system shall have a Human Machine Interface that gives access to operational functionality: capture images from the live video feed, see system status and AI results.	A
UC-002	**System integration**	REQ-001-0005	The mounting of the base to a fixed horizontal surface shall be fascilitated through 4 x M8 bolt holes in the base.	A
		REQ-001-0006	The system shall be be able to connect to an external 24 V DC power supply.	A
UC-003	**The working area specifications**	REQ-001-0007	All system operations shall take place in a circular area of Ø1200 mm with the robot arm base in the middle.	A
		REQ-001-0008	The work area shall be divided into 4 zones: pick Up zone, Placement zone, processing zone and robot base zone.	A
		REQ-001-0009	The base shall be able to rotate 300 degrees in the horizontal plane.	B
UC-004	**Moving plants using robotic arm**	REQ-001-0010	The robotic arm shall grip plants that are delivered to the pick up zone.	A
		REQ-001-0011	Once the robotic arm has gripped the plant, the robot shall move it to the next area as instructed by the control unit.	A
UC-005	**Root cutting**	REQ-001-0012	The root cutting mechanism shall perform a clean root cutting on first attempt at least 95% of the time.	C
		REQ-001-0013	The system shall verify that roots have been cut off correctly.	C
		REQ-001-0014	The system shall perform 3 attempts to cut roots off a plant successfully before discarding the plant.	C
		REQ-001-0015	The system shall stop the cutting process if >x N resistance is detected.	C

Figure 20: Requirements for US-01

User Story ID: US-02
User story short title: Plant recognition

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-006	**Capture an individual plant in the pickup area**	REQ-002-0001	The sensors shall detect an individual plant from the scanning area with at least 95% accuracy.	A
		REQ-002-0002	The system shall calculate pick-up coordinates in X, Y, Z dimensions with sufficient accuracy to pick up plant with a success rate of 90 %.	A
		REQ-002-0003	The vision system shall isolate a plant from other overlapping plants with an accuracy of at least 90%.	B
UC-007	**Classify plants**	REQ-002-0004	The system shall use AI technology to classify the plant types: Crispi, Multiblond and Ruccola	A
		REQ-002-0005	The system shall use AI technology to estimate plant health based on colour of leaves.	C
		REQ-002-0006	The system shall use AI technology to estimate if plant growth stage based on size.	C
		REQ-002-0007	If plant classification confidence falls below 80 %, the system shall attempt again 3 more times before requesting human verification.	C
		REQ-002-0008	The AI classification algorithms shall run with at least 95% success rate over 100 test samples.	B

Figure 21: Requirements for US-02

User Story ID: US-03
User story short title: Optimal handling

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-008	**Determine if plant should be harvested with or without roots**	REQ-003-0001	The system shall determine whether the plant should be harvested with or without roots, based on type.	C
UC-009	**Determine the best way to grip the plant.**	REQ-003-0002	The system shall decide the best method for the grabber to grip the plant for pickup.	A
UC-010	**Determine optimal and efficient handling.**	REQ-003-0003	The system shall support soft touch gripping to avoid damaging the plant.	A
		REQ-003-0004	The gripper needs to handle the salad in a effective and gentle way. Without sensors (soft touch).	A
		REQ-003-0005	The gripper needs to be able to grip, carry and release leafy greens, with weight up to 1 kg (gripper force).	C
		REQ-003-0006	The gripper needs to rotate around the end effector. <180 degrees (torque).	A

Figure 22: Requirements for US-03

User Story ID: US-04
User story short title: Safety and efficiency

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-011	**Detecting failures**	REQ-004-0001	Each joint shall have it's own defined movement boundary that is enforced by limit switches	A
		REQ-004-0002	Each joint shall have calibration functionality integrated based on limit switches.	A
		REQ-004-0003	The system shall use rotary encoders to report back on stepper motor position to help detect missed steps.	A
		REQ-004-0004	Current sensing (StallGuard) - The system shall use motor current sensors to detect overloads or out of bounds resistance.	B
UC-012	**Safety Factors**	REQ-004-0005	Motors shall work within safe limits.	A
		REQ-004-0006	Mechanical elements (shafts etc) should handle expected stressors.	A
		REQ-004-0007	Electronics (connectors, Mosfets, Tracewidth, powerplane, resistor ratings) shall operate safely within expected environmental conditions.	A
UC-013	**Notify users**	REQ-004-0008	The system shall log gripping errors.	A
		REQ-004-0009	The system shall log cutting errors	A
		REQ-004-0010	The system shall log success.	B

Figure 23: Requirements for US-04

User Story ID: US-05
User story short title: Modularity and expansion

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-014	**Modularity and expansion**	REQ-005-0001	The architecture shall allow new hardware and software functions, such as root cutting, to be integrated as independent modules, provided they follow defined communication interfaces.	A
		REQ-005-0002	The software architecture shall support modifiability by allowing components to be changed or extended with minimal impact on system-wide behaviour.	A

Figure 24: Requirements for US-05

User Story ID: US-06
User story short title: System scalability

Use Case ID	Use Case	Requirement ID	Requirement	Priority
UC-015	**System scalability**	REQ-006-0001	The system architecture shall be designed for scalability, such as for introducing multiple robots.	B

Figure 25: Requirements for US-06

6 Risk Management

We will identify, assess, prioritize current and potential risk, and implementation of mitigation strategies. The book used for reference is systems engineering Theory and practice [7, P. 88-100]
For a bachelors thesis it is not required to have a fully complete risk management, we need to show that we have a good understanding of what it is and how to deal with it and not overcomplicate it.
Risks can be poor decision making or oversight of the system. This can lead to a temporary stop of development or even backtracking to fix and mitigate risks in the system.

6.1 Risk identification and assessment

DAB | *VMN*

Identify - assess – implement mitigation strategy. Identification of current and potential risks that can disrupt the project and affect the achievement of the goal set. Identification must be an ongoing activity throughout the project to ensure continuous progress with minimal problems. All team members must contribute and participate in the discussion of identifying risks in their respective disciplines and roles.

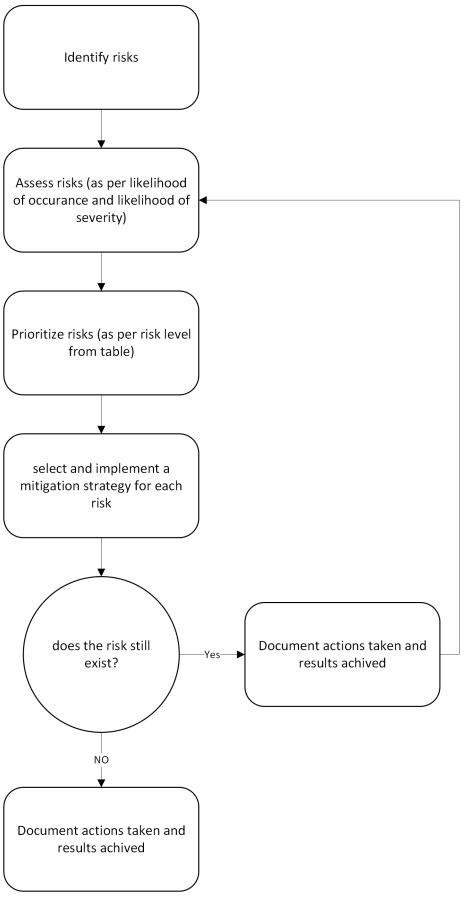


Figure 26: Risk management process

The figure has been modified to our needs from book systems engineering theory and practice [7, P. 92]

After that is done, we assess each risk and rank them in the figure 27. In the table, we group them into the categories Technical, Group, Financial, and Human error. With the category's set we give them a unique id to them like RT1–risk technical 1. This simplifies the process of looking through the risk table.
The identified and categorized risks we assign a number of what is the effect/consequence and likelihood of occurrence by looking at risk matrix table:2. The risk matrix is 5x5 to have good accuracy.
We have set up an excel sheet that automatically calculates risk factors based on the in-

putted values. if needed, we can write a description in the risk table if required.

6.2 Risk management strategies

DAB | *VMN*

Finding the best action to reduce or eliminate the likelihood or consequence of the risk. There are several ways to deal with risks, such as acceptance, contingency, reduction, transference, and prevention, discussed in the book.
Our risk table includes a mitigation strategy tab; there we write a short action as of how to deal with the risk and to avoid overcomplicating, and also at the same time show that it has been under consideration.

6.3 Risk analysis

DAB |

likelihood	1	**low**	**low**	**Low/Med**	**Low/Med**	**Low/Med**
	2	**low**	**Low/Med**	**Low/Med**	**Medium**	**Medium**
	3	**Low/Med**	**Low/Med**	**Medium**	**Medium**	**Medium/high**
	4	**Low/Med**	**Medium**	**Medium**	**Medium/high**	**High**
	5	**Low/Med**	**Medium**	**Medium/high**	**High**	**High**
		1	2	3	4	5
		Effect/ consequence				

Table 2: Risk matrix

Risk list	Description	ID	Effect/ consequence	Likelihood	Risk	mitigation strategy
Technical:						
Maintenace of robot arm	bad saefty mesures	RT1	4	3	**Medium**	stop button, maintnace program that shuts everything off for maintenance.
Power supply	loss off power to the system	RT2	3	2	**Low/Med**	buying higher quality psu
Signal integrity	signal issues	RT3	4	2	**Medium**	better shielding of the electronics
Corrosion		RT4	2	3	**Low/Med**	paint, or elektroplating
Contamination of food		RT5	5	3	**Medium/high**	rutinely check for damage of gripper At end of week
Motor mising steps	wrong position in space	RT6	4	4	**Medium/high**	in tandem with RT7. need to not overload, and soft correctly reads encoders. Tolorance stacking in parts. Need acceleration curve for stepper motors to avoid lost steps
Motor overload	motor cant move due to exiiding holding torque	RT7	5	5	**High**	when designing neeed to be under holding torque of the motor with some saefty margin.
Gripper squishing the salad		RT8	3	4	**Medium**	calibrating properly of the force requiared,
Communication error		RT9	3	3	**Medium**	ensuring fail safe, when message not recived after amount of tries safe shutdown
Camera placement	bad positioning=bad data in, lens geats dirty, or broken	RT10	3	3	**Medium**	
Printing bachelor thesis	having printing problems	RT11	5	3	**Medium/high**	planing in advance when to print,having backup method where to print
Groupe:						
Sick-1 person		RG1	3	1	**Low/Med**	home office, or a ersen overtakes work
Conflict	Delayd development,bad groupe dynamics	RG2	5	3	**Medium/high**	kontrakt, møte med karoline
Drops out of group		RG3	5	2	**Medium**	kontrakt
Sickness most of the groupe	2-3 peaple sick at the same time	RG4	4	3	**Medium**	digital teams meeting to maximize performane while at home.
Financial:						
Over budget	passer ikke på ugifter ut	RF1	5	4	**High**	holde kontinuerlig logging av utgifter
Price of parts	øking på pris av deler	RF2	5	4	**High**	finne billigs leverandår og printe innhouse deler
Human error:						
Breaking a 3d print part		RH5	1	4	**Low/Med**	3d print a new part
Breaking a electronics part		RH6	2	5	**Medium**	need to order new if time allows it, have backup or have a alternative replacpent
Breaking a motor	driving motors in the arm	RH3	5	5	**High**	just don't
Ordering wrong parts	pulley,belt, PCB and more	RH4	4	3	**Medium**	Try to adjust/modify the parts to fit if possible. If time alloows order new parts if needed
		RH5				
		RH6				

Figure 27: Risk Table

7 System Architecture

This chapter gives an overview of the system architecture that supports the Leafy Automation robotic harvesting system. It will begin by presenting the academic literature which has greatly inspired and influenced the Leafy Automation architectural design. The design drivers and constraints will then be presented, followed by the architectural objectives that have steered design choices in the right direction. From this foundation, we will present how the hardware and software components of Leafy Automation are structured and interact together to enable sensing, planning and execution of motor control. The systems layered architecture approach combined with the communication methods by Robotic Operating System 2 (ROS2) will be presented in detail, along with an overview of the software components themselves.

7.1 Literature review

EG |

Our aim is to base our System Architecture decisions on established research and best practices. For each source reviewed, we provide a summary of main points, gaps relevant to this thesis, and an evaluation of how this source can support our upcoming architecture.

Software Architecture in Practice, Clements, and Kazman (2021)[8]

This textbook by Bass, Clements, and Kazman presents foundational principles and tactics for designing software architectures. Early in the book, the authors discuss quality attributes associated with software architecture, why they are highly important, how to identify the most relevant attributes, and how to design for them. Most relevant to this thesis are the quality attributes of modifiability, scalability, and testability. The book presents several tactics that promote each attribute, providing an honest balance between benefits and trade-offs. A key tactic presented in this book is the layered software architecture pattern. The authors define layers as vertical groupings of related modules with strictly controlled interfaces, arguing that changes confined to one layer do not affect the others. Benefits include easier extension by inserting modules at the correct layer, focused testing scoped per layer, and clearer separation of concerns to ease understanding and enable isolated development.

While this book has proven to be a valuable resource, it offers relatively few domain-specific examples relating to embedded robotic applications. The authors focus primarily on large-scale businesses.

We evaluate that this work provides good framework from which to identify the most important quality attributes for the Leafy Automation architecture. It also provides a strong theoretical foundation for our layered ROS 2 architecture and justifies the strict layer boundaries and

interfaces between them.

Robot Operating System 2 (ROS2)Based Frameworks for Increasing Robot Autonomy: A Survey by Bonci et al. (2023).

Bonci and co-authors survey the state of the art in using ROS2 as a middleware to facilitate perception, planning, and control modules, particularly in fixed-base robots. The paper reviews existing ROS 2 features and tools, and proposes a high-level modular architecture for a pick-and-place proof-of-concept. The main points we draw from this source:

Middleware The authors propose ROS2 as the glue that brings together sensors, algorithms, and robot controllers. They contrast it with other frameworks (such as YARP, OROCOS, MOOS) and highlight ROS2s extensive ecosystem of libraries, including MoveIt2 for planning and the ROS-OpenCV bridge for vision.

Layered, task-based architecture They decompose autonomy into seven core tasks: Perception, Recognition, Behavior Planning, Trajectory Planning, Trajectory Re-planning, Motion Control, and Manipulation. These are grouped into three layers (Perception + Recognition, Planning, Control) to enforce clear interfaces and facilitate modularity.

Planning and control Implementation Their proof-of-concept uses MoveIt2 to generate and adapt pick-and-place trajectories, and two ROS 2 nodes (Cobot Driver and End-Effector Driver) to send commands to a commercial cobot and its gripper.

However, while ROS2s micro-ROS project is mentioned, the paper acknowledges that many embedded platforms still lack robust ROS2 clients, leaving a gap for lightweight protocols or custom bridges.

This source offers a practical, robotics-centered blueprint for structuring a ROS2based autonomy stack, proving valuable for teams aiming to assemble perception-to-control pipelines quickly. Their task-layer mapping directly tells us how to group our own nodes and define clear topic or service interfaces between them. The overview of MoveIt2 integration and case-study citations also serve as guides for our proof-of-concept work. However, to address our embedded-hardware constraints call for further research, particularly on protocol bridging.

7.2 Project Constraints and Architectural Drivers

EG | *BMR*

Developing a system architecture should begin with a prioritised list of quality attribute requirements. These form the basis for design decisions and guide trade-offs throughout the development process. [8, p. 7–20] This section presents an expanded list of priorities, covering

both key project constraints and the most relevant architectural quality drivers. It reflects the strict limitations on time and resources, as well as the central architectural quality goals of modifiability and scalability.

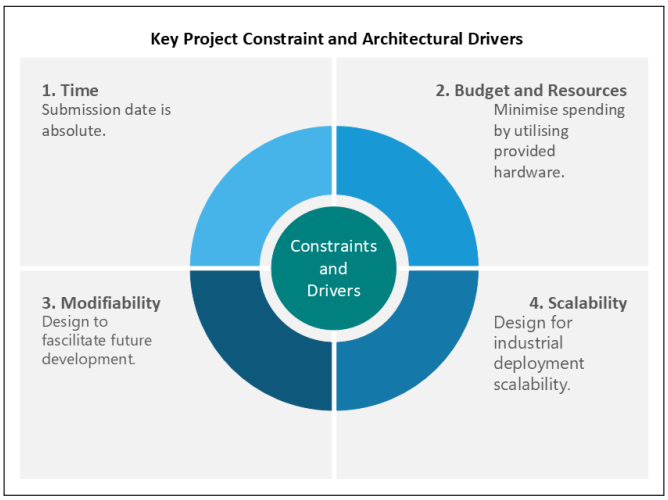


Figure 28: Key Project Constraints and Architectural Drivers

Time constraints

The bachelors thesis in engineering at the University of South-Eastern Norway (the University) spans one semester and includes a number of mandatory events and deliverables. The final submission deadline is fixed and non-negotiable, meaning the project must be completed within this timeframe.

Budget and Resources Requirement REQ-001-0003 (see section 5) states that development must keep expenditure to a minimum by primarily using resources provided by Hydroplant Technologies and the University. This constraint has led to choices such as utilising supplied hardware, opting for open-source software (such as ROS2, which is further explained in section 7), and avoiding third-party or proprietary solutions where possible.

Modifiability

The Leafy Automation harvesting system is an early-stage, proof-of-concept research project that extends beyond the scope and timeline of this bachelors thesis.Therefore, a key objective is to develop a software architecture that is both modifiable [8, p. 117–130] and flexible, enabling further development and adaptation after the initial prototype phase.

To support this goal, modifiability was identified as an essential quality attribute from the start. A modular architecture provides clear separation of concerns and well-defined interfaces between components, allowing team members to design, implement, and test their contributions independently.[9] This is also an advantage for a team working in parallel and under a tight deadline.

Modularity also plays a critical role in the long-term modifiability and scalability of the system. Well-encapsulated modules are easier to understand, replace, or upgrade without requiring significant changes to the rest of the system.[9] This is especially relevant in a research and development setting where future changes, improvements or extensions, such as new sensors, gripper designs, or additional subsystems like plant pot or root removal, may be added to the system. With a modular foundation, such additions can be integrated with minimal disruption to the system at large.

Scalability

Scalability is an important goal for the Leafy Automation harvesting system. Although this project delivers a proof-of-concept, the system is intended to grow over time, both in complexity and scale.

A scalable design allows the same architecture to support higher output rates, additional functionality such as root cutting or plant health checks, or the ability to operate multiple robotic arms along the same production line. To make this possible, the software needs to be designed around reusable components with clearly defined interfaces, so that software components can be extended or duplicated without major structural changes or significant rewrites.

In line with REQ-006-001 (see section 5), the system is built from modular and abstracted components that can be tested, updated, or replaced independently. This fits well with agile development principles and recommendations from modular software architecture research, highlighting that well-separated, low-dependency components are essential for scalable and maintainable systems.[9]

7.3 System Objectives

EG | *BMR*

The Leafy Automation system is a project that aims to fully automate the harvesting of leafy green vegetables in an industrial, agricultural environment. At this early proof-of-concept stage, the robot is required to carry out a defined sequence of basic tasks: detecting and classifying a plant, picking it up, transporting it, and finally placing it in a designated placement zone. This functional sequence forms the backbone of the harvesting process and is illustrated in Figure 29. The sequence is executed by a robotic arm, which is the main focus of this bachelors thesis. The arm is positioned in the center of the working area, as shown in Figure 30, with further detail provided in the zone layout shown in Figure 31. For a more thorough explanation of the

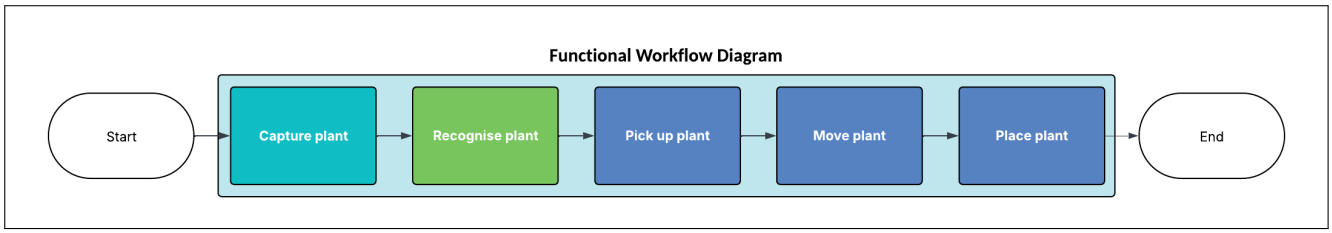


Figure 29: Functional Workflow Diagram

working area layout, see Section 8.4 *Definition of Working Area*.

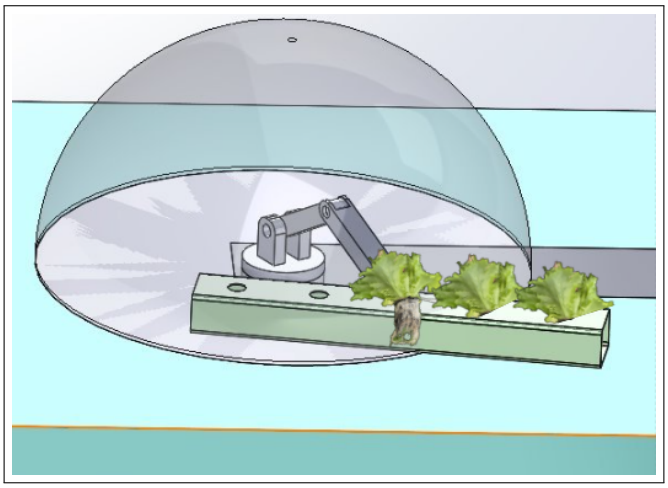


Figure 30: Positioning of the robotic arm.

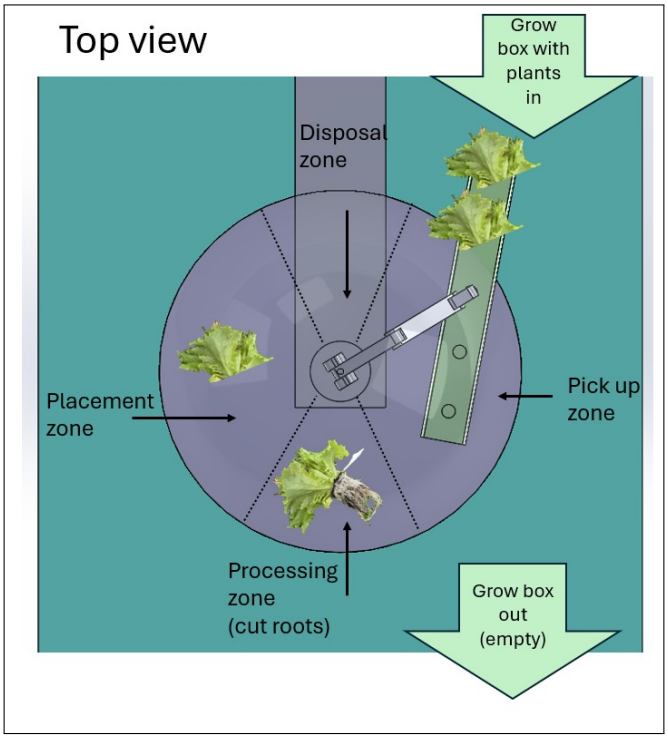


Figure 31: Working area zone partitioning

The workflow shown in Figure 29 has been derived from the project requirements defined in

section (see section 5). These requirements specify the functions that the robot must perform. Based on this workflow, a set of high-level system objectives has been identified. These objectives describe the main capabilities that the system must perform and serve as a basis for guiding key design decisions. Table 3 provides an overview of the architecture objectives and their corresponding requirement references.

ID	System Objective	Corresponding Requirement
SO-01	Capture and analyse plants	REQ-002-001 REQ-002-002 REQ-002-003 REQ-002-004 REQ-002-005 REQ-002-006 REQ-002-007 REQ-002-008 REQ-001-008 REQ-001-009
SO-02	Motion planning	REQ-001-0008 REQ-001-0009
SO-03	Task planning	REQ-003-0001 REQ-003-0002
SO-04	Motor control	REQ-001-0009
SO-05	Human interface	REQ-001-0004

Table 3: High-level system objectives.

System Objective SO-01: Capture and Analyse Plants The workflow begins when the camera captures the image of one or more plants in the Pickup-zone. The image must then be processed using AI-based image analysis to identify the type of plant and to estimate the best gripping coordinates for the pickup operation.

System Objective SO-02: Motion planning Motion planning is an essential component responsible for generating safe movement paths for the robotic arm based on input from the AI-based recognition system. It should calculate trajectories that guide the end-effector, for example, from its current position to the plant pickup coordinates, coordinating all five joints to ensure smooth and precise motion.

System Objective SO-03: Task planning The system should support a coordinated sequence of harvesting operations, including plant detection, pickup, placement, and state handling. This functionality must be organized and structured to allow for adaptations or new tasks to be added over time.

System Objective SO-04: Motor control The system shall translate planned motion paths into joint and gripper movements, enabling the robotic arm to carry out its tasks correctly and in the correct sequence.

System Objective SO-05: Human Machine Interface The Human-Machine Interface (HMI) should allow users to start and stop operations, monitor system status, and view error messages. This is valuable in testing, debugging, demonstrations, and production.

7.4 System Overview

EG | *BMR*

A layered architectural approach has been chosen, which separates the system into functional blocks: perception, recognition, planning, high-level, and low-level motor control. The software modules of each layer communicate through well-defined interfaces, allowing components to be developed and tested independently. Further details of the software layers are provided in later subsections.

Below is a summary of the main system components.

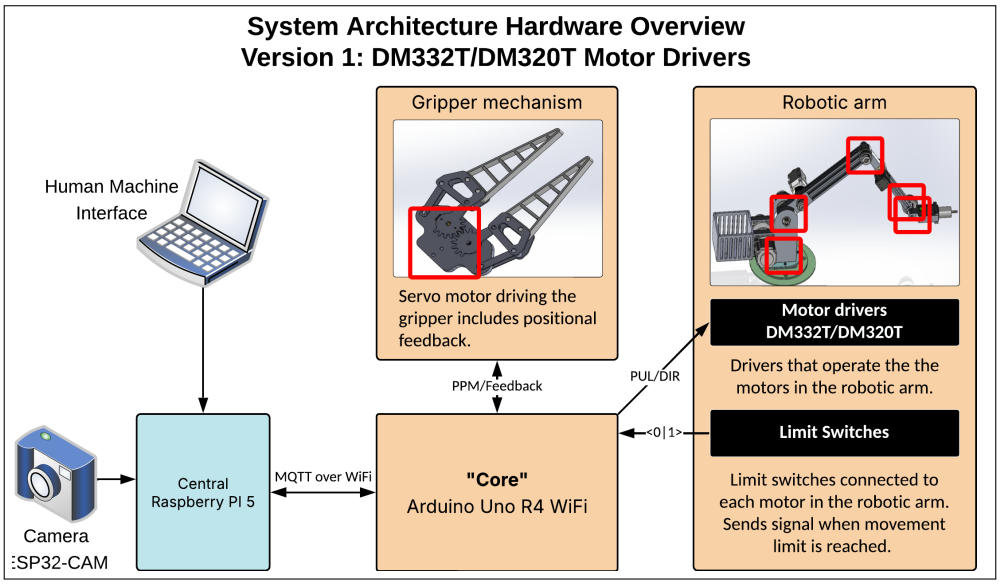


Figure 32: System Architecture Hardware Overview

7.4.1 RaspberryPi 5 (Central) for high level computations

EG | *BMR*

The development team was provided with one Raspberry Pi 5 and two Arduino Uno R4 WiFi boards for use in the Leafy Automation project. The Raspberry Pi 5, referred to as Central throughout the project, was selected to serve as the central processing unit. It is responsible for receiving camera input, performing AI-related computations, executing motion and task planning, and handling high-level motor control.

These tasks are computationally intensive and require a platform capable of multitasking and multithreading. The Raspberry Pi 5 is equipped with a quad-core Arm Cortex-A76 CPU cluster [10], making it significantly more powerful than the Arduinos. It also offers better memory, native support for multitasking and the ability to run ROS2 [11]. ROS2 and its place in the Leafy Automation system is explained in *Section: Robot Operating System 2* 7.

In addition to general-purpose processing, the Raspberry Pi 5 includes a dedicated Image Signal Processor (ISP) and a hardware video scaler, which provide strong support for camera input and image processing. This further supports selecting the Raspberry Pi 5 (Central) for perception tasks and AI-based recognition models.

7.4.2 Arduino Uno R4 Wifi for low level motor control

EG | *BMR*

The Arduino Uno R4 WiFi, referred to as Core is responsible for low-level motor control and direct communication with the stepper motor drivers powering the robotic arm, and the servo motor driving the gripper. It was selected due to its real-time capabilities [12] and suitability for handling precise motor signals, as well as being one of the boards provided to the group by Hydroplant Technologies.

During early development, we considered using Micro-ROS, however the Arduino Uno R4 lacks stable support for this integration. Although community-driven support was released late 2024, implementing into the system architecture was considered too high a risk given the short development timeframe. Instead, communication between Central and Core is implemented using MQTT, with the broker hosted on the Raspberry Pi. This setup enables communication between the high-level planning software and low-level motor control, while maintaining modularity and allowing development to continue within the project's hardware and time constraints.

Although less integrated than a full micro-ROS solution, the Arduino is a sound choice for low-level motor control and supports the systems layered architecture.

7.4.3 Motor drivers

EG | *BMR*

As is explored in *Section Stepper Motor Drivers* 10, motor control is initially handled using DM332T and DM320T stepper drivers. These drivers came with the motor kit and are controlled using simple pulse and direction signals. The only motor feedback the Arduino receives is from limit switches, which detect end stop conditions to avoid over-traveling. A transition to a more sophisticated TMC5160 driver is underway, providing additional features such as Rotary Encoder feedback for motor position status, and UART-based communication. This upgrade will take place once the custom PCB and supporting hardware are completed internally, and delivered from the external fabricator. For full details about the stepper motor driver development and setup, see section **??**.

7.4.4 Camera and AI

JCDH | *SME*

The camera gives us the vision of our system. This provides us with knowledge of where the robotic arm is placed in relation to the plant. A picture may be worth a thousand words, but we need more concrete data points. This is where AI models come in, as they give us a way to synthesize the picture into concrete data points which will be useful to the robotics. More information about the camera can be found in Section 11.6 and information about how AI is used in this project, is provided in Section 11.7.

7.4.5 Communication

EG | *BMR*

The Leafy Automation system combines a layered architecture with publish/subscribe messaging to achieve modular, distributed control. Within the ROS2 domain, discussed more extensively in *Section Robot Operating System 2* 7, the layered software components, often referred to as nodes, exchange data via Topics. This communication approach maintains loose coupling between components while supporting scalability and fault isolation. [8, p. 117–130]

Between Central and the Core, we use MQTT, which is a lightweight publish/subscribe protocol

with the broker running on the Raspberry Pi. In early development we used HTTPS for communication testing, but since switched to MQTT to reduce coupling and maintain faster, more direct communication. More on this can be read in *Section MQTT set-up between Central and Core* 7.

7.5 Layered Software Architecture

EG | *BMR*

The software for the Leafy Automation system is structured based on the layered architectural pattern, which is described in Software Architecture in Practice (Bass, Clements, Kazman, 2012)[8, p. 128–129] as an advised tactic for achieving modifiability:

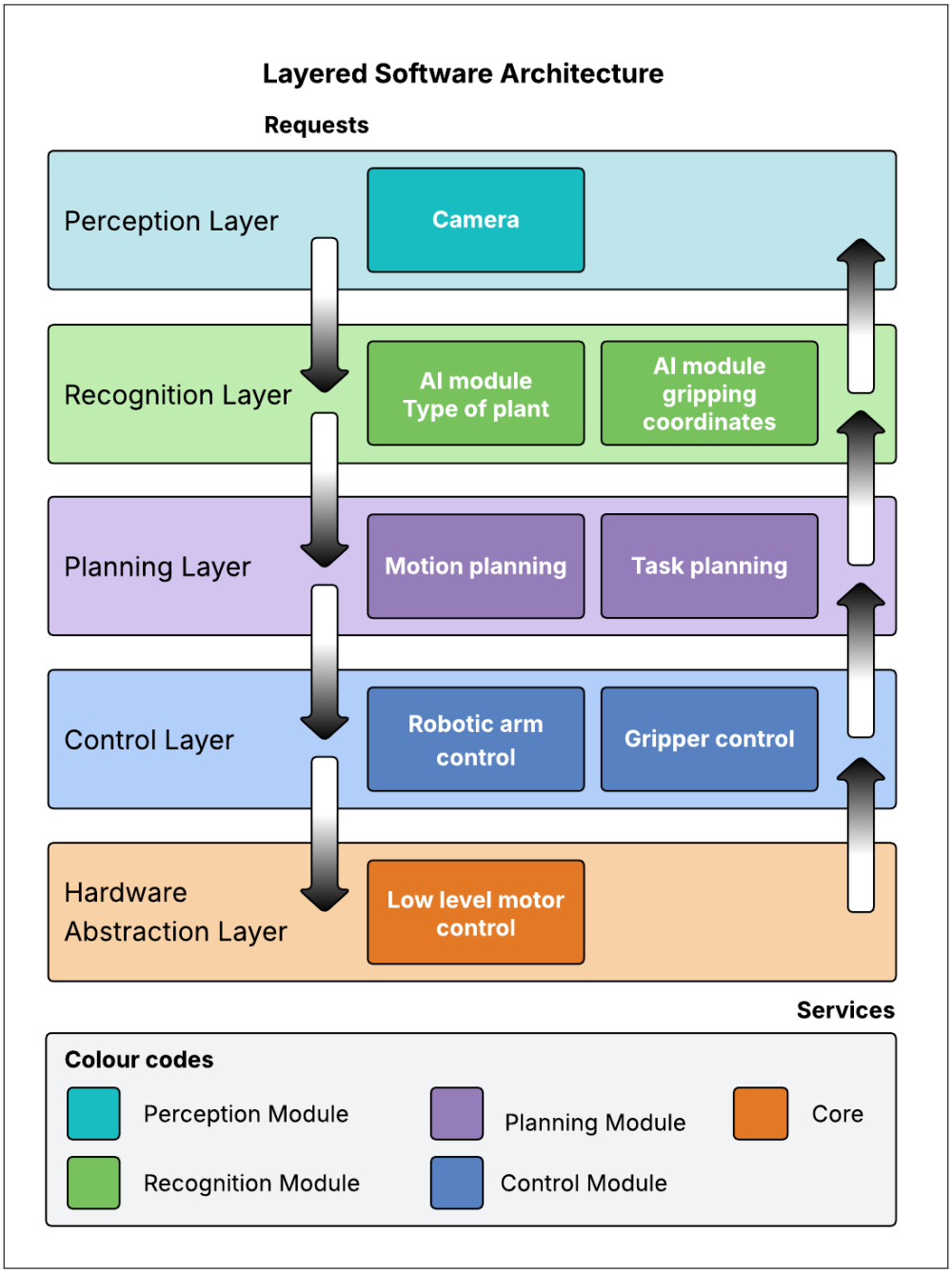


Figure 33: Layered software architecture for Leafy Automation

In a layered architecture, the software is divided into a hierarchy of vertical layers. Each layer brings together modules that share common concerns. [13] An example of this might be two control modules, one focused on operating the robotic arm and one for the gripper. As seen in Figure 33 these two modules are grouped together in the Control Layer.

Communication between the layers is strictly controlled. An architectural layer may only use the services provided by the layer immediately below it and may equally so only provide services to the layer directly above it through a well-defined public interface. This grouping and separation of concerns has several key benefits to the robotic application:

Modifiability is improved as logic within one layer can be modified without affecting layers above, as long as the interfaces remain the same. This separation also supports multiple developers to work on different layers concurrently, without facing integration issues.

Extendability A layered architecture makes it easy to extend functionality by introducing new modules at the appropriate layer, without affecting the existing layers.

Easier to understand Well-defined interfaces between the layers simplify understanding and debugging. A developer can focus on the single layer without having to understand the whole system.

Easier to test Limiting complexity supports testability [8, p. 190–191]. A layered approach reduces complexity by encapsulating concerns. This makes it much easier to define test cases as they can be scoped down to a single layer, thus reducing the number of test scenarios.

7.5.1 Layered module-based design

EG | *BMR*

The software components of the Leafy Automation system are organised into five layers, each with a clearly defined functional responsibility: Perception, Recognition, Planning, Control, and Hardware Abstraction. Each layer relies on the services provided by the layer directly below it and exposes a well-defined interface to the layer above. The hierarchy maintains strict boundaries where for example high-level decision logic never interferes with low-level motor control, and hardware-specific code remains hidden from the vision and planning layers.

The Leafy Automation architecture is greatly inspired by and adapted from the work done by Bonci, Gaudeni, Giannini and Longhi for their article Robot Operating System 2 (ROS2)-Based Frameworks for Increasing Robot Autonomy: A Survey [14].

As seen in Figure 33, the Leafy Automation software is organised into five. Each layer is responsible for a distinct aspect of the pick-and-place pipeline:

Perception Layer The purpose of this layer is to receive and, when applicable, preprocess

sensory data. For the scope of this thesis, this layer only contains a single camera node that captures images of the plants in the pickup zone and sends them downstream for processing.

Recognition Layer The purpose of this layer is to process the information received from the Perception layer and turn it into actionable data. This layer contains two AI-modules. One module is to estimate plant type, and the other module estimates the most optimal object gripping coordinates for the pickup sequence. This information is sent down to the Planning Layer. *The plant-type estimation node is not used further in this project due to time constraints. It was meant to adjust handling parameters according to plant type to optimise plant handling.*

Planning Layer The Planning layer is responsible for the pick-and-place workflow by turning pose estimates into a sequenced series of actions. It includes two tightly coupled software modules: the Task Planner, which decides the next operation (e.g. move to pre-grip, close gripper, transport, open gripper, return home), manages calibration and error-recovery logic, and publishes each step as a task goal. The Motion Planner, which is one recipient of these task goals, calculates joint trajectories within the robot's physical constraints.

Control Layer The Control layer sits between Planning and the Hardware Abstraction layer. Its purpose is to translate abstract trajectories and grip actions into motor instructions. It formats and dispatches MOVE and GRIP commands to the firmware, monitors acknowledgments (e.g., message receipts and motioncomplete signals). As the Control Layer is responsible for sending firmware instructions and enforcing timing and safety checks, the Planning Layer can remain focused on strategy. This layer contains two software modules; one for arm control and one for gripper control.

Hardware Abstraction Layer The Hardware Abstraction Layer handles low-level motor control. Unlike the other layers that all reside on the ROS2 -based Raspberry Pi Central, this layer is located in its entirety on the Arduino. This layer is responsible for translating control messages received from the Control Layer into exact pulse and direction signals for the stepper motors and PWM signal for the servo motor in the gripper. It monitors sensor signals from the connected Limit switches and maintains communication with the Control Layer by sending a heartbeat and status updates. By isolating these tasks, the upper layers in the structure can remain hardware agnostic. This facilitates future upgrades for example to a different motor driver with minimal changes, which is an important aim for this project.

7.6 Communication model

EG | *BMR*

Effective messaging is vital for any autonomous robot system. In Leafy Automation, we use the ROS2 publish/subscribe system for internal communication and a MQTT bridge to link the ROS2 Control Layer to the Arduino firmware, handling low-level motor control. Figure

On the Raspberry Pi, ROS2 nodes share AI results, motion plans and task instructions over well-defined topics. This decouples data producers from consumers and lets each layer evolve independently. When it comes to real-time motor control, the Control nodes send MOVE and GRIP commands over MQTT to the Arduino client. The MQTT broker is hosted in a Docker container on the Pi to keep the setup portable and isolated.

This section first outlines the current ROS2 topic -based implementation. It then details the MQTT bridge configuration between the Control layer and Arduino firmware. Finally, it offers recommendations for future development such as services and actions implementation and Quality of Service (QoS) tuning.

7.6.1 Robot Operating System 2

EG | *BMR*

Robot Operating System 2 (ROS2) Jazzy Jalisco (the most recent long term supported version, at time of writing) is an open-source software platform specifically designed for developing robotic applications. [15] It is distributed under the Apache 2.0 License [16], which grants users a significant freedom to modify, apply and redistribute the software, without obligations to contribute back. [17]. ROS2 provides an extensive suite of tools and libraries, encompassing drivers, commonly used algorithms (such as perception, simultaneous localisation, mapping) and various development utilities. [17]. There is also a strong development community supporting ROS2, with a plethora of open-source packages available on the ROS Index and active discussion forums with active contributors from both industry and academia.

At its core, ROS 2 is built around software nodes which are independent, encapsulated processes. And with the ROS2 nodes being language-agnostic, they let developers choose the most suitable programming language for each task at hand.

ROS2s communication is based on DDS, Data Distribution Service protocol from the Object Management Group. DDS provides peer-to-peer publish/subscribe middleware with configurable Quality of Service policies, which is valuable for robotic applications that are often resource constrained. [15] The three communication methods are:

Topcis provide an asynchronous, many to many messaging channel using standard or custom made message types (for example, `sensor_msgs/Image` [18]). Any node can publish data, and any number of nodes can subscribe. In Leafy Automation, the `camera_node` in the Perception layer publishes raw images on the topic `/leafy_automation/images`. Downstream perception nodes subscribe to this topic, perform image analysis, and then publish their outputs to topics consumed by the Planning layer. ROS2's many message types can be explored extensively at the ROS Index [19]. [20]

Services offer a synchronous request/response pattern where a client node sends a request and blocks until it receives a reply. This works well for oneoff operations, such as motor calibration.

In Leafy Automation, the Task Planner calls the `/leafy_automation/calibrate_motors` service using `std_srvs/SetBool`[21] to calibrate all joints before a new pickandplace cycle. By using a ROS 2 service, no other actions proceed until the calibration has completed. [22][20]

Actions Actions extend services to support longrunning tasks with progress feedback and cancellation. For example, the `motion_planner_node` sends a trajectory goal via the `control_msgs/FollowJo` action to the `arm_control_node`. The action server returns periodic updates (e.g., percentage complete) and allows the goal to be canceled in case of an emergency stop. This pattern is ideal for pick-and-place operations that require real-time monitoring and safe abort capability. [24][20]

7.6.2 ROS2 communication proposal for proof-of-concept development

EG | *BMR*

Figure 34 shows the proof-of-concept proposal designed for this thesis. This overview is further elaborated in Table 35, where more detail is added. This design is based exclusively on the use of topics. This more simplistic design gives us fast access to testing and prototyping. A more robust communication model, better suited for an operational environment, that also is presented in the Future Works chapter 15.1.

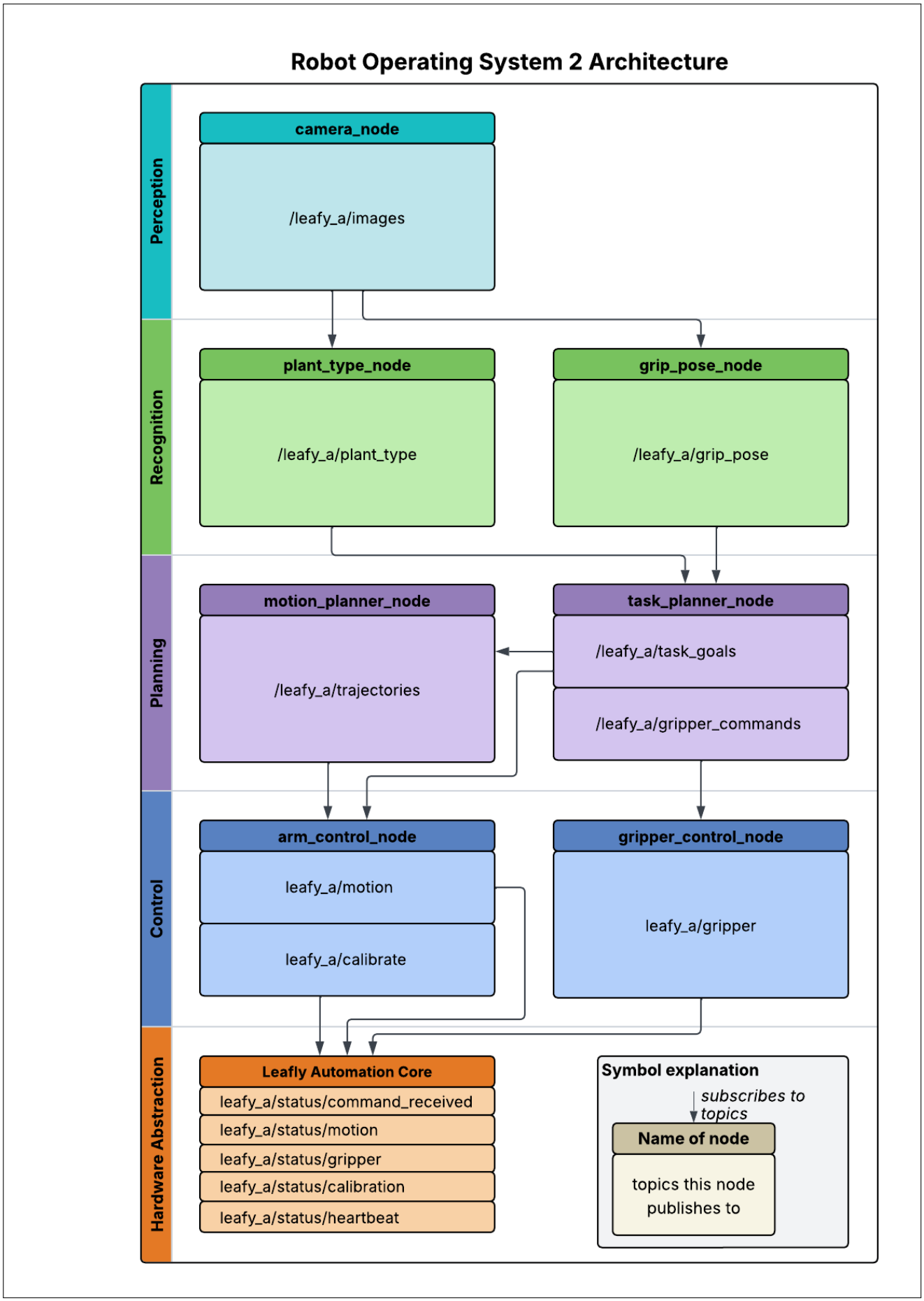


Figure 34: Communication between nodes via ROS2 topics.

Node	Subscribes	Publishes	Message Type
camera_node		/leafy_a/images	sensor_msgs/Image
plant_type_node	/leafy_a/images	/leafy_a/plant_type	custom_msgs/PlantID[]
grip_pose_node	/leafy_a/images	/leafy_a/grip_pose	geometry_msgs/PoseStamped[]
task_planner_node	/leafy_a/plant_type /leafy_a/grip_pose	/leafy_a/task_goals /leafy_a/gripper_commands	custom_msgs/TaskGoal custom_msgs/GripperCommand
motion_planner_node	/leafy_a/task_goals	/leafy_a/trajectories	trajectory_msgs/JointTrajectory
arm_control_node	/leafy_a/task_goals /leafy_a/trajectories	leafy_a/motion leafy_a/calibrate	std_msgs/String (MOVE J0 J1 J2 J3 J4) std_msgs/Bool (calibrate)
gripper_control_node	/leafy_a/gripper_commands	leafy_a/gripper	std_msgs/String (GRIP 1 / GRIP 0)
Arduino via MQTT	leafy_a/motion leafy_a/gripper leafy_a/calibrate	leafy_a/status/command_received leafy_a/status/motion leafy_a/status/gripper leafy_a/status/calibration leafy_a/status/heartbeat	

Figure 35: The Leafy Automation software nodes communicating via defined topics.

Remember to elaborate on the custom messages!

7.6.3 MQTT set-up between Central and Core

EG | *BMR*

The Arduino Uno R4 WiFi lacks micro-ROS support and therefore cannot run as a native ROS2 node. Given our architectural constraints and drivers (see *Section: Project constraints and architectural drivers* 7), this architecture avoids adding extra hardware and instead bridges the Raspberry Pi and Arduino via MQTT over Wi-Fi, as seen in Figure 36.

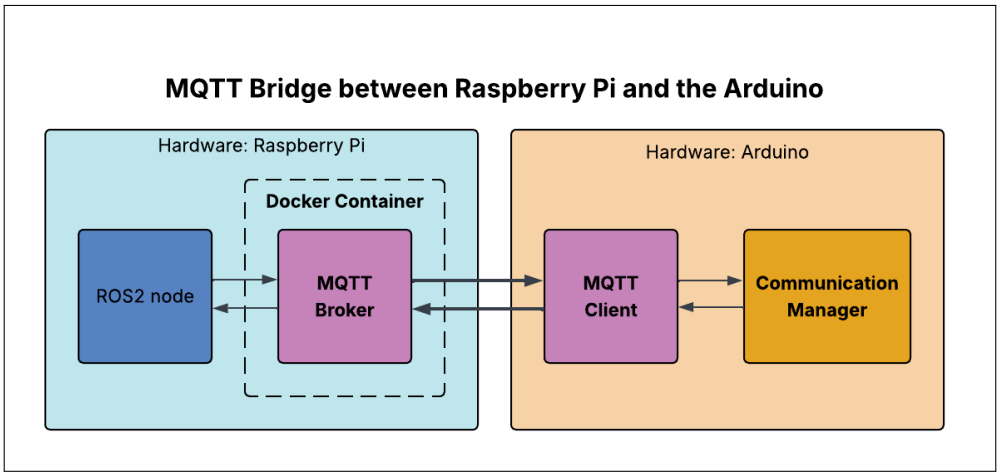


Figure 36: Diagram illustrating how the MQTT bridge fascilitates communication between ROS2 control nodes and the Core Communication Manager

The MQTT broker runs in a Docker container on the Raspberry Pi, which encapsulates all of its dependencies, configuration, and runtime environment. This isolation ensures that the broker can be updated, replaced, or rolled back without affecting the rest of the ROS2 system, simplifies deployment on fresh Raspberry Pi images, and keeps the overall easier to set up and reproduce.

The specific messages being sent over MQTT are listed along the "Arduino via MQTT" row in Table 35.

7.7 Critical technologies

JCDH | *SME*

Based on what has been defined in requirements and the risk analysis, critical technologies defines the list of technologies which are required for optimal system functionality. This subsection outlines the critical technologies, along with brief descriptions of each technology. You may read more about the details of these systems in following sections. When it comes to defining these technologies it is helpful to visualize the requirements and risks to our system, and then to build from there.

Table 4 shows a high-level overview of our systems critical technologies.

7.8 Earlier work

JCDH | *SME*

Please refer to Appendix 4 & G.16 for initial work on architecture.

8 Physical concept

This section contains information about some of the initial work done to establish a set of frames/boundaries around the physical concept of this bachelor project. This includes choice and definition of robot type, outline of working area and reach of the robot.

8.1 Comparison of robot types

SME | *JCDH*

To determine the most suitable pick-and-place robot type for our system, we did a comprehensive analysis of five different robotic arm types. Each type was evaluated based on key performance criteria that align with our specific project requirements. By comparing these options, we aimed to identify the most efficient and effective robotic arm for our application. See appendix J for table of robot types evaluated.

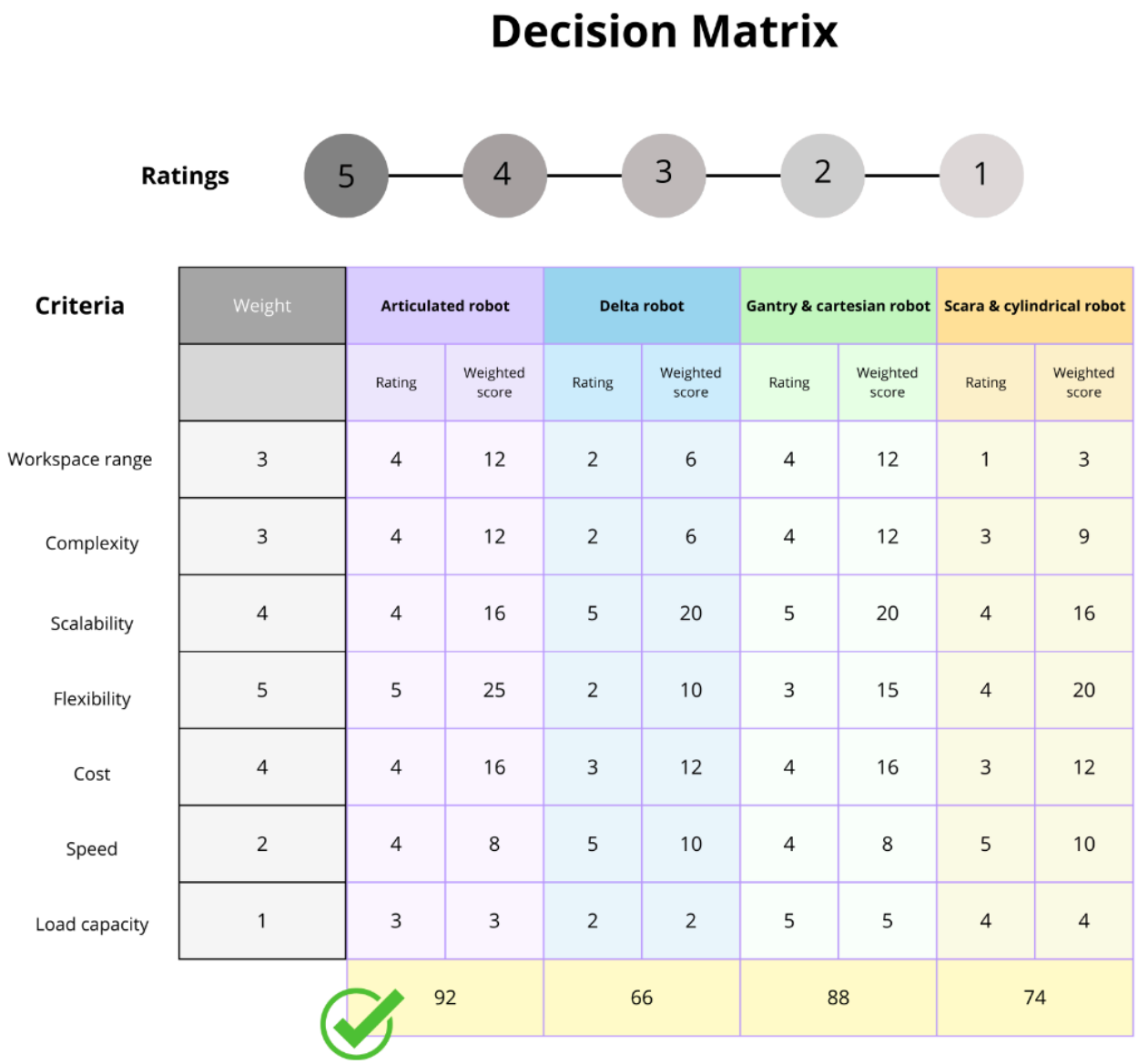
To facilitate this evaluation, we used a Decision Matrix, which is a structured analytical tool designed to compare multiple alternatives based on predefined criteria. This matrix provides

a clear framework for assessing the advantages and disadvantages of each robot type while prioritizing the factors most critical to our projects success.

The Decision Matrix method enables a systematic ranking of the selected robotic concepts by assigning weighted scores to essential evaluation criteria. By calculating the total score for each robot type, we can directly compare their suitability and select the one that best meets our performance requirements. The evaluation criteria were weighted on a scale from 1 to 5, where 1 represents the lowest level of importance and 5 the highest. Since some factors are more critical than others (particularly those that affect flexibility, scalability, and cost), this weighting system helps reflect their relative significance. Similarly, each type of robotic arm was assigned a general performance score within the same 1 to 5 scale to facilitate a direct comparison.

The Decision Matrix is structured into columns that represent the evaluation criteria, weight, the different robot types, and the total score assigned to each type. This structured approach allows an overview of the selection process to identify the most suitable robot type for our project. For a detailed breakdown of the evaluation and results, refer to the Decision Matrix table.

8.1.1 Decision Matrix



Criteria	Weight	Articulated robot		Delta robot		Gantry & cartesian robot		Scara & cylindrical robot	
		Rating	Weighted score	Rating	Weighted score	Rating	Weighted score	Rating	Weighted score
Workspace range	3	4	12	2	6	4	12	1	3
Complexity	3	4	12	2	6	4	12	3	9
Scalability	4	4	16	5	20	5	20	4	16
Flexibility	5	5	25	2	10	3	15	4	20
Cost	4	4	16	3	12	4	16	3	12
Speed	2	4	8	5	10	4	8	5	10
Load capacity	1	3	3	2	2	5	5	4	4
		92		66		88		74	

Figure 37: Decision matrix table

8.2 Choice of robot type

SME | *BMR*

In modern automation, various types of robots can be employed for pick-and-place operations, each with distinct advantages depending on the task requirements. The choice of robot type is influenced by several factors, including the nature of the load, environmental constraints, precision and the overall system flexibility. For this project, we have chosen to design and implement an articulated robotic arm, which closely resembles the movement and functionality of a human arm.

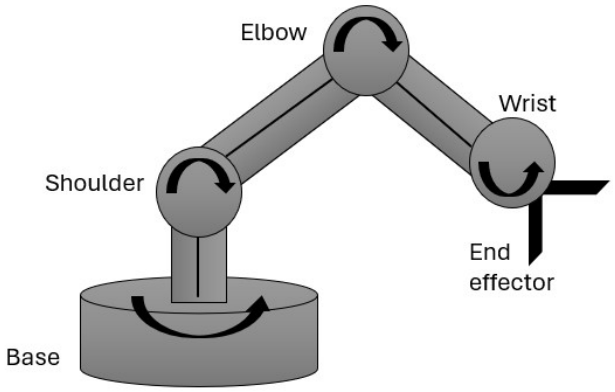


Figure 38: Articulated robot arm with rotational base

Agriculture remains as one of the industries where manual labor is dominantly used, particularly in tasks as harvesting plants. Despite technological advancements, the automation of plant harvesting is still in its early stages. Our objective is to develop a robotic solution that can efficiently handle plants with precision while offering adaptability for different scenarios.

An articulated robotic arm is characterized by multiple joints and a rotational base - see figure 38, which provides the necessary flexibility to achieve these goals. Several key factors influenced our decision to select this type of robot:

- **Flexibility in the orientation of the end effector/gripper.** A critical requirement for our project is the ability to manipulate plants from various angles and positions. Unlike other robotic configurations, such as Cartesian or SCARA robots, an articulated robot arm allows for greater range of motion, enabling precise handling of plants regardless of their orientation. Additionally, as the position and the orientation of the plants may vary within the working environment, having a highly maneuverable robotic system ensures consistent performance.
- **Compact volume and reduced footprint for increased adaptability**. The relatively small volume of an articulated robotic arm provides greater flexibility regarding its replacement within the working environment. Agricultural settings, particularly indoor farming or greenhouse operations, often contain spatial constraints or physical obstacles

that limit the placement of automation systems. A compact design ensures that the robot can be integrated seamlessly into various working environments without significant modifications to the infrastructure.

- **Interchangeable end effectors for diverse applications**. While the primary focus of this project is on harvesting leafy green salads, the system should be adaptable to handle different types of plants. By designing an interchangeable gripper mechanism, the robotic arm can be easily modified to perform various agricultural tasks, such as re-potting tomato plants, sowing seeds, or handling different crop species. This modularity enhances the robot‘s versatility and extends its applicability across the multiple agricultural processes.

- **Ability to grip and lift plants from multiple angles.** In practical agricultural applications, plants may not always grow in a perfectly upright position. Leafy green salads, in particular, can become tilted or positioned irregularly due to natural growth patterns or environmental factors. Therefore, our robotic gripper must have the capability to approach and lift plants from different angles, mimicking the complexity of a human hand. This ensures efficient handling of plants in real-world scenarios where variations in plant positioning are common.

- **Integration with a mobile platform or rack system to expand the working range**. To further enchance the flexibility of our robotic system, we aim to design it so it can be mounted on a mobile unit or a rack. This allows for an extended operational range, making the robot suitable for larger-scale agricultural operations. By incorporating mobility into the system, the robot can be deployed dynamically within different areas of a farm og greenhouse, increasing its overall efficiency and productivity.

By selecting an articulated robotic arm for this project, we ensure that the system is not only capable of precise plant handling but also adaptable to varying environmental conditions and future agricultural automation applications. The combination of flexibility, compact design, modularity, and mobility makes this robotic configuration an ideal choice for advancing automation in agriculture.

8.3 Robot arm diagram

SME | *JCDH*

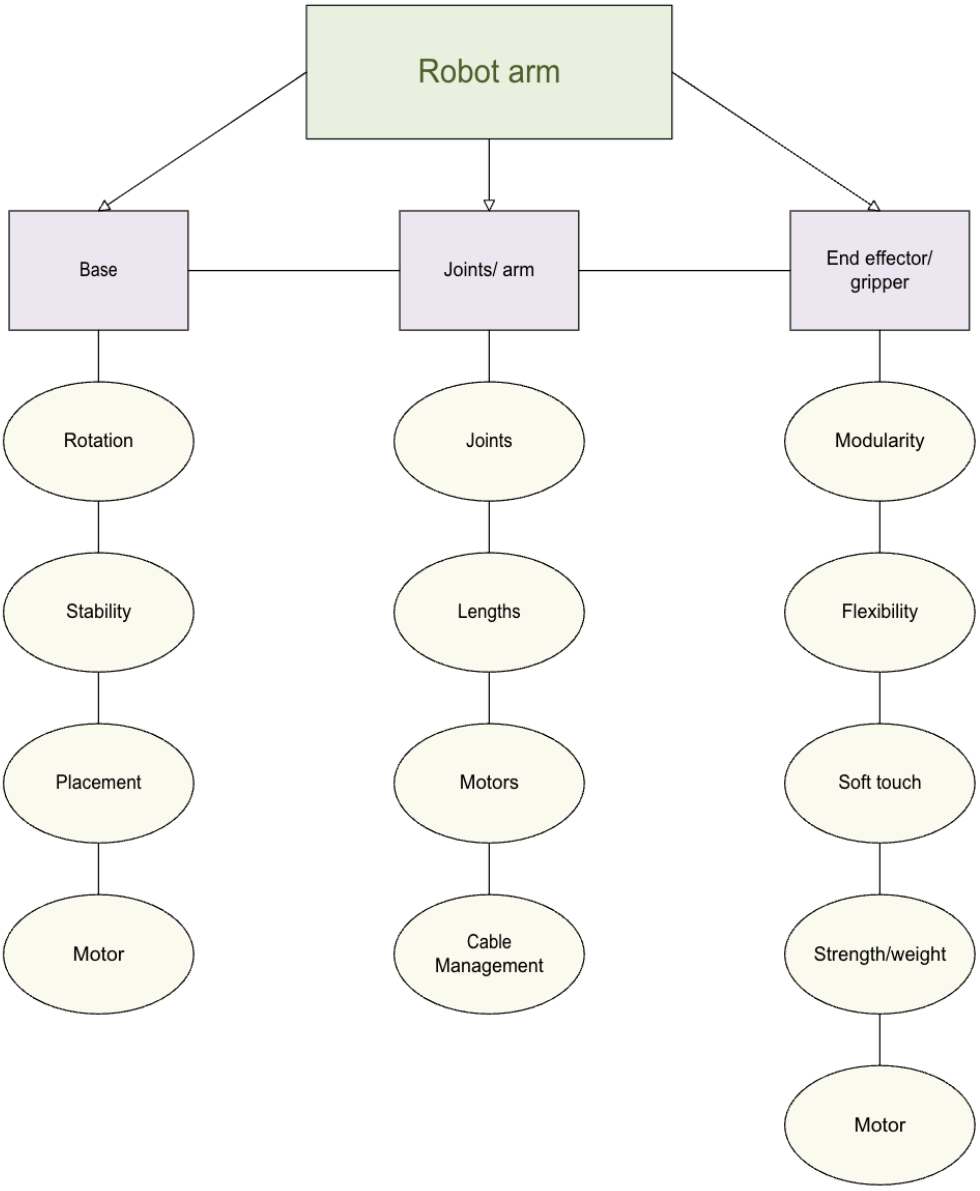


Figure 39: Robot Arm Diagram

This is an overview of the implementations for our robot arm system. The robotic arm developed for this project is structurally divided into three primary subsystems: the base, the joints and links (arms), and the end effector (gripper). Each of these plays a distinct role in ensuring that the robot can operate efficiently and precisely within the defined work area.

Base

The base forms the foundation of the robotic arm and is responsible for anchoring the system to the ground or mounting surface. As illustrated in Figure 42, the base is positioned centrally to provide a stable platform for the arms operation.

A critical function of the base is to allow for rotation, enabling the robotic arm to access the full work area as discussed in Section 8.4. This rotational capability is achieved by incorporating a motor within the base that facilitates smooth and controlled movement around the vertical axis.

In addition to rotational mobility, stability is of great importance. A stable base ensures the accuracy and repeatability of the robotic movements, which is especially crucial during precise tasks such as harvesting delicate plants. The physical placement of the base is determined by the layout and constraints of the target work area, and must be carefully planned during installation.

Joints and Links

The robotic arm consists of multiple joints, connected by links, which collectively determine the arm's overall reach and degrees of freedom. For a detailed overview of the joints and linkages, refer to Figure 42.

Each joint is equipped with a dedicated motor, which provides actuation for rotation or angular displacement. These motors are responsible for driving the movement of the links, allowing the arm to perform complex and positional adjustments during operation.

Between the joints, rigid links are used to maintain the structural integrity of the arm and to define its length and range of motion. The configuration and dimensioning of these links directly influence the robot's accessibility to the objects it interacts with.

Cable management

An often overlooked but essential aspect of joint design is cable management. Proper routing and securing of electrical cables is necessary to prevent damage during dynamic movements, reduce mechanical wear, and enhance operational safety. The design should ensure that cables do not interfere with the robots workspace or limit its motion.

End Effector (Gripper)

The end effector, commonly referred to as the gripper, is attached to the distal end of the robotic arm. It is the primary interface between the robot and its environment, and is responsible for handling the plants during pick-and-place tasks.

To accommodate various agricultural applications, the end effector must be modular and interchangeable. For instance, while the current gripper is designed to handle Crispi lettuce, future use cases may include tasks such as seed sowing, which require a more delicate and precise gripping tool. Therefore, designing the gripper as a swappable module enables flexibility and scalability for additional agricultural tasks or new crop types.

Moreover, the gripper must offer a high degree of compliance and flexibility, allowing it to adjust to variations in plant size, orientation, and shape. Plants may not always be uniformly aligned, and the gripper should be able to grasp them from multiple angles without compromising grip stability.

A soft-touch interface is also essential to avoid bruising or damaging the produce. This can be achieved through the use of soft materialssuch as siliconeon the gripping surfaces, which help to cushion contact with delicate plant tissue.

In terms of load requirements, the end effector must be capable of handling a maximum payload of 1 kg, which is a design constraint established based on the heaviest anticipated plant or product. This must be factored into both the mechanical and motor selection processes to ensure reliable operation.

A dedicated motor is integrated into the end effector to control the gripping mechanism, providing the force necessary to securely hold or release objects as needed.

8.4 Definition of working area

BMR | *EG*

To identify the optimal robotic concept, we first defined the robots working area and surrounding environment. The placement and range of the robot have not been part of the requirements given by the contractor beforehand, so it has been important to specify and quantify these together with Hydroplant Technologies (see Section 1 for details on integrating the robot into the HPT system).

The working area, also referred to as the task space, is where the designated task of the robot shall take place. For this project it has been defined as a horizontal plane where the leafy green in its planter appears and is ready to be picked up (they might arrive via conveyor belt, but

the method of transportation lies outside the scope of this thesis). The robots type and design will define its configuration space, which is all the coordinates which the end effector can reach. The configuration space should fully encompass the task space, see fig. 40.

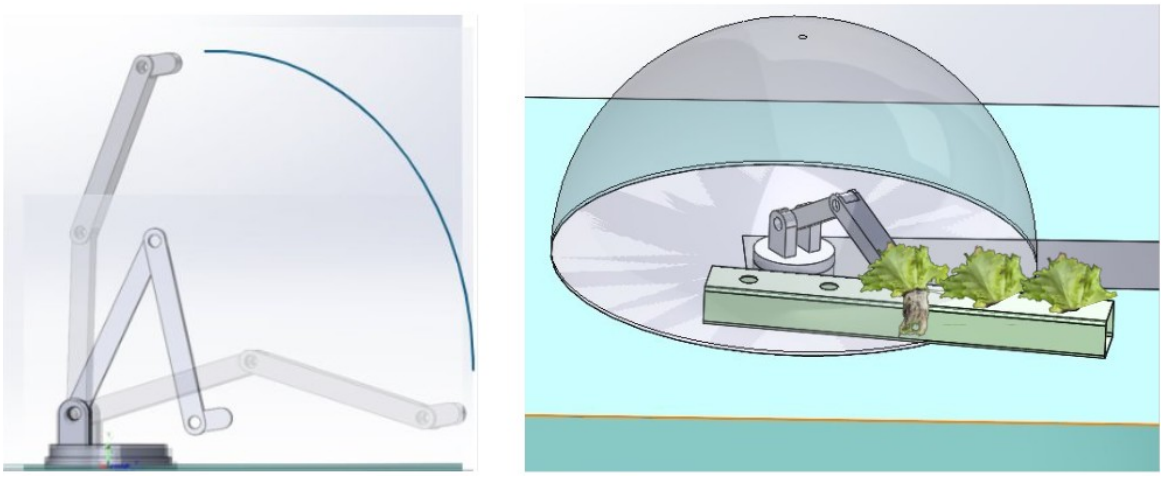


Figure 40: Configuration space and task space graphics

8.4.1 Working area quantification

BMR | *EG*

- The horizontal working area will be a circle of 1200 mm diameter at table height, except for a 200 x 200 mm square in the center where the robot is mounted.

- The reach of the robot arm will be half a globe of radius 600 mm, with some limitations directly above and around the the robot's mounting point.

- The robot arm will be mounted on the same plane and height as the table which receives the plant trays.

A visualization of the quantified working area can be seen in fig. 41, including an example division of task-specific zones.

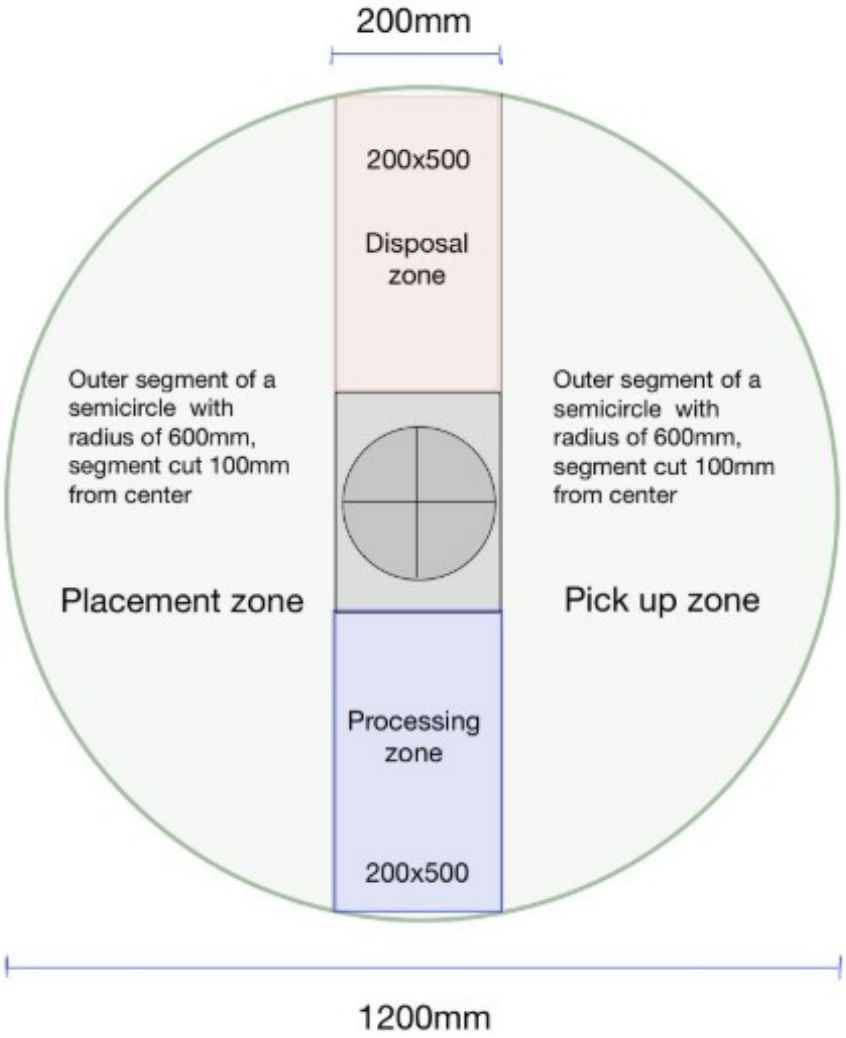


Figure 41: Top view working area quantification

8.5 Definition robot arm

BMR | *EG*

Working on the robotic arm development as a group has made it important to use consistent names and definitions for the distinct parts of the robot. This common vocabulary made it easier for us to adopt design changes across disciplines and prevent interface conflicts. Each link, joint and motor has it's own name and abbreviation, see fig. 42.

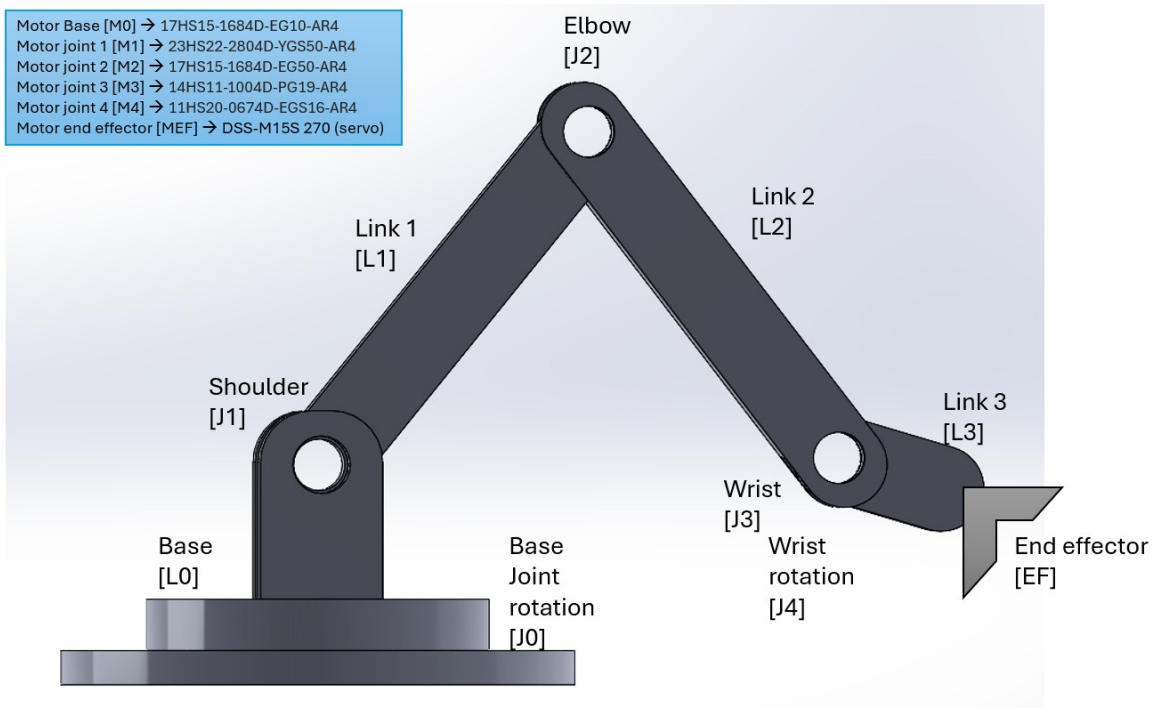


Figure 42: Robotic arm with labels

Alongside the digital model, we built a full-scale MDF model (see appendix D). This physical prototype has been valuable for presentations and for discussing design ideas and challenges within the team.

8.5.1 Degrees of Freedom

BMR | *EG*

The robotic arm consists of rigid bodies (links) connected by joints. Attached to ground at the base and ending in an end effector, this is also called a serial kinematic chain. The Degrees of Freedom (DOF) of a robotic arm refers to the different possibilities of movement of the arm as a whole, and is dependent on the amount and type of links and joints. DOF is an important aspect when planning to build a robotic arm, since it directly effects its suitability for the given task and the designated workspace.

Higher DOF offers more flexibility and precision, while lower DOF brings more constraints on the robotic arm. Despite this, during development we should strive for the minimum DOF needed to meet the task specifications. Adding more DOF's requires additional actuators, which in turn increases the complexity (both the physical build and the computational path-planning. It also makes the robot arm heavier and more expensive to produce.

Degrees of Freedom for the Leafy Automation robotic arm

For the robotic arm we have chosen to have 5 joints to ensure good flexibility also for future applications. The joints are of the type revolute joints where one link can rotate relative to another around the same axis (this type of joint has one DOF, see fig. 43), and so the robotic arm will have 5 degrees of freedom. The end effector will have it's own DOF depending on type.

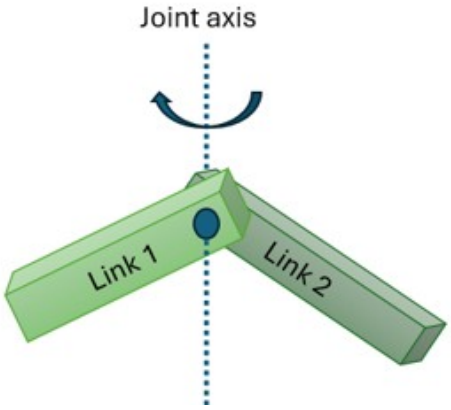


Figure 43: Revolute joint

8.5.2 Joint angle restraints

BMR | *EG*

After determining how many DOF's the robot arm shall have and how they will be arranged to create the desired configuration space, some restrictions can be added to the rotation of the separate joints. Since all joints are revolute, the maximum angle they can move is 360 degrees. Only the base joint needs up to a full revolution to cover the desired work space, the other joints will all have limited rotation. The range will be defined in the code and will have a direct effect on the solution space for the inverse kinematic calculations. The angle limits will also affect the design of the joints and the wire lengths required. The joint restraint angles can be seen in table 44 and fig. 45.

Joint:	Name:	Around axis:	Angle:	Range:
J0	Base	Z_{J0}	Θ_{J0}	0 - 300 °
J1	Shoulder	Z_{J1}	Θ_{J1}	0 - 90 °
J2	Elbow	Z_{J2}	Θ_{J2}	0 - 150 °
J3	Wrist	Z_{J3}	Θ_{J3}	0 – 180 °
J4	Wrist	Z_{J4}	Θ_{J4}	0 – 180 °
J5	End Effector	Z_{J5}	Θ_{J5}	

Figure 44: Joint restraint angles

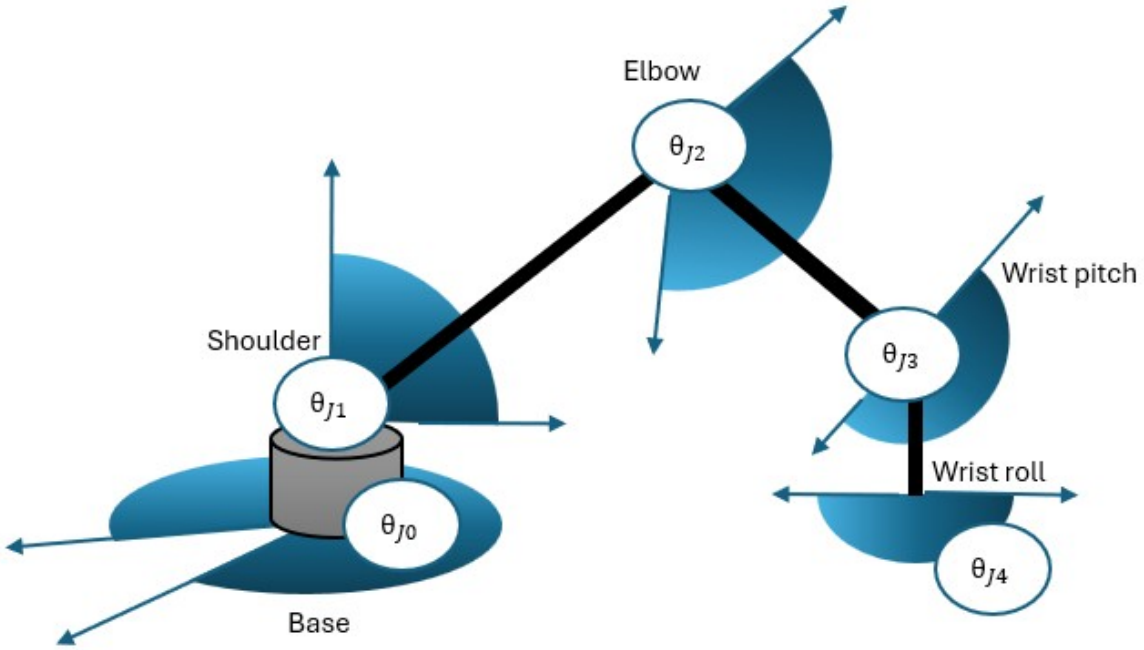


Figure 45: Visualization joint restraint angles

8.5.3 Configuration space

BMR | *EG*

The configuration space, meaning every point within the end effectors reach, is generated by using link lengths and joint-angle restraints. This is best done using a robot simulation tool (like MATLAB) for ease of calculation, but to get an initial understanding we have made a 2D scatter plot in Excel. Figure 46 shows a side-view of the configuration space alongside working-area boundaries and sample lettuce placements for reference.

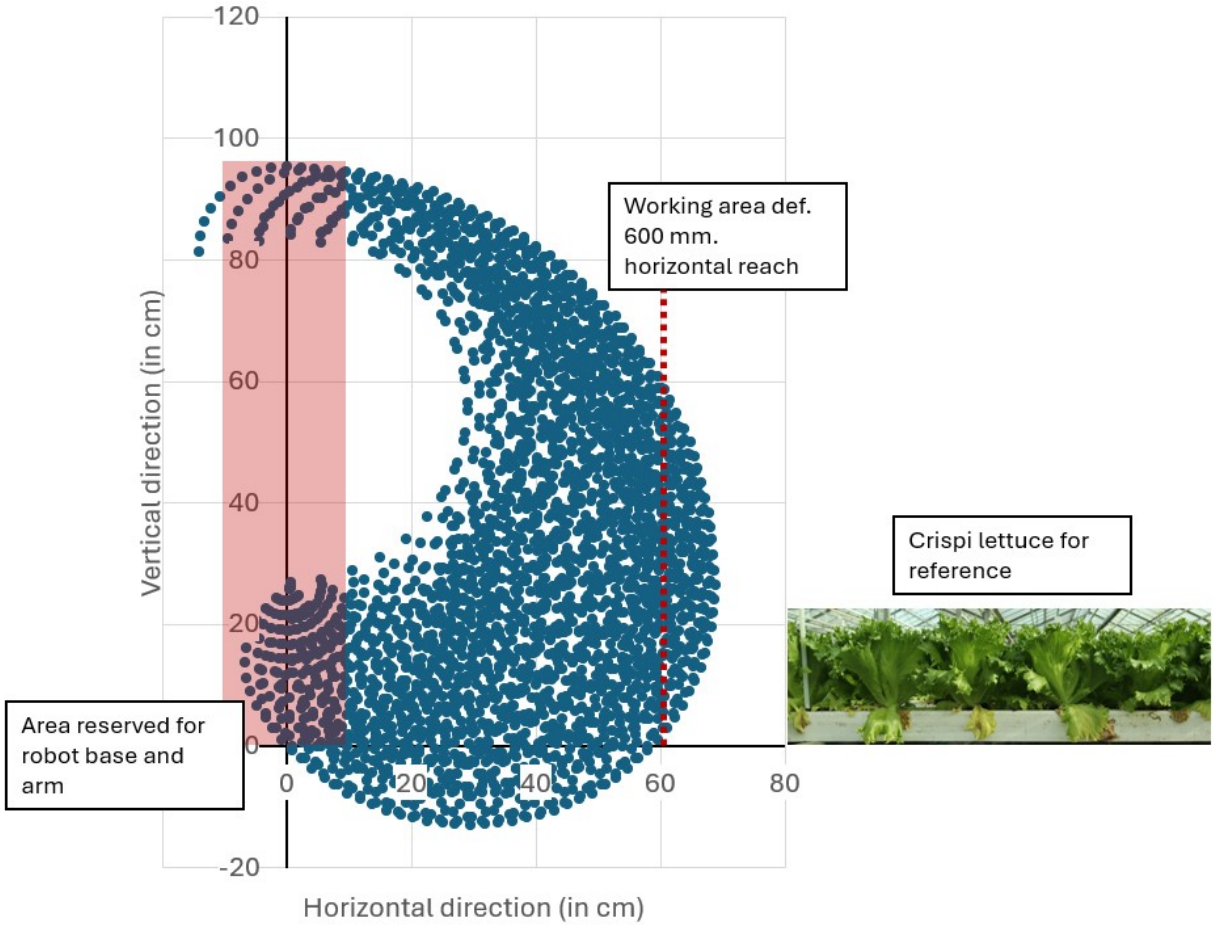


Figure 46: Scatter plot of configuration space

The scatter plot was made using a straightforward equation for planar forward kinematics that use trigonometry (see fig. 47). When using this equation it is important to put in the values for the restrained joint angles measured counterclockwise as seen in fig. 48. To get the right values for the y- coordinates of our robot arm, we added the height from the bottom of the robot base to the first axis to the equation to shift all the points up.

$$P_x = l_1 * \cos(\theta_1) + l_2 * \cos(\theta_1 + \theta_2) + l_3 * \cos(\theta_1 + \theta_2 + \theta_3)$$

$$P_y = l_1 * \sin(\theta_1) + l_2 * \sin(\theta_1 + \theta_2) + l_3 * \sin(\theta_1 + \theta_2 + \theta_3)$$

Figure 47: Equation for planar forward kinematic [2, p. 2]

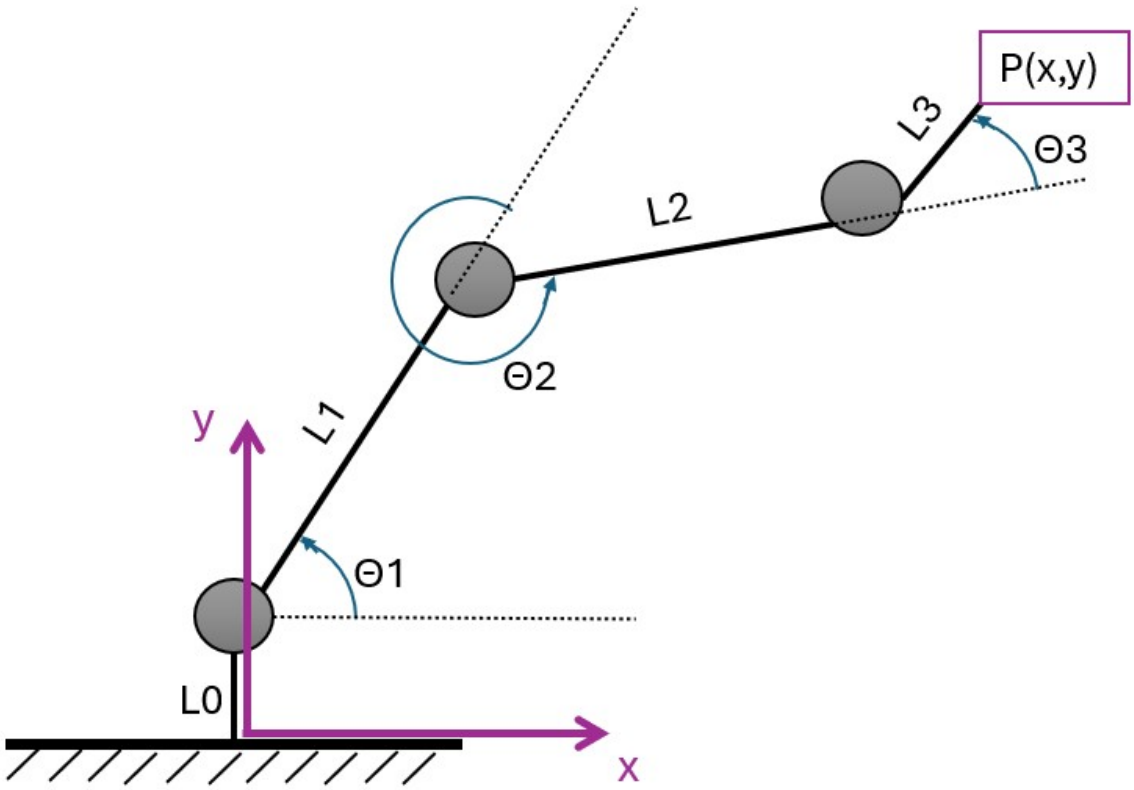


Figure 48: Diagram for forward kinematic equation

This type of plot can help with figuring out where the dead zones are and where the robot must be restrained to avoid colliding with, for example, the base or the table. The plot only shows the outer reach of the 3. link, since the end effector will be exchangeable and so have its own unique length. The excel sheet is made dynamic and could be used to test how different link lengths or restriction angles will effect the reach of the robotic arm, for example for future applications (see appendix H).

Table 4: Critical technologies

Technology	**Purpose**	**Key Challenges**	**Considerations**
AI recognition	Identifies and classifies the plant	Wrong classification, accuracy	High-quality training data, real-world testing
Robotic handling	Picks up and moves plant	Mechanical failure, precision	Durable components, precise handling
Root cutting mechanism	Cuts the roots of the plants	Mechanical failure, precision	Durable components, precise handling
Water proofing and environmental protection	Protects electronic components from water and humidity in the hydroponic environment	Ensuring full water resistance, preventing corrosion	Use water proof enclosures, IP-rated components
Human Machine Interface (HMI)	Provides a user friendly interface for monitoring and controlling the system	Designing a user friendly UI, ensuring responsiveness, accessibility	Web dashboard, mobile app, status indicators
Power management	Manages the power delivery to robotics, sensors and micro controllers	Correct voltages and amperage to all components	General electrical safety, safety in case of water spills
Failure detection and recovery	Detects failures and ensures system continuity	Sensor reliability, software robustness	Redundant sensors, error-handling algorithms
Leafy Automation Core (embedded devices)	Responsible for handling processing and sensor IO	Processing speed	Safety in case of water spills
Leafy Automation Central	Central hub which does AI processing and a web server	Security and complexity	Simple and easy to connect to the embedded system
Communication protocols	Facilitates the communication between the embedded systems, the Leafy Automation Central and the HMI	Choosing the right protocol based on current knowledge	Transmission speed
Motors	Movement, facilitates robotics	Budget	Torque, speed

9 Mechanical

This chapter outlines the mechanical development process of the robotic gripper, detailing the design, prototyping, and evaluation of its key components, namely the base, joints, and gripper. Together, these parts form the structural and functional core of the robotic arm. As seen in figure 39

The design evolution of each mechanical subsystem is presented, from initial CAD modelling and material selection to 3D-printing, assembly, and testing. Particular emphasis is placed on the iterative nature of the design process, including the rationale for structural modifications and the technical considerations that informed each decision.

The chapter also compares prototypes, highlighting improvements in mechanical robustness, manufacturability, and functional performance based on empirical testing and user feedback.

9.1 The base

BMR |

The base is a revolute joint with a vertical axis, and the motor shaft is the connection point between the stationary part attached to the table, and the rotating part attached to the rest of the arm (see fig. 49). The base is subjected to both axial load from the combined weight of the arm with payload and moment load from off center mass (to read more about this see appendix D). For rotation of the base we selected a Nema 17 motor with a 10:1 gear ratio for direct drive. For the base interface to the HPT system, see appendix D.

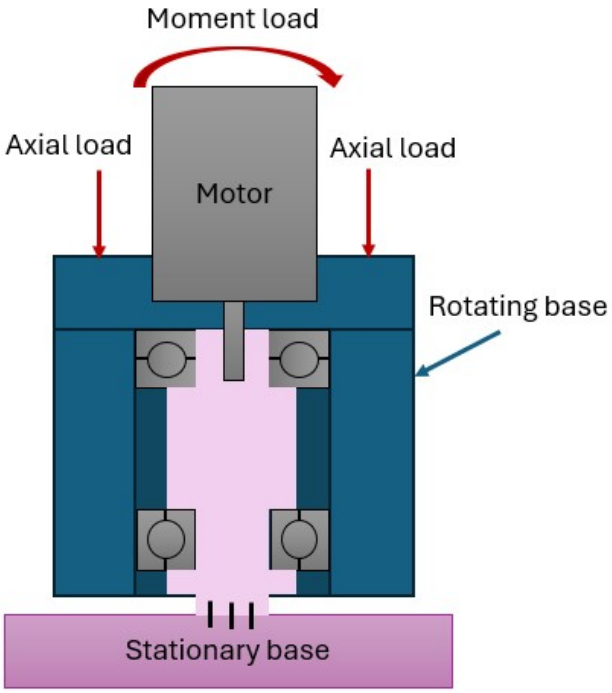


Figure 49: Different forces acting on base

9.1.1 Material choice

BMR |

Important for the base material is that it is rigid, long lasting and strong. Weight restriction is not critical, as is the case for the rest of the robotic arm. Low carbon steels (AISI 1020-1050) are a good choice for shafts ([25, p 375]. The base consist mainly of an installation flange, bearings, a housing and a shaft, so a low carbon steel will be the preferred material. This material will also combine well with the motor shaft and bearing, since these are also made of steel. The robotic arm will possibly operate in an environment with high humidity, so an AISI 1035 stainless steel would be a good choice to avoid corrosion.

Since the material for most of the robotic arm is aluminium, it is important to take account of possibilities for galvanic corrosion in the interfaces between the materials.

9.1.2 Design process for the base

BMR |

Since this project uses an agile workflow, the design has undergone many changes along the way (see fig. 50). The changes are brought on by feedback from our client (HPT), or the other team members, mostly due to interfacing challenges and possibilities.

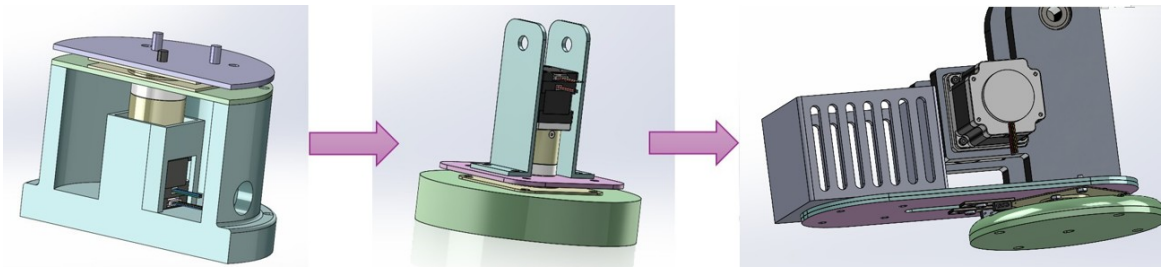


Figure 50: Initial design ideas V1, V2 and V3

Base V1

In the first design, the base was built as a simple container for the motor with a rotational top. An advantage with this design was that the motor wires were kept away from rotation, and the flat top surface gave a very adaptable interface to the arm.

Base V2

The second design idea came from realization that even though the motor wires for the base would be kept away from rotation, the rest of the wiring from the arm would not. All these wires needed connection to the same control units, so it seemed a good idea to gather them at a single point on the base. The motor was flipped upside down so the shaft was going down to the stationary plate and the motor rotating itself and the arm on top. This way all the wiring coming from the arm itself could combine with the wires from the base motor.

Base V3

Building on this idea, the base plate could extend to hold a box with all the control units and all connections for the wiring could be assembled on top of the rotating base. This way there would only be one cable, the main power supply, that needed to follow the rotation of the arm. This design would not only keep the wires organized and free from rotational drag, but would also make the whole robot more easily portable.

The control unit box would contain the motor drivers, the arduino and the Rasberry Pi. To help with heat dissipation a small fan could be implemented together with a perforated wall.

Base V4

To avoid conflict between the required working area and the elongated base plate under rotation, it was decided to attach the control unit box to parts of the base link (L0) with brackets and to design it to build in height to keep the base footprint as small as possible (see fig. 51). The configuration plot (section 8.1.1) could be used here as an insight to were the dead zones

of the end effector were, so the control unit box could be placed intersecting that area.

Limit switches are used to stop unwanted rotation outside the limits of the operating area and is also used as a positioning reference (see section 10). Two limit switches were positioned beneath the rotating base plate, and a removable block was implemented to the stationary base part as a trigger, see fig. 51. The limit switches would take away some of the possible 360 degree rotation, and we decided to make a model of the base in MDF to test how the limit switches would work and how much internal distance was necessary, see fig. 52. The trigger block could be removed for the initial testing, to not damage the limit switches.

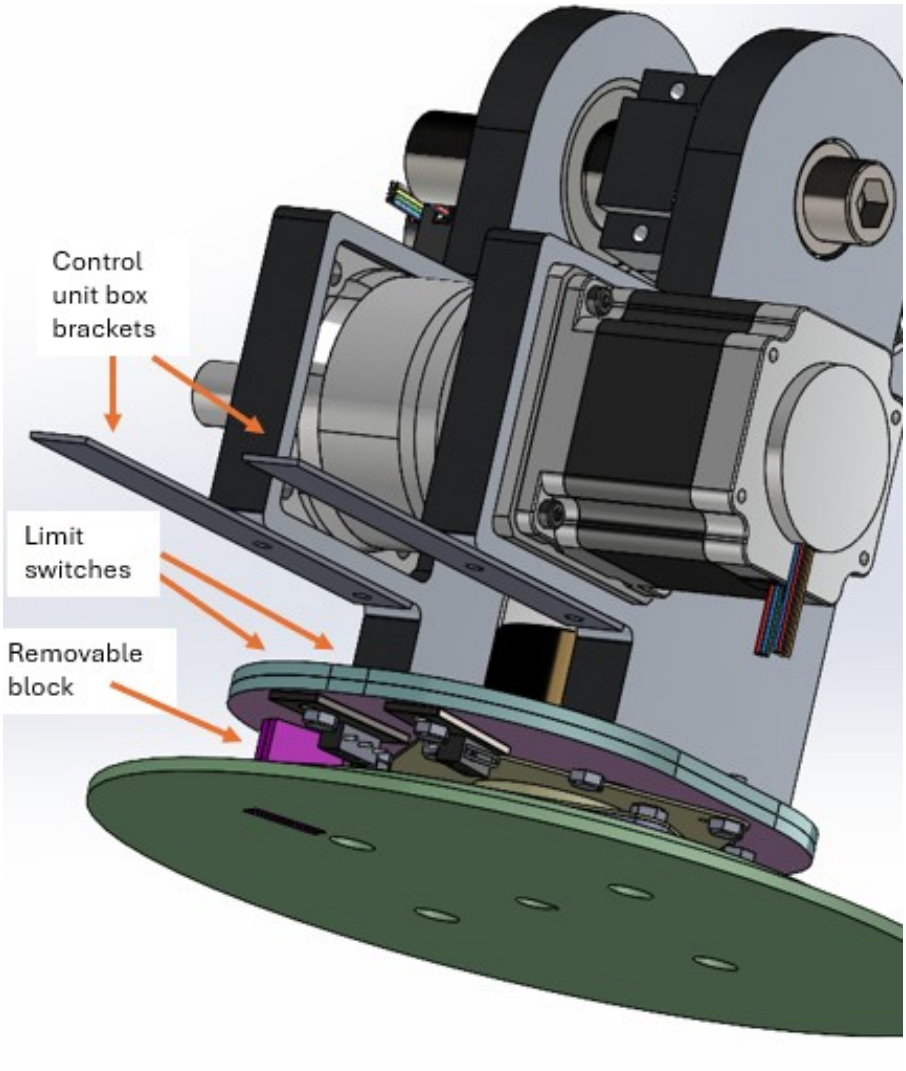


Figure 51: MDF CAD model V4 (CAD motors from [3])

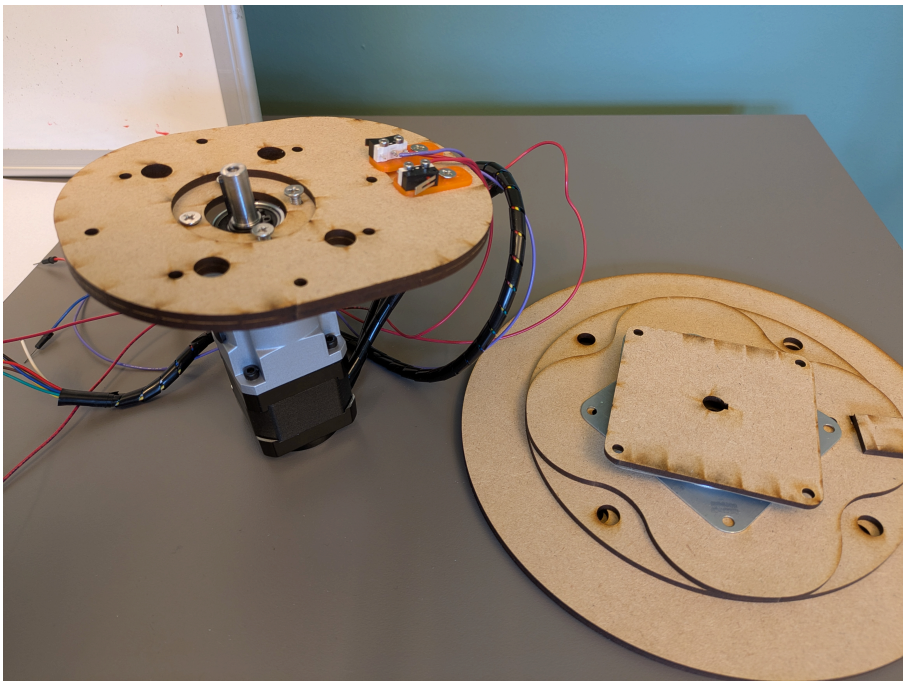


Figure 52: MDF model of base for testing limit switch position

For the finished design for the prototype see section 13, and for thoughts about future work on the base see section 15.1.

9.2 The Joints / arm

DAB | *VMN*

In this section we talk about joint designs, design decisions and designing process. Need to choose material for the parts. Material choices are based on weight reduction, ease of assembly and cnc machinability including the price. The joints and links should handle the moment and bending forces of its own weight including whit a set gripper weight that can be up to 1kg and a payload of 850-1000g

9.2.1 Material Choice

DAB | *VMN*

Joints

Aluminium 6061-T6 was chosen due to being most common and popular alloy grade to use and the website JLCCNC [26]uses specifically 6061-T6 alloy. The alloy is available in SolidWorks material library so we don't have to find and input it manually to the library

Link/arm

The links are made from aluminium profile V-slot type V2020 which is extruded from 6063-T5. 2020 stands for the profile dimension 20mmx20mm. It was chose due to being supplied by our employer, witch means we don't have to use our budget on this. But the extruded profiles are really affordable to buy per meter.

the employer orders from website zeptobit[27], there we can find what type of aluminium its

extruded from. It is also in the standard solidworks material library

9.2.2 Direct drive joints

DAB | *VMN*

Consideration of direct drive was to simplify and ease the assembly process. With a direct drive you need to consider bending forces acting on the gearbox shaft, which will impact the lifetime of the gearbox and bearings, since the weight of the assembly will be resting on the shaft.
Because of this we need to add a second bearing to counter act/minimize the moment forces acting on the gear box bearing. The second bearing will be integrated in the joint and is gonna be removable to ease servicing of the robot arm.
A positive thing with direct drive is that it can rotate full 360°.
V0.2

(a) normal view

(b) Exploded view

Figure 53: Joint V0.2

The joint interface between aluminium profile and the joint are four extruded pegs that are able to slide inside the aluminium profile

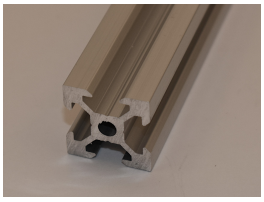


Figure 54: Aluminium profile

The aluminium profile can be seen in figure 54.
Why didn't we also create a small extrusion around the profile for more stability. It was a

aesthetic choice to have the design flush fit against the link.
since we want a flush fit the aluminium profile dictates the dimension of the raw part material at the interface point to be 20mm x 20mm of profile or 20mm x 40mm of two profiles.
Look at the surface where the motor mount interface in figure 53 (b), it is located at the part with the bearing mount. Motor used in this joint is a NEMA 17 it uses 4 x M4 screws for mounting.
The thickness of the motor mount location on the part is reduced to allow for the shaft to protrude enough for the opposite part to interface with the shaft key.
The Bearing mount surface is machined directly in the joint, this causes the part to be higher than 20mm in thickness. the bearing surface is machined to be a interference fit, so that it can be pressed on, and a ring with evenly spaced five m4 holes is pressed on the outside of the bearing. this ring is for ease of assembly and so that the whole arm dosnet need to be dissembled just to press out a bearing. the bearing used i a ID 20mm, OD 27mm, and thickness is 4mm
I speculate that problems with this iteration is that the pressed on ring may slide because of bending moments without no backing. Also the thickness of the part needs to thin enough to withstand the forces while maintaining good shaft protrusion to opposite part.

V0.3

This iteration is with adjustment and modifications to try resolve issues and address possible issues.

(a) normal view (b) Exploded view

Figure 55: Joint V0.3

Differences from last iteration is where the bearing presses on. A removable bearing adapter part was made see figure 55 (b) so that the part is 20mm when machine and can be easily removed with the bearing. The bearing adapter is screwed trough the back side of the motor mount. bolts used for bearing adapter are 4 x m4 screws. So you screw together the arm first and then mount the motor. There are indents in the bearing side piece so that you can screw in the motor screws.
A useful unintentional design is that when you want to separate the bearing and bearing adapter

you can screw in longer screws to push the bearing off.
Interface between aluminium profile and joint added extra height in middle 6mm wide to be flush with the alu profile se figure 55. Also added screw holes two M4 crews to tighten against alu profile to prevent moment and slip. while also removed to legs to alu profile interface so that the parts are more easily machinable, since now there is good space for the end mill to get in-between and machine properly. the piece just need to be flipped to machine the other side. since the tight small gaps in-between the legs are almost impossible to machine or requires fine machining which costs allot. this design is limited only to 180°angle. this can be easily adjusted by just moving the bearing interface face forward to increase movement angle.

V0.4
this is the last iteration created of direct drive joint before moving on to belt driven joint design.

(a) normal view (b) Exploded view

Figure 56: Joint V0.4

This iteration have small changes. moved away from two aluminium profiles used to one instead for weight reduction, also moved the bearing forward for greater angle of rotation mentioned in the last iteration to be done, and a circular extrusion to keep the motor surface covered for a nice aesthetic look.
figure 56 shown is for joint 2 to joint 3. but joint 3 will be the same just adjust the design for use of one aluminium profile and a nema 14 motor instead of nema 17.
joint 1 v0.4 has not been create before moving to belt design.

9.2.3 Belt drive joints

DAB | *SME*

The experience and knowledge we gained from designing the direct drive, we brought with us in the process of designing the belt version.
The choice to use belt drive mechanisms, was to offset the weight more towards the centre of

the robot so there would be less strain on the motor, which will improve lifting capabilities. With the addition of pulleys we can adjust the gear ratio, by increasing or decreasing it. This helps if limitation on torque would occur on some of the motors. The site used to find pulleys and belts that would be suitable for our robot arm, you can find here [28], which was introduced to us by our employer.

The pulleys we found for our shaft sizes.

J1

- 162230K14
  - 30Teeth, 14mm bore
- 162230K20
  - 30T, 20mm
- Gear ratio of 1:1

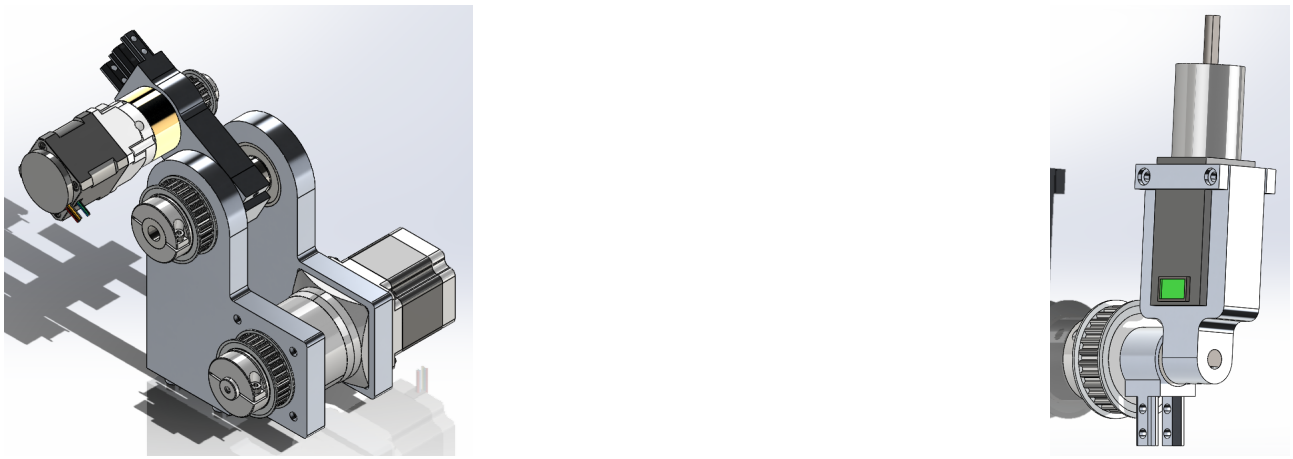
J2

- 162216K08
  - 16T, 8mm
- 162222K12
  - 22T, 12mm
- Gear ratio of 1.375:1

J3

- 162214K05
  - 14T, 5mm
- 162228K08
  - 28T, 8mm
- Gear ratio of 2:1

These pulleys use a belt type T5 with a width of 10mm. The pulleys are made from Aluminium similar to EN AW2017A stated on their website. You can see more detailed explanation for belts and pulleys in Appendix C.



(a) normal view (b) Normal view

Figure 57: Joint V0.1B J1 and J3

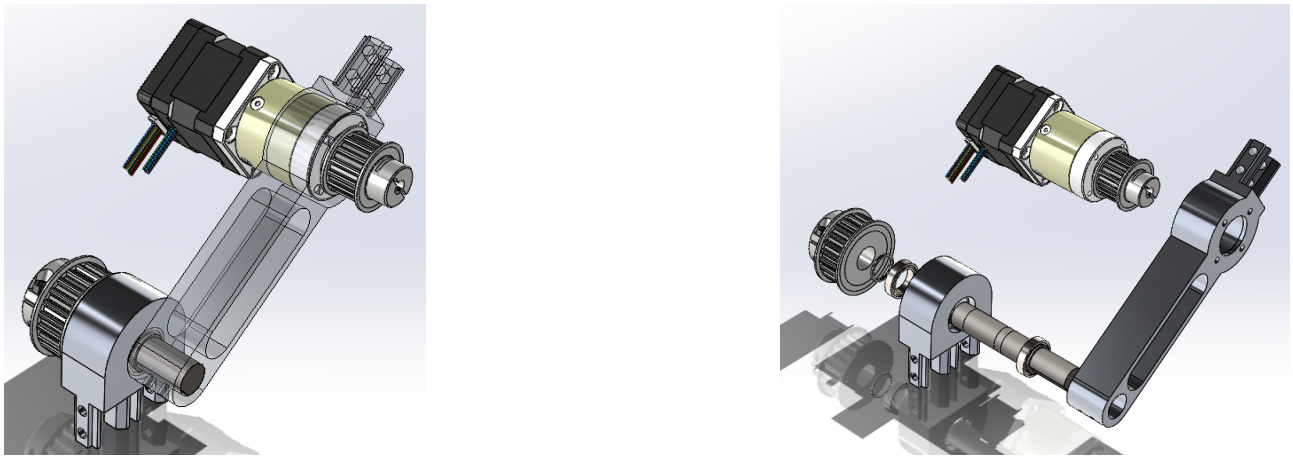
We can start with the part that interact with the base (a) in figure 57. These two plates supports the NEMA 23 motor that has a offset from the centre at the same plane as the arm so

that some of the weight of the motor acts as a counter balance for the arm, and is mounted so that the main body of the motor is in the middle to minimize bending forces on the mounting plates. The plates have also dowels underneath to have correct alignment with the base.

The joint uses 2 bearings for a 20mm shaft size, the bearings are pressed in to each plate for good stability.

The transmission of rotational and moment forces to the robotic arm was inspired by the method used to fasten connecting rods to a crankshaft. One piece is the main part going to the rest of the arm with a mounted NEMA 17 for joint 2, and the end cap that is bolted on. Both surfaces, and shaft are equipped with Ø4mm holes to use a shaft-pin instead of the usual shaft key. It is easier to assemble this way, since you can disassemble the whole arm without disassembling joint 1.

Joint 3 (b) in figure 57 is almost identical to J2, but one difference is that it goes from single profile to single profile, 2 bearing to support a 8mm shaft and stabilize moment loads. We also have a small motor for rotation of the gripper. One small problem that occurred is that the motor is quite long when attached to the gearbox. So because of that, the bracket around it goes up to the motor and clamps around it for a good snug fit. One adjustment to help with this, could be adding two brackets for good stability from moment forces.



(a) normal view (b) Exploded view

Figure 58: Joint V0.1B J2

J2 in figure 58 is connected to the motor for joint 3, and the bearing that is used is a thin section bearing to support a shaft of 12mm. The part where a NEMA 14 motor is mounted needs to be long enough from the shaft to motor shaft so that we have adequate range of motion before motor touches the frame.

9.2.4 Belt tensioner design

DAB | *SME*

We tried making a simple belt tensioner that fits in the aluminium profile and is fastened in place by a M4 screw. It is made so that you do not have to disassemble the arm to put in the aluminium profile, just put in place and twist it so that the legs move under the aluminium profile (See figure 59).

An M4 screw is threaded into the component, which causes the legs to expand since the hole is smaller. This makes it a better and more snug fit, and once the screw reaches the bottom of the hole, it also pushes it up and presses it more against the aluminium profile.

Also the design has a 1mm thin 14mm spanner that holds it in place, so that it is easier to insert the screw.

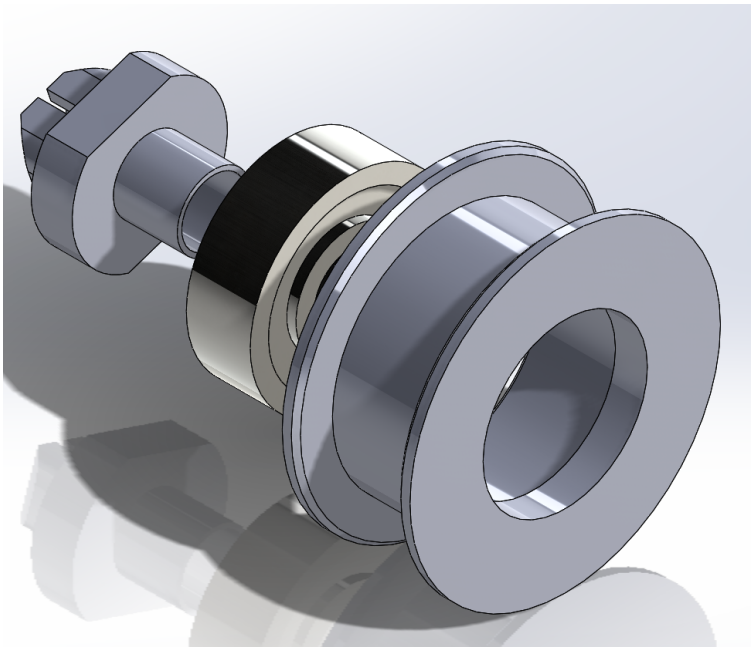


Figure 59: Tensioner for belt

The tensioner for joint one is more tricky to design, since we need to create a small rail where it can slide in and be set in place by a screw.

Bearings used for the tensioners are the same used in joint 3 since we had quite a few of those bearings.

Generally these tensioners were designed to be 3D-printed and not to be made in any other specific material.

If we would like to have better tensioner we would go for a off the shelf tensioner pulley, and only manufacture the interface between aluminium profile and tensioner pulley.

9.2.5 Limit switch placement

DAB | *SME*

There must be 2 limit switches per joint and for gripper rotation to define end stops. Read more about limit switch in later section 10. And for correct working angles required, refer to section 8.1.1 figure 44.

For joints 2 and 3

We were thinking that the easiest placement would be on the Aluminium profile since we can easily change its position along the profile and also add a joint to adjust the angle of the switch for fine tuning. It gives us plenty of flexibility for testing.

Joint 1

Need to make holes in the side plates in joint 1 so that we can create limit switch brackets to get the correct angle we desire.

For the Gripper

Made small extension from the motor mount bracket. It extends almost to the edge of the gear box and the switches are stacked to reduce used space and ease of installation using only 2 x M2 screws to hold the switches in place.

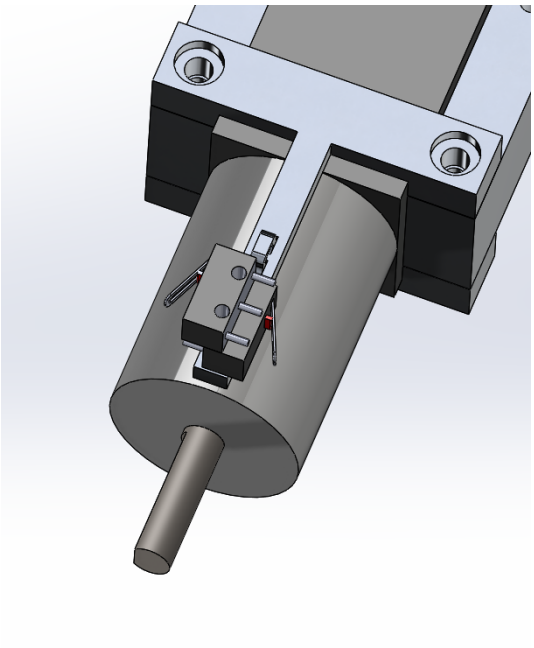


Figure 60: Gripper limit switches

9.2.6 Single(mono) design vs multi part design

DAB | *SME*

We have created a multi part design, in this section we will discuss and compare it to a single piece design.

Modularity

With multi part design each piece can be modified, upgraded, or replaced without redesigning the whole product. It gives greater flexibility for future adjustments, and customizations requested by the employer.

But with a mono piece you need to make a whole new part which leads to more expensive development, repair, and future development.

Cost

Multi parts can be more easily produced using standard manufacturing processes to potentially reduce overall costs. However, it requires more precision in assembly to maintain quality and performance. With a mono piece it is the cost of machining the whole thing that can get expensive fast if the parts are large, and if it is not possible to use standard production techniques.

Maintenance and repair

Multi parts allow for easier repair since a malfunctioning part can be replaced independently, while with a mono piece you have to dissemble everything which introduces a longer down time and increased repair costs. Individual component failure can be harder to detect and fix compared to an assembly.

Design challenges

One challenge with multi parts is that we need to ensure a proper fit between parts and have good structural integrity in junction points, like joints, aluminium profile to end parts, and more.

While mono parts eliminate a lot of connection points reducing misalignment and give better structural integrity through out the part, it demands high control over manufacturing tolerances since there is no opportunity to correct errors during assembly stage.

9.3 Gripper

SME | *JCDH*

In the process of selecting an appropriate gripper for our robot arm, we did a comprehensive review of existing robotic gripper technologies. This included conventional designs such as jaw and finger grippers, as well as more advanced soft robotic solutions like pneumatic soft grippers and Fin-Ray grippers. Each concept was evaluated on the basis of its adaptability, mechanical complexity, and suitability to handle fragile produce such as lettuce.

Of particular importance was the integration of soft-touch technology, which is essential in agricultural applications to avoid bruising or damaging delicate plants during harvesting. Materials such as silicone and Thermoplastic Polyurethane (TPU) were identified as favorable due to their flexibility, cushioning properties, and compliance with food-contact safety standards. Techniques like silicone molding offer customizable and biocompatible gripper surfaces that enhance both precision and gentleness during operation.

The Fin-Ray gripper emerged as a strong candidate due to its simplicity, compliance, and natural ability to conform around irregular shapes without the need for complex sensors. This, combined with the ability to 3D-print the design using food safe materials, makes it a cost effective and scalable solution for controlled environments like hydroponic facilities.

A more in-depth technical overview of each gripper type, along with discussions on material selection and soft-touch design principles, is included in Appendix C.C.

Materials and Techniques

The gripper components in this project were manufactured using Fused Deposition Modeling (FDM), which is one of the most accessible and widely used 3D-printing methods. This technology melts thermoplastic filaments and extrudes them layer by layer to form the final geometry. To read more about 3D-printing, go to Section 13.

Two primary materials were used for the gripper:

- Polylactic Acid (PLA): Chosen for its ease of printing, rigidity, and availability. PLA was used for all structural components where stiffness and dimensional accuracy were required.

- TPU: Selected for the Fin-Ray fingers in Prototype 2 and final design, TPU offers the flexibility and elasticity necessary for soft-touch gripping. It allows the fingers to deform around the object being handled, reducing the need for sensors while enhancing adaptability. We used TPU 75A, which is very soft and flexible. We also tested TPU 95A, that is much harder and more rigid. For the purpose of the Fin-Ray gripper it was most suitable with the 75A since we wanted a more stretchier TPU that had higher elongation.

Moreover, modularity was prioritized to facilitate reprinting and replacement of individual components without the need to manufacture the entire assembly again. This modular approach was especially valuable for testing and educational environments with limited budgets and access to industrial manufacturing tools.

Benefits and Limitations

The primary advantage of using 3D-printing in this context was the ability to rapidly iterate on the design at minimal cost. Changes could be made in Computer-Aided Design (CAD) and quickly verified with a physical prototype. However, 3D-printing is not ideal for all application components that experience high mechanical loads or require long term durability, and may eventually need to be replaced with machined metal versions. These considerations are explored in more detail in chapter 15.1.

9.4 Design Process for The Gripper

SME | *JCDH*

Gripper V1

The first prototype focused on developing a jaw-style gripper, primarily intended for initial testing, data collection, and integration with electronic components. This version was not designed to serve as the final gripper design for the robotic arm but rather as a preliminary functional model to support early-stage development.

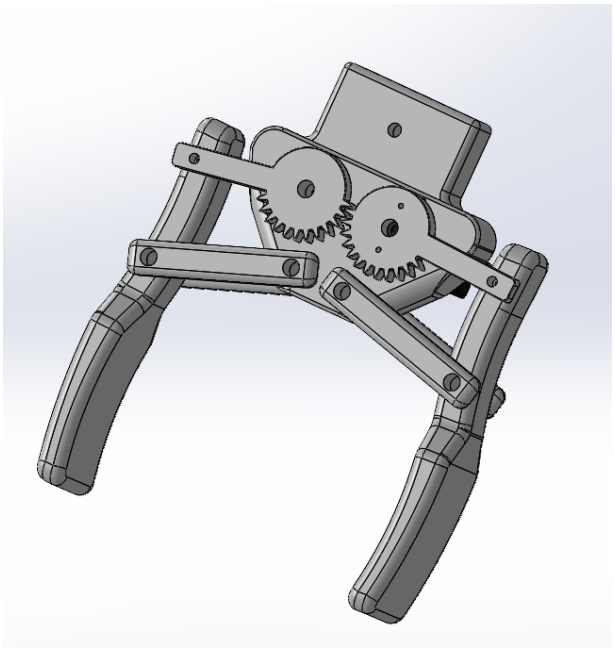


Figure 61: Gripper V1

The design was intentionally kept simple to ensure the ease of manufacturing, cost-effectiveness, and rapid iteration. An important consideration from this design was modularity since the gripper needed to be easily replaceable to allow for quick upgrades in later prototypes. At this stage, we also evaluated whether to implement soft-touch technology or to postpone this feature until a later version more aligned with the final design goals. For prototype process and development of V1, read more in Section 13.

Gripper V2

The second prototype shifted focus toward developing a more advanced gripper capable of imitating the flexibility and functionality of a human hand. This iteration introduced finger-based gripping mechanisms inspired by the Fin-Ray principle, which is a form of soft-touch technology that allows for passive adaptive grasping.

Figure 62: Gripper V2

This design enhancement significantly improved the grippers ability to conform to objects with complex geometries and fragile surfaces. Such adaptability is crucial when han-

dling delicate produce, such as leafy greens, in agricultural automation settings. By applying an iterative design approach, the V2 prototype demonstrated a promising balance between flexibility, precision, and gentleness which are key attributes for robotic harvesting applications. For prototype process and development of V2, read more in Section 13.

Final Gripper Design

The final gripper design is almost identical to the V2 prototype, with only minor modifications. The most notable change was the addition of two brackets mounted on the sides of the gripper body. These brackets were introduced to enable the integration of limit switches on the end effector, ensuring reliable contact during rotational movements of the gripper. This addition supports position feedback and enhances overall control of the system.

Following the successful testing of the V2 prototype, the design proved to be functional and well suited for proof-of-concept validation. Given that all components were 3D-printed, and no parts were machined, the Fin-Ray inspired gripper was selected as the final concept. Its soft touch, adaptive nature, combined with the practicality of additive manufacturing, made it the most viable solution for the intended agricultural use case.

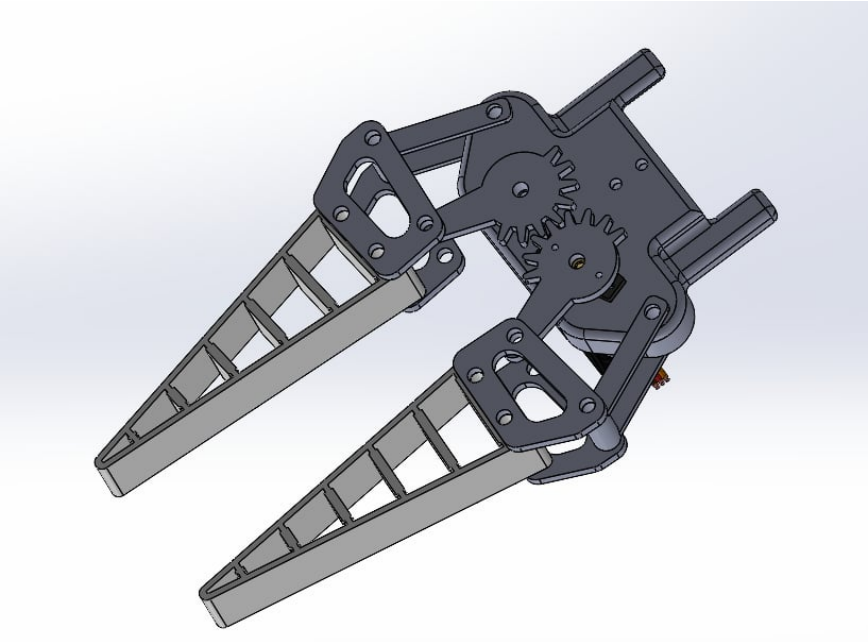


Figure 63: Final Design

9.4.1 Calculations

SME | *JCDH*

Gear calculations

Sources for maximum torque and force/ torque formula you can find here. [29].

Servo max torque = 18kg

Convert to Nm → Torque (Nm) = kg x cm x 0.09807 (1 kg cm = 0.09807 Nm)

T = 18 x 0.09807 = 1.76526 Nm ≈ 1.77Nm (worst case scenario)

Formula for force:

$$F = \frac{T}{r}$$

F = Force (N)
T = Torque (Nm) = 1.77 Nm
r = Radius (m) = 12 mm (distance centre hole to surface)

$$F = \frac{1.77\ Nm}{0.012\ m} = 147.5\ N$$

$$\text{F per tooth} = \frac{147.5\ N}{2} = 73.75$$

Figure 64: Calculations: force on gear teeth

9.5 Structural integrity

BMR | *EG*

Once the robot type is selected, we must identify all the forces acting on its structure. These force calculations provide the foundation for the design process, guiding motor selection and placement.

Initially, we focus on static equilibrium moment calculations at each joint. These initial calculations are valuable as they help us to develop an intuition into force magnitudes. For example, we can determine the minimum holding torque required for each actuator in its corresponding joint. These results can then be compared against motor specifications and gearing ratios. Because this arm is not intended for high-speed applications, we disregard inertial and acceleration effects in this thesis.

9.5.1 Static moment calculations

BMR | *EG*

Since the robot operates under conditions with normal gravitational acceleration, the forces

from both the load and the arm itself will always point vertically downward. The robotic arm is situated indoors and is therefore not subjected to horizontal forces from, for example wind.

Moment calculation and the effect of robotic arm configuration

A moment is defined as a force acting at a distance (M = F * d), the moment acts around a point and the distance is the perpendicular distance from the force vector to this point. The aim of the calculations is to find the minimum values needed to counteract the moment in each joint and achieve a static equilibrium. For this reason, it is imperative to calculate from the arm configuration that gives the highest moment values. Figure 65 shows how the positioning of a link at different angles will affect the moment created by vertical loads. The conclusion will also be valid for when the link is pointing downwards. The calculations show that the moment will always be at it's maximum when the robotic arm is fully outstretched horizontally. Therefore, this configuration is the one used in the static calculations as a "worst case scenario".

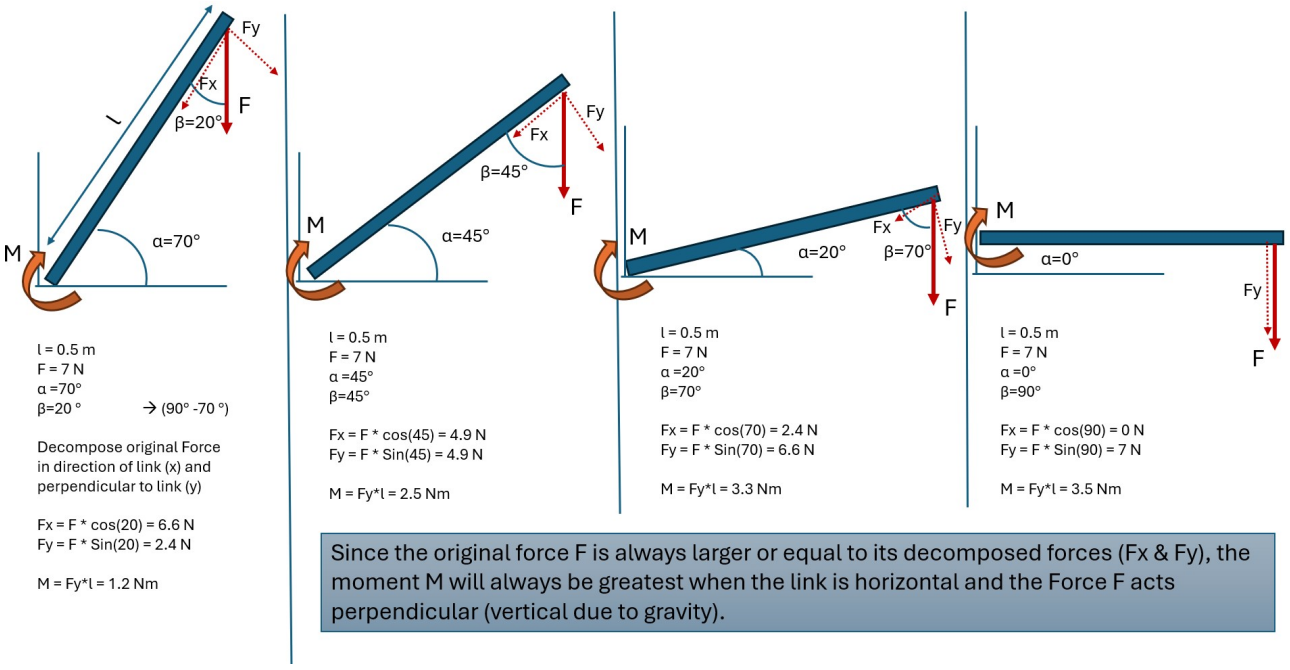
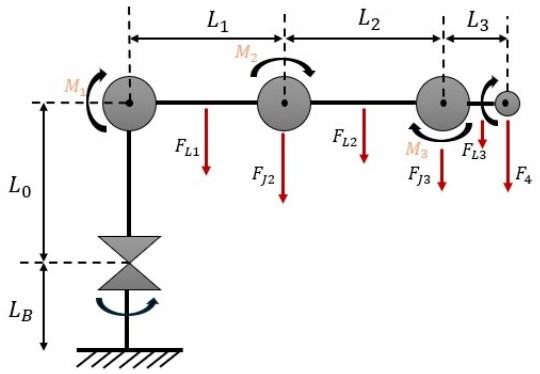


Figure 65: Decomposing forces

Static moment calculation V1

The moment in each joint is created by the different loads, in this case the weight from the payload, the weight of each joint and the weight of each link, see fig: 66. These calculations are best done iteratively, since the weights of the joints and links are not preset. For this an excel sheet was created with input parameters for the different masses and link lengths (see appendix H). Fixed constants like the gravitational constant g is used to convert the masses to forces (in newtons), and the moment equations are defined. The gravitational center of each link is in this setup 1/2 of the link length and where the equivalent force is situated. The equivalent

force is a force that acts at a single point and replaces the forces that are distributed along an object.



$$\boldsymbol{M_1} = \left(\frac{L_1}{2} * F_{L1}\right) + \left(L_1 * \ F_{J2}\right) +$$
$$\left((L_1 + \frac{L_2}{2}) * F_{L2}\right) + \left((L_1 + L_2) * F_{J3}\right) +$$
$$\left((L_1 + L_2 + \frac{L_3}{2}) * F_{L3}\right) + \left((L_1 + L_2 + L_3) * F_4\right)$$

$$\boldsymbol{M_2} = \left(\frac{L_2}{2}\right) * F_{L2}) + (L_2 * F_{J3}) +$$
$$\left((L_2 + \frac{L_3}{2}) * F_{L3}\right) + \left((L_2 + L_3) * F_4\right)$$

$$\boldsymbol{M_3} = \left(\left(\frac{L_3}{2}\right) * F_{L3}\right) + \left(L_3 * F_4\right)$$

Figure 66: FBD of robot arm with moment equations

As seen by the moment equations and Free Body Diagram (FBD) in fig. 66, the moments at each joint is only affected by the forces acting to the right (towards the end effector). Beginning with finding the moment at joint 3, choosing a suitable motor and plotting in its corresponding weight and then working the way inwards to joint 1 gives all the values needed for the moment calculation.

To illustrate how the payload weight affects these moments, we plotted each joints moment against payload mass in Fig. 67. The relationship is linear, and it is evident that joint 1 sees the largest increase in moment as payload weight grows.

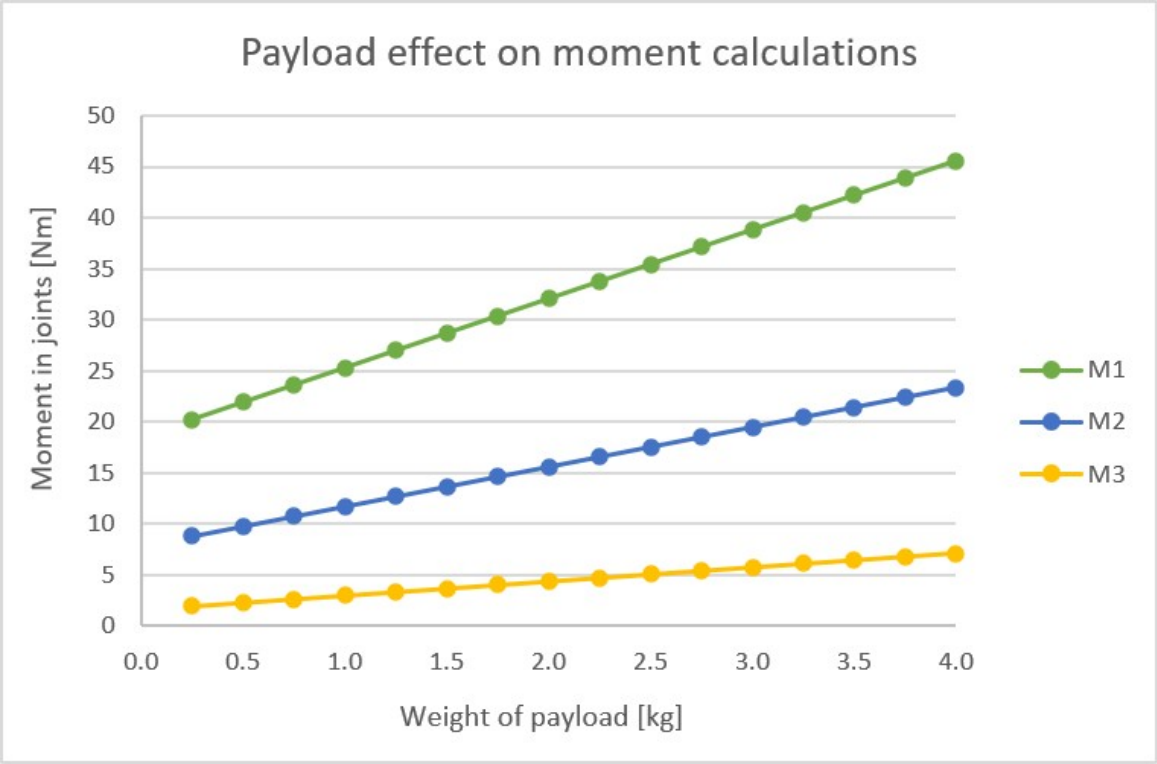


Figure 67: Payload effect on moment in joints

Static moment calculation V2

After some changes in the design where the motors were moved closer to the base and away from the joints (direct drive to belt drive, see section 9), the need for more specific calculations appeared. As the motors contribute with a substantial part of the total weight, the center of gravity was no longer in the middle of the links, and needed to be found to adjust the distance between the equivalent force and the joint, see fig. 68. The center of gravity was found using the equation seen in fig. 69. New moment values could be calculated using the updated distances and a FBD seen in fig. 70. The excel sheet can be seen in appendix H.

	L1			
X1	X2	X3	X4 (L1)	
0.03	0.06	0.177	0.295	[m]
m1	m2	m3		
0.155	0.727	0.173		[kg]
X(cg)				
0.07478	[m]			

	L2			
X1	X2	X3	X4(L2)	
0.045	0.09	0.172	0.255	[m]
m1	m2	m3		
0.066	0.313	0.06		[kg]
X(cg)				
0.09444	[m]			

	L3	
X1	X2(L3)	
0.07	0.14	[m]
m1		
0.443		[kg]
X(cg)		
0.07		

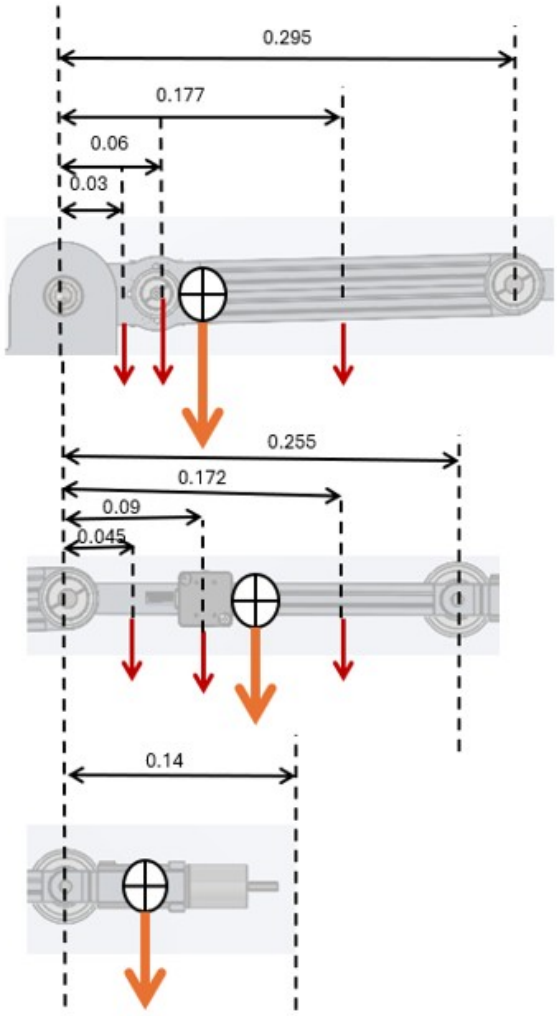


Figure 68: Center of gravity for each link

$$x_{cg} = \frac{m_1 * x_1 + m_2 * x_2 + m_3 * x_3 + \cdots}{m_1 + m_2 + m_3 + \cdots} = \frac{\sum_i m_i x_i}{\sum_i m_i}$$

Figure 69: Equation for center of gravity [4, p. 365]

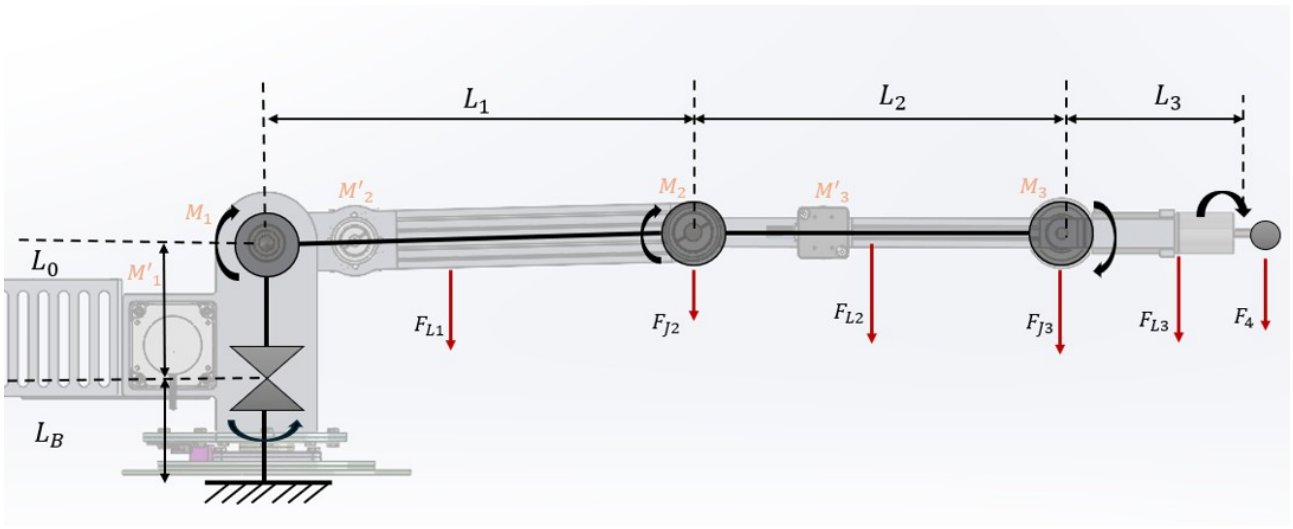


Figure 70: FBD of robot arm (V2 with belt drive)

Conclusion moment calculations

The calculated moments were compared to the maximum holding torque specifications on the motors selected (see section 10). Figure 71 shows which motors were chosen for joint 1-3 and their respective safety factors.

V2 - belt drive design						
	Torque at joint:			**Max holding torque (w. gear):**		
	[Nm]		Nema 23 (1:50)	47.94		
M1	19.913		Nema 17 (1:50)	18.33		
M2	9.913		Nema 14 (1:19)	1.92		
M3	3.051		Nema 17 (1:10)	3.74		
	Torque at motor:	**Pulley ratio:**		**SF**	**Motor:**	
M1'	19.913	1		2.41	Nema23(1:50)	
M2'	7.2098	1.375		2.54	Nema17(1:50)	
M3'	1.5255	2		1.26	Nema14(1:19)	
M3''	1.5255	2		2.45	Nema17(1:10)	(alternative motor)

Figure 71: Motor selection from moment calculation

10 Electronics

Electronics is a crucial aspect of a robotic arm. It is needed to control it and get feedback to the control unit. To achieve this, motors, drivers, and sensors are needed.

10.1 Sensors

VMN | *DAB*

Sensors are used to measure a change in the environment, as a example one of the most common sensor is a temperature sensor that measure the ambient temperature and then reports it to a microcontroller which then displays the temperature on a display.

Incremental Rotary Encoder

An incremental rotary encoder is used to know the speed and direction of rotation. It has two phases, called A and B, and an optional phase Z. A and B are used to calculate the speed and direction, and Z is used to know the centre position of the encoder. To know the exact location a reference position is required, and the easiest solution is to add limit switches.

Absolute Rotary Encoder

The absolute encoder is very similar to an incremental rotary encoder, but it has more phases so it can determine the exact position instead of only speed and direction. For a robotic arm, this can be used to determine the exact angle between the links on power up. There are two commonly used absolute encoders, magnetic or optical. Optical absolute encoders are the easiest to interpret since they often use Gray code, so for every pulse, the binary code only changes by one bit.

Limit switch

A limit switch is a switch that gives a signal when it has reached the limit of the operating area. This is often used with open-loop stepper motors to signal when it has reached an reference position. It can be useful in a robotic arm to stop the movement of an axis when it has reached the limit of the defined working angle it should operate in.

Camera

A Camera (image sensor) is a collection of light sensors (pixels) that measure the light level of specific colors. The gathered values are then combined into an image using signal processing.

10.2 Electric motors

VMN | *DAB*

An electric motor is an actuator that converts electrical energy to mechanical energy. A motor

is required to move the robotic arm to interface the mechanical parts with the electronics and software.

Stepper motor

A stepper motor is a BLDC motor that uses steps and moves incrementally and is usually an open-loop motor. Most stepper motors have a step angle at 1.8°, so the shaft will rotate in increments of 1.8°. An open-loop stepper motor does not have logic, so it requires a stepper motor driver, which can have varying functionality. Stepper motor drivers can detect missing steps, detect if it is stalling, and change the running or idle current in software.

Closed-loop stepper motor

A closed-loop stepper motor is a stepper motor with an added encoder to have a feedback system. This makes it work similar to a servomotor.

Servomotor

A servomotor is a form of closed-loop motor that has a feedback loop that includes a position encoder. This is useful in robotics because you want to know the position and correct errors if they occur. Servomotors can consist of an absolute encoder, a BLDC motor, a reduction drive, and a motor driver.

10.3 Choosing a Motor

VMN |

All of the motors listed in the previous section can be used for a robotic arm, but it depends on the requirements. One of the most common motors used for robotics is servomotors. Since the project focuses on costs, servomotors have been ruled out because they cost a lot more than a stepper motor and are more readily available.

Motor kit	Motor Size	Holding Torque w/o gearbox (Nm)	Gear Ratio
17HS15-1684D-EG10-AR4	NEMA17	0.39	10
23HS22-2804D-YGS50-AR4	NEMA23	1.02	50
17HS15-1684D-EG50-AR4	NEMA17	0.39	50
11HS20-0674D-EGS16-AR4	NEMA11	0.14	16
17E19S1684MB4-200RS-AR4	NEMA17	0.44	1
14HS11-1004D-PG19-AR4	NEMA14	0.125	19
Stepper motor drivers	RMS current	Input voltage	
DM332T	2.29A	10-30VDC	
DM320T	1.56A	10-30VDC	
Power Supply	Current	Input voltage	Output Voltage
LYD2409000	9A	100-240VAC	24VDC
		Total price before VAT and Shipping:	$657.43

Figure 72: Motor kit BOM

For the project, the kit from stepperonline in Fig. 72 has been chosen. This kit consists of six stepper motors, whereas five are being used. These motors come in a few different sizes with different gear ratios. All the motors are fitted with incremental rotary encoders. This kit contains stepper motor drivers for each motor with different current ratings. It also includes a 216W (24V, 9A) power supply.

The gripper uses a DSS-M15S servomotor that can rotate a total of 270° and has an analog feedback to know the current position. This servomotor has a holding torque of up to $12kg \cdot cm$ depending on the supply voltage. It can pull up to 1.76A at 7V.

10.4 Stepper motor drivers

VMN | *DAB*

DM320T/DM332T are the stepper motor drivers that are provided with purchased kit, which have settings for microstep resolution and current. They are controlled using two signals called pulse and direction. The direction signal chooses the direction it should move and the pulse signal determines how many steps it should move. It does not have an input for the included encoder, so missed steps have to be compensated in software by having the encoder directly connected to the microcontroller. These stepper motor drivers have limited functionality and are not really suited for a robotic arm. It can not detect whether it is stalling or it is missing steps itself. This can be solved by changing to another stepper motor driver.

Trinamic TMC5160 has a lot of features that are useful for robotics. It has features like CoolStep, StealthChop, and StallGuard. StallGuard is a feature to detect when the motor is stalling, and it has to be configured for a specific load, it measures the current going through the coil to determine if the motor is stalling. CoolStep is a feature that can be used if the StallGuard has been tuned and adjusts the current flowing through the coil depending on the load applied to the motor. This feature is useful to reduce power consumption and the heat generated by the motor and driver. StealthChop is a proprietary chopper mode created by Trinamic, this feature has a reduced noise level compared to a regular chopper mode. Chopper mode is just a way to regulate the current going through a coil by turning an H bridge on and off repeatedly and varying the time it is turned on.

The interfaces available to use with the TMC5160 are Step / Dir, UART, SPI, or RS485. Step/Dir gives almost no control in standalone mode, and the features have to be enabled at hardware level instead of in software, so it is not an interface that is applicable to this usage. SPI can be used to configure the stepper motor driver when using the STEP/DIR interface, but it can also be used to control the driver. UART can be used to configure and control the stepper motor driver. UART also allows for the control of multiple drivers with one wire. RS485 is almost the same as UART except that it uses two wires to have a differential signal.

For this project, the RS485 protocol has been chosen, since it requires less wires than SPI, but it has a slower data rate which is not a concern for this application. UART could have been chosen over RS485, as it is the exact same signal and requires one less wire than RS485. UART only requires two wires, the signal wire and a ground wire. RS485 requires three wires, which are the two differential signal wires and a ground wire.

10.5 Component selection

VMN |

NTD3055L104T4G has been chosen as the MOSFET used in the external H bridge that is required for the TMC5160 to operate. The NTD3055L104T4G was chosen because it meets the requirements specified in the TMC5160 datasheet. This MOSFET can conduct 10A $100°C$ continuously and 12A $25°C$. This MOSFET also has a low miller charge of $4nC$ which is used to select the gate resistor and choose the drive strength in software. With the specified miller charge, the TMC5160 datasheet recommends that the drive strength be set to 0 and the gate resistor be $\leq 15\Omega$. The switching characteristics of this MOSFET are slower than the AO4882 used for the TMC5160-BOB board and the BSC072N08NS5 used for the TMC5160-EVAL board. This can be compensated for in software by tuning changing two parameters, called the parameters for break-before-make time parameters.

EBQA-04-C-C is a terminal block connector that was chosen to connect the stepper motors to the driver. This is because it is a standard connector with a 5.08mm pitch. This connector is rated for up to 15A which gives a 1.5 safery factor with a load of 10A, which is the most the TMC5160 can handle. This is the minimum safety factor for this stepper motor driver. The stepper motor with the highest current rating is the NEMA23 motor at 2.8A, this gives a 5.35 safety factor.

TBP02R1-381-06B is a terminal block with a pitch of 3.81mm and is used for connecting the encoder. It is a vertical 6 position connector that mates with a TBP02P1-381-06BE, it was chosen since it supports the wire gauge of the rotary encoders.

TLV9101IDBVR is a general purpose operational amplifier. It has a bandwidth of 1.1MHz, this is needed since the maximum output frequency of the rotary encoder is 400kHz, so it is well within the bandwidth of the operational amplifier. The TLV9101IDBVR works on a 16V single supply, or with dual supply +/- 8V. It is used with a 5V single supply. This operational amplifier was compared with other operational amplifiers from Texas Instruments and Analog Devices in SPICE, and had the best results. It had no overshoot and had a fast enough response to work on the absolute maximum frequency the incremental rotary encoders can output. The

MAX40079 was one of the other alternatives that were tested in SPICE, but it had problems with overshoot if the supply voltage was 5.5V before it would go to the correct value.

Sense resistors are used to measure the current passing through the motor coils. Since the current rating for each motor is different, multiple resistor values are needed. For the resistor values and power ratings required for this project, the 2512 imperial footprint has been chosen. The resistor values that are used are $75m\Omega$, $120m\Omega$, $150m\Omega$, and $220m\Omega$ which are taken from the table under "Selecting Sense Resistors" in the TMC5160 datasheet .

Voltage regulator was considered to supply the encoder and the operational amplifiers, but is not required since the TMC5160's internal 5V voltage regulator can supply 30mA. The operational amplifiers and the incremental rotary encoders requires a total of 20.25mA, so it should be sufficient. Adding a 5V voltage regulator could be something to consider in a future revision.

10.6 MOSFET

VMN |

Metal-oxide-semiconductor field-effect transistor (MOSFET) is used to amplify or switch signals. The TMC5160 controls the motor using MOSFETs instead of doing it directly, this is to allow for higher current and voltage.

The TMC5160 uses a chopper mode to regulate the current flow through the stepper motor. This is done by switching the MOSFETs on and off fast, and with different duty cycles to regulate the current going throught the coils.

10.7 Operational Amplifier circuit

VMN |

10.7.1 Simulation of OPAMP

VMN |

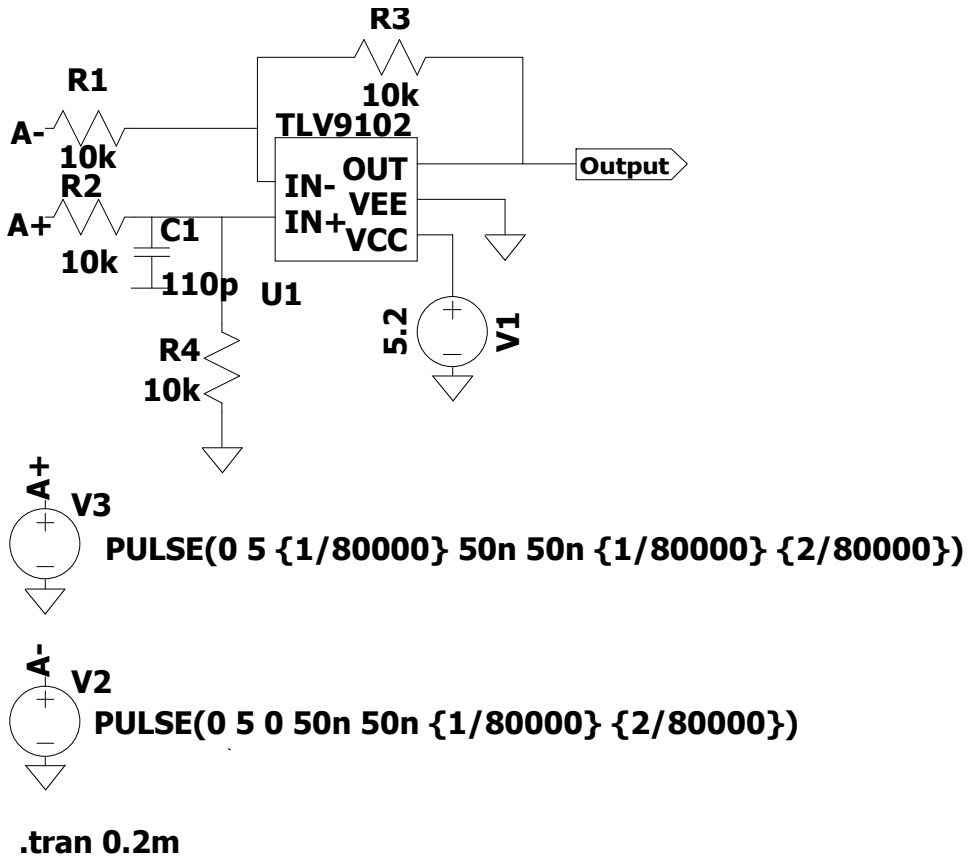


Figure 73: Operational Amplifier SPICE test circuit

The operational amplifier that was chosen was tested with spice simulation. With test signals at 400kHz which is the maximum frequency that the magnetic incremental rotary encoders can output. With the circuit shown in Fig. 73 it can handle the signal, but it is not perfect. This is not a problem, since the motor will never reach speeds that will cause it to output a signal of 400 kHz. Since the motor will only spin at 1000 RPM or less. 1000 RPM can be used to determine a more realistic frequency. The rotary encoder outputs 1000 pulses per revolution, which at 1000 RPM would give a frequency of $\frac{1000RPM}{60s} \cdot 1000PPR = 16666.67Hz$, but to have some headroom it can be increased to 40kHz which is $\frac{1}{10}$ of the maximum output frequency. This gives us the results in Fig. 74, and we can see that it does not overshoot and that it can handle the frequency.

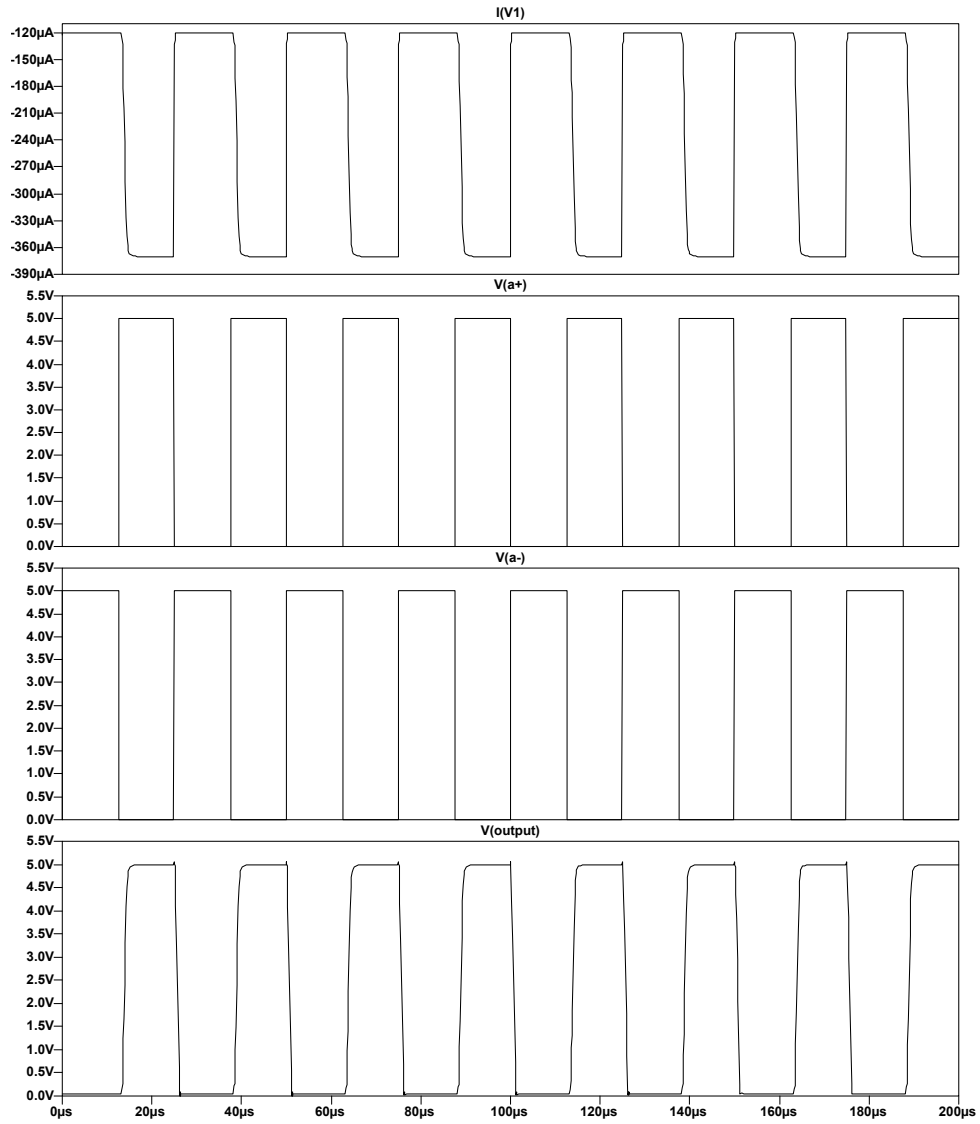


Figure 74: OPAMP result with test signals at 40kHz

10.8 PCB Design

VMN |

A Printed Circuit Board (PCB) consists of copper and dielectric layers, the most common dielectric material is Fire-Retardant 4 (FR4). PCBs are used to connect components together. The copper layers are etched to provide the traces that connect the components together. vias are used to connect traces or planes together that are located on different layers.

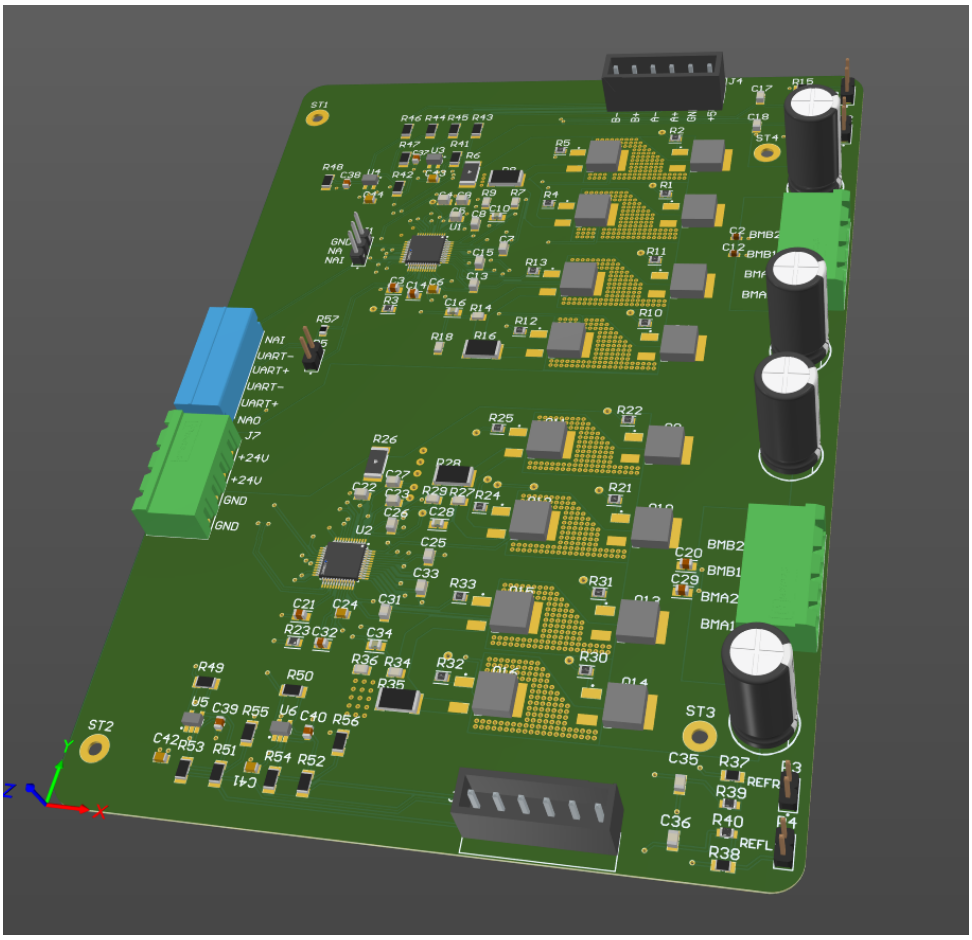


Figure 75: 3D view of the PCB

10.8.1 Schematic

VMN |

A schematic diagram is needed to create a PCB, this is used to know all of the components that are in the circuit and how they are connected. The schematic is also used to create the Bill of Materials (BOM). The program that has been used to create the schematic and pcb layout is Altium Designer, this is because it has a lot of features and is free for students.

The current schematic for the stepper motor drivers can be seen in Fig. E in Appendix E. This is based on the datasheet[30] and the evaluation board TMC5160-BOB[31]. Originally, the MOSFETs used was the AO4882 taken from the TMC5160-BOB, but it was swapped out with the NTD3055L104 in the final design. This was done so the current limit could be increased in the future. This MOSFET also uses a DPAK footprint so it could be replaced by other DPAK MOSFETs. The NTD3055L104 has a drain current of 10A at 100°C, compared to the drain current of the AO4882 of 3A at 70°C.

10.8.2 BOM

Figure 76: BOM PCB

Index	DigiKey Part #	Manufacturer Part Number	Manufacturer	Description	Quantity	Backorder	Unit Price	Extended Price
1	490-3291-1-ND	GRM188R71E474KA12D	Murata Electronics	CAP CER 0.47UF 25V X7R 0603	2	0	0.787	NOK 1.57
2	311-1140-1-ND	CC0805KRX7R9BB104	YAGEO	CAP CER 0.1UF 50V X7R 0805	10	0	0.28	NOK 2.80
3	399-C0805C224K1RACTUCT-ND	C0805C224K1RACTU	KEMET	CAP CER 0.22UF 100V X7R 0805	10	0	1.792	NOK 17.92
4	NTD3055L104T4GOSCT-ND	NTD3055L104T4G	onsemi	MOSFET N-CH 60V 12A DPAK	16	0	6.9008	NOK 110.41
5	A129727CT-ND	CRGCQ0805F15R	TE Connectivity Passive Product	RES 15 OHM 1% 1/8W 0805	16	0	0.1906	NOK 3.05
6	541-1.00AAFCT-ND	CRCW25121R00FKEG	Vishay Dale	RES SMD 1 OHM 1% 1W 2512	2	0	1.865	NOK 3.73
7	175-TMC5160A-TA-TCT-ND	TMC5160A-TA-T	Analog Devices Inc./Maxim Integrated	IC MTR DRVR BIPOLAR 8-60V 48TQFP	2	0	66.48	NOK 132.96
8	399-15437-1-ND	C0805C111J1GACTU	KEMET	CAP CER 110PF 100V C0G/NP0 0805	4	0	1.792	NOK 7.17
9	2057-EBQA-04-C-C-ND	EBQA-04-C-C	Adam Tech	TERM BLOCK HDR 4POS 5MM	2	0	4.621	NOK 9.24
10	311-10.0KFRCT-ND	RC1206FR-0710KL	YAGEO	RES 10K OHM 1% 1/4W 1206	16	0	0.186	NOK 2.98
11	296-TLV9101IDBVRCT-ND	TLV9101IDBVR	Texas Instruments	IC OPAMP GP 1 CIRCUIT SOT23-5	4	0	4.434	NOK 17.74
12	478-KGM21NR71H102KTCT-ND	KGM21NR71H102KT	KYOCERA AVX	CAP CER 1000PF 50V X7R 0805	4	0	0.518	NOK 2.07
13	738-CSNL2512FT75L0CT-ND	CSNL2512FT75L0	Stackpole Electronics Inc	RES SHUNT	4	0	4.755	NOK 19.02
14	541-10.0KCCT-ND	CRCW080510K0FKEA	Vishay Dale	RES SMD 10K OHM 1% 1/8W 0805	10	0	0.435	NOK 4.35
15	PCE5211-ND	EEU-FS1K221	Panasonic Electronic Components	CAP ALUM 220UF 20% 80V RADIAL TH	4	0	8.982	NOK 35.93
16	490-3326-1-ND	GRM21BR72A474KA73L	Murata Electronics	CAP CER 0.47UF 100V X7R 0805	4	0	2.683	NOK 10.73
17	13-RT0805FRE0722KLCT-ND	RT0805FRE0722KL	YAGEO	RES SMD 22K OHM 1% 1/8W 0805	10	0	0.228	NOK 2.28
18	445-1420-1-ND	C2012X7R1C225K125AB	TDK Corporation	CAP CER 2.2UF 16V X7R 0805	4	0	1.057	NOK 4.23
19	399-C0805C222K5RACTUCT-ND	C0805C222K5RACTU	KEMET	CAP CER 2200PF 50V X7R 0805	10	0	0.394	NOK 3.94
20	478-KGM21AR72A223KUCT-ND	KGM21AR72A223KU	KYOCERA AVX	CAP CER 0.022UF 100V X7R 0805	2	0	0.87	NOK 1.74
21	S1012EC-03-ND	PREC003SAAN-RC	Sullins Connector Solutions	CONN HEADER VERT 3POS 2.54MM	1	0	3.21	NOK 3.21
22	102-6484-ND	TBP02R1-381-06BE	Same Sky (Formerly CUI Devices)	TERM BLOCK HDR 6POS 3.81MM	1	0	6.84	NOK 6.84
23	541-2.2TACT-ND	CRCW08052R20JNEAHP	Vishay Dale	RES SMD 2.2 OHM 5% 1/2W 0805	2	0	1.285	NOK 2.57
24	P47.0CCT-ND	ERJ-6ENF47R0V	Panasonic Electronic Components	RES SMD 47 OHM 1% 1/8W 0805	10	0	0.539	NOK 5.39
25	A126336CT-ND	CRG0805F120R	TE Connectivity Passive Product	RES SMD 120 OHM 1% 1/8W 0805	2	0	0.373	NOK 0.75
26	102-6408-ND	TBP01R1-508-04BE	Same Sky (Formerly CUI Devices)	TERM BLOCK HDR 4POS 5.08MM	1	0	4.97	NOK 4.97
27	408-KRL3264E-C-R220-F-T1CT-ND	KRL3264E-C-R220-F-T1	Susumu	CURRENT SENSE RESISTORS, 2512, 2	1	0	7.87	NOK 7.87
28	283-MFHA2512R1200FECT-ND	MFHA2512R1200FE	Eaton - Electronics Division	RES 0.12 OHM 1% 2W 2512	2	0	3.118	NOK 6.24
29	13-PE2512FKF7W0R15LCT-ND	PE2512FKF7W0R15L	YAGEO	RES 0.15 OHM 1% 2W 2512	2	0	2.818	NOK 5.64
30	10129379-902001BLF-ND	10129379-902001BLF	Amphenol ICC (FCI)	CONN HEADER R/A 2POS 2.54MM	2	0	0.477	NOK 0.95
31	102-6472-ND	TBP02P1-381-06BE	Same Sky (Formerly CUI Devices)	TERM BLOCK PLUG 6POS 3.81MM	1	0	13.99	NOK 13.99
32	2057-EBHA-04-C-ND	EBHA-04-C	Adam Tech	TERM BLOCK PLUG 4POS 5MM	2	0	11.085	NOK 22.17
33	2057-EBHA-04-D-ND	EBHA-04-D	Adam Tech	TERM BLOCK PLUG 4POS 5.08MM	1	0	15.33	NOK 15.33
					Module price w/o VAT			NOK 489.77
					Module price w/ VAT			NOK 612.21
					Total (3 modules) w/o VAT			NOK 1,959.08
					Total (3 modules) w/ VAT			NOK 2,448.85

Fig. 76 is a list of all the components for the PCB and the connector that are used. This list includes the total component price for one board and three board. The exported BOM from Altium Designer can be viewed in Appendix E.E

10.8.3 PCB layout

VMN |

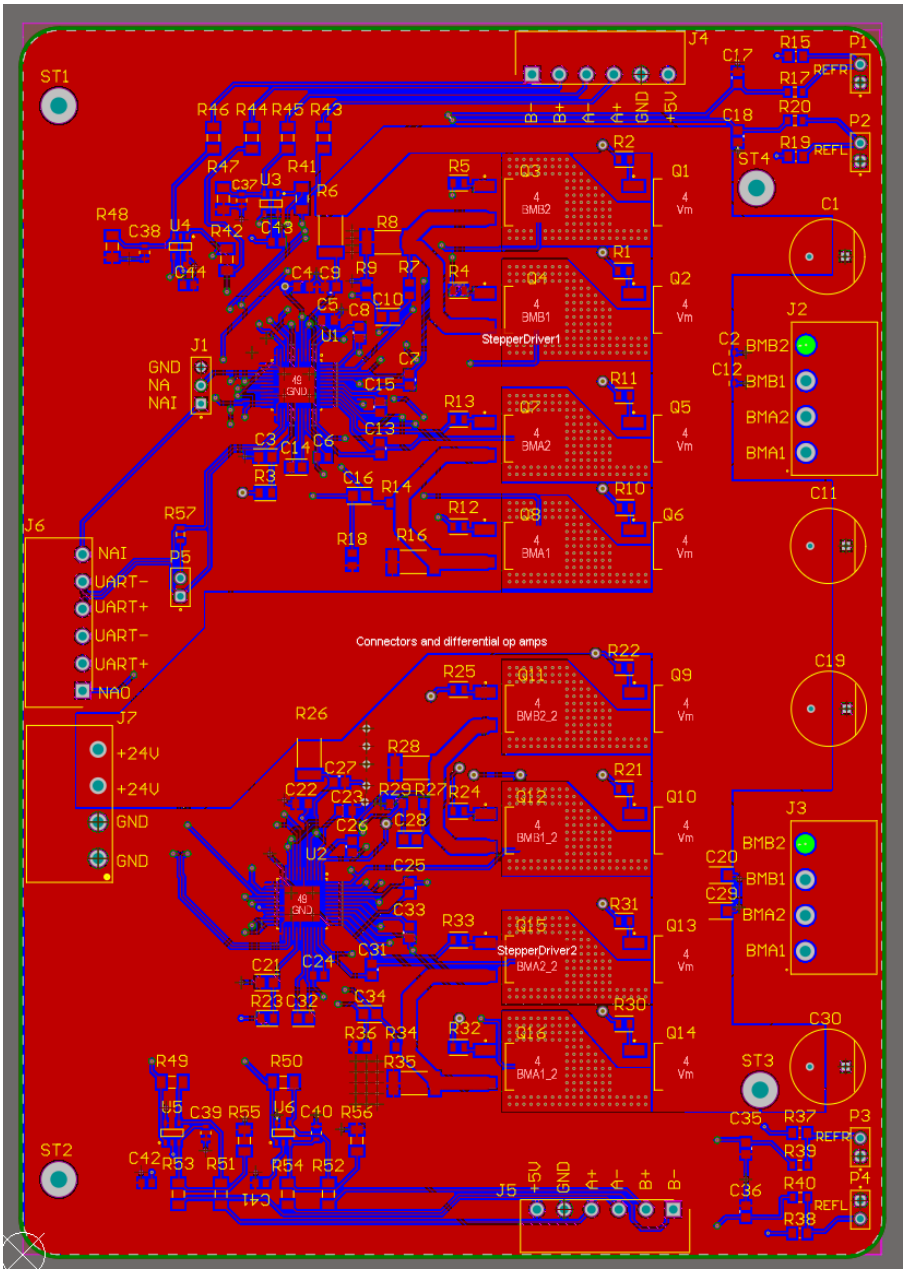


Figure 77: PCB layer view with all layers visable

In Fig. 77, we can see the PCB layers of the PCB with designators and outlines of all the components. I assigned the schematic sheets to rooms to restrain where the components can be placed and to help move the components to the correct room. Rooms also has other functionalities like copying the layout of another room if the components are the same, this function

was not used since I used a flat design for the schematic instead of a hierarchical design. To connect the MOSFETs together and to the connector, polygon pours have been used instead of traces to help with heat dissipation. The top and bottom layers are connected with a lot of small vias to help with heat dissipation.

The decoupling capacitors that are used for the IC are placed close to the pins that require them. This is done for EMC concerns.

10.8.4 Electromagnetic Compatibility

VMN |

Electromagnetic Compatibility (EMC) is one of the most important aspects to consider when designing a PCB and circuit. EMC concerns the generation of Electromagnetic Interference (EMI) and how EMI affects the circuit. This affects the functionality of the circuit and/or whether it will affect other circuits/systems.

One mitigation for EMI is decoupling capacitors close to an IC's pins. This is done to keep the pin voltage stable and avoid EMI from external and/or internal sources. Decoupling capacitors are also used to avoid creating EMI.

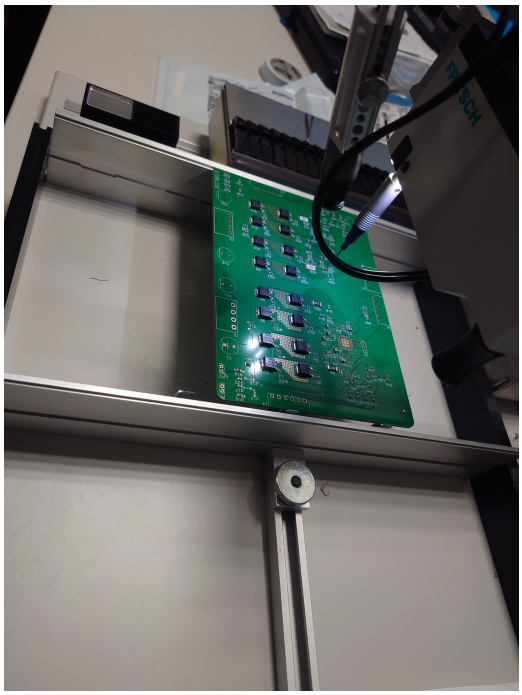
The layer stackup of a PCB is also important to consider since it can help mitigate EMI. The layer stackup used for this PCB is signal+pwr/gnd/gnd/signal+pwr. The outer layers are used for signals and power, while the inner layers are used for ground. This is to have ground close to the signal and also to be able to use vias where ground is needed. Another layout that could have been used is signal/gnd/pwr/signal. The internal ground layers are used to prevent interference between the top and bottom layers.

The main concern for this PCB and EMC is the switching of the MOSFETs, and how much EMI it would create. EMC is a very complex subject and can be difficult to design around.

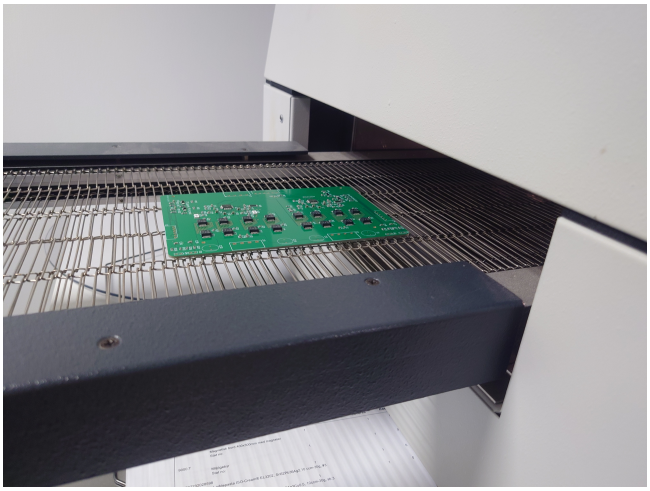
10.8.5 PCB Assembly

VMN |

The first revision that was ordered were assembled at the university using the pick-and-place machine, and reflow oven as seen in Fig. 78



(a) Pick-and-Place machine



(b) Reflow Oven

Figure 78: PCB assembly

After being soldered, the PCB was tested by connecting it to the power, but it ended up being shorted. This was caused by a mishap when generating the drill file. Some of the silkscreen was also missing. To remedy this, some troublesome areas were drilled and cut to remove direct shorts from the power rails to the ground. This was a success, but there was still something wrong with it. New PCBs were ordered the next day with the correct files, but will not arrive before the deadline for this report. It may arrive before the USN Expo or the day after. At least it will be assembled and tested before the final presentation.

10.9 Microcontroller & Computer

VMN |

To control the stepper motors we are using an Arduino UNO R4 WiFi provided to us by Hydroplant Technologies. This has just enough pins for the stepper motor drivers, the servomotor, and the limit switches. With the DM320T and DM332T the Arduino does not have enough pins for the rotary encoders, to add this functionality Arduino UNO R4 WiFi could be swapped out for another microcontroller or use port expanders to add additional I/O ports.

10.10 Electrical Signals & Communication

VMN |

10.10.1 DM320T/DM332T Stepper Motor Driver

VMN |

Direction & Pulse

Direction and pulse are the two most common ways to control a stepper motor driver. The direction signal has two states, high or low. This will simply choose between spinning clockwise or counterclockwise. The pulse signal also has two states, high or low. When the pulse goes high, it signals to the stepper motor driver to go one step. The logic level for this setup is +5V.

Rotary Encoder Signals

The rotary encoder has 6 signals, which are A+, A-, B+, B-, Z+, and Z-. The rotary encoder has two signals for each phase, as it is a differential signal. This is done to minimize noise. For this application, the only signals that will be used are A+ and B+. This could be changed in the future by implementing an differential operational amplifier before the input

Enable Pin

The Enable pin is used to enable the stepper motor driver when the pin is set to high (+5V), and disable it when set to low (0V/GND).

10.10.2 TMC5160 Stepper Motor Driver

VMN |

UART/RS485

UART is used to control all of the stepper motors, with the configuration of the TMC5160 it is used in a differential signal mode. This is almost the same as the RS485 standard. The TMC5160 operates on half-duplex so it will send data or receive data, but it cannot do both at the same time. Most of the configuration is done by setting parameters in software, except for the maximum current limit, which is set on a hardware level. The UART interface is also used to send movement controls and receive data from the TMC5160. It is a two-wire interface and addresses are used to control multiple drivers with the same two wires.

NAI / NAO Next Address In and Next Address Out are used to connect the drivers together and increment the address of the drivers that have been connected.

11 Software

This section outlines the various software components required for the whole system to work as required. Please refer to Appendix F for detailed Doxygen-generated code documentation.

11.1 Leafy Automation Central

JCDH | *SME*

The Leafy Automation Central provides a HTTP server which hosts the Human Machine Interface (HMI), Robot Operating System 2 (ROS2), AI models, ML algorithms and other utilities critical to system functionality.

Leafy Automation Central is an essential piece of software. It is where a lot of the heavy lifting and data processing happens. Microcontrollers are often very limited in available resources (often only kilobytes of ram), which is why a piece of software running on dedicated hardware is necessary. Leafy Automation Central is intended to run on the Raspberry Pi 5, although there is no functional limitation on which type of device it can run on (provided sufficient processing power).

In order to facilitate communication between all devices of the Leafy Automation system, a local network must be setup. Initial testing began using a hotspot on a computer running Microsoft Windows 10, but the intention will be to host a hotspot on the Raspberry Pi 5, thus removing the need for a standalone networking device. As the Leafy Automation system will be deployed to many different types of networks, testing on a wide area of possible network configurations gives us valuable evidence to verify system general stability.

11.1.1 Areas of responsibility

JCDH | *SME*

The Leafy Automation Central is divided up into the following main areas of responsibility:

Table 5: Leafy Automation Central - Areas of responsibility

System	**Purpose**
AI and CV processing	Identifies and classifies the lettuce using various AI models and algorithms.
Robotic conversions	Converts the data points compiled by the *AI recognition* to concrete real-world coordinates usable by the robotics (grip point).
Web server	Allows the Leafy Automation Core to communicate with the Leafy Automation Central
Human Machine Interface (HMI)	A Graphical User Interface (GUI) which let's you interface with the system, give commands and see information

Please refer to Appendix G.14 for the HTTP based API implementation of Central.

11.2 HMI

JCDH | *SME*

The Human Machine Interface (HMI) is the part of our system which interfaces with a human. A Graphical User Interface (GUI) is presented to the user where they can view useful information like system status and camera feeds. This interface also allows for initiating / stopping system functions. [1]

Although our system is autonomous, human oversight is still required, especially for initial testing and for taking spot checks or quality control. The HMI is accessible from the Leafy Automation Central IP address. Figure 79 shows how the user interfaces with the HMI, and in turn how the HMI interfaces with the rest of the system.

[1] The HMI is part of the Leafy Automation Central codebase, but contained in its own module.

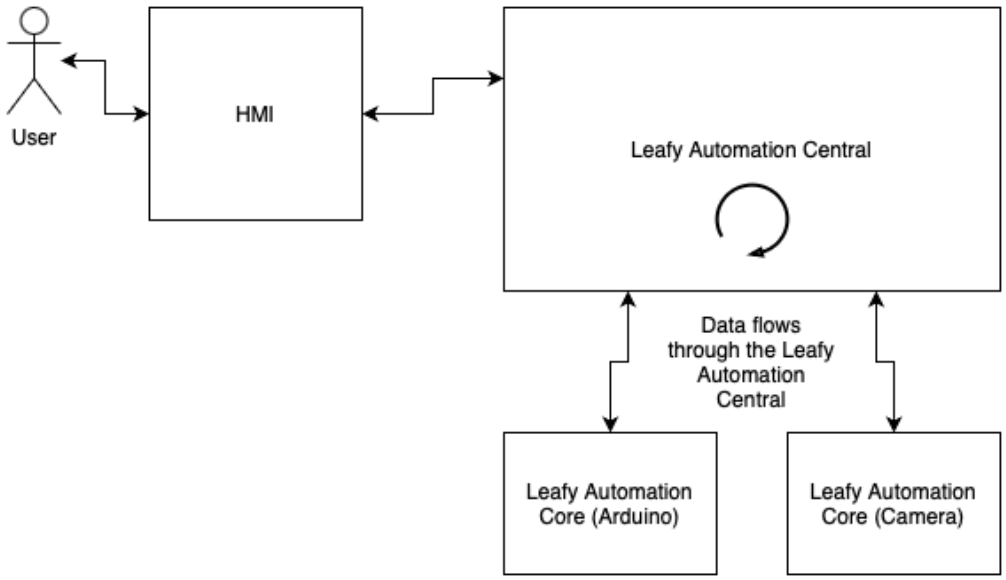


Figure 79: HMI diagram

The following are the aims for the HMI:

- User friendly
- Snappy
- Accessible

11.2.1 Dashboard

JCDH | *SME*

Figure 80 shows the HMI dashboard in action. It includes a camera feed, system modules state, controls for starting / stopping the system, a log panel and login information. Please refer to Appendix G.11 for an earlier iteration the dashboard.

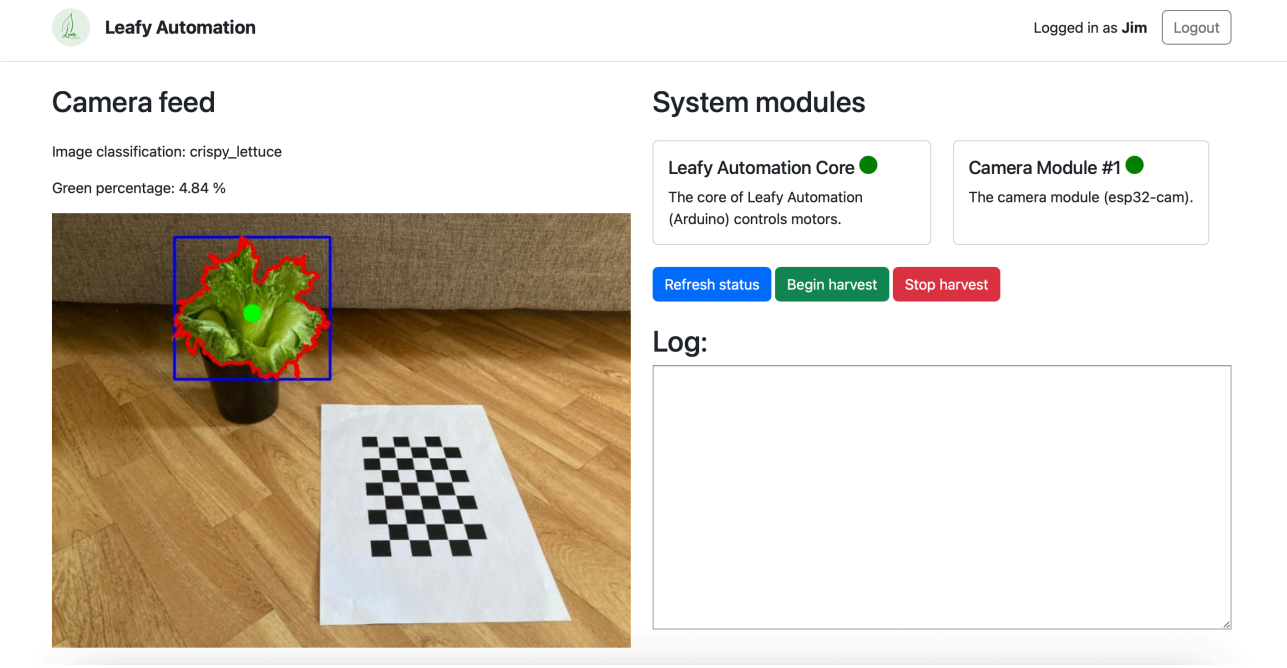


Figure 80: HMI dashboard

Underlying HMI architecture

The HMI itself is written in HTML, CSS and JS, utilizing Bootstrap for the layout and main styling, which is a common web-app configuration. The reason for this configuration is because it makes quick prototyping / development easy. On the backend Python Flask is used.

JQuery is used to connect the pieces of the webapp. The HMI utilizes a Model View Controller (MVC) architecture for optimal code organization [32], as seen in Figure 81.

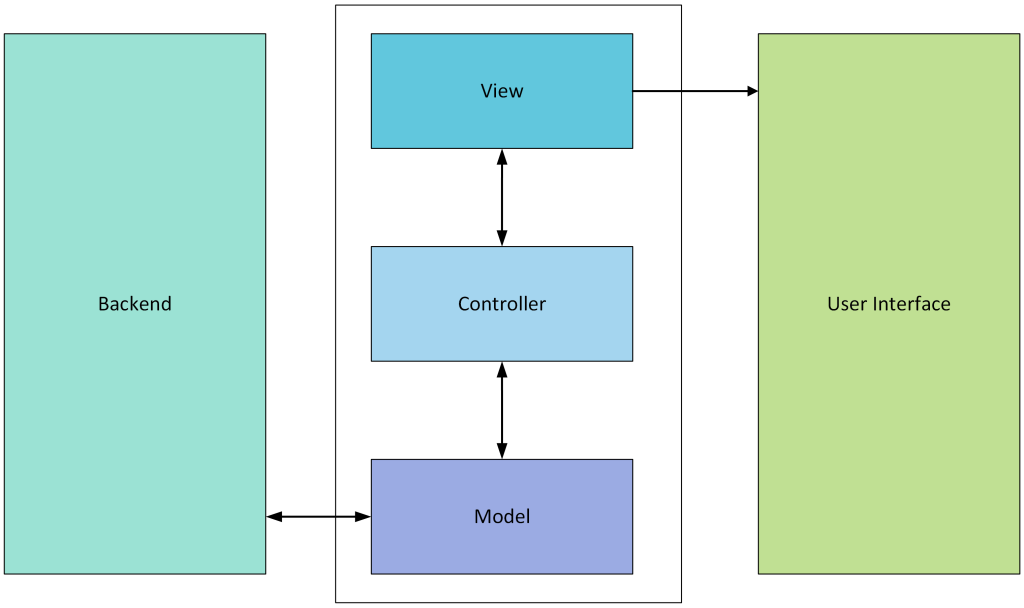


Figure 81: MVC - control flow

Camera feed

The initial iteration of the camera feed started with a HTTP-based endpoint to stream the camera feed. This was later extended with a WebSocket implementation, which enabled low latency communication. Please refer to Appendix G.10 for more info on earlier iterations. [2]

11.2.2 Login, authentication and session handling

JCDH | *SME*

The user is greeted with a login page before gaining access to the Dashboard as seen in Figure 82.

[2]Research was done on using a ROS2 compatible JavaScript library for complete integration with the ROS2 ecosystem, but this work was set aside due to time constraints [33].

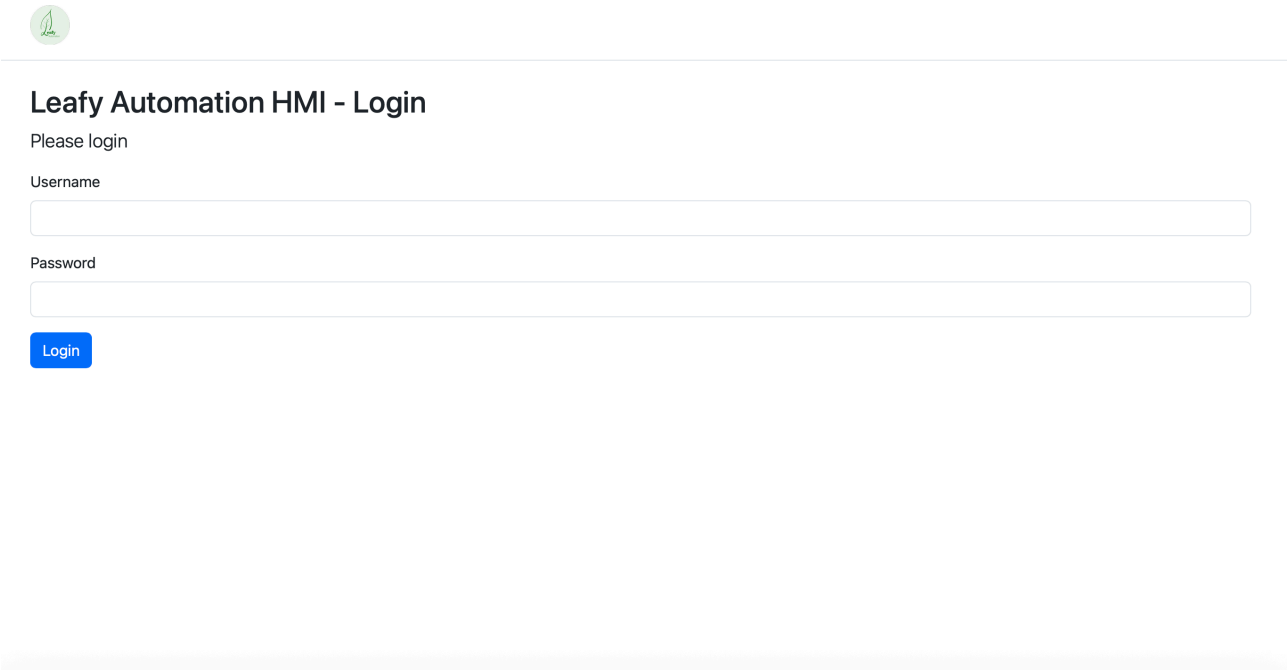


Figure 82: HMI login page

Authentication and password hashing

Authentication and password hashing is implemented using functionality from Werkzeug [34]. The reason we hash passwords is that in case of a data leak, or hacking attempt, the users password will not actually be known as there will never be stored a clear text version of the users password, and only a hashed version of the password which for all practical purposes will be impossible to decipher. The hashed password which is stored in the users table (Table 7), is stored in the scrypt format, which you can read more about in Appendix G.10.4.

Session cookie

A session cookie implements a way for the Human Machine Interface (HMI) to remember the user after they have logged in, so that the next time they visit the HMI they do not have to login again.

11.3 Database

JCDH | *SME*

Every system of moderate complexity needs a database to organize persistent data in a structured way. Leafy Automation Central keeps this data in a database which is called **SQLite**. A small, fast and self-contained SQL database engine. [3]

Currently, the main responsibility of the database is to store which users have access to the system. The user authentication and authorization is implemented in the HMI.

Here is a **list of the primary data types in SQLite** (the list is short, which shows the simplicity of database, as opposed to other larger database software like MySQL) [36].

- NULL - No value.
- INTEGER - Signed integer (only positive values).
- REAL - Floating point value (decimal number).
- TEXT - Text string (utf-8).
- BLOB - Binary data, stored exactly "as is".

11.3.1 The need for a database

JCDH | *SME*

Data needs to be persisted between reboots of the server and, whereas storing data in variables would be fine for some use cases, the data would be lost upon the next restart of the server. [4]

Why we use SQLite over other databases

We believe that using SQLite is the correct approach for our system for the following reasons:

[3] SQLite is actually the most used database in the world, being used in everything from mobile apps to embedded devices. According to the SQLite website there are over 1 trillion active SQLite database instances in use today [35].

[4] While visiting a dry-cleaner which made use of AI models and robotics for folding laundry, they emphasized the importance of keeping historic data from the folding process in order to fine tune the AI models. In order to achieve this a database had to be deployed to facilitate the storage.

1. SQLite implements a much smaller subset of functionality, which is similar to MySQL, which reduces complexity.

2. Upgrading to MySQL would require minimal refactoring of existing SQL queries due to their similarities in syntax.

11.3.2 Database tables in use

JCDH | *SME*

Figure 83 shows an overview of database tables and their relationships. Please refer to Appendix G.13 for in-progress work on the database.

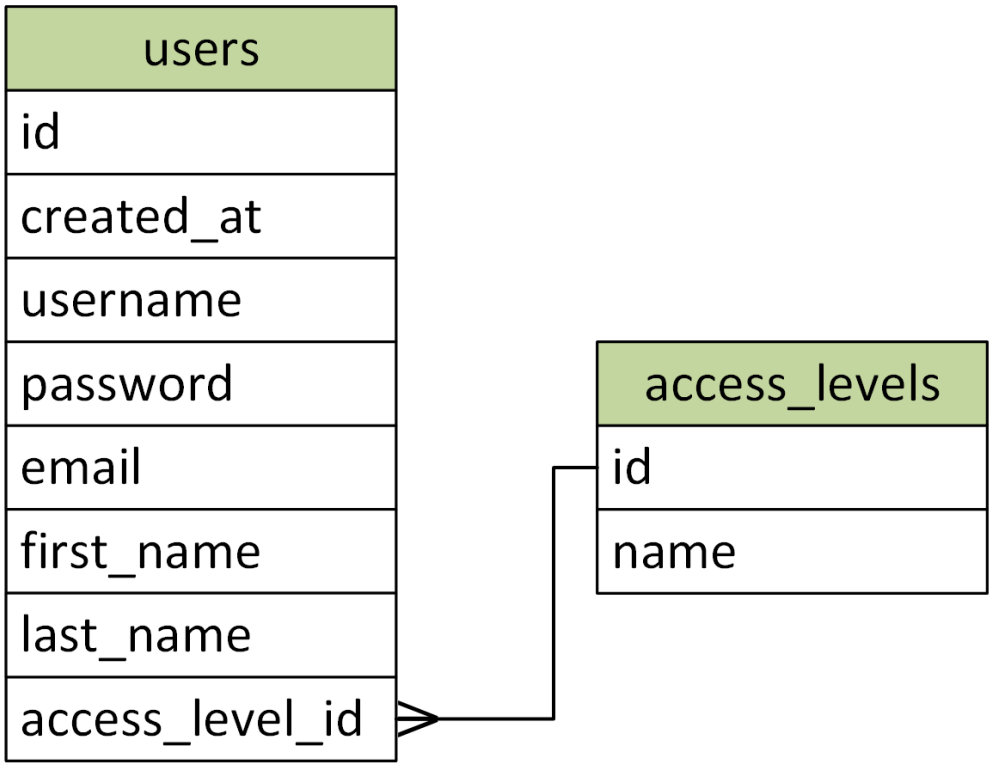


Figure 83: Database overview diagram

Note: Arrows indicate foreign key relationships, and the crows foot indicate a one-to-many relationship.

11.3.3 Users and Access levels tables

JCDH | *SME*

In this context **AUTO INCREMENT** means that the column (which in most cases will be *id*) will automatically count +1 for each new row in the table [37], **PRIMARY KEY** uniquely

identifies each row in the table, **CURRENT_TIMESTAMP** stores the current UTC (Co-ordinated Universal Time) time in the format *YYYY-MM-DD HH:MM:SS* and **UNIQUE** ensures that no duplicate values will be used [38].

Users table (users)

Name	Description	Datatype	Metadata
id	The unique id of the user	INTEGER	AUTO INCREMENT, PRIMARY KEY
created_at	The timestamp for when this user was created	TEXT	CURRENT_TIMESTAMP
username	The unique username of the user	TEXT	NOT NULL, UNIQUE
password	The hashed password of the user	TEXT	NOT NULL
email	Users email	TEXT	NOT NULL
first_name	The first name of the user	TEXT	NOT NULL
last_name	The last name of the user	TEXT	NOT NULL
access_level_id	The access level of the user, which is a foreign key to the specific access_level in the access_levels table	INTEGER	FOREIGN KEY to access_levels.id

Table 6: Database: Users table structure

id	created_at	username	password	email	first_name	last_name	access_level_id
1	2025-04-07 11:20:05	admin	[hashed-password]	admin@example.com	John	Green	1
2	2025-04-08 09:39:29	spectator	[hashed-password]	spectator@example.com	Leafy	Green	2

Table 7: Database: Users table example

Access levels table (access_levels)

Name	Description	Datatype	Metadata
id	The unique id of the access level	INTEGER	AUTO INCREMENT, PRIMARY KEY
name	The name of the specific access level	TEXT	NOT NULL, UNIQUE

Table 8: Database: Access levels table structure

Each user gets one of two access levels:

- admin
- spectator

Where *admin* corresponds to a user who has access to both managing and viewing the system, and *spectator* which only has the ability to view system info.

id	name
1	admin
2	spectator

Table 9: Database: Access levels table example

11.3.4 Side notes

JCDH | *SME*

Appendix G.12 outlines additional research and notes on the database.

11.4 Leafy Automation Core

JCDH | *SME*

The Leafy Automation Core has multiple important responsibilities. This is the software which runs on the embedded devices within the Main System. It is the codebase which runs on the ESP32-CAM module and the Arduino (which handles direct signaling between robotics). This codebase is divided up into the two above mentioned parts, and the code to compile is decided based on the target devices.

This codebase communicates directly with the Leafy Automation Central which serves as a *"base of operations"* keeping track of the current state of the overall system and deciding which state the system is in.

Please see Appendix G.7 for more details about Leafy Automation Core (initial HTTP iteration and benchmarking), and Appendix G.17 for earlier research.

11.4.1 Why WiFi was chosen over a wired connection

JCDH | *SME*

Networking is a central piece of the system as a whole. Without networking, the individual components would fail to communicate with each other. It is of course possible to communicate across devices and modules using a wired connection, but we believe connection over network protocols like TCP / UDP as in our case creates a much more modular and easy to work with system. Also, this approach allows for easy integration with other IoT devices in the future. [5] [6]

11.4.2 Code reuse

JCDH | *SME*

Both the esp32-cam and Arduino share the same codebase. This might seem strange at first, but there is a simple reason for it, which is code reuse. Most of the network stack is identical between these devices, therefore it simplifies the codebase greatly by removing the need for duplicate code. Which part of the codebase will be used (ESP32-CAM or Arduino robotics) is decided using compile time switches, as seen in Listing 1.

[5] Protocols like MQTT, HTTP and ROS2 all make use of TCP / UDP as the underlying protocol.

[6] The goal when it came to the first iteration of networking was to get something up and running as soon as possible. Based on earlier experience, and availability of documentation it was decided to just build this iteration on HTTP, which builds on top of the TCP protocol.

```
void setup() {
    #ifdef PLATFORMIO_ENV_UNO_R4_WIFI
        main_base_setup();
    #elif PLATFORMIO_ENV_ESP32CAM
        main_cam_setup();
    #endif
}
```

Listing 1: Leafy Automation Core codebase compile time switches

11.4.3 Real-Time Operating System (RTOS) future proofing

JCDH | *SME*

The code in Leafy Automation Core is organized in a way where there are intermediate abstractions between the system drivers and "user space" code. This allows for a smooth transition in the future if a switch to a RTOS like Zephyr is to be made, our code can be mostly reused.

At this stage of proof-of-concept development, our code is running on an Arduino. Because the Arduino has simple abstractions around the underlying driver, these abstractions lend themselves to be easily ported to other microcontrollers.

The diagrams in Figure 84 outlines how a future transition to an RTOS could look like.

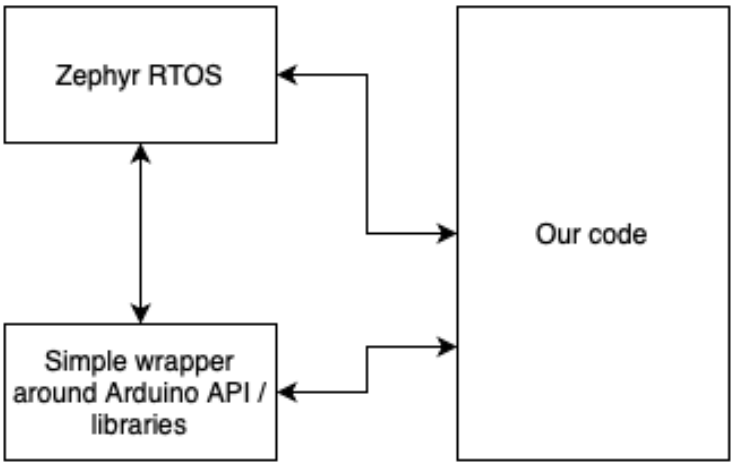


Figure 84: Real-time OS future proofing

11.5 Design and Implementation of Arduino Firmware

EG |

In this chapter, we present the design and implementation of the low-level firmware that runs on the Arduino R4 WiFi (Core) and serves as the real-time controller for the Leafy Automation robotic arm. This firmware is responsible for interfacing with the systems wireless network (as described earlier in this chapter 11), decoding high-level commands issued by the Raspberry Pi 5 (Central) ROS2 Control Layer nodes (as described in the *Layered module-based design*, section 7), driving stepper and servo motors to execute pick-and-place motions, and reporting status updates back to Central.

11.5.1 Hardware components and their identifiers

EG |

Before delving into the firmware logic, it is important to understand the physical hardware that is controlled by Leafy Automation Core, along with the naming conventions that we use. For the purpose of this chapter, Figure 85 shows a simplified naming overview which is further elaborated witha more technical detail in Table 10. More information about the project naming conventions can be seen in section *10.4 Definition robot arm* 8.1.1, and further technical detail about the motors in section *13.4 Stepper motor driver* **??**

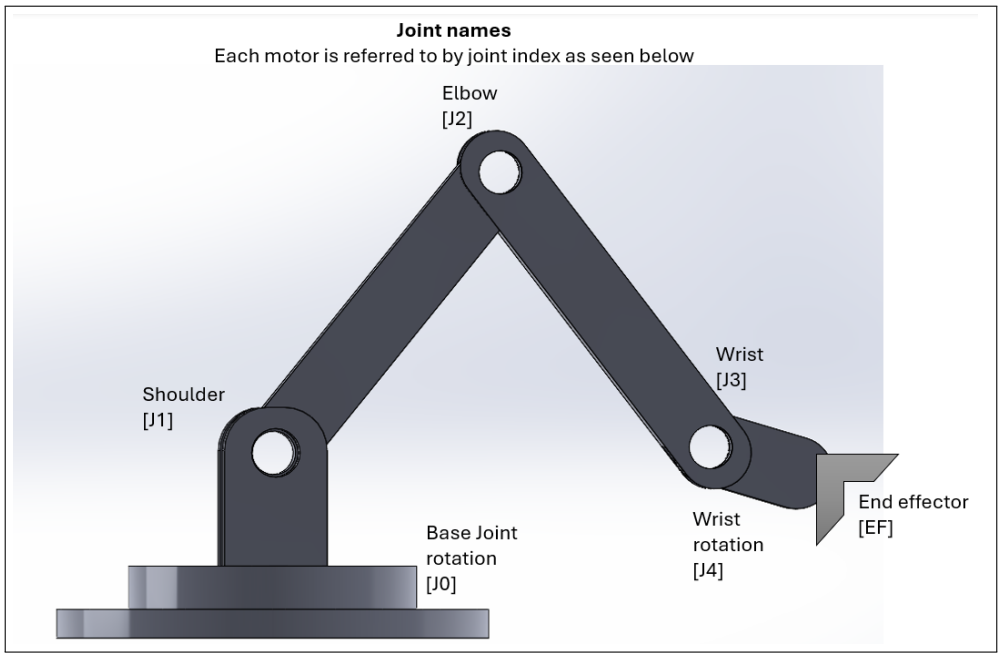


Figure 85: Naming of joint motors, including driver and gear ratio information

Table 10: Naming conventions including key technical detail.

Joint	Joint ID	Motor driver	Gear ratio
Base rotation	J0	DM332T	1/10
Shoulder	J1	DM332T	1/50
Elbow	J2	DM332T	1/50
Wrist	J3	DM320T	1/19
Wrist rotation	J4	DM320T	1/16
End effector (gripper)	EF	Servo	N/A

Each joint is assigned a short identifier (J0 ... J4 for the five stepperdriven axes, and EF for the servo driven end-effector controlling the gripper). These IDs appear in our code (for example moveJoint(J2, <number of steps>) and in code documentation.

A brief mention about gear ratios

Gear ratio, such as for example 1/50 for the motor running J1, means the motor must turn 50 times to produce one full joint revolution. In the Core firmware, this gear ratio setting is used to convert the desired joint speed (in RPM) into the correct motor pulse rate. This means that a higher gear reduction lets us run the motor faster while the joint moves at a slower, more controllable speed. More information about this can be read in **SECTION VETLE?**

Centralised configuration in config.h

Hardwarespecific constants and operating constraints relating to Core are defined centrally in the /include/config.h file. Examples include Arduino pin assignments, microstep settings, gear ratios, and maximum RPM values. Keeping these parameters in one place allows developers to quickly and easily retune the system for different mechanical configurations, without having to modify the control logic directly.

11.5.2 Overview of Library dependencies

EG |

The Core firmware relies on several, well established Arduino libraries. Table 11 gives a short overview, and the sections that follow we will explain how they are used in the various software modules.

Table 11: Arduino library dependencies

Library	Version	Purpose
AccelStepper [39]	v1.64	Non-blocking stepper motor control.
PubSubClient [40]	Hydroplantno fork	MQTT client for publish/subscribe messaging.
WiFiS3 [41]	Arduino SA	Wi-Fi connectivity.
Servo [42]	Arduino SA	PWM-based control for the gripper servo.

11.5.3 The Software build on Leafy Automation Core

EG |

Priority when designing the software for Core has been on building for simplicity to facilitate rapid testing and modularity to make the code easy to understand and modify. These priorities align with the architectural drivers detailed in section *10.1 Project Constraints and Architectural Drivers 7.*

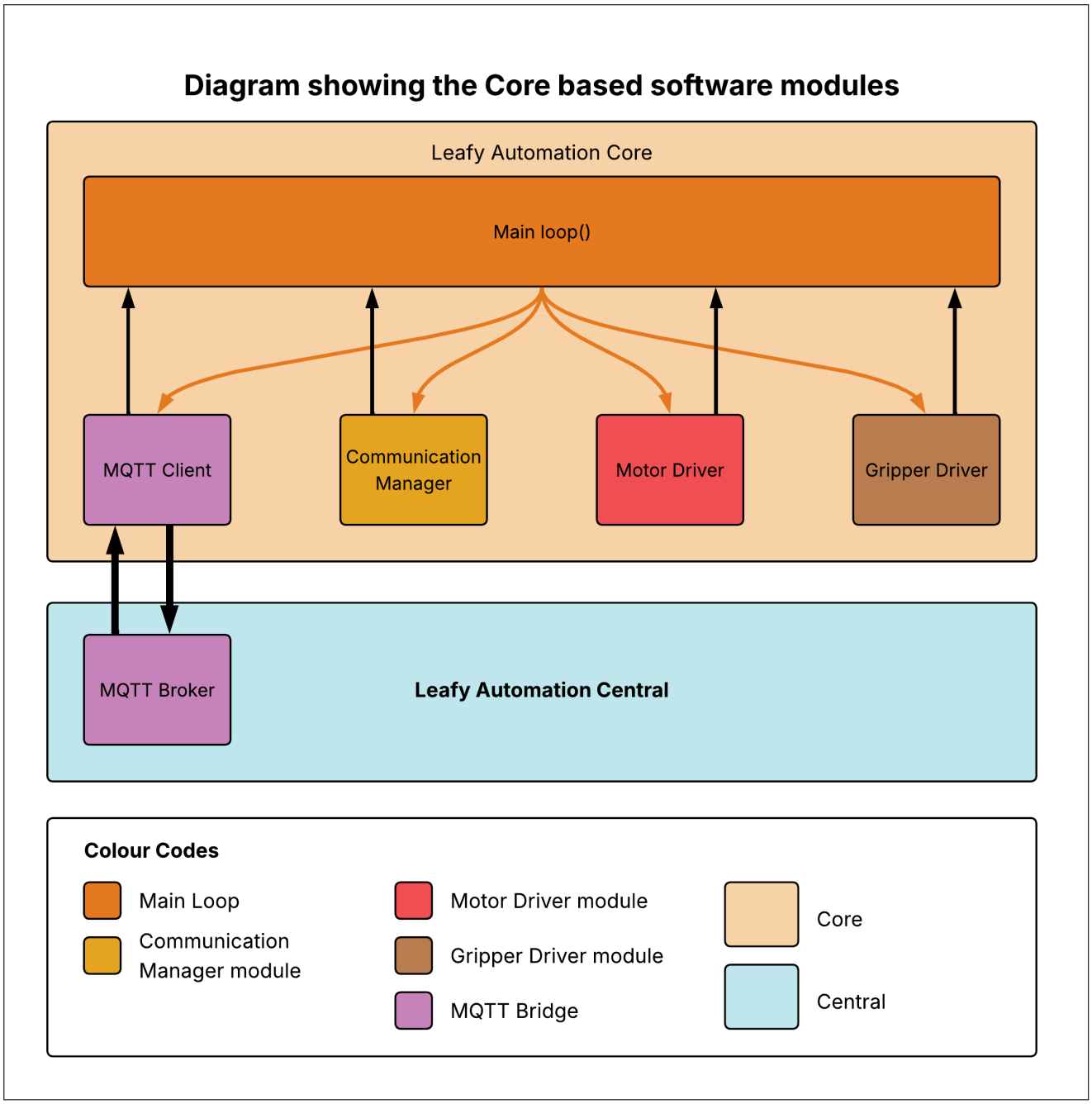


Figure 86: Diagram showing the main software components of Leafy Automation Core, including the MQTT-based connection to Central.

As discussed in Section **??**, communication between Central and Core is achieved by using a MQTT bridge. In this setup, the MQTT broker resides on Central side, with Central publishing high-level commands (for example MOVE or GRIP) to specific topics that Core subscribes to. On the other side, by utilising the PubSubClient library [40], Core publishes status updates and heartbeats back to Central via dedicated topics, ensuring a clear, decoupled exchange of control and feedback.

The Leafy Automation firmware is built based on a cooperative super-loop scheduling pattern. In this pattern, all tasks are placed sequentially inside a single, never-ending main loop, where

each task runs to completion until the next one begins.[43]

This solution was selected because it minimises code and memory overhead, provides predictable timing which is important for real-time control, and because it facilitates rapid development and debugging, thus paving the way for fast proto-typing. This makes it an efficient solution in line with the project constraints as detailed in section 7. [43]

It is important to note that with this setup, all software modules must use only non-blocking operations to keep timing consistent and predictable. In practice, this means advancing motors in small increments (using AccelStepper::run()) and basing state-machine transitions on elapsed-time checks (millis()), rather than any blocking calls like delay().

We shall explore how all the software components fit together, and how each pass of the Main Loop services MQTT traffic, advances the robotic-arm movements, updates the gripper, and sends status updates in a non-blocking cycle.

11.5.4 Main Loop()

EG |

Building on the cooperative super-loop model described above, the Main Loop implements a fixed sequence that drives every aspect of Cores operation. During each iteration, the following five routines see in Table 87 are invoked in the following order:

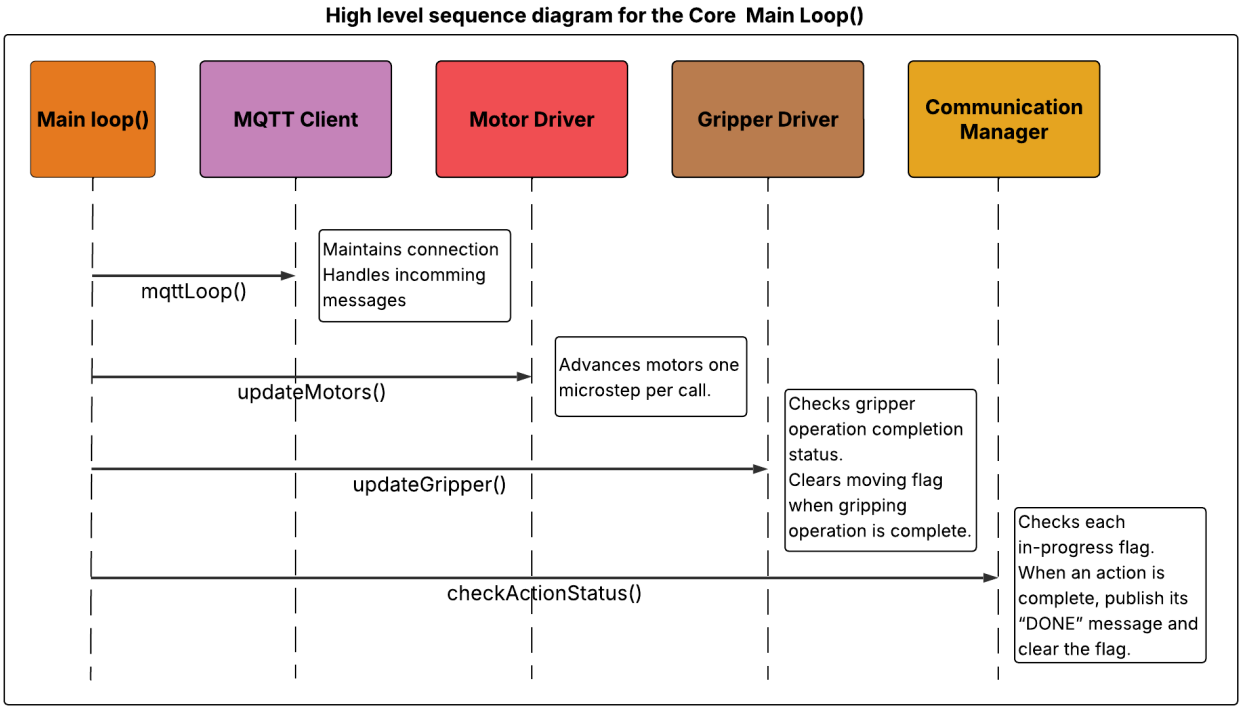


Figure 87: Sequence diagram showing the Main Loop() on Core.

mqttLoop() Maintains the MQTT connection to Centrals broker, performing reconnection attempts with exponential back-off [44] and dispatching any received messages to the Communication Manager.

updateMotors() Cycles through each stepper joint and invokes AccelStepper::run() [39]. When a motors internal timing states that a microstep is due, it executes exactly one step. By invoking run() on every joint each run of the main loop, all axes advance in parallel in a non-blocking fashion.

updateGripper() Executes a timed state-machine check **(see Figure: XXX)**. If the elapsed time since the last moveGripper() call exceeds the configured travel duration (defined in /include/config.h), the internal moving flag is cleared. (This strategy inspired from Arduino documentation [45]).

checkActionStatus() Each loop, it checks if any ongoing action (move, grip, or calibrate) has finished. If so, it sends the corresponding DONE message and resets that actions flag.

sendHeartbeat() Publishes a periodic alive signal to inform Central that Core remains operational.

11.5.5 MQTT Client Module

EG |

The objective of this module is to bridge high-level Central commands and the low level Core motor control together, using MQTT. Thereby, minimising coupling while ensuring reliable message delivery. By isolating all networking logic in this module, the higher-level control code remains agnostic of transport details. [40]

Key interfaces:

initMQTT() Configures PubSubClient with the broker address and callback function, then attempts an initial connection and subscribes to the control topics (motion, gripper, calibrate).

mqttLoop() Periodically invoked to sustain the network link. Unsuccessful connection attempts employ an exponential back-off strategy (2 s, 4 s, , 60 s) to prevent network congestion [44].

publishStatus(const char*, const String) Publishes status strings (e.g. MOVE DONE) to designated topics, logging any transmission failures to the serial console.

setMessageHandler(void (*)(const String)) Tells the MQTT Client which function to run whenever a new message arrives. In our code, we pass handleIncomingCommand, so every time

Core receives an MQTT payload, it automatically calls that function with the message text. Inspired by code example [46].

11.5.6 Communication Manager

EG |

The objective of the Command Manager is to interpret the textual commands received over MQTT and translate them into motor instructions, while maintaining reliable message received and instruction complete handshakes. Figure 88 demonstrates how Core processes a MOVE command, based on the Command Manager's functionality. Here, Central publishes a MOVE message to the MQTT broker, which Core receives in mqttLoop(). Cores Communication Manager dispatches the motion, the drivers execute until completion, and checkActionStatus() publishes MOVE DONE back through MQTT to Central.

This decoupling of parsing, dispatch, and status logic promotes modifiability as new commands can be added by extending this module alone, without altering networking or driver code.

Main functions in the Command Manager:

Command parsing with handleIncomingCommand() This function, inspired by code example [46], starts by trimming off white spaces. It then separates the message based on the command keyword (MOVE, GRIP, CALIBRATE), and publishes the raw command to topic status/command_received to acknowledge receipt. Then, the following actions happen, depending on which command has been received:

A MOVE command: Parses five integer positions, invokes moveJoint() for joints J0 J4, and sets movementInProgress = true. A GRIP command: Calls moveGripper(state) and sets gripperInProgress = true.

A GRIP command: On a GRIP message, the function publishes the incoming command, parses a single integer argument (0 or 1), and passes it to moveGripper(state). It then sets gripperInProgress = true so that the next status poll can detect completion.

A CALIBRATE command: When a CALIBRATE command is received, this function publishes the payload, parses the integer argument (0 = cancel, 1 = start), and immediately acknowledges with CALIBRATE RECEIVED. If starting, it calls calibrateAllJoints(), waits for the homing sequence to complete, then publishes CALIBRATION DONE. On cancel, it clears the flag and publishes CALIBRATE CANCELED.

The process is illustrated in Figure 89

Completion monitoring with checkActionStatus() Periodically checks whether each in-progress action has completed its task, and publishes the corresponding DONE message exactly once.

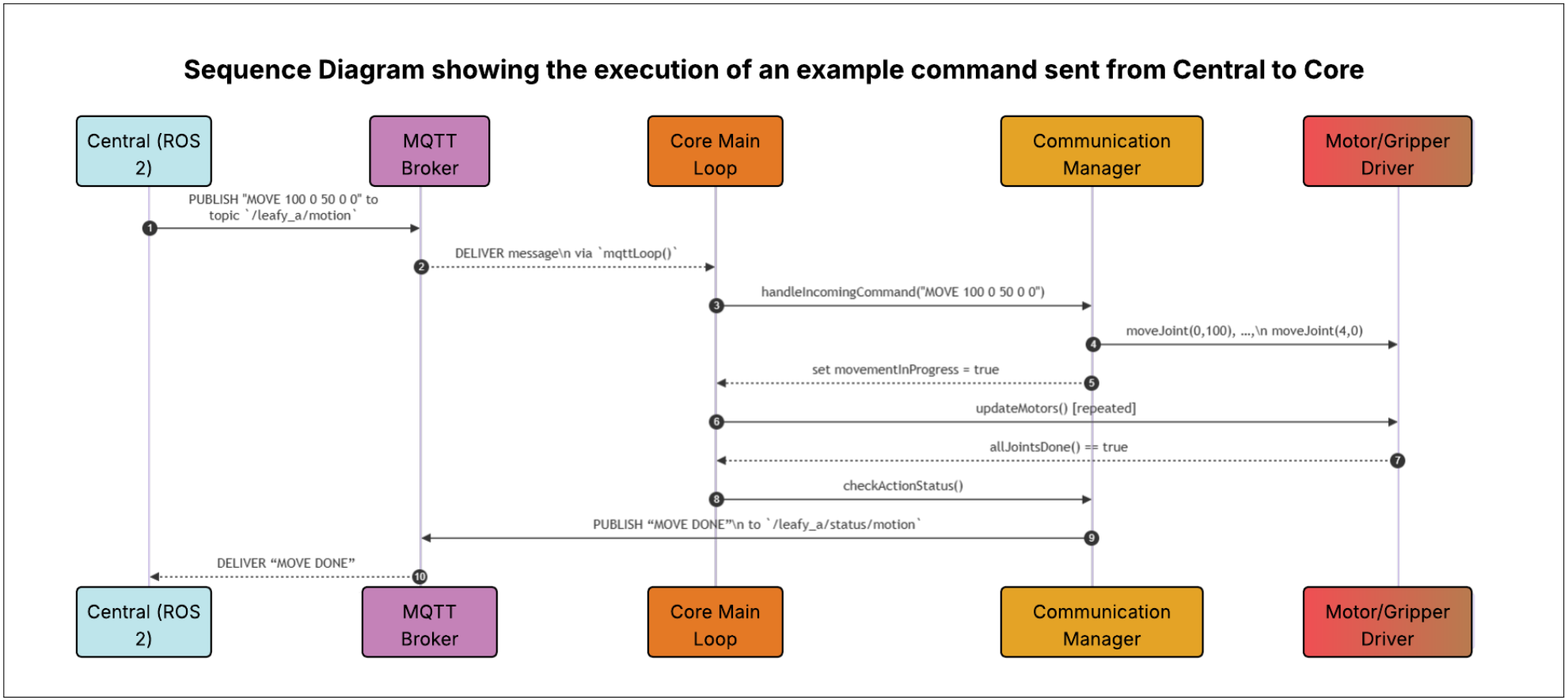


Figure 88: Sequence diagram showing the execution of an example MOVE command sent from Central to Core

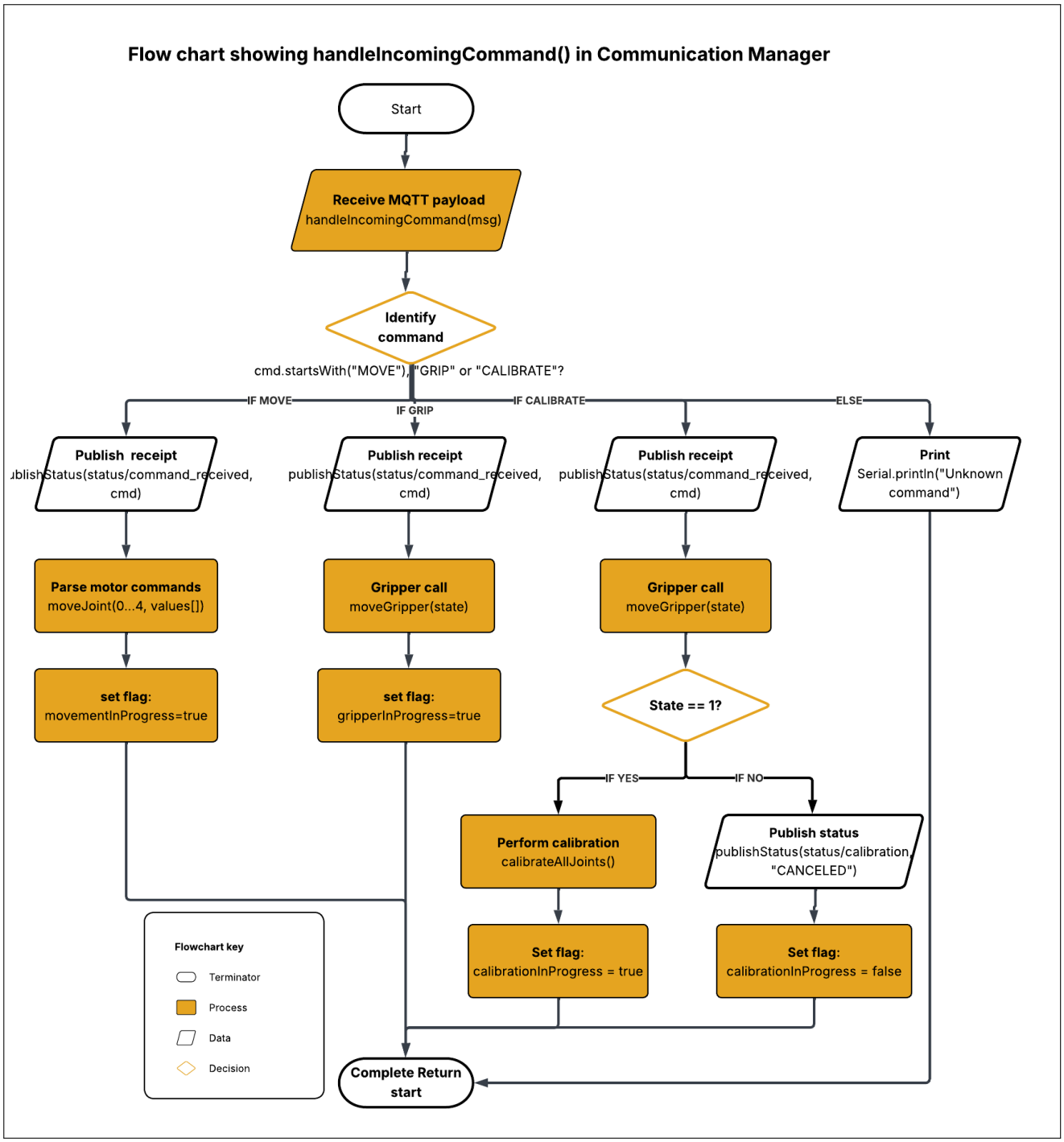


Figure 89: Flow chart showing handleIncomingCommand()

11.5.7 Motor Driver

EG |

The role of the Motor Driver is to provide precise, concurrent control of five stepper-driven joints with non-blocking motion and a homing routine for calibration.[39] [47]

Key functions:

initMotors()
Reads per-joint constants (`MICROSTEPS`, `GEAR_RATIO[]`, `MAX_OUTPUT_RPM[]`, pin mappings) from `config.h`, computes the required step-rate via the following formula:

$$\text{steps/s} = \frac{\text{MAX\_OUTPUT\_RPM}}{60} \times \frac{\text{MICROSTEPS}}{\text{GEAR\_RATIO}}$$

and then configures each `AccelStepper` instances maximum speed and acceleration.

moveJoint(index, stepCount) Queues a relative microstep move for the specified joint.

updateMotors() Calls run() on each stepper once per loop pass, ensuring concurrent stepping.[47]

calibrateAllJoints() Performs a blocking routine to locate left and right limit switches, measure travel range, return to the midpoint, and zero the position for each joint. [48]

allJointsDone() / calibrationDone() Reports whether queued motions or the homing sequence have completed. By using the AccelStepper librarys [39] non-blocking API, the module performs multi-joint motion within the super-loop. Homing is the only blocking operation. *We considered this to be justified by it being a critical safety feature, given that the initial motor drivers provide no other means for step position feedback.*

11.5.8 Gripper Driver

EG |

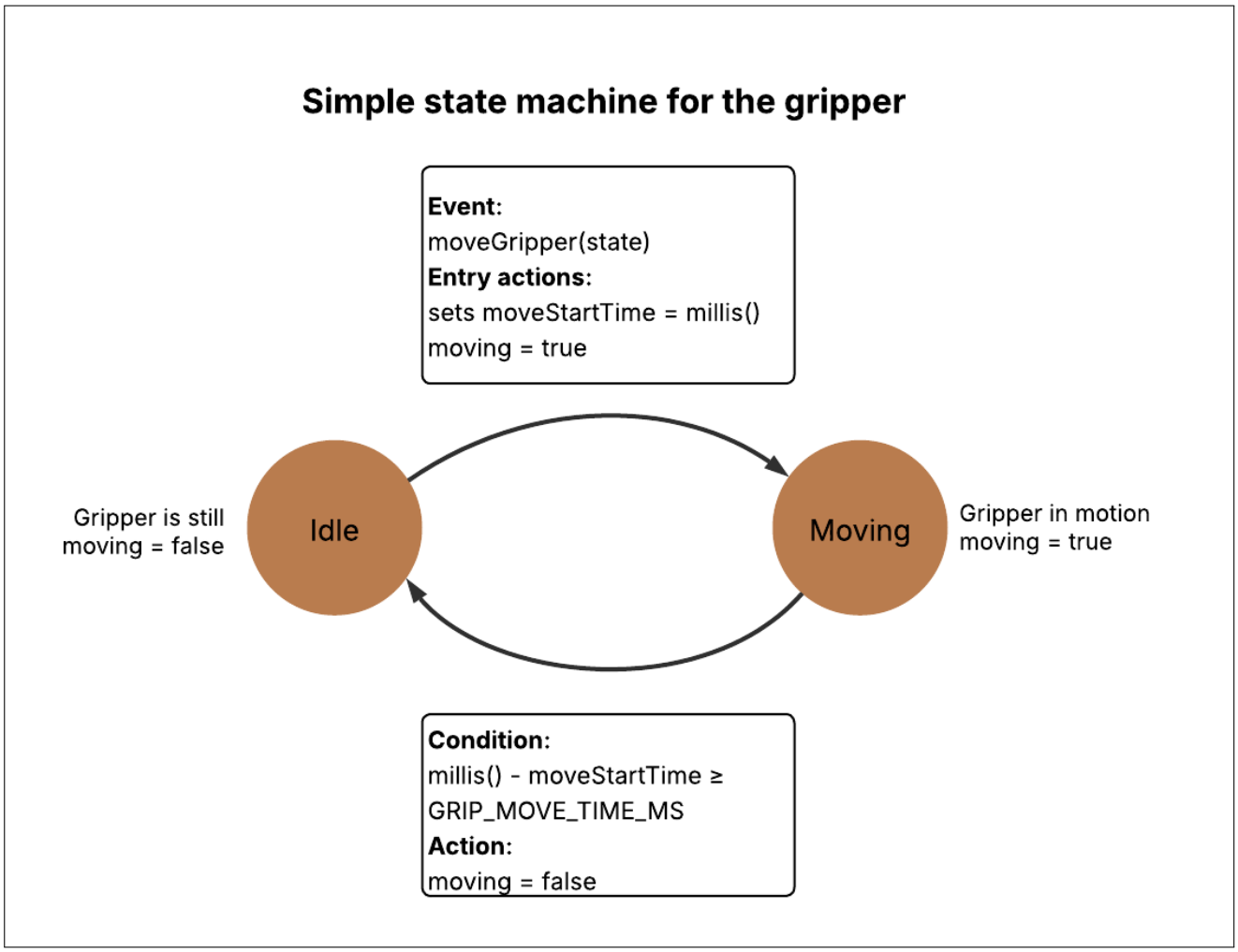


Figure 90: Simple state diagram showing the state transition conditions and actions.

The Gripper Driver module controls the gripper endeffector (`EF`) servo via a simple, nonblocking, timebased state machine (See Figure 90). When a `moveGripper(state)` command is called, the function records the current time in `moveStartTime` and sets an internal `moving` flag. On each pass of the main loop, `updateGripper()` computes:

$$\Delta t = \texttt{millis()} - \texttt{moveStartTime}$$

this is then compare to the predefined duration `GRIP_MOVE_TIME_MS`. The `moving` flag is cleared when $\Delta t \geq$ `GRIP_MOVE_TIME_MS`, showing that the gripper `EF` has reached it's target position.

Two advantages to this strategy:

1. By avoiding any blocking calls, the Main Loop is free to service motor updates and MQTT traffic at high frequency.

2. The elapsed-time check ensures the gripper completes its motion in a fixed, known interval, without creating slowdown elsewhere in the system.

This strategy is inspired by the following sources: [42] [49]

11.6 Camera

JCDH | *SME*

The ESP32-CAM is a low-cost camera development board intended for various IoT and industrial applications, including smart agriculture. It integrates the ESP32 chip directly on the same board, which makes it function as its own computer [50].[7] Our particular board uses the OV2640 camera module, which has a max resolution of 1600x1200 pixels [51].

11.6.1 ESP32-CAM supported resolutions

JCDH | *SME*

The ESP32-CAM supports the following resolutions (lower resolutions takes less time for the ESP32, network, AI models and Computer Vision (CV) to process) [52]:

- FRAMESIZE_96X96, // 96x96
- FRAMESIZE_QQVGA, // 160x120
- FRAMESIZE_QCIF, // 176x144
- FRAMESIZE_HQVGA, // 240x176
- FRAMESIZE_240X240, // 240x240
- FRAMESIZE_QVGA, // 320x240
- FRAMESIZE_CIF, // 400x296
- FRAMESIZE_HVGA, // 480x320
- FRAMESIZE_VGA, // 640x480
- FRAMESIZE_SVGA, // 800x600
- FRAMESIZE_XGA, // 1024x768

[7] The original intent was to connect the camera module to the Arduino, but because it was later found that this camera module has its own ESP32 chip it proved easier to just stream the data directly from the ESP32-CAM to the Leafy Automation Central.

- FRAMESIZE_HD, // 1280x720
- FRAMESIZE_SXGA, // 1280x1024
- FRAMESIZE_UXGA, // 1600x1200

11.6.2 Handling low light conditions

JCDH | *SME*

The testing indicated that the camera quality suffered greatly in low light conditions. Figure 91 shows an image of the printed chessboard from Figure G.53 on a neutral background. This image taken by the ESP32-CAM indicates how important targeted lighting or direct daylight is for the handling area. [8]

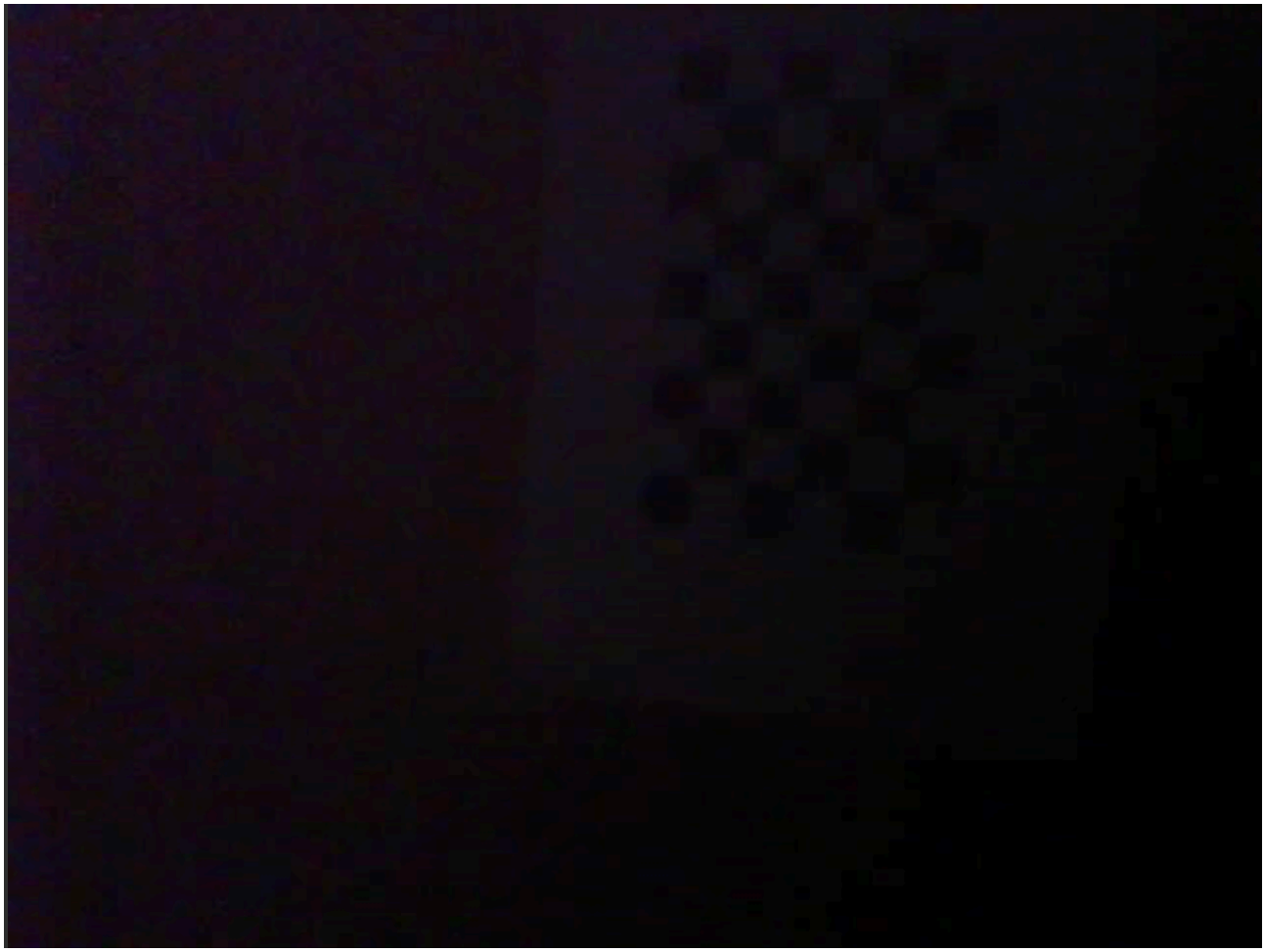


Figure 91: ESP32-CAM low light conditions

[8]Note that viewing the content of the image in Figure 91 is difficult, but it emphasises the importance of a brightly lit scene.

11.6.3 Verifying feasibility of detecting lettuce at a distance of 0.3 meters

JCDH | *SME*

This distance was chosen because it is well within the reach of the robot arm. The lens of the particular ESP32-CAM we use has a Field Of View (FOV) of 65 ° [53].

Using the formula for calculating the *viewer distance* from the Valve Developer Community [54], we can input known values and rearrange them to deduce the *screenwidth* (how much of the scene width is visible in the image):

$$\text{viewer distance} = \frac{\text{screenwidth}}{2\tan(\frac{FOV}{2})}$$

$$\Rightarrow \text{screenwidth} = \text{viewer distance} \times 2\tan(\frac{FOV}{2}) = 0.3 \times 2\tan(\frac{65}{2})$$

$$= 0.382242156 \text{ meters} \approx \mathbf{0.38 \text{ meters}}$$

Measurements indicate that a normal-sized Norwegian crispy salad is roughly **20 cm** in width, which proves that the camera captures the salad within the image when located on the robot arm. [9]

11.6.4 Calibration and handling camera distortion

JCDH | *SME*

Cameras can introduce distortion to images like radial distortion, which makes straight lines appear curved, and tangential distortion which means that the camera lens is not perfectly aligned with the imaging plane [55]. Figure 92 shows an example of tangential distortion, where the camera sensor is not parallel with the imaging plane. Image calibration using a chessboard aims to solve this issue, although if intrinsic camera values are known beforehand, or manually fine-tuned, this step can be skipped.

[9] Manual testing with the esp32-cam and a measuring tape was done to double check results.

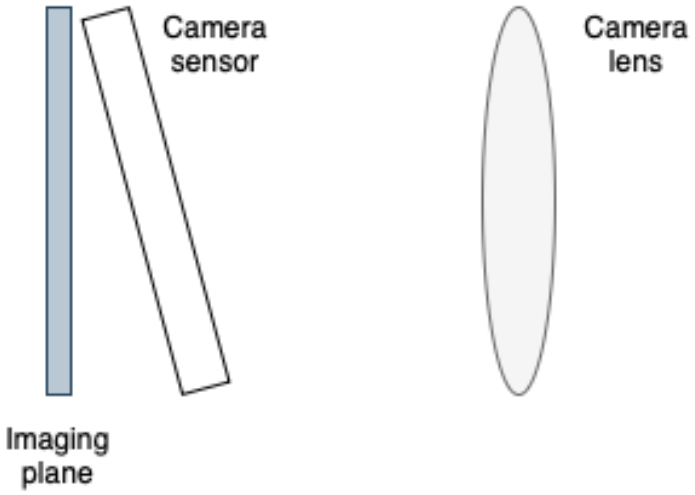


Figure 92: Example of unaligned camera lens in relation to imaging plane (tangential distortion)

11.7 Artificial Intelligence (AI) and Computer Vision (CV)

JCDH | *SME*

Artificial Intelligence (AI) and Machine Learning (ML) are important technologies and critical parts of image recognition. This section outlines the underlying technology of modern AI models and ML algorithms, and how they are used in our system. [10]

These days, a lot of fuzz is made about AI. Why do we need it, and what does it do? Well, because of today's environment around AI technology, we have to be careful in validating if a technology is actually useful and adds real value to our system goals.

11.7.1 What is Artificial Intelligence?

JCDH | *SME*

Artificial Intelligence (AI) refers to computer software which mimics the intelligence of biological life. [11] Appendix G.1 outlines AI concepts, technical analysis, training and benchmarking in great detail.

[10] Machine Learning (ML) is actually a sub-category of Artificial Intelligence (AI), but the terms are used interchangeably in many cases.

[11] Artificial Intelligence was actually an integral part of developing the COVID-19 vaccine as quickly and efficiently as was done. As a part of this process millions of people contributed collectively trough projects like Folding@home which allowed people to run protein folding AI models locally on their computers, providing valuable insights into viral mechanisms and potential treatments. [56]

Overall, AI models have showed to be a valuable resource for achieving the requirements of Hydroplant Technologies, especially depth estimation. Fine-tuning object detection models to a satisfactory point has proven to be time-consuming and resource intensive, and although further work must be done for this type of model to become useful, the research we have done will be a good baseline to work on in the future. It is believed that object detection AI models are superior for diverse sets of plants, based on testing.

For more info on the initial AI testing, please refer to Appendix G.3.

11.7.2 What is Computer Vision (CV)?

JCDH | *SME*

Computer Vision generates useful data points from a visual source, like an image. [12] Appendix 10.4 outlines Computer Vision concepts, technical analysis and development in detail.

11.7.3 Combining multiple different AI and CV technologies

JCDH | *SME*

The consensus is that combining different technologies can give improved accuracy for the robot arm, and each technology is a tradeoff between performance and complexity. The goal of each technology is to create an abstraction of the real world, whether this abstraction actually is close enough is something that must be tested and considered thoroughly, but it is our belief that the plethora of AI and CV technologies explored creates a useful baseline for further development.

11.7.4 Classifying a plant and creating a "plant_type"

JCDH | *SME*

Figure 93 shows the process of predicting a plant type. The model is fed a list of possible values, which are called "candidate labels". From on the image supplied, a score for each candidate label is generated which indicates how confident the AI model is in its prediction. The AI model used for this task is called "openai/clip-vit-base-patch32". This is a unique kind of image classification model which hasn't been specifically trained on the source images [57] [58].

[12] In many cases, Computer Vision tries to mimic how vision works in biology.

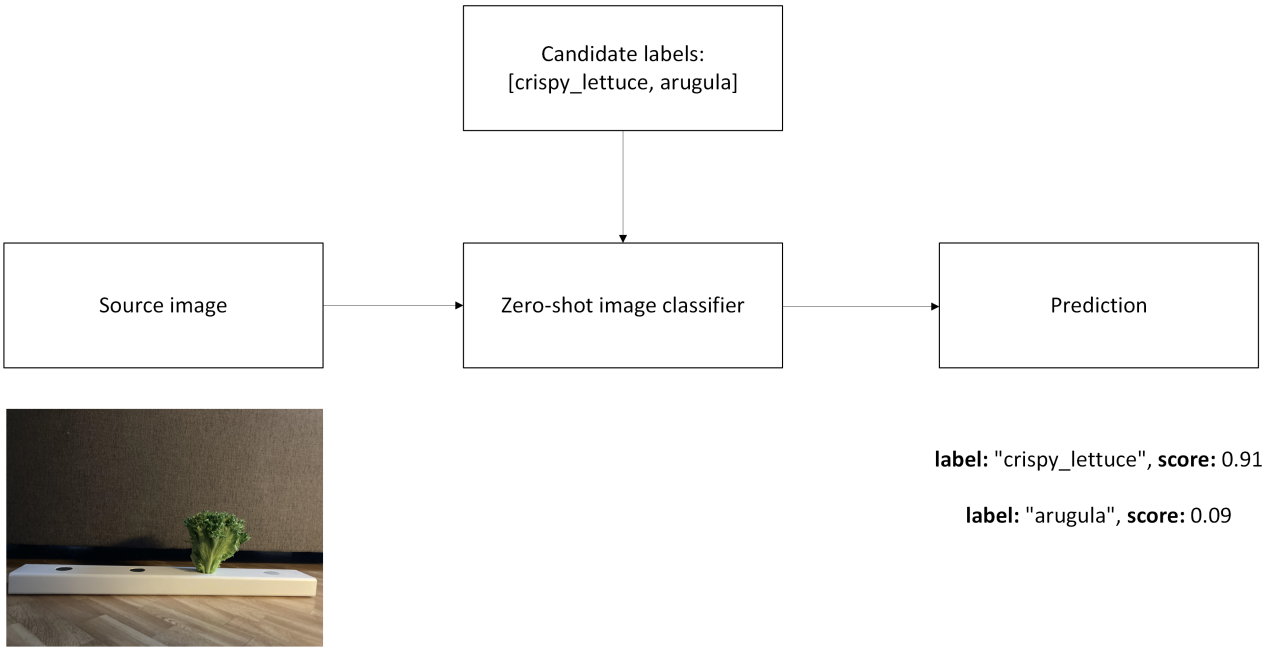


Figure 93: Plant type pipeline

11.7.5 Creating a "grip point"

JCDH | *SME*

In order to facilitate the proper movement of the robot arm, a grip point has to be generated. This is done using a combination of AI models and custom developed CV algorithms. Figure 94 shows the process of generating a gripping point from a source image. [13]

Depth estimation is done using the "depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf" [59] [60] AI model, while mask and geometry generation is done using custom developed algorithms which rely on PlantCV and OpenCV.

[13]The chessboard seen in Figure 94 is used for image calibration and testing purposes.

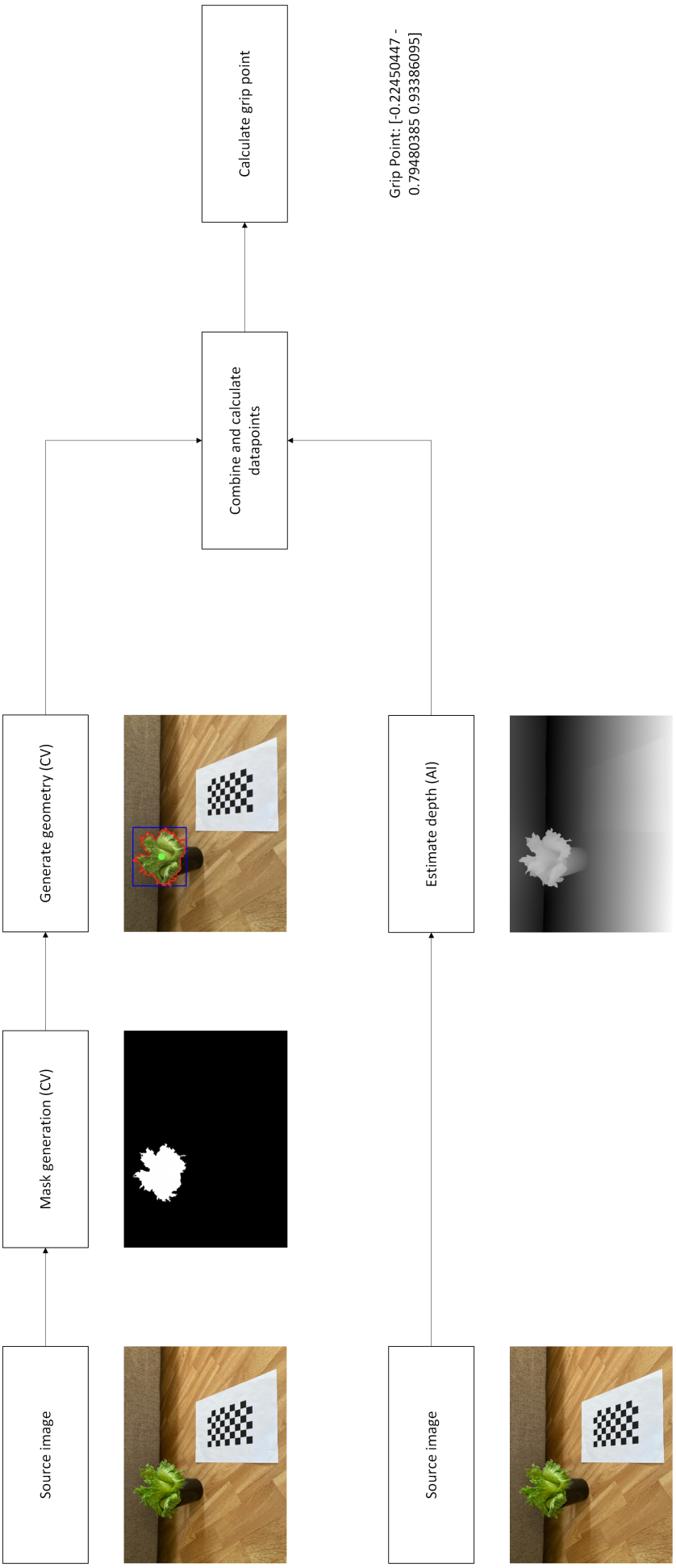


Figure 94: Grip point pipeline

11.7.6 AI models and CV algorithms used in Leafy Automation

JCDH | *SME*

The following is a list of all AI models and CV algorithms used in the system (more on these in Appendix 1):

- **Image Classification** - Classifies what objects are within an image. In our case, which type of lettuce.
- **Depth Estimation** - Estimates a distance in meters from the camera sensor for each pixel in an image.
- **Mask Generation** - Separates an object of interest from the rest of the image but giving it a white color, while everything else is a black color. [14]

The different AI and CV codebases are divided up into *tasks*, which can be seen in Figure 95. The reason for using the *tasks* concept is that its a well established term within the Hugging Face documentation, and extending it to mean any **AI or CV related task makes sense from an application organization perspective** [61].

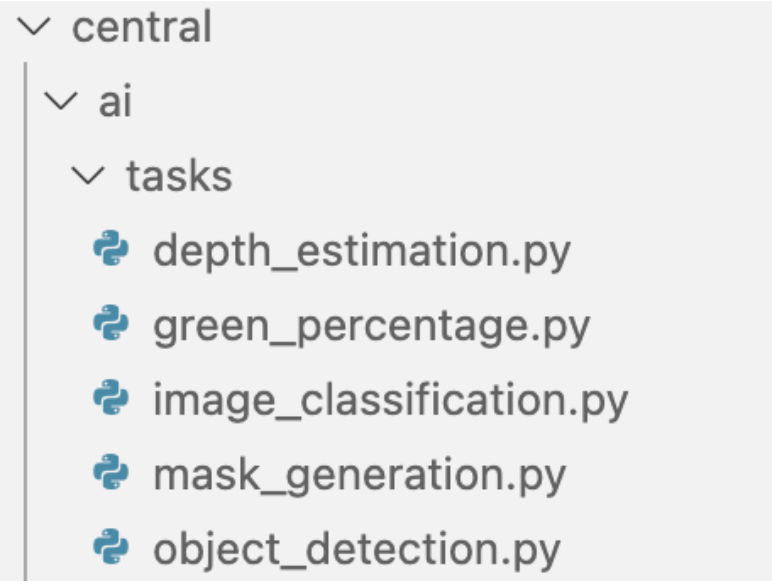


Figure 95: AI tasks directory structure

[14] Testing indicated that using a custom developed Computer Vision algorithm for mask generation gave quicker and more accurate results than using an AI model. This due to the process of identifying lettuce being highly specialized, and not general purpose which most AI models are optimized for.

11.8 Code quality and maintainability

JCDH | *SME*

This subsection outlines different approaches we use for writing high quality and maintainable code.

11.8.1 Type hinting

JCDH | *SME*

Python is a dynamically typed language which means that variables does not require you to define variable types, this stands in contrast to languages like C++ which do enforce types (statically typed languages). Statically typed languages offers many benefits like **improved code readability** and **reduction in bugs** related to wrong assumptions about variable types (e.g. passing a string such as "2" instead of the integer 2, which is a common mistake in programming). We make use of both Python and C++ in our codebase, and altough Python is not a statically typed language it supports "type hinting" (meaning labeling a variable as a specific type) which we make extensive use of.

```
def auth(username: str, password: str) -> bool:
    ...
```

Figure 96: Python type hinting example

As seen in **Figure 96** the “auth” function defines that it requires two arguments (*username*, *password*) of type *str* (string), and returns a *bool*.

11.8.2 Unit testing

JCDH | *SME*

We need a way to confirm that our system works as expected after implementing new features in our codebase. Therefore unit testing is certainly one of the key methods to measure system stability by. Unit testing is the practice of testing small units of code (e.g. functions), and asserting that these units of code actually return the expected values.

11.8.3 Benchmarking class

JCDH | *SME*

In order to deduce the performance of an arbitrary block of code, benchmarking must be done.

This was achieved by using a custom developed class which allows you to measure the time it takes to execute a piece of code and export the data as a diagram. Please refer to Appendix G.G for details on using this script.

11.8.4 Environment variables

JCDH | *SME*

Environment variables are used to store variables which should not be in the source code, and which are only loaded at runtime. Examples are passwords and access tokens. A file called ".env" is used to achieve this for the Leafy Automation Central, and a file called "secrets.h" in Leafy Automation Core, which is separated from the commited codebase by using a ".gitignore" file.

12 Design Review

A formal design review was conducted with both our internal and external supervisors to present the current status of our robotic arm project and receive valuable feedback for the remaining project period. This review served as a checkpoint for assessing progress across all departmentsmechanical, electrical, and softwareand to identify critical tasks for completion.

Mechanical Department

The mechanical team presented the fabricated components and demonstrated their functionality with attached and programmed motors. A prototype of the gripper was showcased, successfully demonstrating its gripping force. However, the rotational torque could not be tested during the review due to a loose connection between the stepper motor and the gripper body. Additionally, we presented printed components for the joints and arms, as well as a prototype of the base that successfully demonstrated rotational motion.

Software Department

The software team provided an overview of the system architecture, including the Arduino microcontrollers, AI and the central control unit based on a Raspberry Pi. The feedback regarding our approach to system architecture was positive. We received valuable advice concerning motion planning and were encouraged to explore integration with ROS2 for improved modularity and scalability. Another point of discussion was the importance of demonstrating some level of motion in the final prototype, to visually showcase the robotic arm's capabilities. Furthermore, the camera system was reviewed, including considerations for optimal placement, the number of cameras (single vs. dual), and appropriate angles for maximizing visual coverage and accuracy during plant recognition and harvesting.

Electrical Department

The electrical team was tasked with connecting all electrical components while ensuring efficient and safe cable management. Proper cable routing and space claim are critical to prevent interference between moving parts and electrical lines. One essential improvement highlighted during the review was the need to integrate limit switches for the stepper motors responsible for rotational movement. Without these, the system risks mechanical failure due to over-rotation. Additionally, if time permits, it was advised that we implement safety mechanisms to handle potential failuressuch as broken limit switches or misaligned componentsto protect both the hardware and users.

Summary and Reflection

The feedback from our supervisors was both insightful and constructive. One key takeaway is the importance of time management in the final project phase. With limited time remaining, we must focus on completing the most critical features. Any components, functions, or improvements that cannot be implemented within this timeframe will be documented in a dedicated chapter titled Future Work **??**. This ensures continuity and provides a clear foundation for Hydroplant Technologies AS to continue development if desired. It is important to emphasize that this project is a proof-of-concept. While functional components have been developed, and preliminary integrations have been made, the current system is not yet suitable for commercial or industrial use. Further refinement, testing, and validation will be required to achieve full operational readiness.

13 Prototype

We will look at the latest iterations of parts to be chosen as the prototype to be built. The robot arm parts are continuously developed to meet the requirements to move effectively in the designated area. We will also introduce the process of 3D-printing, since almost all parts are printed.

13.1 3D printing

We will shortly describe how 3D-print is used in the industry. Also what materials we used and the techniques we used to improve and reinforce prints.
Group member DAB has his own printer at home that we can use, and we can also request parts printed by teacher Richard at USN Kongsberg.

13.1.1 Rapid prototyping

DAB | *SME*

Rapid Prototyping is widely incorporated in many different project models because it is an iterative process. It also helps to reduce time used in development, increases the end products quality, makes it easier to visualize concepts, has low cost and fast production time of prototypes.

13.1.2 Materials used

DAB | *SME*

PLA–Polylactic acid
PETG–Polyethylene terephthalate glycol
TPU–Thermoplastic polyurethane
The most commonly available plastics for 3D-printing is PLA and PETG. PLA has more tensile strength but is more brittle, and more sensitive to moisture.
PETG is not as durable as PLA. It is a lot more flexible, has higher UV resistance and is more resistant to heat and chemicals.
TPU is a flexible filament which is hard to print because it can slip in the extruder gears. Low speed and high temp printing is required.

13.1.3 Printing

DAB | *SME*

This section will discuss a bit about general printing and some things you need to think about.

There are plenty of articles and videos online going in full detail for each topic.

Bed Adhesion

It is critical to have good bed adhesion so that the parts that are printed does not loosen from the print bed wile printing, or the print lifting in some areas which will lead to warping or worst case the extruder hits the print and knocks it off while printing.

To improve adhesion, you can use tape, glue sticks and hair spray. It is most common to use hair spray, since it has good effect. You can also upgrade the printer bed by applying steel PEI sheet. PEI is a Polyetherimide, which is a a thermoplastic coating with good adhesion properties that requires little to no surface prep for many different types of filament. Some of the PEI sheets come with two sides, one smooth and one textured surface, for better grip. Another benefit is that it is made out of steel, which makes it easier to remove parts from the steel plate since it loosens more effortlessly.

Print temperature, print speed and cooling

These three thing are related to each other. Printing temperature is critical to attain good flow of material at the speed range you want to print, without overheating the filament and having correct fan speed to cool the layer at a correct speed to achieve good adhesion of the layers and minimal stringing of material.

Part print orientation/layer direction

Knowing which direction to print the parts, is also critical, especially if it is bearing some kind of load in a specific direction. Since you can adjust the part orientation to achieve desirable layer direction to bear load, instead of splitting the layers as easily.

Infill and wall line count

Infill describes how much the inside of the printed part is filled in (%) and what type of structure the infill has. The infill at full, alternates layers 90°line pattern.

It is also possible to choose the defined amount of wall lines you want, this is calculated automatically, but sometimes it is better to have thicker walls for strength overall, or if you want to have heat inserts or drive screws directly.

Layer height

With a finer layer height, you can have a more detailed part, but it will not be as strong as thick layer lines since you reduce number of layers required. If you have a large nozzle with a high layer height, it will produce stronger parts but will not be visually appealing.

13.1.4 Printing arm part

DAB | *SME*

We designed the parts in Solid Works in the material it is supposed to be in the industrial context, so no adjustments were made just for a 3D-printed version. The only change was that we used PETG since it is more flexible and it increases the layer lines so that the screws have good amount of space to screw in and form treads nicely in the plastic.

We had some problems with the print. It split up in the layer lines because of the force from screws being tightened, and the screw was tapping its own threads. Unfortunately, we could not change the direction of layer lines since the load acting on the arm would not allow me to print it.

13.2 The base prototype

BMR |

This version was designed mainly for additive manufacturing (3D-print) for the prototype of the robot arm. A section view of the assembly can be seen in fig. 97, the green and blue hues are part of the rotating part of the base attached to the arm, while the pink and orange hues are stationary and secured to the shaft and the table. An exploded view showing the assembly can be seen in fig. 98.

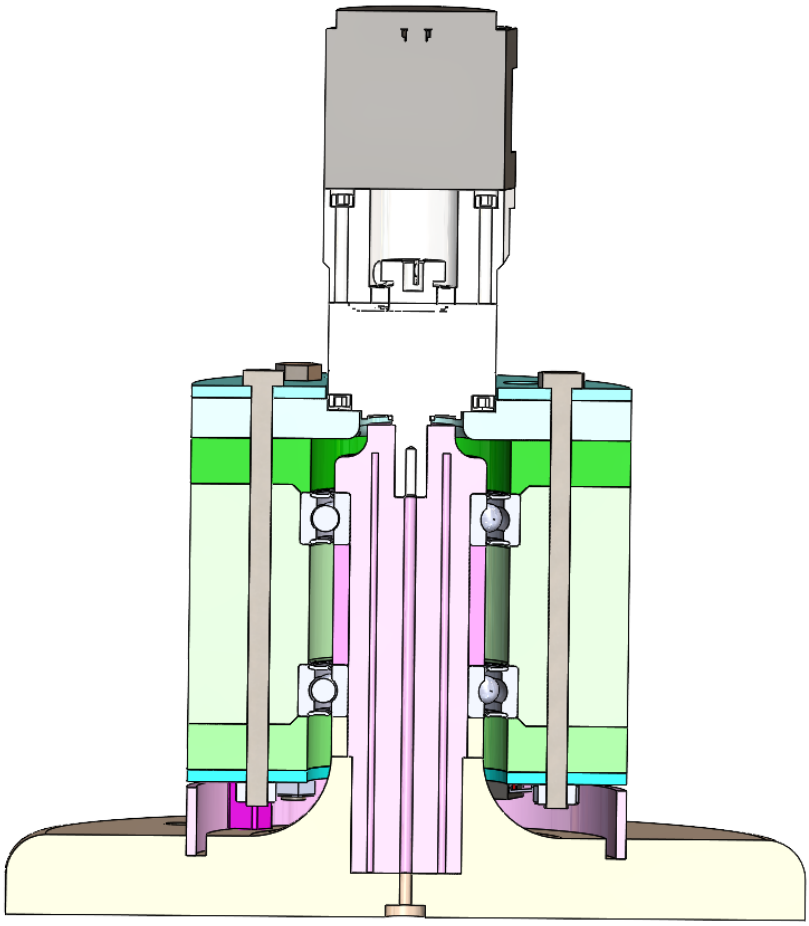


Figure 97: Section view of base assembly (CAD bearings from [5] and CAD motor from [3])

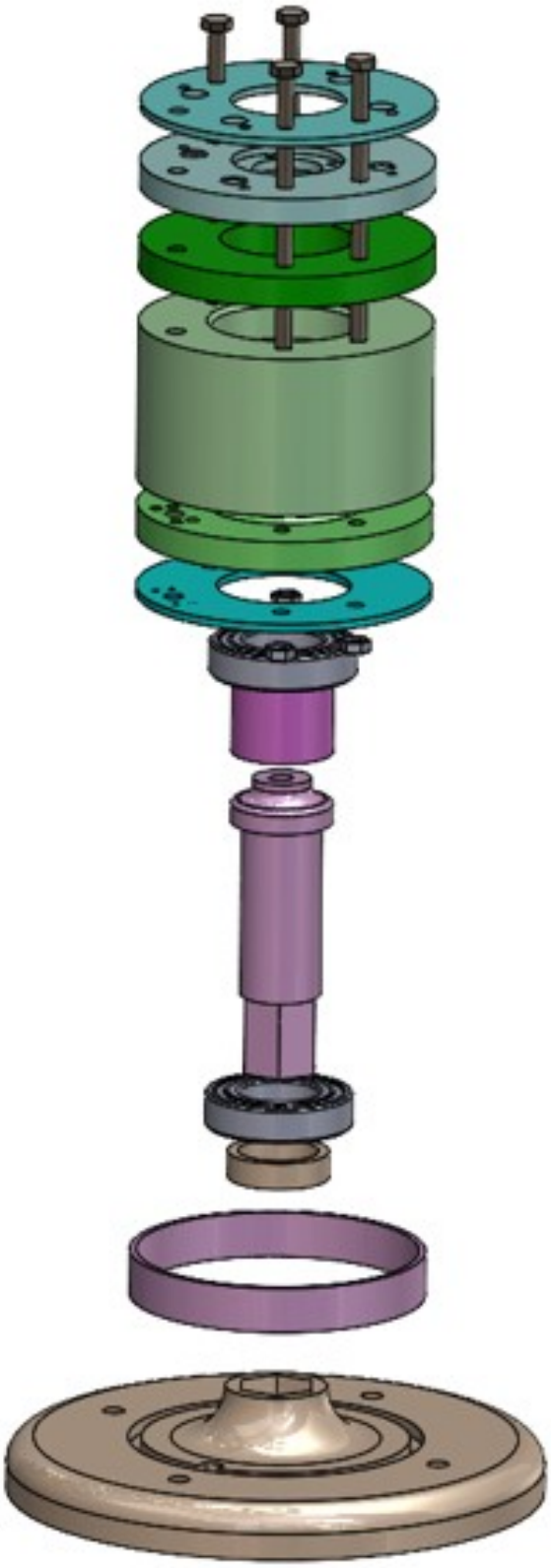


Figure 98: Exploded view of base assembly (CAD bearings from [5])

The inner shaft is reinforced with 6 x Ø2.5 mm steel rods to counteract the weakness of the layer lines in the 3D print (as discussed in section 13). The outer housing polymer parts are sandwiched between two disks of MDF that are laser cut for better tolerance in the interface

to the arm, and also provides a more even load distribution. A hole going through the outer housing from the limit switches to the top allows for the wires to connect to the control units at top. All parts can be seen in fig. 99, and assembled in fig. 100.

The bearings used are two deep groove ball bearings (SKF 6006-2RS1), that can take both radial and axial loads. These are supported between the shaft and outer housing.

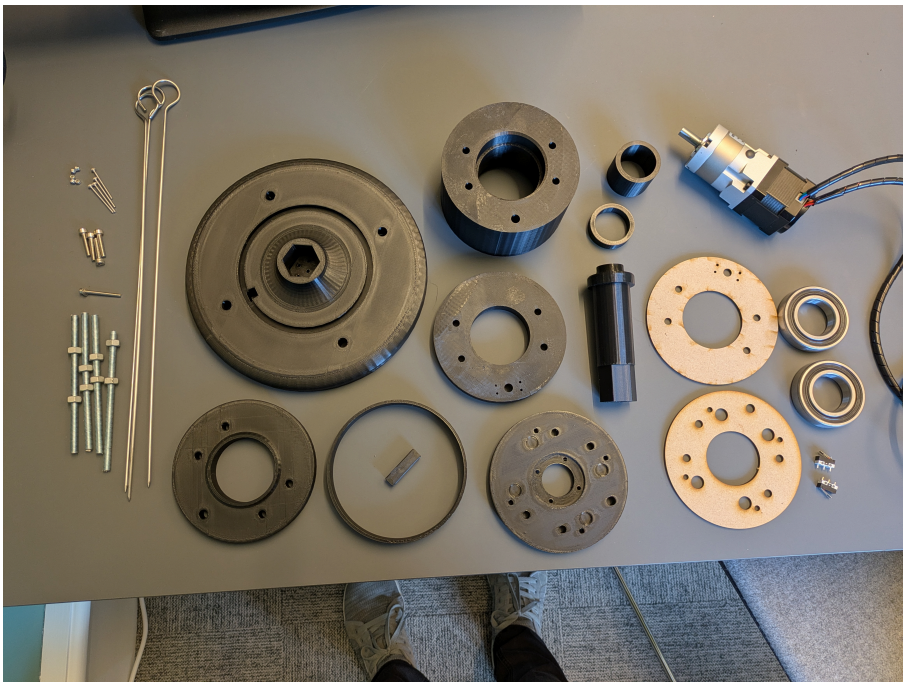


Figure 99: 3D printed parts for the base before assembly

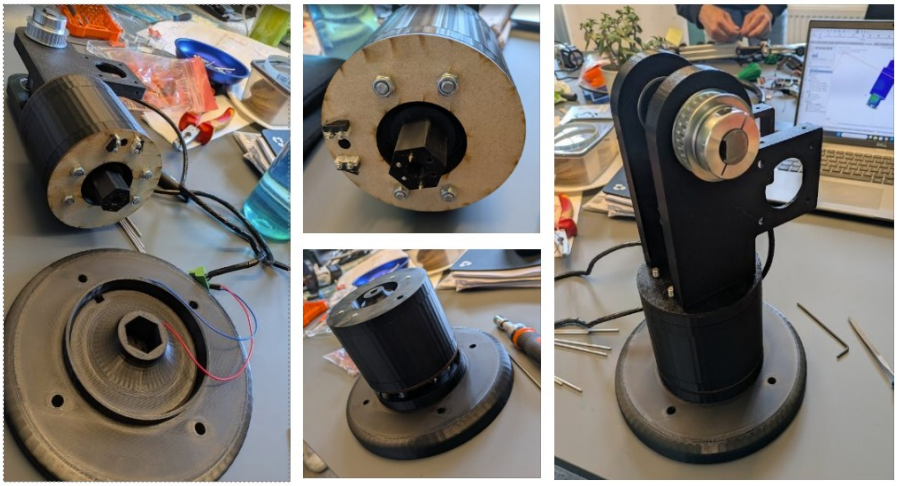


Figure 100: The base prototype assembly

13.3 The joints/arm

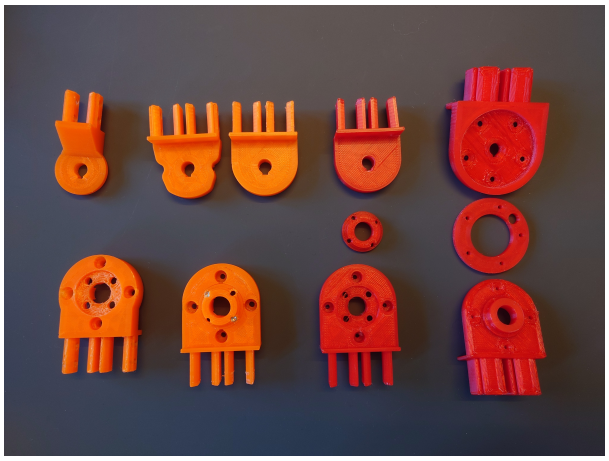
DAB | *SME*

In this section we will look at the printed iterations and discuss problems discovered or tweaks needed.

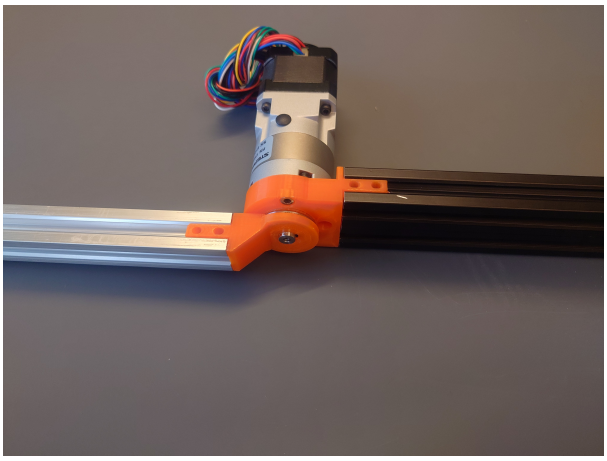
13.3.1 Direct drive joints

DAB | *SME*

We printed several variations/iteration to check how it looks physically and to check fitment. This method of printing and testing, or visual check is called rapid prototyping which is widely used and implemented in various production models across the world. More about rapid prototyping/3D-printing you can find at the beginning of this section. 13.



(a) iterations



(b) latest test fit

Figure 101: direct drive Joint prototypes

13.3.2 Belt drive

DAB | *SME*

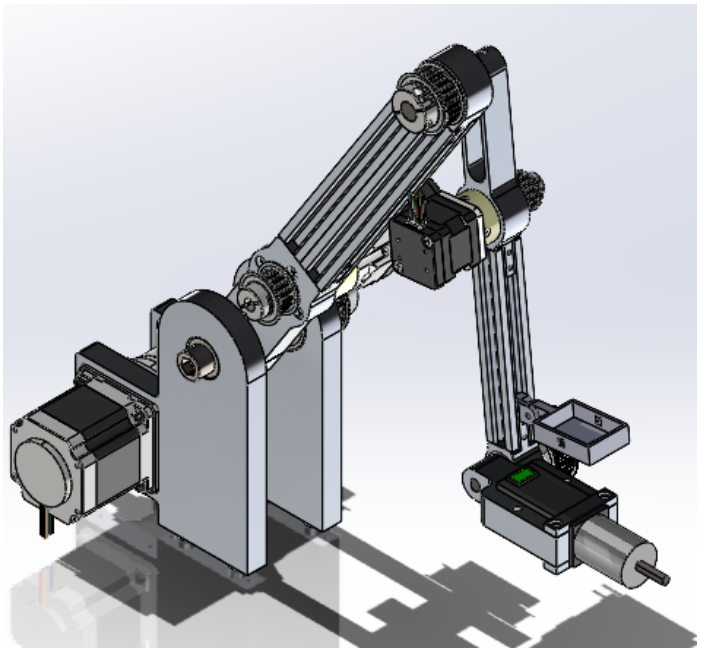


Figure 102: Robot arm V0.1B

We chose to go with a belt driven version for the ability to adjust the gear ratio after gear boxes of the motors (See section 10 for more information about motors). This will allow us to fine tune the torque to our requirements. Another benefit is that we offset the motors closer to the base, which in turn helps with weight management of the robot arm.

The biggest motor in joint one is offset off centre to work as a counter balance weight. NEMA 23 motor gross weight is 1.67kg which works well for us, since the arm itself without motors is around 950 grams with parts in aluminium and not 3D-print. With motors included it is around 2251 grams.

The robot arm meets our requirement for angle of rotation, and flexibility of the design. This implies that the design can be easily adjusted for future modifications or adaptation for other task because of multi assembly design.

Design can bee realistically developed with the available resources, and budget supplied by our employer, but because of our time constraint, we can only manage to 3D-print a prototype.

The durability should be adequate by looking in what working conditions the robot arm is in. Aluminium does not rust like steel, this is important since the robot is placed in 23℃high humidity environment.

Aluminium has a good protective oxide layer, and is used widely across the world as food safe material in food production and more. You can also get antibacterial anodizing coating which is comprised of quaternary ammonium compounds, used in high contact areas. By looking on research papers, it kills 99.9% in 5 min [62].

Compared to AR4-MK2 kit [63]
The total reach of our arm is 60cm but working range is 50cm. Payload 850-1kg, do not have weight requirements, but try to get as low as possible.
The DIY kit has a total reach of 62.9cm. Payload 1.9kg, total weight 12.25Kg

Quick moment calculation

We did some rough hand calculations to check if we exceed the holding torque of the motors after gearing. Since our arm has to have a payload of 1kg, and also consider the gripper to have a maximum load of 1kg, we need to ensure that we have no problems down the road.
Joint one has a small margin of holding torque, joint 2 is with a wide margin and joint 1 is overkill we have more than we need.

Price
Price per meter for aluminum profile is 110kr in zeptobit store [27] where our employer buys it.
Aluminium profile length used in our prototype is 190mm link 1 double row, but link 2 length is 120mm single row.
Total length of alu profile needed is 500mm. 110kr/1000mm=0.11 price per mm. so total price for aluminium profile for robot is 500*0.11=55kr.

Also looking up the cost of CNC machining for joints and shafts in JLCCNC if we chose to order. The total price with shipping is 184.23 without 25%mva, With mva 230.29.

13.4 Gripper development and Testing

SME | *JCDH*

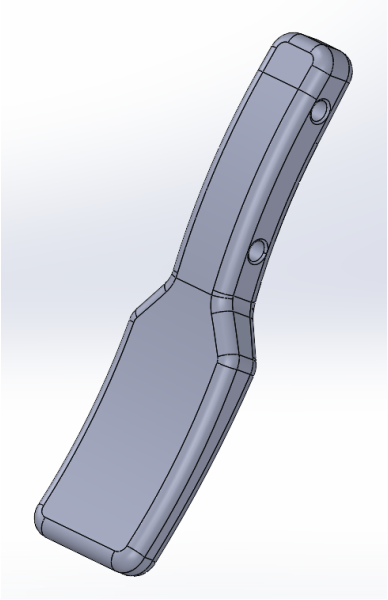
The first prototype design focused on a jaw-style gripper, designed to facilitate initial testing, data collection, and integration with electronic components. This prototype was not intended to serve as the final end-effector for the robotic arm but rather as a preliminary functional model to streamline early-stage testing and optimize the development process. Given its purpose, the jaw gripper will be kept relatively simple, ensuring that it remains an efficient and cost-effective starting point. Additionally, a key design consideration will be the modularity and replaceability of the gripper, which must be accounted for from the early development stages. Furthermore, whether to incorporate soft touch technology in this prototype or delay its implementation to a later iteration that more closely aligns with the final design must be evaluated.

For the subsequent prototypes, the focus will shift toward developing a more advanced gripper that better replicates the flexibility and functionality of a human hand. These iterations will incorporate finger-based gripping mechanisms along with soft touch technology, such as the Fin-Ray gripper principle, to enhance adaptability when handling objects with complex geometries. The incorporation of soft touch elements is essential to prevent damage to delicate produce, such as salad greens, during pick-and-place operations. By refining the design through an iterative prototyping process, the final gripper will achieve an optimal balance of flexibility, precision, and delicacy, meeting the specific requirements of agricultural automation.

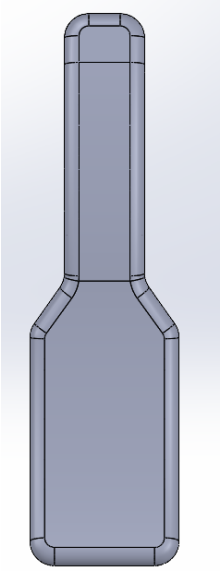
13.4.1 Gripper V1

SME | *JCDH*

This initial prototype was developed with the objective of creating a simple, quick-to-assemble design that could be efficiently modelled in SolidWorks, and easily 3D printed and assembled. It serves primarily as a test model to validate the fundamental design and assembly process and does not represent the final product. The design process began with the gripper's fingers, which are shaped somewhat like spatulas. The prototype features two fingers designed to work in unison to gently lift a salad plant and place it into a designated area.



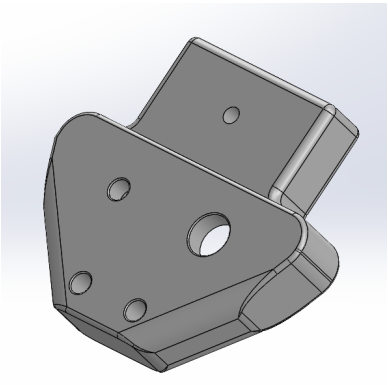
(a) Finger gripper



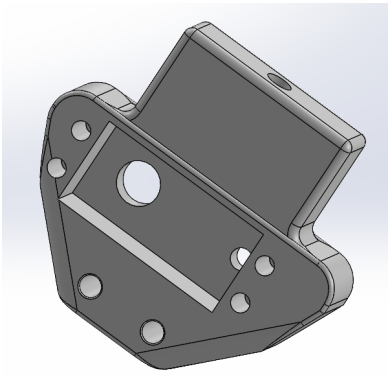
(b) Finger gripper front

Figure 103: Gripper fingers

To integrate the gripper with the NEMA 11 stepper motor (See section 10 for more information about motors) and the rest of the robotic arm, a structural base was designed. An indent was added on the underside to securely place the servo motor, along with holes for fastening the screws. A central hole was also incorporated to allow the motor shaft to connect to the internal gearing mechanism responsible for actuating the fingers.



(a) Gripper body



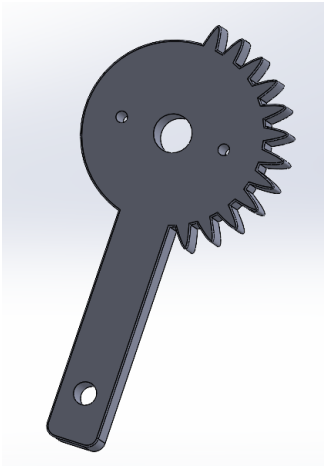
(b) Gripper body, underneath

Figure 104: Gripper body

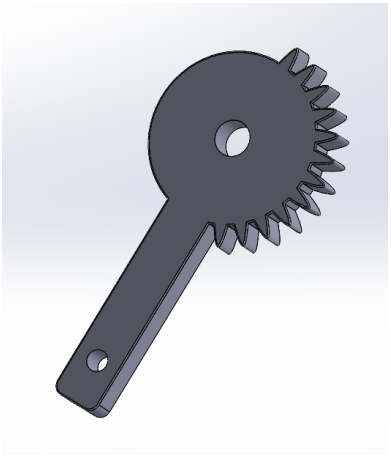
Subsequent modifications included the addition of screw holes to mount the gear components and structural supports. The body was further extruded to accommodate the stepper motor, and a through-hole was created for the Z-screw, enabling rotational movement of the gripper. This extruded section also partially encloses the stepper motor to protect it from dust and

mechanical interference.

Following this, the gearing components were designed. Two nearly identical gear holders were created, differing only in that one interfaces directly with the servo motor and includes mounting holes for attachment to the motor fan. These gear holders are mechanically linked to the gripper fingers.



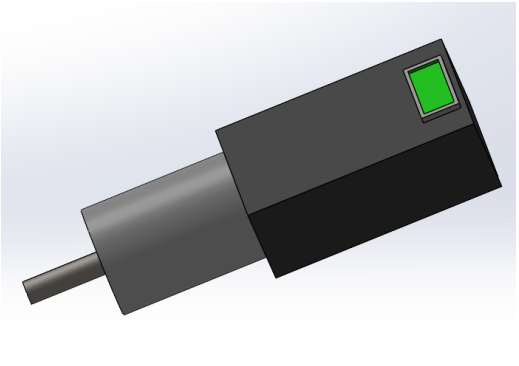
(a) Gear part 1, connection to servo motor



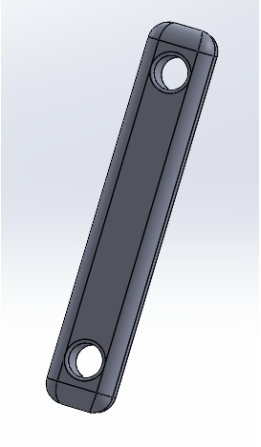
(b) Gear part 2

Figure 105: Gearing components

To enhance structural integrity and minimize the risk of mechanical failure, two additional support components were developed to link the gripper fingers to the main body. Due to the unavailability of an accurate NEMA 11 stepper motor model in the CAD library, a simplified version was modelled based on real-world measurements. This enabled accurate dimensional validation and ensured a proper fit during the assembly process. All components were subsequently assembled in a SolidWorks® (SolidWorks) environment. As emphasized, this is an early-stage prototype intended for testing and design validation, not the finalized product.



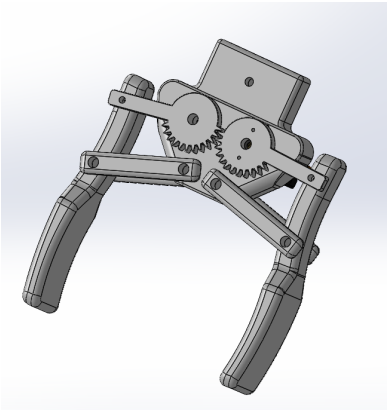
(a) NEMA 11



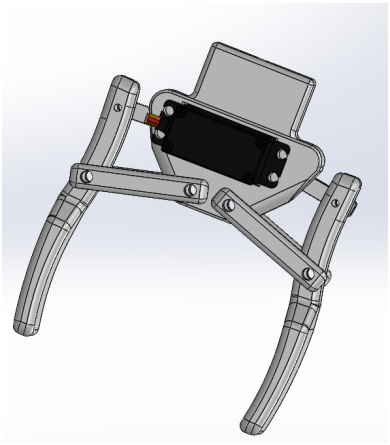
(b) Support components

Figure 106: Support and stepper motor

Throughout development, several iterations were made to the body design. Ultimately, the enclosing section around the stepper motor was removed to reduce material usage and decrease the overall weight of the gripper. Since the motor is securely fastened using a Z-screw, the enclosure was deemed unnecessary at this stage. However, a secondary fastener may be added near the motor tip in future versions to provide additional stability. This decision will be made following further structural evaluation, including Finite Element Method (FEM)



(a) Gripper assembly front



(b) Gripper assembly back

Figure 107: Gripper assembly

The final version of Prototype 1 appears as shown. Upon completion of the design, the SolidWorks models were exported as STL (STL) files and sent to a 3D printer. Printing began with the support structures and the components interfacing with the stepper motor, followed by the gripper body and the two fingers. The first print attempt of the body and fingers failed due to poor bed adhesion, necessitating a restart. The second attempt was successful.

After printing, all components were assembled using screws. The servo motor (See section 10 for more information about motors) was then connected to the gear-driven component responsible for actuating the gripper. However, the motors side sliders interfered with full insertion into the designated cavity, requiring light sanding with sandpaper to ensure a snug fit. Once all parts were in place, the screws were tightened, and the prototype was prepared for initial testing.

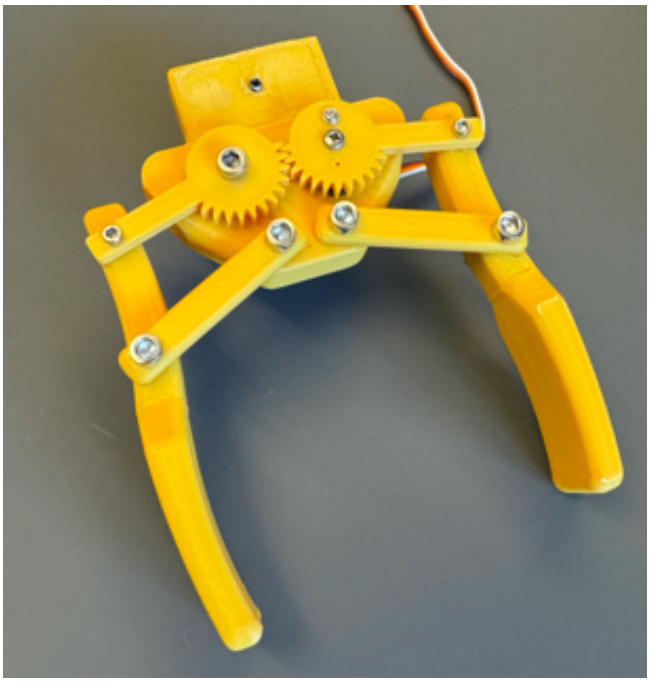


Figure 108: Gripper prototype 1

13.4.2 V1 - Assembly and Initial Testing

SME | *JCDH*

Upon completion of the 3D printing process for all components of Prototype 1, the assembly phase was initiated. The parts were mechanically fastened using appropriate screws, and the assembly tolerances were sufficient to allow all components to fit together as intended. Following mechanical integration, the servo motor was mounted and coupled with the internal gear mechanism to assess the functionality of the gripper.

Once all electrical systems were connected and verified, a control script was uploaded to the microcontroller to test the actuation of the gripping mechanism. The servo-driven gripper successfully executed basic open-and-close motions, confirming that the mechanism was functionally operational in its initial configuration.

During this test phase, one observation was that the gear engagement could have been optimized. Specifically, the mating gears did not achieve full rotational contactlimiting the overall

range of motion slightly. Although this did not hinder basic gripping functionality, the limited contact area reduced potential grip span and torque efficiency. This shortcoming was documented and directly informed improvements implemented in Prototype 2, where the gear profiles and tolerances were refined for better performance.

In terms of rotational capability, the stepper motor was mounted through a central bore in the gripper body and secured using a Ø4 mm Z-screw. While the Z-screw initially fit, the tolerances between the printed hole and the screw shaft were too close, resulting in material wear during repeated threading. This caused the hole to widen over time, and during the design review demonstration, the Z-screw failed to maintain its positionrendering the rotation functionality inoperable during that phase. As a result, while gripping force was demonstrably successful, the rotational movement of the gripper could not be showcased.

This issue highlighted the need for reinforced mounting solutions for the stepper motor. In the subsequent iteration, additional structural support and fastening mechanisms were incorporated to ensure the motor remained securely attached during repeated use and under applied torque.

These lessons from Prototype 1 provided critical feedback for mechanical robustness, modularity, and motion fidelity, which were all addressed in the next design iteration.

13.4.3 Gripper V2

SME | *JCDH*

Building upon the insights gained from the first prototype, the second gripper iteration was designed to improve both functionality and structural integrity while maintaining a modular and cost-effective approach. Since the final prototype will not undergo industrial deployment at this stage, and given the limited project budget, it was decided to not use machined components in favour of 3D-printed parts using PLA which was used for the structural components due to its rigidity and ease of printing, while TPU which was chosen for the Fin-Ray fingers thanks to its elasticity.

Although machined steel parts would undoubtedly offer greater mechanical robustness, particularly relevant for agricultural applications, these will be discussed in section 15.1. The exclusive use of 3D-printed components enables rapid iteration and low-cost manufacturing. Moreover, 3D printing is easily accessible and aligns well with the proof-of-concept nature of the project. Given the initial success of Prototype 1, Prototype 2 was designed as an improved

and functionally enhanced version with a focus on soft-touch handling.

Fin-Ray Concept for Soft Gripping

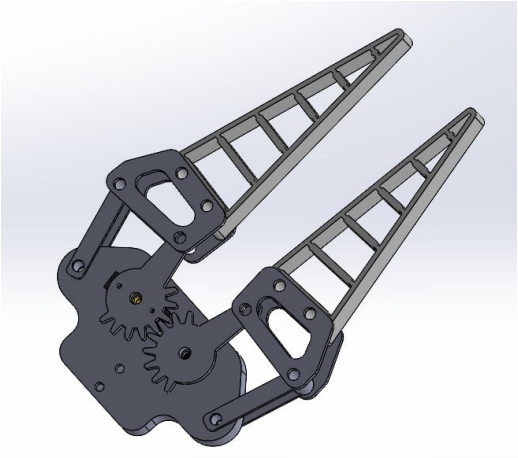


Figure 109: Fin-Ray gripper

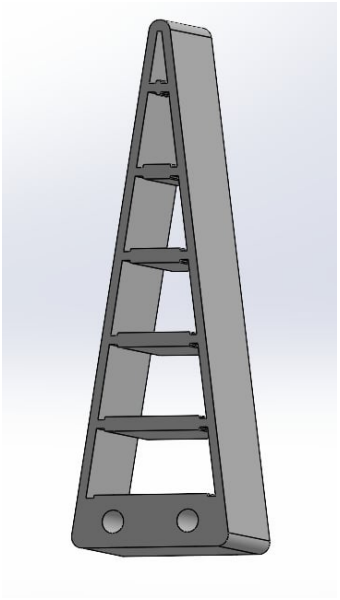
To meet the requirements for handling delicate agricultural produce, such as leafy greens, the Fin-Ray gripping principle was selected for this prototype. This bio-inspired design allows the gripper fingers to adapt passively to the shape of objects, ensuring a gentle grip without the need for integrated sensors. Increased force causes the Fin-Ray fingers to wrap more completely around the object, thereby minimizing the risk of damage to sensitive plant matter.

Design and Material Considerations

As with the initial prototype, SolidWorks was used as the CAD platform for designing all components. The development process began with the most critical elements: the Fin-Ray fingers. These components were designed to be printed in TPU, allowing for the necessary flexibility and deformation properties. The design was inspired by existing Fin-Ray concepts found in literature and online resources, but was scaled and adapted to fit the dimensions of the robotic arm. The finger geometry draws inspiration from human hand proportions to enhance ergonomic compatibility.

The initial finger design featured an internal triangular structure with wall segments to facilitate controlled deformation. To improve flexibility, the wall thickness was reduced near the edges. Two M4 mounting holes were included at the base of each finger to secure them to a connecting plate.

(a) Finger, front



(b) Finger, side

Figure 110: Fin-Ray fingers

Structural and Mechanical Integration

Following the development of the fingers, a solid intermediate mounting plate was designed to connect the fingers to the main body. This plate includes cutouts to reduce weight and material usage while maintaining mechanical strength. The plate is affixed to both the fingers and the body using M4 screws.

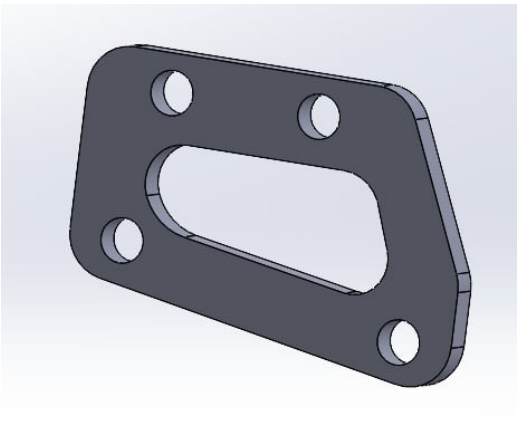


Figure 111: Mounting plate

To secure the mounting plate to the gripper body, support brackets were developed. These symmetrical components extend from the front to the underside of the gripper, providing structural rigidity and minimizing flex during operation. Like the plate, they are fastened using M4 hardware.

(a) Support bracket, front

(b) Support bracket

Figure 112: Support brackets

Gripper Body and Motor Housing

The central body of the gripper serves as the interface for all mechanical and motor components. On the left side of the body, a Ø9.3 mm hole was dimensioned to accommodate the shaft of a servo motor responsible for actuating the fingers. This aperture was precisely sized to allow the motor shaft to pass through and be secured from the front side. On the right, a Ø3 mm hole was included for an auxiliary support shaft, which will be fixed using an M4 screw.

An indent was added on the underside of the gripper body to seat the servo motor, with Ø3.5 mm holes for M4 screws to secure it in place. Additional side holes allow for attachment of the support brackets mentioned earlier.

To integrate the NEMA 11 stepper motor at the base of the gripper, a Ø6 mm hole was included. The hole was shaped to match the motor shaft profile for improved fit and stability. Two Ø3 mm holes were also added from the top to allow screws to secure the shaft directly, providing greater rigidity compared to Prototype 1, which relied solely on a Z-screw.

(a) Gripper body

(b) Gripper body, underside

Figure 113: Fin-Ray gripper body

Gear Mechanism

The gear components responsible for transferring torque from the servo motor to the gripper fingers were subsequently developed. These gears feature teeth modeled using a freehand approach and will be tested for fit and performance. One gear interfaces directly with the servo motor via one center hole of Ø4mm and two Ø1.5 mm holes that align with a fan mount. Due to a 2 mm offset created by this mounting interface, a corresponding 2 mm extruded base was added to the opposing gear to ensure collinearity. Both gear components are secured with M4 screws.

(a) Gearing, motor

(b) Gearing part without motor

(c) Gearing underside

Figure 114: Gearing components

Final Assembly and Validation

All components were assembled in SolidWorks to assess fit and functional alignment. This assembly phase revealed the need for minor adjustments to part lengths and hole placements to ensure proper alignment and fastening. Also, we needed to add two brackets on the sides of the gripper body, as seen in Figure 115, to make sure the gripper will interface with the limit

switches that is connected to the end effector. The final assembly emphasizes modularity, ease of maintenance, and material efficiency.

Although the design closely resembles Prototype 1 in form, significant improvements were made in structural robustness and finger actuation. The current design remains optimized for 3D printing, but could be further refined for metal machining as discussed in later sections 15.1.

Once the design was finalized, STL files were generated, and the 3D printing of individual components began.

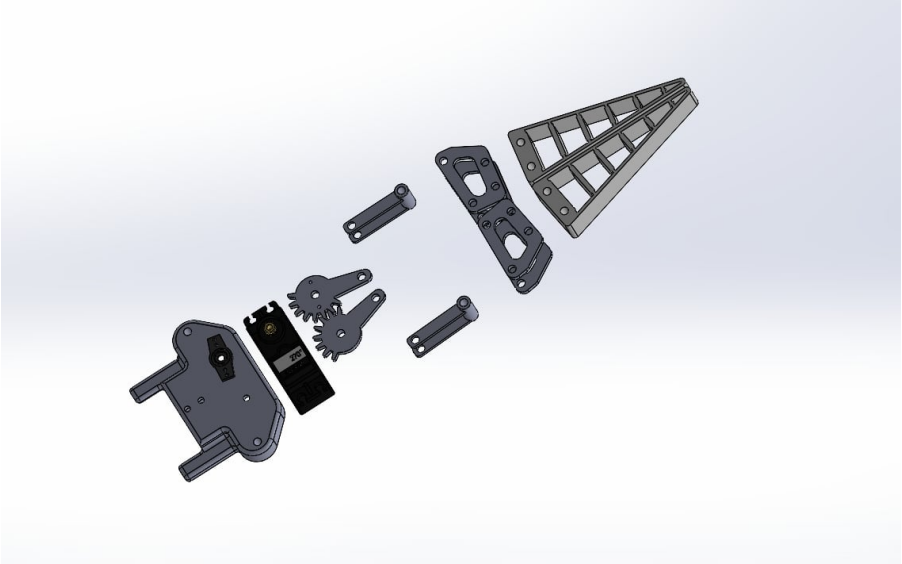


Figure 115: Fin-Ray gripper, exploded view

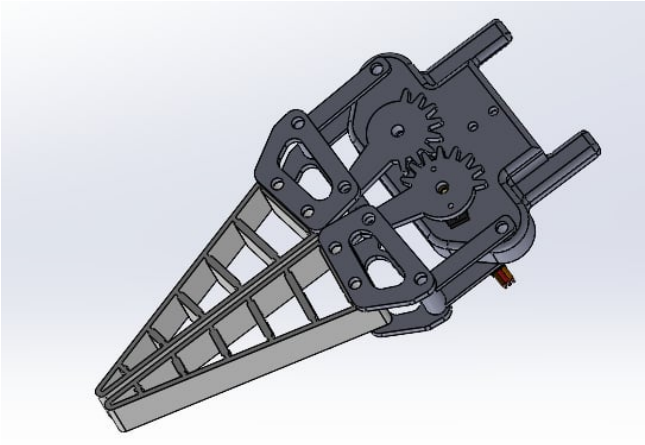


Figure 116: Final Fin-Ray assembly

13.4.4 V2 - Assembly and Initial Testing

SME | *JCDH*

Once the SolidWorks model of the second gripper prototype was completed, the components were 3D printed and subsequently assembled using standard screws. The components fit together well, and the Fin-Ray-inspired gripper was ready for initial testing.

The gripper was mounted onto the end-effector using two Z-screws, which securely fastened it to the shaft of the stepper motor. Functional tests were then conducted using various objects to evaluate the grippers adaptability and performance. One important observation was that the surface of the gripper fingers, printed in TPU, was very smooth. This caused the gripper to slide on smooth objects, reducing its gripping effectiveness. This limitation is addressed further in the Future Work section 15.1, where improvements such as textured or coated finger surfaces are discussed.

Since the robotic system is primarily intended for harvesting lettuce, a practical test was conducted using a Crispi lettuce head. The aim was to evaluate how well the Fin-Ray fingers conform to the organic shape of the lettuce and whether it could be lifted securely. The gripper performed successfully in this test, easily picking up the lettuce. During this experiment, displacement measurements of the gripper fingers were recorded by a caliper (Figure 118) and later used as input for a Finite Element Method analysis, visualizing deformation when gripping a crispi salad. (see Figure 119).

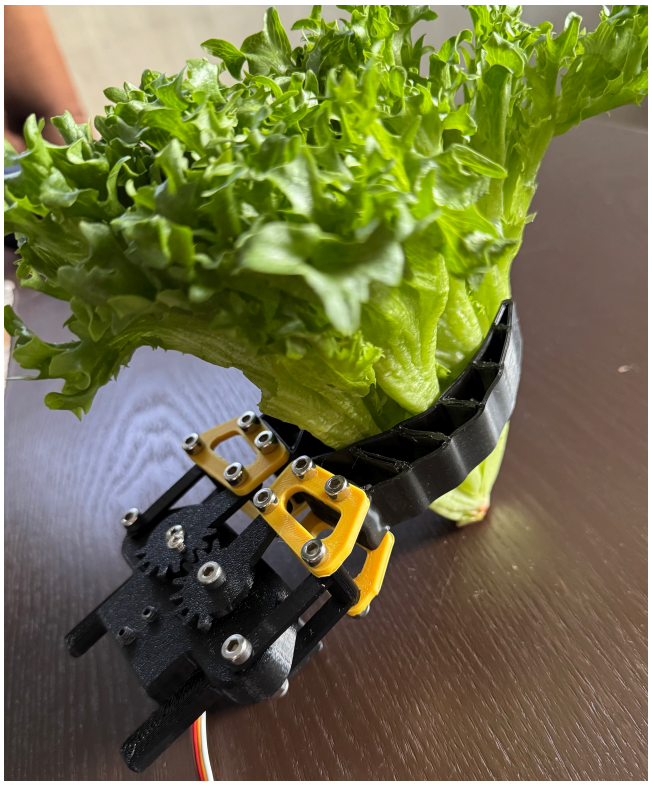


Figure 117: Fin-Ray gripping test

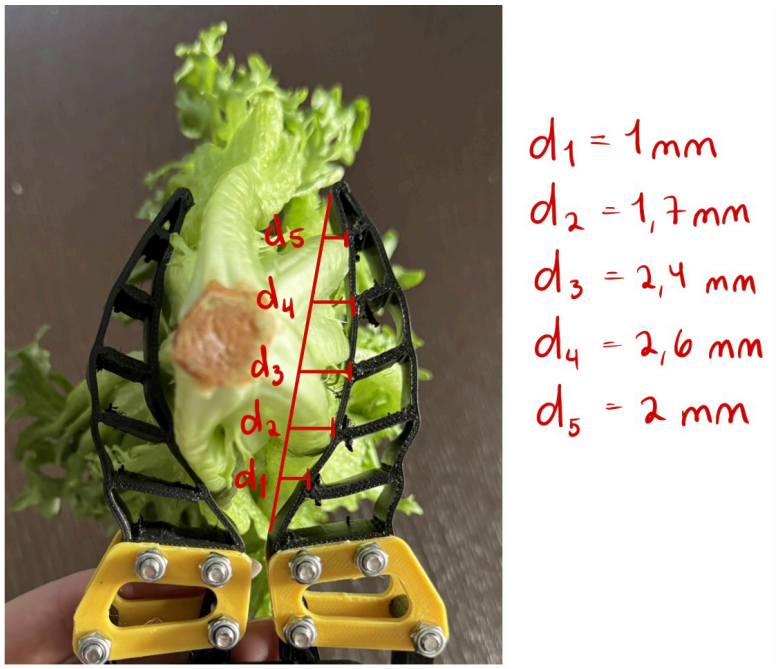


Figure 118: Displacement measurements from practical test

It is worth noting that the TPU material used for the fingers was not included in the default SolidWorks materials library. Therefore, to conduct the FEM study, we relied on empirical data obtained from the physical tests. For accurate stress and deformation analysis, especially in industrial applications, precise material properties should be implemented in FEM simulations. This is crucial to determine a reliable Factor of Safety (FOS), which cannot be fully trusted without correct input data.

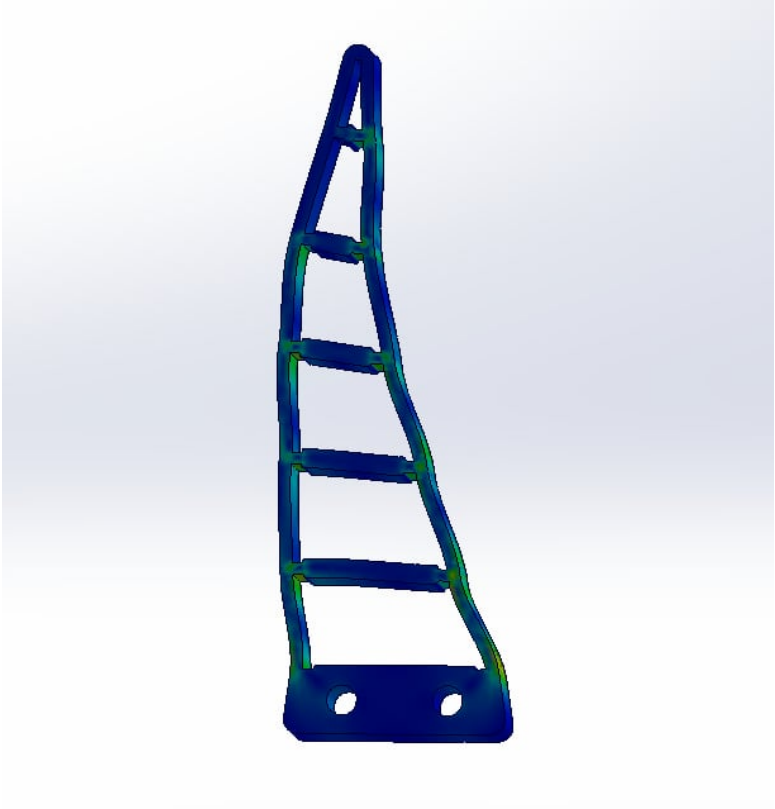


Figure 119: Fin-Ray finger displacement FEM

In addition to the soft gripper analysis, a separate FEM study was carried out on the gear components that actuate the gripper. Before initiating the simulation, hand calculations were performed to determine the forces acting on the gear teeth 64. SolidWorks contact analysis revealed that only two teeth are engaged at any given time. Although the gear consists of a total of eight teeth, the load is shared by just two. The total tangential force was therefore distributed evenly across these two engaged teeth. A detailed breakdown of the force calculations can be found in Figure 64.

These calculated forces were then used as input in SolidWorks for the FEM study. The resulting simulation (Figure 120) highlights the areas of maximum and minimum stress on the gear. Given that the gears are 3D printed in PLA, the FOS (Factor Of Safety) is relatively low and is not suitable for demanding industrial environments. For real-world implementation, these components should ideally be manufactured using machined materials. This point is further discussed in chapter 15.1.

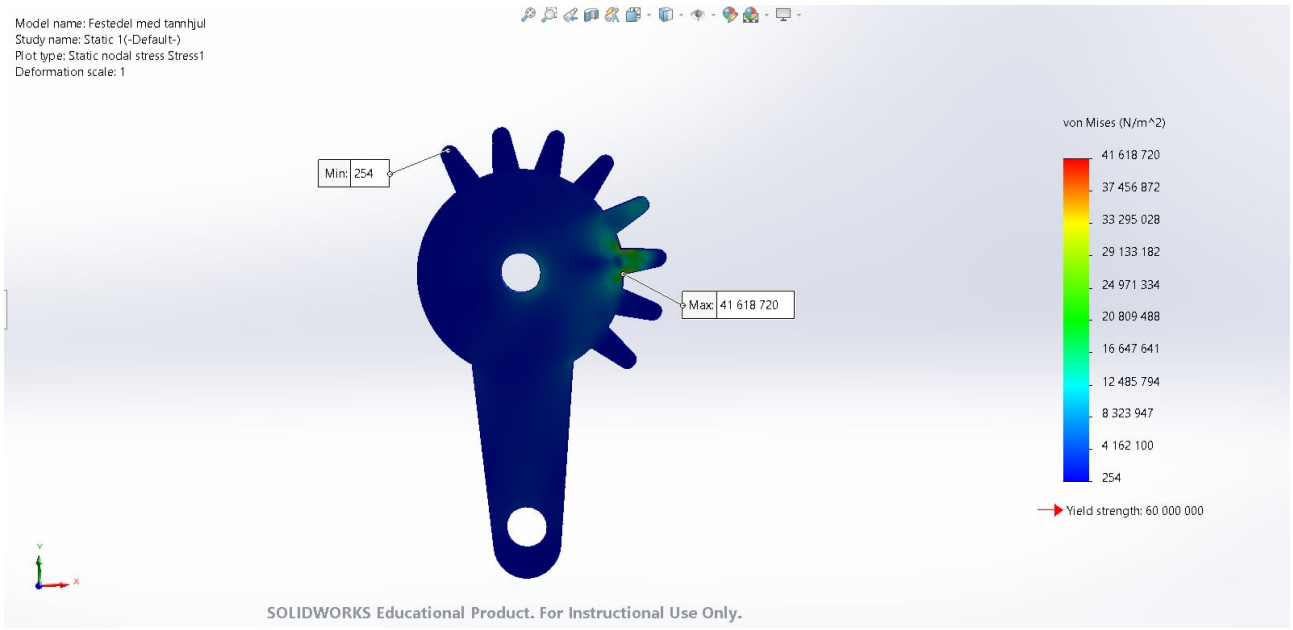


Figure 120: Stress Analysis gear components

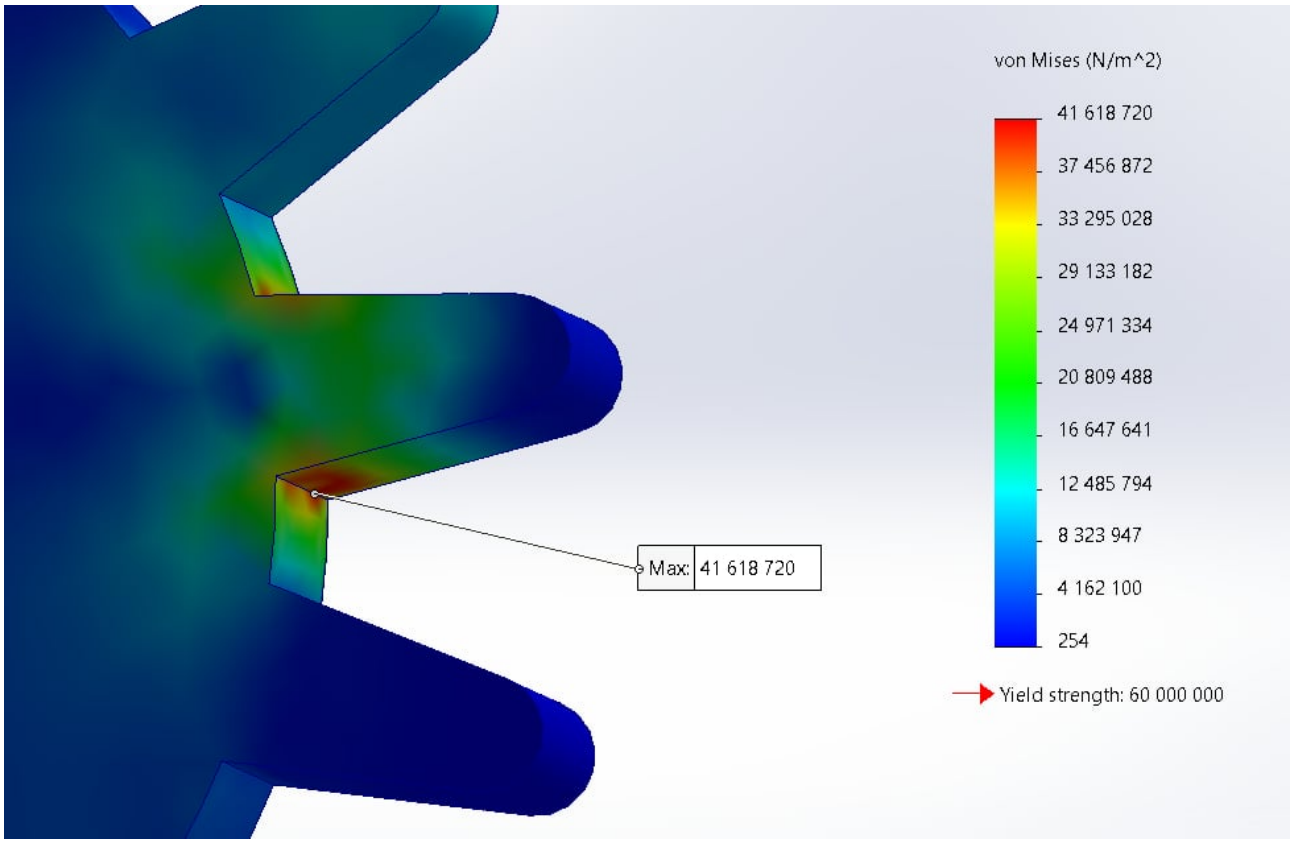


Figure 121: Close up, maximum stress

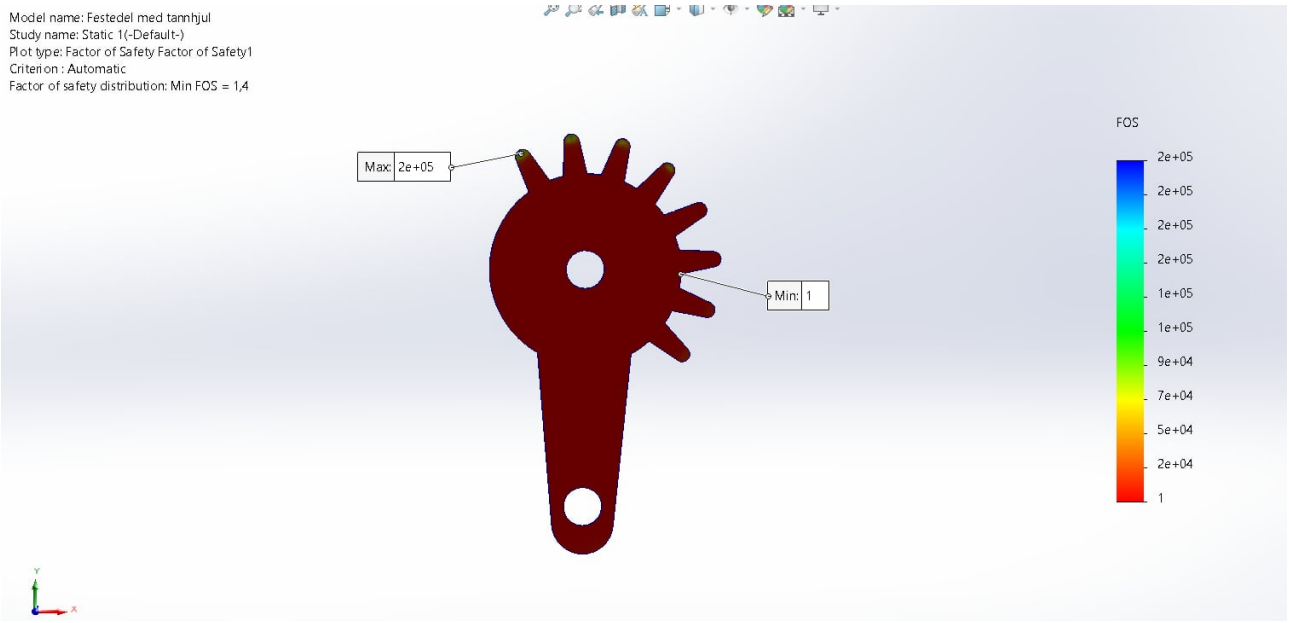


Figure 122: Safety Factor

Since PLA was also not available in the SolidWorks materials library by default, it was manually added. The material properties were sourced online and entered into the software [64].

Conclusion of Initial Testing

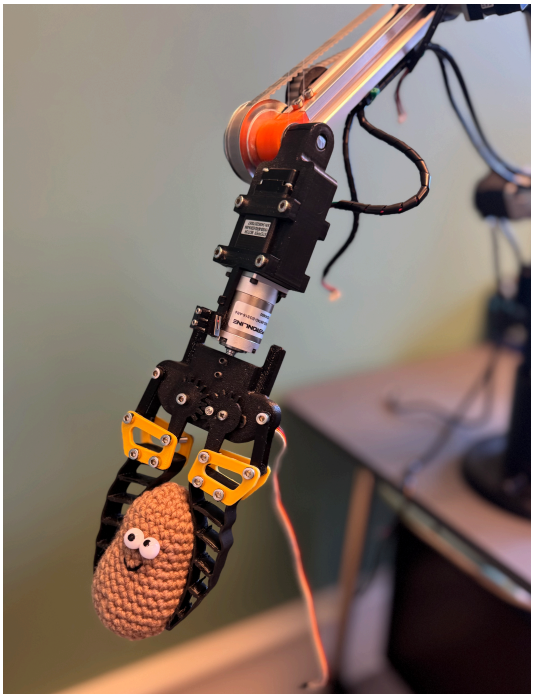
The initial tests demonstrate that the second gripper prototype functions effectively as a proof of concept. The Fin-Ray fingers adapt well to organic shapes and successfully pick and place lettuce heads without causing damage. While some improvements are needed to enhance grip stability and material accuracy in FEM simulations, the prototype serves as a solid foundation for further development.

13.4.5 Physical model

DAB |

Pictures of physical model

(a) Robot Arm



(b) Close up Gripper

Figure 123: Finished Physical Model

13.5 Specifications robot arm prototype

BMR |

The configuration space quantified (built on calculations from section 8.1.1).

Model	**Robot arm prototype - Leafy Automation Bachelor**	
Type	Articulated robot with rotational base	
Degrees of freedom	5 (excl. end effector)	
Axis spec.	**Operating axis**	**Max angle range**
	Base roll (J0)	0˚ ~ 300 ˚
	Shoulder pitch (J1)	0˚ ~ 90 ˚
	Elbow pitch (J2)	0˚ ~ 150 ˚
	Wrist pitch (J3)	-90 ˚ ~ 90 ˚
	Wrist roll (J4)	-90 ˚ ~ 90 ˚
Payload max.	1 kg	
Max speed	20 RPM	
Driving motors	Brushless Stepper motors	
Working range	690 mm max. reach (See figures B & C)	
Mass	7 kg	
Installation	Horizontal mounting (see figure A)	
End effector	Exchangeable	

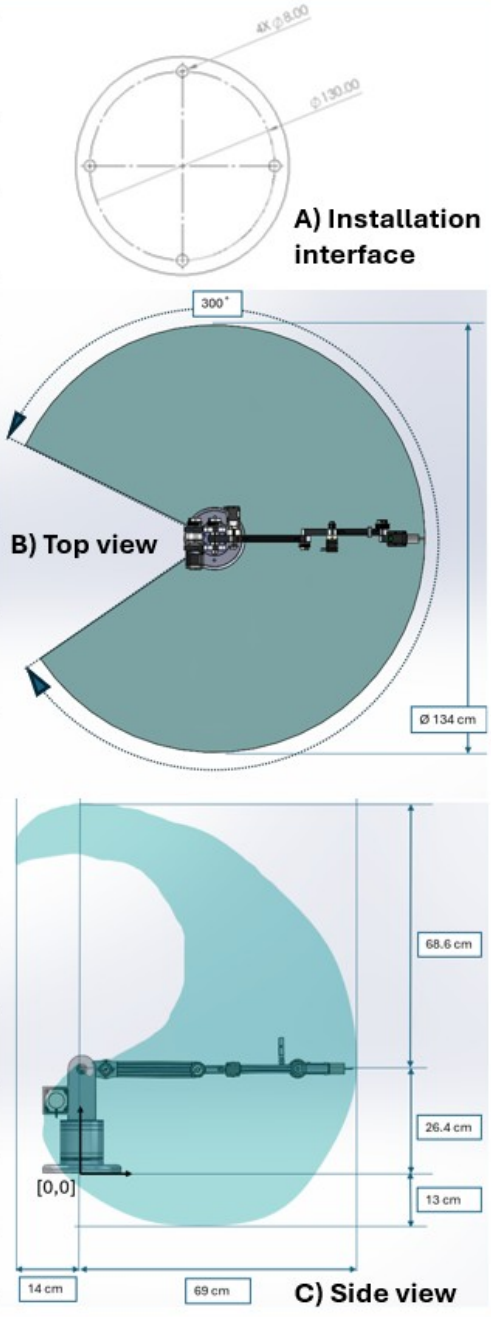


Figure 124: Prototype specification

14 Conclusion

This Bachelor thesis, carried out in collaboration with Hydroplant Technologies AS, aimed to develop a autonomous and versatile robotic system for harvesting leafy greens in farming environments. As a proof-of-concept project, we focused on the design and integration of a robotic arm capable of gentle harvesting and relocating produce such as lettuce. The system is intended to play a vital role within Hydroplant‘s future goal of creating an automated pipeline from seeds to packaged product, contributing to more sustainable and efficient food production.

Throughout the project, our multidisciplinary teams effort was applied across mechanical design, electronics and software systems. Several prototypes were developed and evaluated. Mechanically, a modular robotic arm and multiple gripper designs were implemented and iterated by using assembly techniques and 3D-printed components to allow rapid prototyping and testing. As far as electronics goes, custom PCBs were created to control motors and sensors. While on the software side, the system was partially integrated with ROS2, forming the foundation for future autonomous behaviour. Additionally, initial efforts were made in developing AI-based object recognition and a database for logging plant interactions.

The final prototype demonstrates that the system is viable and can be developed further into a innovative solution for the agricultural automation domain. However, significant work needs to be done to transition the prototype into a fully deployable product. This includes redesigning parts for machining, improving mechanical tolerances and component durability, enhancing software architecture, and verifying system behaviour through comprehensive simulations and field testing. A more detailed roadmap for these improvements is described in the 15.1 Chapter.

This project has not only served as a technical challenge, but also highlighted the potential impact of robotics in future food systems. By reducing manual labour, increasing hygiene and optimizing autonomous harvesters like the one prototyped in this project, could play a central role in transforming the agricultural industry towards more sustainable and scalable practices. The insights, designs, ans experience gained through this proof-of-concept form a strong foundation for Hydroplant Technologies to further develop and refine the system into a production ready solution.

15 Reflection

Throughout the project period the group has had a steep learning curve, both when it comes to the technical aspects of the project and the more administrative sides. Nobody in the group had prior knowledge with robotics, so there was many new concepts and expressions to get familiarized with.

Group collaboration

We had some initial ideas about how we wanted to structure the project work and the collaboration between the different engineering disciplines in our group. Despite different ways of working, we managed to find a good way to collaborate and communicate.

We had some challenges working with a set project model (scrum), and ended up working around a hybrid solution where we choose to adapt some of the elements that worked well for the group, like daily stand up meetings. The expectations from the University, that each student had his/her own work description, sometimes conflicted with the idea of collaborating towards a main goal.

Time planning

The group got the project assignment 6 weeks into the bachelor period and had some startup challenges. As a result of the delayment, many key planning and decision making meetings occurred relatively late in the process, which introduced time management difficulties and increased the pressure on both development and integration phases. We also faced some challenges with task planning and time estimations, mainly due to lack of experience around how much time was needed for the specific tasks. Having a better overview and more milestones could maybe have improved the work flow and integration process.

Group structure

At the onset of this thesis, we decided to arrange ourselves in a flat leadership structure believing that this would encourage creativity and a strong sense of shared ownership. As we now near the end of this experience, we acknowledge that there are mixed feelings about this decision as it did not come without cost. Examples of this are more and longer meetings than we might have needed. Lack of individual accountability also led to tasks being left unattended. It is therefore believed by some members of our team that the flat leadership structure took a toll

on our overall efficiency. Retrospectively, it is considered that splitting areas of responsibilities into clearly defined roles with attached accountability could have been a better hybrid solution.

Hydroplant partnership

For the partnership with our employer Hydroplant Technologies, we have had a really good communication and relationship with them.
The employer was very engaged in the progression of the project, very responsive, and always tried to meet our requests.

15.1 Future work

This chapter will outline the recommendations for future development of our project, and the robotic system. Covering mechanical design, electrical integration and data-related improvements. While the current prototype demonstrates a proof-of-concept, additional development is required to enhance the performance, durability and application range. We will go through proposed directions that will guide this project from a proof-of-concept prototype, to a fully operational and scalable solution for agricultural automation.

15.1.1 Future work - Base

BR |

Future work for the base includes:

- adjusting the design for machining in the actual material (for example AISI 1035 stainless steel).
- Test the parts and assembly with FEA.
- Choose suitable bearings and bearing arrangements.
- fine-tune the shaft and housing design to fit the chosen bearings.

To see some of the initiating thoughts around these topics, see appendix D

15.1.2 Future work - Joints

DAB | *SME*

Creating hollow shafts for wire to pass through. This will eliminate wire bending and will only twist the pairs of wires together which lessens the strain.

Tensioner mechanism between CNC part and aluminium profile. It is quite crucial since it will eliminate play between the parts which will lead to reduced tolerance stacking and in turn increase accuracy of the robot overall.

Further FEA testing of parts. see appendix further work in appendix D

After these improvements, it would be nice to see and test actual CNC parts.

Casing for the arm is more for the aesthetic purposes, and also so that the belts and frame parts are protected against gunk, dust and more depending on the environment it will be used inn.

15.1.3 Future work - Gripper

SME | *JCDH*

Transitioning from Prototyping to Production

Although the current gripper prototypes serve as effective proof-of-concept models, additional work is necessary to transition the system into a production-ready state suitable for real-world agricultural automation. The following areas have been identified for future improvement:

- **Machined Components for Strength**

While 3D printed PLA and TPU parts are sufficient for testing and demonstration, they are not ideal for the strict demands of agricultural environments. Components such as the gripper body, gear assemblies, and support structures would benefit from being machined in aluminum or stainless steel to enhance durability, precision, and resistance to environmental factors like humidity, temperature, and mechanical shock.

- **Improved Gear Mechanisms**

The current gearing system was created manually and may not provide optimal efficiency or alignment. Future iterations should include precision-designed gear profiles and potentially integrate bearings or bushings to reduce friction and wear. Simulation tools like the Finite Element Method FEM and motion studies in SolidWorks can also be used to improve the testing and verification of stress distribution and torque transmission.

- **Sensor Integration**

Although the Fin-Ray fingers provide passive compliance, future versions could integrate tactile or force sensors to enable active feedback and adaptive control. This would enhance the robots ability to handle a wider variety of crops with varying fragility and shapes, improving reliability in dynamic environments.

- **Surface texture enhancement for Fin-Ray fingers**

During testing of the Fin-Ray fingers, it was observed that the TPU print has a very smooth surface finish, which limits the gripping ability on smooth objects. While the current surface works for soft and deformable objects like lettuce, it lacks sufficient friction for handling a wider range of agricultural produce. Future iterations should explore the integration of textured coatings, surface patterns, or friction-enhancing materials such as silicone overlays or rubber based inserts. These modifications would broaden the gripperṫs applicability beyond lettuce and improve overall grip reliability.

- **Integration of bearing support**

In the current gripper design, the gear component that is not directly connected to the servo motor is secured only with a screw, which may be sufficient for short-term prototyping and testing, but lacks the mechanical stability required for long-term use. To reduce wear, friction and axial play during rotation, it is recommended that future iterations incorporate a bearing system for this passive gear. Proper bearing support would significantly improve durability, ensure smoother operations under load, and enhance the overall reliability of the gear mechanism in further gripper development.

- **Calculations and simulations**

Future work should also include detailed calculations of gripping force and torque requirements to ensure the gripper can reliably handle various payloads. These calculations should be performed both analytically and by using Finite Element Method (FEM) simulations to verify stress distribution and structural integrity under different loading conditions.

Particular attention should be given to the potential for bending in the stepper motor shaft, as this could impact precision and long-term durability. Incorporating these analyses will contribute to the optimization of component sizing, material selection, and overall gripper performance.

- **Environmental Testing**

The prototypes have yet to undergo field testing under realistic agricultural conditions. Evaluating performance in outdoor settingsexposed to dust, moisture, and temperature variationwill be essential to validate long-term stability and identify any points of mechanical or electronic failure.

15.1.4 Future work - Electronics

VMN |

- PCB redesign
  - EMC
  - Dual package MOSFETs for smaller PCB
  - Smaller component packages (resistors, capacitors)
- PCB Assembly
- PCB Testing
- Add option for UART or SPI
- Tune the driver for each motor

15.1.5 Future work - Software

JCDH | *SME*

- Complete AI object detection model with custom dataset.
- Complete Database work for logging plant identification logs.
- Connect up HMI to robotics.
- Transition to a Real Time Operating System.

15.1.6 Future work - Architecture and robotics

EG |

Upgrading the ROS2 Architecture. Once initial proof-of-concept testing is complete, the architecture ought to be upgraded with ROS2 services and actions. We suggest the following adaptations:

Synchronous Task Negotiation via Service We propose that instead of publishing the next task on a topic, that a service is called with a custom leafy_msgs/TaskGoal. This way, the Task Planner sends out a request and waits until the service provider confirms it received and accepted the goal. This handshake prevents loss or out-of-order tasks and lets reject invalid requests before starting execution.

Motion via Action We propose that the Motion Planner node is set up as an action server. This would replace the one-way /planned_trajectory topic as the client instead sends a trajectory_msgs/JointTrajectory goal and receive periodic progress feedback, as well as a final result indicating success or failure. This upgrade would also support action cancellation in response to events such as obstacle detection. This upgrade would improve safety and flexibility for the system.

Benefits Introducing services for important exchanges and actions for long-running tasks aligns well with ROS2 design guidelines. It introduces some added setup (service and action servers/-clients) but pays dividends in robustness, explicit error handling, and future extensibility (e.g., dynamic replanning or conditional preemption).

These upgrades would naturally call for some adaptations on the Arduino Core. Our initial thoughts suggest the following:

New MQTT Topics: Add the following subscriptions to initMQTT():

- /leafy_a/execute_trajectory/goal
- /leafy_a/execute_trajectory/cancel alongside existing command topics.

Cancel Handling: handleTrajectoryGoal() or handleTrajectoryCancel() should be introduced to the Communication Manager message dispatching.

Stop motors stopAllMotors() should be introduced to the Motor Driver module.

Feedback updateMotors() needs to be adapted to provide /leafy_a/execute_trajectory/feedback via (publishStatus()).

Software libraries to consider:

MoveIt 2 This library is the used as an industry-standard for motion planning, kinematics, collision

References

[1] "Hivemind — itfag.usn.no," https://itfag.usn.no/grupper/D01-23/, [Accessed 29-03-2025].

[2] V. Kumar, "Robot geometry and kinematics," accessed: 2024-05-08. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d875497e3d8e2f313112d6d20426ba9986b90dc9

[3] Stepperonline, "Nema 17 stepper motor," accessed: 2024-05-02. [Online]. Available: https://www.omc-stepperonline.com/nema-17-stepper-motor-l-39mm-gear-ratio-10-1-high-precision-planetary-gearbox-17hs15-1684s-hg10

[4] H. D. Young, "Sears and zemansky's university physics." Place of publication not identified, 2016.

[5] SKF, "6006-2rs1," accessed: 2024-05-02. [Online]. Available: https://www.skf.com/group/products/rolling-bearings/ball-bearings/deep-groove-ball-bearings/productid-6006-2RS1

[6] "ISO/IEC/IEEE 29148:2018 — iso.org," https://www.iso.org/standard/72089.html, [Accessed 18-05-2025].

[7] A. Sols, *Systems engineering: theory and practice.* Madrid Universidad Pontificia Comillas 2014, 2014, oCLC: 892528234.

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, fourth edition. ed., ser. The SEI series in software engineering. Boston: Addison-Wesley, 2022.

[9] S. Mbugua, J. Korongo, and S. Mbuguah, "On software modular architecture: Concepts, metrics and trends," *International Journal of Computer & Organization Trends*, vol. 10, pp. 3–10, 03 2022.

[10] Raspberry Pi Ltd, "Raspberry pi 5 raspberry pi," 2023, accessed: 2024-05-02. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[11] Open Source Robotics Foundation, "Installing ros 2 on raspberry pi ros 2 documentation: Jazzy," 2024, accessed: 2024-05-02. [Online]. Available: https://docs.ros.org/en/jazzy/How-To-Guides/Installing-on-Raspberry-Pi.html

[12] Arduino AG, "Arduino uno r4 wifi," 2023, accessed: 2024-05-02. [Online]. Available: https://store.arduino.cc/products/uno-r4-wifi

[13] S. T. Mbugua, J. Korongo, and S. Mbuguah, "On software modular architecture: Concepts, metrics and trends," *International Journal of Computer & Organization Trends*, vol. 12, no. 1, pp. 3–10, 2022. [Online]. Available: https://www.researchgate.net/publication/360726289_On_Software_Modular_Architecture_Concepts_Metrics_and_Trends

[14] A. Bonci, F. Gaudeni, M. C. Giannini, and S. Longhi, "Robot operating system 2 (ros2)-based frameworks for increasing robot autonomy: A survey," *Applied sciences*, vol. 13, no. 23, p. 12796, 2023.

[15] Open Robotics. (2025) ROS 2 documentation (jazzy). Accessed: 2025-05-14. [Online]. Available: https://docs.ros.org/en/jazzy/

[16] The Apache Software Foundation. (2004) Apache license, version 2.0. Accessed: 2025-05-14. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.html

[17] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[18] Open Robotics. (2020) Ros noetic api: `sensor_msgs/Image` message. Accessed: 2025-05-15. [Online]. Available: https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Image.html

[19] ——. (2025) Ros index. Accessed: 2025-05-15. [Online]. Available: https://index.ros.org/?search_packages=true

[20] Udemy. (2025) ROS2 for Beginners [online course]. Accessed: 2025-05-15. [Online]. Available: https://www.udemy.com/course/ros2-for-beginners/learn/lecture/20260476#overview

[21] Open Robotics. (2018) Ros melodic api: `std_srvs/SetBool` service. Accessed: 2025-05-15. [Online]. Available: https://docs.ros.org/en/melodic/api/std_srvs/html/srv/SetBool.html

[22] ——. (2025) Understanding ros 2 services. Accessed: 2025-05-15. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[23] ——. (2020) Ros noetic api: `control_msgs/FollowJointTrajectory` action. Accessed: 2025-05-15. [Online]. Available: https://docs.ros.org/en/noetic/api/control_msgs/html/action/FollowJointTrajectory.html

[24] ——. (2020) Understanding ros 2 actions. Accessed: 2025-05-15. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

[25] R. G. Budynas, "Shigley's mechanical engineering design," USA, 2021.

[26] "Online CNC Machining Service - JLCCNC." [Online]. Available: https://jlccnc.com/

[27] "ZeptoBit AS." [Online]. Available: https://www.zeptobit.com/index.php?product=10

[28] [Online]. Available: https://www.maedler.de/

[29] Core Electronics. (2025) Df metal geared 15kg standard servo 270ř (dss-m15s). Accessed: 2025-05-18. [Online]. Available: https://core-electronics.com.au/df-metal-geared-15kg-standard-servo-270-dss-m15s.html

[30] *TMC5160A Stepper Motor Driver*, Analog Devices, 2023, available at https://www.analog.com/media/en/technical-documentation/data-sheets/TMC5160A_datasheet_rev1.18.pdf, Revision V1.18.

[31] *TMC5160-BOB Evaluation board*, Analog Devices, 2021, available at https://www.analog.com/media/en/technical-documentation/data-sheets/TMC5160-BOB_datasheet_rev1.10.pdf, Revision V1.10.

[32] ardalis, "Overview of ASP.NET Core MVC — learn.microsoft.com," https://learn.microsoft.com/nb-no/aspnet/core/mvc/overview, [Accessed 14-05-2025].

[33] "GitHub - RobotWebTools/roslibjs: The Standard ROS JavaScript Library — github.com," https://github.com/RobotWebTools/roslibjs, [Accessed 15-05-2025].

[34] "Werkzeug - Werkzeug Documentation (3.1.x) — werkzeug.palletsprojects.com," https://werkzeug.palletsprojects.com/en/stable/, [Accessed 06-04-2025].

[35] "SQLite Home Page — sqlite.org," https://www.sqlite.org, [Accessed 30-03-2025].

[36] "Datatypes In SQLite — sqlite.org," https://www.sqlite.org/datatype3.html, [Accessed 30-03-2025].

[37] "SQLite Autoincrement — sqlite.org," https://www.sqlite.org/autoinc.html, [Accessed 30-03-2025].

[38] "CREATE TABLE — sqlite.org," https://www.sqlite.org/lang_createtable.html, [Accessed 30-03-2025].

[39] M. McCauley, "AccelStepper: Flexible stepper motor control library for arduino," https://www.airspayce.com/mikem/arduino/AccelStepper/, 2025, accessed: 2025-05-17.

[40] hydroplantno, "PubSubClient: Arduino mqtt client library (fork)," https://github.com/hydroplantno/pubsubclient, 2025, accessed: 2025-05-17.

[41] A. SA, "WiFiS3: Arduino wi-fi library for the uno r4 wifi," https://github.com/arduino/ArduinoCore-renesas/tree/main/libraries/WiFiS3, 2025, accessed: 2025-05-17.

[42] ——, "Servo: Arduino library for hobby servo control," https://docs.arduino.cc/libraries/servo/, 2025, accessed: 2025-05-17.

[43] Y. H. Hee, M. K. Ishak, M. S. M. Asaari, and M. T. A. Seman, "Embedded operating system and industrial applications: a review," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 3, pp. 1687–1700, June 2021. [Online]. Available: https://beei.org/index.php/EEI/article/view/2526

[44] E. Team, "Ensuring reliable iot device connectivity: Best practices for mqtt client auto-reconnection," https://www.emqx.com/en/blog/mqtt-client-auto-reconnect-best-practices, Sep. 12 2024, accessed: 2025-05-17.

[45] Arduino, "BlinkWithoutDelay: Arduino tutorial on non-blocking timing," https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay, 2025, accessed: 2025-05-17.

[46] D. McAulay, "SerialCommandExample: Arduino serial command parsing library example," https://github.com/kroimon/Arduino-SerialCommand/blob/master/examples/SerialCommandExample/SerialCommandExample.pde, 2025, accessed: 2025-05-17.

[47] M. McCauley, "AccelStepper multistepper example," https://www.airspayce.com/mikem/arduino/AccelStepper/MultiStepper_8pde-example.html, 2025, accessed: 2025-05-17.

[48] Y. . Brainy-Bits, "Homing stepper motors using the accelstepper library," https://www.brainy-bits.com/post/homing-stepper-motors-using-the-accelstepper-library, 2020, accessed: 2025-05-17.

[49] J. Rullan, "StateMachine arduino example," https://github.com/jrullan/StateMachine/tree/master/examples/arduino_state_machine, 2025, accessed: 2025-05-18.

[50] "ESP32-CAM camera development board | 安信可科技 — docs.ai-thinker.com," https://docs.ai-thinker.com/en/esp32-cam, [Accessed 11-05-2025].

[51] "GitHub - espressif/esp32-camera — github.com," https://github.com/espressif/esp32-camera, [Accessed 11-05-2025].

[52] E. Systems, "arduino-esp32: Arduino core for the esp32," https://github.com/espressif/arduino-esp32, 2025, accessed: 17.03.2025.

[53] "ESP32-CAM: The Complete Machine Vision Guide — blog.arducam.com," https://blog.arducam.com/esp32-machine-vision-learning-guide/, [Accessed 14-05-2025].

[54] V. D. Community, "Field of View - Valve Developer Community — developer.valvesoftware.com," https://developer.valvesoftware.com/wiki/Field_of_View, [Accessed 28-04-2025].

[55] "OpenCV: Camera Calibration — docs.opencv.org," https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html, [Accessed 11-05-2025].

[56] "Folding@home - Fighting disease with a world wide distributed super computer. — foldingathome.org," https://foldingathome.org, [Accessed 29-03-2025].

[57] "openai/clip-vit-base-patch32 Hugging Face — huggingface.co," https://huggingface.co/openai/clip-vit-base-patch32, [Accessed 13-05-2025].

[58] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[59] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," *arXiv:2406.09414*, 2024.

[60] "depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf Hugging Face — huggingface.co," https://huggingface.co/depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf, [Accessed 13-05-2025].

[61] "Tasks - Hugging Face — huggingface.co," https://huggingface.co/tasks, [Accessed 07-05-2025].

[62] J. Jann, O. Drevelle, X. G. Chen, M. Auclair-Gilbert, G. Soucy, N. Faucheux, and L.-C. Fortier, "Rapid antibacterial activity of anodized aluminum-based materials impregnated with quaternary ammonium compounds for high-touch surfaces to limit transmission of pathogenic bacteria," *RSC Advances*, vol. 11, no. 60, pp. 38 172–38 188. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9044312/

[63] "Annin Robotics." [Online]. Available: https://www.anninrobotics.com

[64] National Library of Medicine. (2025) Mechanical properties of 3d-printing polylactic acid parts subjected to bending stress and fatigue testing. Accessed: 2025-05-18. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC6926899/

[65] DornaRobotics, "Types of robot grippers and their applications," 13. october, 2023. [Online]. Available: https://dorna.ai/blog/types-of-grippers-for-robots/

[66] Alan Brown, "Seven big advances in soft robotic grippers," 22. April, 2020. [Online]. Available: https://www.asme.org/topics-resources/content/seven-big-advances-in-soft-robotic-grippers

[67] Lucia Beccai, "A deep learning method for vision based force prediction of a soft fin ray gripper using simulation data," 25. May, 2021. [Online]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.631371/full

[68] Source Robotics, "Soft robotic grippers - fin ray effect," 11. December, 2024. [Online]. Available: https://source-robotics.com/blogs/blog/soft-robotic-grippers-fin-ray-effect

[69] Yahia A. AboZaid, Mahmoud T. Aboelrayat, Irene S. Fahim, Ahmed G. Radwan, "Soft robotic grippers: A review on technologies, materials, and applications," 17.April 2024.

[Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0924424724003741

[70] H. Face, "Image classification," 2025, accessed: 29.03.2025. [Online]. Available: https://huggingface.co/tasks/image-classification

[71] "What is Object Detection? - Hugging Face — huggingface.co," https://huggingface.co/tasks/object-detection, [Accessed 15-04-2025].

[72] Unsplash, "Photo by Bhong Bahala on Unsplash — unsplash.com," https://unsplash.com/photos/a-couple-of-people-that-are-walking-down-a-street-npv1LcdKOqY, [Accessed 16-04-2025].

[73] R. 100, "lettuce pallets dataset," https://universe.roboflow.com/roboflow-100/lettuce-pallets , may 2023, visited on 2025-05-18. [Online]. Available: https://universe.roboflow.com/roboflow-100/lettuce-pallets

[74] "Deed - Attribution 4.0 International - Creative Commons — creativecommons.org," https://creativecommons.org/licenses/by/4.0/, [Accessed 18-04-2025].

[75] F. Ciaglia, F. S. Zuppichini, P. Guerrie, M. McQuade, and J. Solawetz, "Roboflow 100: A rich, multi-domain object detection benchmark," 2022.

[76] Ultralytics, "YOLO Data Augmentation — docs.ultralytics.com," https://docs.ultralytics.com/guides/yolo-data-augmentation/, [Accessed 11-05-2025].

[77] H. Face, "Depth estimation," 2025, accessed: 29.03.2025. [Online]. Available: https://huggingface.co/tasks/depth-estimation

[78] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, "Depth anything: Unleashing the power of large-scale unlabeled data," 2024. [Online]. Available: https://arxiv.org/abs/2401.10891

[79] R. P. Ltd, "Raspberry Pi 5," https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf, [Accessed 01-04-2025].

[80] "Tokenizer — huggingface.co," https://huggingface.co/docs/transformers/en/main_classes/tokenizer, [Accessed 11-04-2025].

[81] "time Time access and conversions — docs.python.org," https://docs.python.org/3/library/time.html, [Accessed 13-04-2025].

[82] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," 2020.

[83] "MySQL :: MySQL 8.4 Reference Manual :: 1.2.1 What is MySQL? — dev.mysql.com," https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html, [Accessed 06-04-2025].

[84] "DB Browser for SQLite — sqlitebrowser.org," https://sqlitebrowser.org, [Accessed 02-04-2025].

[85] "Home — opencv.org," https://opencv.org, [Accessed 15-05-2025].

[86] "PlantCV — plantcv.org," https://plantcv.org, [Accessed 15-05-2025].

[87] R. v. HSV for Computer Vision | Rehan Guha Portfolio & Blog, "RGB v. HSV for Computer Vision — rehanguha.github.io," https://rehanguha.github.io/articles/rbg-vs-hsv-for-computer-vision, [Accessed 06-05-2025].

[88] P. D. Team, "Watershed Segmentation - PlantCV — plantcv.readthedocs.io," https://plantcv.readthedocs.io/en/stable/watershed, [Accessed 04-05-2025].

[89] "OpenCV: Image Segmentation with Watershed Algorithm — docs.opencv.org," https://docs.opencv.org/4.x/d3/db4/tutorial_py_watershed.html, [Accessed 08-05-2025].

[90] "OpenCV: Create calibration pattern — docs.opencv.org," https://docs.opencv.org/4.x/da/d0d/tutorial_camera_calibration_pattern.html, [Accessed 29-04-2025].

[91] "OpenCV: Camera Calibration and 3D Reconstruction — docs.opencv.org," https://docs.opencv.org/4.11.0/d9/d0c/group__calib3d.html, [Accessed 09-05-2025].

[92] S. Josefsson, "RFC 7914: The scrypt Password-Based Key Derivation Function — datatracker.ietf.org," https://datatracker.ietf.org/doc/html/rfc7914, [Accessed 08-04-2025].

Bibliography

[Bib1] H. Ott, *Electromagnetic Compatibility Engineering*, 1st ed. Wiley, 2009. [Online]. Available: libgen.li/file.php?md5=5afbcaac73ac4851f5f53d5513350fd3

[Bib2] M. O. Hara, *EMC at Component and PCB Level*, 1st ed. Chantilly: Elsevier Science & Technology, 1998.

Appendices

Appendix A

Requirements earlier work

1 Requirements

This section is still under development and should only be seen as collaborative notes and under no circumstances as finished work at this current stage.

Introducing Requirement development
Talk about the importance of requirements and why they must be customer and stakeholder led, and well defined (quantifiable etc, for validation, verification..). Something worthwhile in one of the ISO standards on this? ISO 15288, IEEE 29148?
Talk about how requirements should guide the design process. Follow the main principles from Sols book, i.e. talking about problem domain (user stories, use cases etc..) and solution domain (system requirements. Note the step-by-step process on page 139 that involves stakeholder requirements guiding us towards selecting the preferred design concept.
Stakeholder→Stakeholder requirements → System requirements → Verification methods

Requirements may be categorised as follows:

- Product requirements
- Process requirements
- External requirements
- (this canshould? be adapted to our project needs.)

How will we be developing requirements?

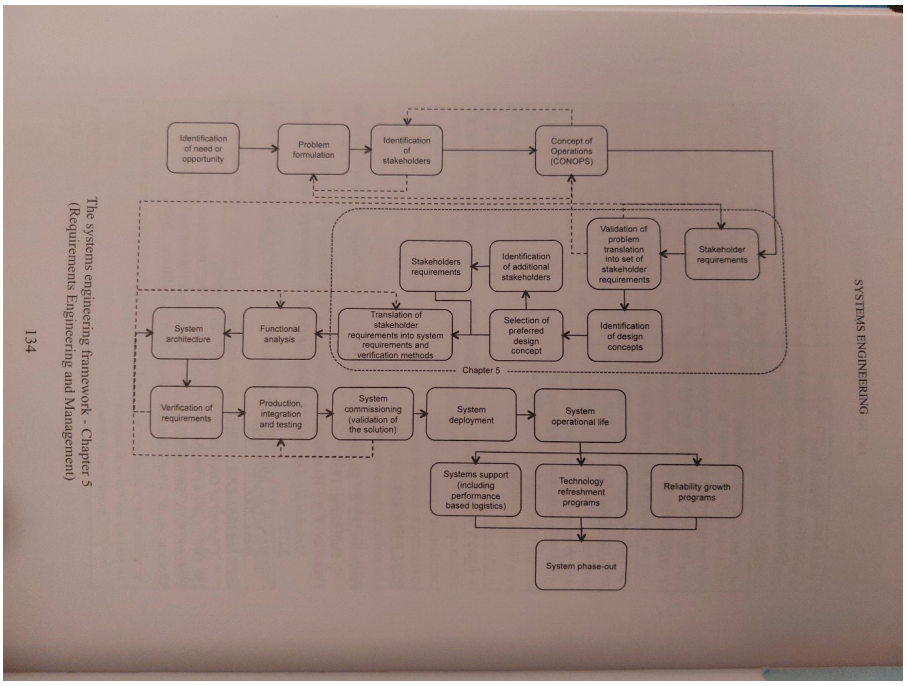


Figure A.1: Taken from page 134 in Alberto Sols' book will adapt.

Talk a about stakeholders and their role in helping us develop our requirements. (More on this in a later chapter)

- How were building our requirements hierarchy from a structural sense (Sols and the ISO/IEEEs)
  - Problem domain, understanding the problem, written in language suitable for non technical stakeholders:
    User stories →Use cases
  - Solution domain
    - System requirements (Acceptance criteria?), Verification and Validation testing.
    - For high level requirements in particular, remember to write about the feedback process w/ customer/key stakeholders, to ensure were all on the same page etc.
  - Traceability. Being able to trace any requirement all the way up to stakeholders and their needs?
  - Process for drilling down into system specification requirements from top level use cases
  - Categories of priority (A, B, C).
  - Categories for easier reading: Functional, non-functional, performance, constraints, etc
  - Perhaps mention about requirement characteristics ref Sols pg. 148. But this might be overkill?

– Traceability. Being able to trace any system requirement all the way up to stakeholder need

1.0.1 How will we go about finding requirements?

EG |

- Stakeholder analysis (refer to that chapter)
- Visiting leafy green producers
- Learning from our key stakeholder; HP Technologies
- Documentation dive
- Literature review helpful here?
- Perhaps mention about consideration to food safety and machine regulations but this is probably covered in stakeholder analysis

SH-ID	US-ID	User story	UC- ID	Use case	REQ-ID	System Requirement	Priority
		As a farmer I want the system to recognise plants so that it can perform optimal handeling of the plant		not damaging		The "gripper" shall not damage the leaves	
				3 different lettuce types		The system shall recognize salad type A,B,C	
				loosing leaves (plant biology)		classify?	
				placement		The system shall	
				håndtere 3 ulike salater (kjerne, løse blader, osv)		The system shall be able to handle/lift? salad A,B,C	
		As a farmer, I want the system to harvest different types of leafy greens, so that I can improve my product offering.		bytte modus for høsting a s1, s2, s3			
		As a farmer I want the system to harvest salads autonomously so that manual labor is not needed					

Figure A.2: Draft Requirement Matrix

- SH-ID: Stakeholder ID
- US-ID: User Story ID
- UC-ID: Use Case ID
- REQ-ID: Requirement ID

A.2

1.0.2 Verification and Validation

EG |

- Talk about how well go about this.

- Story cards and acceptance criteria? Not yet defined

Initial ideas for Acceptance criteria summary (MUST be worked more on!)

Requirement ID	Requirement description	Acceptance criteria	Priority (A, B, C)	Verification method	Responsible team/person	Status

Status descriptions	
Pending	The requirement is documented, but no testing or validation has started yet
In Progress	The requirement is being tested, results are being collected
Passed	The requirement has met all the acceptance criteria and is considered verified
Failed	The requirement has not met one or more of the acceptance criteria, and needs improvement
Blocked	The requirement cannot be tested due to dependencies (hardware or software not ready, component missing etc)

Figure A.3: Draft Acceptance criteria matrix

V&V testing

Test case ID	Requirement ID	Test Description	Test method	Test criteria (Pass/fail)	Responsible team/person	Test status

Status descriptions	
Pending	The requirement is documented, but no testing or validation has started yet
In Progress	The requirement is being tested, results are being collected
Passed	The requirement has met all the acceptance criteria and is considered verified
Failed	The requirement has not met one or more of the acceptance criteria, and needs improvement
Blocked	The requirement cannot be tested due to dependencies (hardware or software not ready, component missing etc)

Figure A.4: Draft Verification and Validation matrix

1.0.3 Current state of requirements

EG | *BMR*

Disse tabellene legges inn for å vise at requirements jobbes med, men vi har ikke rukket å oppdatere videre med delen i rapporten før innlevering. Det gjøres forøvrig oppmerksom om at ID-nummereringen her ikke er riktig. Og sist, vi har en egen tabell hvor vi legger inn forslag til nye requirements mens vi jobber med prosjektet, som tas opp for diskusjon i fellesskap.

User story ID	Short title	User Story	Comments
US-01	Autonomous harvesting	As a farmer, I want the harvesting system to harvest plants autonomously so that I can reduce the need for manual labor.	
US-02	Plant regocnition	As a farmer, I want the harvesting system to classify plants so that it can optimise harvesting for different plant types and handling needs.	
US-03	Optimal handling	As a farmer, I want the system to optimise handling of each plant, so that I can deliver produce meeting the uniformity and quality expectations of my customers.	
US-04	Safety and efficiency	As a farmer, I want the harvesting system to operate efficiently and safely to minimise risk of damages to personnel, equipment or plants.	Efficiency as a separate one?

Figure A.5: User stories

User story short title: Autonomous harvesting

Use Case ID	Use Case	Requirement ID	Requirement
UC-001	System requirements	**REQ**-001-001	The robotic arm shall process X plants per minute.
UC-002	The working area specifications	**REQ**-001-0001	All system operations shall take place in a circular area of Ø1200 mm with the robot arm base in the middle.
		REQ-001-0002	The work area shall be divided into 5 zones: pick Up zone, Placement zone, processing zone, disposal zone and robot base zone.
UC-002	Moving plants using robotic arm	**REQ**-002-0001	The robotic arm shall grip plants that are delivered to the pick-up zone
		REQ-002-0002	Once the robotic arm has gripped the plant, the robot shall move it to the next area as instructed by Central.
UC-008	Root cutting		The root cutting mechanism shall perform a clean root cutting on first attempt at least 95% of the time.
			The system shall verify that roots have been cut off correctly.
			The system shall perform 3 attempts to cut roots off a plant successfully before discarding the plant.
			The system shall stop the cutting process if >x N resistance is detected?

Figure A.6: User story - autonomous harvesting

User Story ID: US-2
User story short title: Plant recognition

Use Case ID	Use Case	Requirement ID	Requirement
UC-01-001	Capture an individual plant in the pickup area	REQ-01-001	The vision system shall detect an individual plant from the scanning area with at least 95%? accuracy.
		REQ-01-002	The system shall calculate pick-up coordinates in X, Y, Z dimensions with **accuracy**
		REQ-001-003	The vision system shall isolate a plant from other overlapping plants with an accuracy of at least 90%?
UC-002	Classify plants	REQ-002-004	The system shall use AI technology to classify the plant types: **X, Y, Z**.
		REQ-002-005	The system shall use AI technology to estimate plant health based on colour of leaves.
			The system shall use AI technology to estimate if plant growth stage based on size.
		REQ-002-006	If plant classification confidence falls below X%, the system shall attempt again Y more times before requesting human verification.
		REQ-002-001	The AI classification algorithms shall run with at least 95% success rate over 100 test samples.

Figure A.7: User story - plant recognition

User Story ID: US-3
User story short title: Optimal handling

Use Case ID	Use Case	Requirement ID	Requirement
UC-003	Determine if plant should be harvested with or without roots	REQ-003-001	The system shall determine whether the plant should be harvested with or without roots, based on type.
		REQ-003-002	The system shall identify the best location for root cutting with an accuracy of XXX
UC-004	Determine the best way to grip the plant.	REQ-003-003	The system shall decide the best location for the grabber to grip the plant for pickup.
		REQ-003-004	The system shall decide the best method for picking up the plant based on plant type, shape and.. growth structure?
		REQ-003-005	The system shall adapt gripping pressure by X-X Newton? based on the plant data?
		REQ-003-006	The system shall support soft touch gripping in the range of X-Y Newton.
		REQ-003-007	The system shall support firm touch gripping in the range of X-Y Newton.
		REQ-003-008	The system shall support forklifting as a method to pick up plants by soil type growth materials.
UC-006	Does the plant conform to acceptance criteria?	REQ-003-009	The system shall estimate if plant qualifies for sale based on plant health estimation.
		REQ-003-010	The system shall estimate if plant qualifies for sale based on growth stage estimation.
		REQ-003-011	The system shall recycle plants that do not meet the predefined sales acceptance criteria.
		REQ-003-012	The system shall allow users to configure sales acceptance criteria per plant type.

Figure A.8: User story - optimal handling

User Story ID: US-4
User story short title: Safety and efficiency

Use Case ID	Use Case	Requirement ID	Requirement
	Detecting failures		
	Defensive action		
	Notify users		The system shall log gripping errors.
			The system shall log cutting errors
			The system shall log successful

Figure A.9: User story - safety and efficiency

Appendix B

General

1 Group Philosophy (initial outlines) JCDH | -

1.1 Introduction JCDH | -

These are the initial outlines of our group philosophy:

- We aim to foster a fun, positive and creative environment that is based on collaboration, all members feel valued. We therefore base ourselves on a flat leadership style, where tasks and responsibilities are shared and assigned based on team discussions and evolving project needs.
- We aim to foster a collaborative environment where all members feel valued.
- We will base our decision making and task allocation on a flat leadership structure, where tasks and responsibilities are shared and assigned.
- Our project should be a time of great learning and fun. We should try aim to keep a sustainable workload, avoiding too much crunch time late in the project. We shall strive for efficiency through being well prepared, exploring options and planning well. A safe and enjoyable environment.
- A safe and fun place to be.

1.2 Flat structure

JCDH | -

Maintain a flat structure, with responsibilities shared and assigned based on team discussions and evolving project needs.

1.3 Iterative process

JCDH | -

Everything can't be perfect on the first try. That's why an iterative process is extremely useful.

2 Project Model earlier work

EG | -

Her har ikke teksten endret seg siden første innlevering (tid). Mesteparten av teksten slik den står nå bil bli flyttet over i appendix.

Multidiciplinary group

At the very beginning, it was agreed by the founding members that a core value of our group collaboration would be to fully embrace the opportunities to broaden our knowledge and our skills as ambitiously and well targeted as we could. Our goal would be to depart in June as a fresh generation of new engineers, confident and well prepared for the future with as much industry-relevant knowledge and real-life experience as possible. Members that since joined were presented with this philosophy and eagerly embraced it.

One of the new and highly interesting learning opportunities that we identified early on was to opt for building a multidisciplinary group, preferably one covering all three engineering disciplines. This felt like a daunting task as we had very little experience with this type of collaboration from beforehand and, furthermore, had only very limited understanding of each others work methodology and workflows. Yet, we considered multidisciplinary collaboration as a relevant skill to hold across all engineering fields, and therefore it was a challenge we eagerly wanted to take on.

With determined optimism we concluded that although this would add extra complexity to our project, the benefits we would stand to gain from choosing this experience would be well worth the added investments of effort required to obtain it. It was recognised, however, that this choice would require extra effort and diligence put into building effective communication, project management, and interfacing.

Balancing the needs of many, with an end goal in sight

An important project management factor found early was that most members in our group are influenced by personal circumstances that may affect their availability or work capacity during the time we intended to spend together. For instance, some live further afield and are likely, at some points, to experience transportation issues during the extremes of Norwegian weather conditions. Some have family commitments that might require them to stay at home during times when children are ill. Also, they may be less available for project work during late nights and weekends, than others who might in fact prefer to work during these times. Furthermore, some are bound by out-of-project work commitments that greatly affect their working schedule and time availability.

It was recognised that each member of the group brings valuable knowledge, experience, skills, and ideas to the table, together with the key ingredients of a positive attitude, collaboration-

and solution-mindedness. It was therefore collectively decided to move forward with the above-mentioned complicating factors; both of building a multidisciplinary focused group, and to create an environment that everyone is able to thrive in.
For this to succeed, certain criteria must be met. It was reasoned that communication must be structured through regular meetings, detailed minutes, and supporting documentation and user guides produced and shared regularly. We agreed on core working hours **(further detailed in a different section? Or here?)**, to facilitate ease of communication and encourage collaborative efforts, alongside our regular set meetings. However, when necessary, exceptions could also be made by group agreement. An example of this was when one of our members was granted a special adaptation of the core working hours setup to accommodate for their work commitments.

Put in a paragraph about starting with a well thought out plan and project model, but to also hold a view of continual improvement through early identification of problems/bottle necks etc, and applying risk based thinking in order to see problems before they arise. Maybe also loosely learn from the ISO way of thinking with the PDCA cycle (Plan, Do, Check, Act). Frequent evaluations of process à improve continuously as we go along à Regular, slightly adapted SCRUM Retrospectives.

Also throw in some good stuff about the importance of clear communication. Plan, responsibilities/accountability, support, sharing documentation. Everyone knowing where to find key information in order to understand what is expected and who does what. Partnering up on tasks.

INTERFACING

Talk about the nice stuff too. Team building, learning from each other, supporting each other, cakes!

Structured as a company, but not quite

Find an elegant way to lead into the two key topics of this subsection:

- Group following a company-like structure
- Discuss how the leadership team works
  - Flat leadership hierarchy no leader
  - Lead over to talking about agile project model → SCRUM

Company-like structure

Talk about how were designing our group as a mini company with a leadership team and its technology departments.

- Think: A company with a leadership team. The team, here consisting of all the group members together, make all the important decisions. We allocate budget, discuss concepts and directions of development, develop requirements, plan and refine the product back log. We monitor progress and make adaptations as necessary. We guide the teams and hold them accountable.
- The company has three development departments: Mechanical (consisting of our three mechanical engineers), Software (our two software engineers, and Electrical. Electrical consists of one electronics engineer who is backed up by one of the mechanical engineers and one of the software engineers. These two members committed themselves from the start to supporting the needs of the electronics department when required, as part of their roles.

Why have we decided to organise ourselves this way?

Each engineering discipline uses their own workflow, tools and processes to conduct their work. However, these methodologies dont always overlap too well. Therefore, instead of imposing the processes and tools preferred by one discipline onto everyone else and expecting them to spend the time finding less efficient work arounds, we opted a different approach. Our take instead was to focus on managing the project on group level where wed apply an agile based model, adapted from SCRUM to manage and structure our progress. Wed also be setting the main standards for tools and templates such as risk management, requirements development and so forth on this level. However, although the product backlog would largely sit at the leadership level, each department would be given the freedom to develop and produce their work using their own preferred techniques. An example to illustrate how this freedom might prove beneficial is where it would mechanical engineers tend to work incrementally whereas software development might prefer to develop their work in a more evolutionary way.

It was decided to focus our leadership team on deciding the work that needed to be done, and to allow the department teams to utilise their own workflows, tools and methods in order to deliver on the orders they receive.

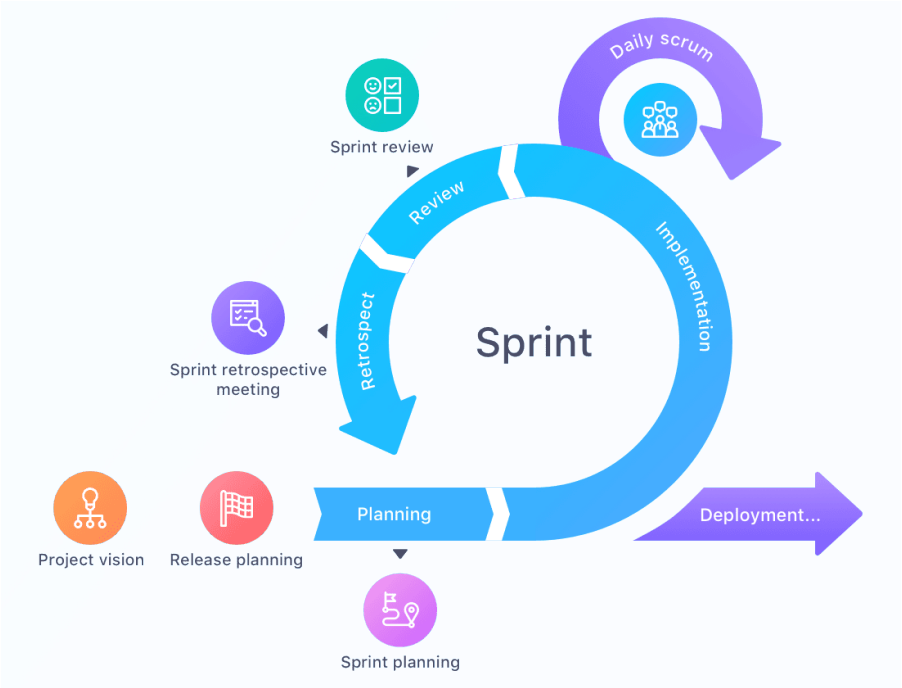
- Partitioning our discipline related efforts into subsections allows us pass important decisions on topics such as concepts, direction and problem domain related requirements on the leadership level, and send these down the line to the departments as work orders

- to follow a main project model on the leadership level, yet at the same time allowing the different departments to follow their own workflow and processes that dont necessarily overlap all too well.

Tricky bit elegantly move over to talking about the bread and butter of this section: The actual project model!

Agile work methodology based on SCRUM

Ive not had chance to write up on this topic yet, but please see the presentation slides I prepared and showed before the oppgave was given, they can be found in the appendix. There are significant amounts of information there.



Dag	Klokkeslett	Møte
1. Onsdag (Sprint start)	09.00 -11.00	Sprint Planning
Hverdager	9.15 - 9.30	Daily standup
2. Fredag (Sprint slutt)	9.00 - 10.00	Sprint review
	10.15 - 11.00	Sprint Retrospective
	11.00 - 12.00	KAKE-FREDAG!

SCRUM supports a flat leadership structure, which is another approach our group was keen to explore.

- Talk about flat leadership hierarchy.
- Why is this a good idea? -All the good references..
- Why did we decide to run with it?
- How have we chosen to apply it, and why?

Tittel	Beskrivelse	Definition of Done
Set up ROS2 workspace and version control	Install ROS2, configure colcon build system, and create basic repo.	ROS2 workspace builds cleanly and version-controlled code is pushed.
Define node architecture and message layout	Sketch full system based on ROS2 layered architecture: - Perception (e.g., vision_node) - Recognition (e.g., root_detector, grasp_detector) - Planning (e.g., task_planner) - Control (e.g., move_group, gripper_controller, cutting_controller) - HAL (e.g., hal_bridge) Define topics, services, and actions using ROS2 message types.	Architecture documented with layers, node responsibilities, and interface types (topics, services, actions). Reference communication diagram and node table.
Create motor control node (publish/subscribe)	Create a control node for arm motion using ROS2 action interface or topic-based command publication to the hardware or simulation driver.	Node publishes to topic or provides action interface. Verified with MoveIt2 or simulated joint interface.
Create gripper control node (service or topic)	Implement a ROS2 service node (e.g., /gripper/set_state) or use topics to toggle gripper state. Integrate pressure feedback from HAL.	Gripper responds to service or topic command and pressure feedback is observed on topic /gripper/pressure.
Design and implement finite state machine (FSM)	Implement FSM in task_planner node to coordinate detection, movement, gripping, cutting, and placing. Support recovery and retry strategies via feedback and result handling.	FSM runs full cycle using simulated or real nodes with transitions between key phases observed.
Connect FSM to individual control nodes	Link FSM to: - move_group action for arm motion - gripper_controller service for grasping - cutting_controller action for cutting Ensure data flow through topics like /detected_root_pose and /gripper/pressure.	FSM successfully sends goals and services, and reacts to feedback. End-to-end cycle completes as expected.
Create test launch file for integrated nodes	Launch script that brings up ROS2 nodes for end-to-end test.	All nodes launch without error and topics/services are active.
Log key data to rosbag	Enable rosbag2 for selected topics (e.g. FSM state, camera, motor commands).	Rosbag files record correctly and can be played back.

Optional Epic: Motion Planning with MoveIt2

Tittel	Beskrivelse	Definition of Done
Define URDF for MoveIt2 compatibility	Ensure the robot's URDF includes joint limits, inertia, and proper link definitions.	URDF loads cleanly in RViz and joints display correctly.
Use MoveIt Setup Assistant to generate config package	Generate the planning group, SRDF, and MoveIt2 configuration files.	MoveIt2 config launches without errors and shows robot model.
Create test motion planning demo in RViz	Plan simple trajectories to a few fixed positions using RViz planning interface.	Planned trajectories visualize correctly in RViz with joint motion preview.
Write basic ROS2 launch file for MoveIt2 and RViz	Launch MoveIt2 and RViz together with correct configuration.	Command launches both tools and robot is interactable.
Connect MoveIt2 planner to simulated controllers	Simulate execution by publishing joint states or using dummy controllers.	Planner output appears as if it were commanding the robot.
Document MoveIt2 architecture and integration plan	Describe how MoveIt2 integrates into control layer. Include /move_group action, joint state updates from HAL, and trajectory execution through controllers. Show planning calls from task_planner.	Integration documented in system architecture section with diagrams, node connections, and MoveIt2-specific components.

3 Project Methodology

EG | -

3.0.1 Earlier work towards creating backlog items for ROS 2 implementation

EG | -

4 Architecture

4.0.1 Early work understanding communication and signals

EG | -

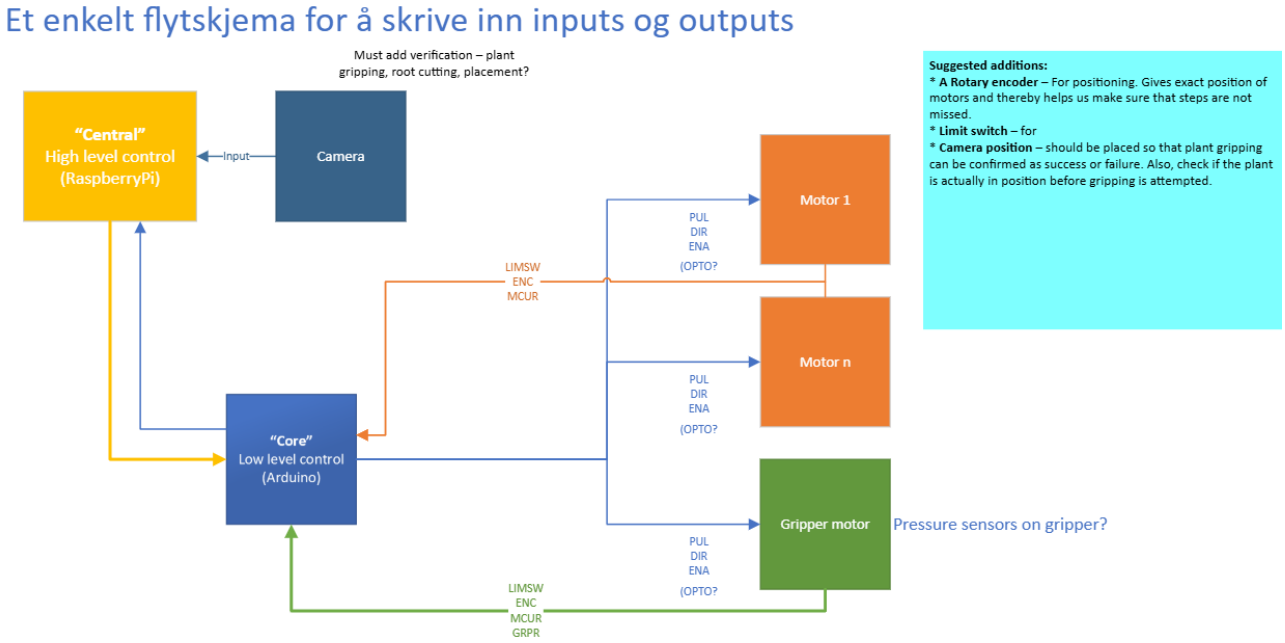


Figure B.1: Interfacing elektro og data

From motor to Core

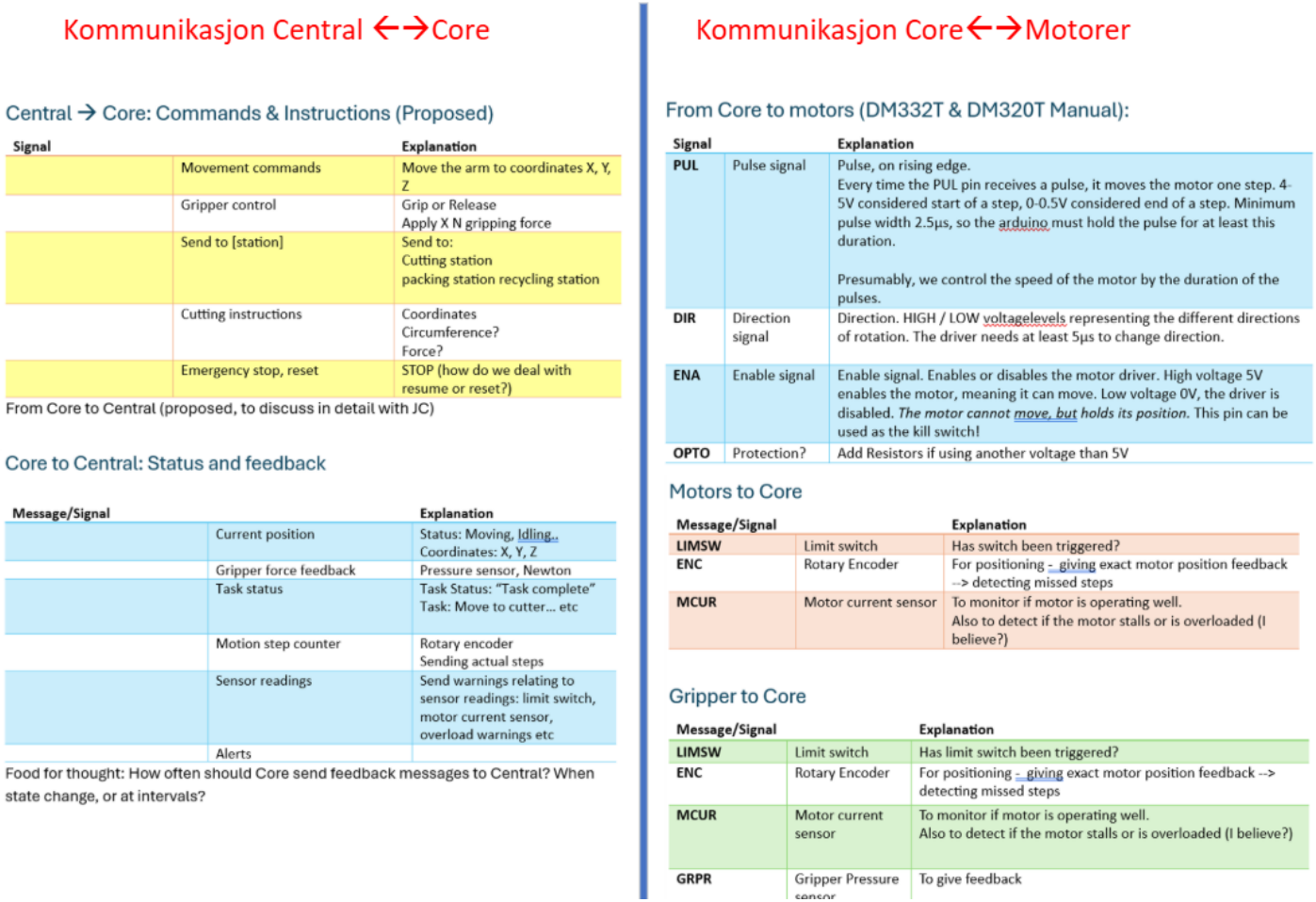
Signal	What it does	Bits	Details	Details for dummies	Important notes
ENCA+	Rotary encoder	1	Rotary encoder	Sends pulse for each step. Each step is represented by a signal change.	
ENCA-	Rotary Encoder	1	inverse of ENCA+ Differential signaling (reduce interferance), not needed	Software: Ignore	
ENCB+	Rotary encoder	1	Rotary encoder	Sends pulse for each step. Each step is represented by a signal change.	
ENCB-	Rotary encoder	1	Differential signaling (reduce interferance), not needed	Software: Ignore	
ENCZ+	N/A		Center position, not needed	Software: Ignore	
ENCZ-	N/A		Center position, not needed	Software: Ignore	
REFR	Limit switch	1	Limit switch for clockwise rotation	When it comes to the limit the signal will be high (1)	
REFL	Limit switch	1	Limit switch for counter-clockwise rotation	When it comes to the limit the signal will be high (1)	
	Motor Current			Mest sansynlig på UART	This will come in vsn 2 driver (internal development)
	Stall Guard				This will come in vsn 2 driver (internal development)

From motor to Gripper

Signal	What it does	Bits	Details	Details for dummies	Important notes
ENCA+	Rotary encoder	1	Rotary encoder	Sends pulse for each step. Each step is represented by a signal change.	
ENCA-	Rotary Encoder	1	inverse of ENCA+ Differential signaling (reduce interferance), not needed	Software: Ignore	
ENCB+	Rotary encoder	1	Rotary encoder	Sends pulse for each step. Each step is represented by a signal change.	
ENCB-	Rotary encoder	1	Differential signaling (reduce interferance), not needed	Software: Ignore	
ENCZ+	N/A		Center position, not needed	Software: Ignore	
ENCZ-	N/A		Center position, not needed	Software: Ignore	
REFR	Limit switch	1	Limit switch for clockwise rotation	When it comes to the limit the signal will be high (1)	
REFL	Limit switch	1	Limit switch for counter-clockwise rotation	When it comes to the limit the signal will be high (1)	

Figure B.2: Signalinterfacing elektro og data

,



Kommunikasjon Central ←→Core

Central → Core: Commands & Instructions (Proposed)

Signal		Explanation
	Movement commands	Move the arm to coordinates X, Y, Z
	Gripper control	Grip or Release Apply X N gripping force
	Send to [station]	Send to: Cutting station packing station recycling station
	Cutting instructions	Coordinates Circumference? Force?
	Emergency stop, reset	STOP (how do we deal with resume or reset?)

From Core to Central (proposed, to discuss in detail with JC)

Core to Central: Status and feedback

Message/Signal		Explanation
	Current position	Status: Moving, Idling.. Coordinates: X, Y, Z
	Gripper force feedback	Pressure sensor, Newton
	Task status	Task Status: "Task complete" Task: Move to cutter... etc
	Motion step counter	Rotary encoder Sending actual steps
	Sensor readings	Send warnings relating to sensor readings: limit switch, motor current sensor, overload warnings etc
	Alerts	

Food for thought: How often should Core send feedback messages to Central? When state change, or at intervals?

Kommunikasjon Core←→Motorer

From Core to motors (DM332T & DM320T Manual):

Signal		Explanation
PUL	Pulse signal	Pulse, on rising edge. Every time the PUL pin receives a pulse, it moves the motor one step. 4-5V considered start of a step, 0-0.5V considered end of a step. Minimum pulse width 2.5µs, so the arduino must hold the pulse for at least this duration. Presumably, we control the speed of the motor by the duration of the pulses.
DIR	Direction signal	Direction. HIGH / LOW voltagelevels representing the different directions of rotation. The driver needs at least 5µs to change direction.
ENA	Enable signal	Enable signal. Enables or disables the motor driver. High voltage 5V enables the motor, meaning it can move. Low voltage 0V, the driver is disabled. *The motor cannot move, but holds its position.* This pin can be used as the kill switch!
OPTO	Protection?	Add Resistors if using another voltage than 5V

Motors to Core

Message/Signal		Explanation
LIMSW	Limit switch	Has switch been triggered?
ENC	Rotary Encoder	For positioning - giving exact motor position feedback --> detecting missed steps
MCUR	Motor current sensor	To monitor if motor is operating well. Also to detect if the motor stalls or is overloaded (I believe?)

Gripper to Core

Message/Signal		Explanation
LIMSW	Limit switch	Has limit switch been triggered?
ENC	Rotary Encoder	For positioning - giving exact motor position feedback --> detecting missed steps
MCUR	Motor current sensor	To monitor if motor is operating well. Also to detect if the motor stalls or is overloaded (I believe?)
GRPR	Gripper Pressure sensor	To give feedback

Figure B.3: Communication and signal details from the early diagram above

4.0.2 Early work on architecture

EG | -

4.0.3 Early work on workflow

EG | -

4.0.4 Early work on understanding communication

EG | -

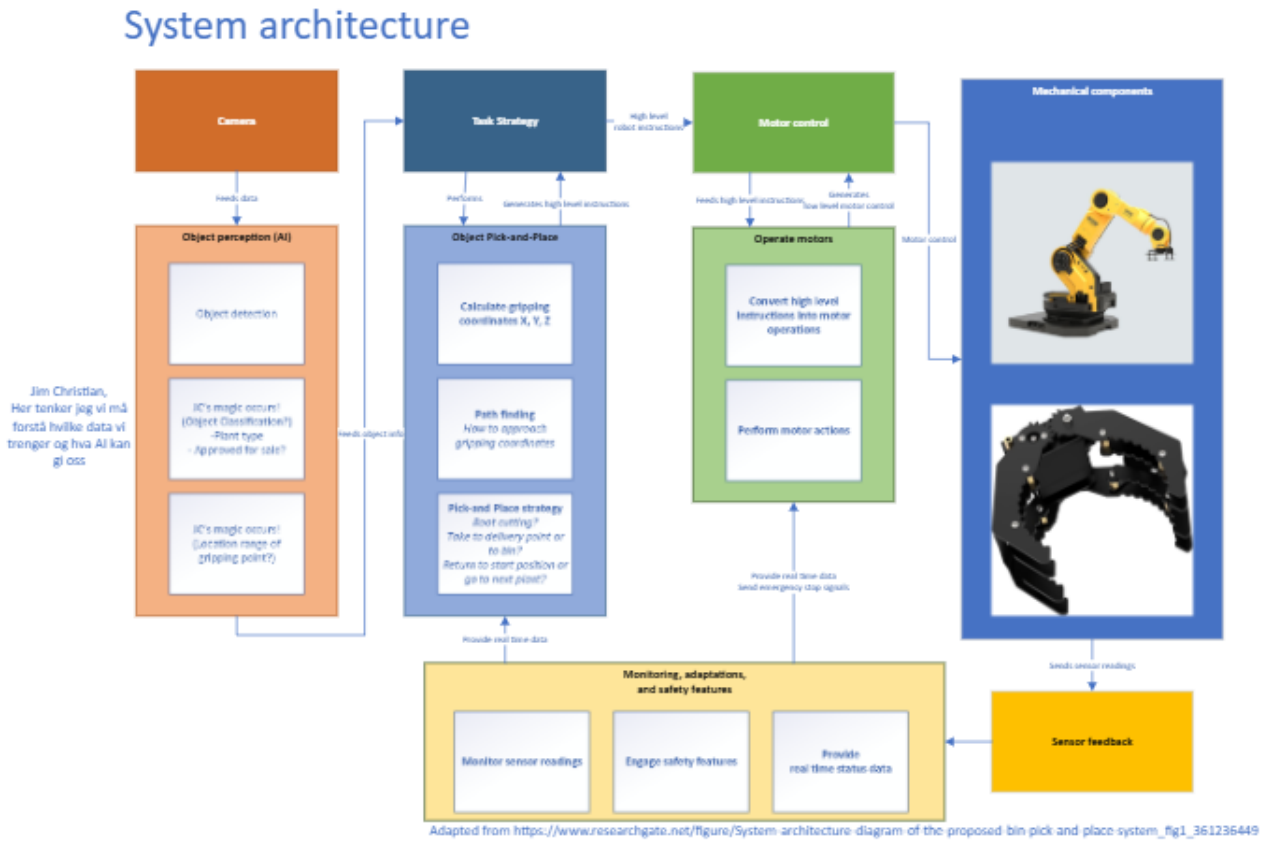


Figure B.4: Early architectural design

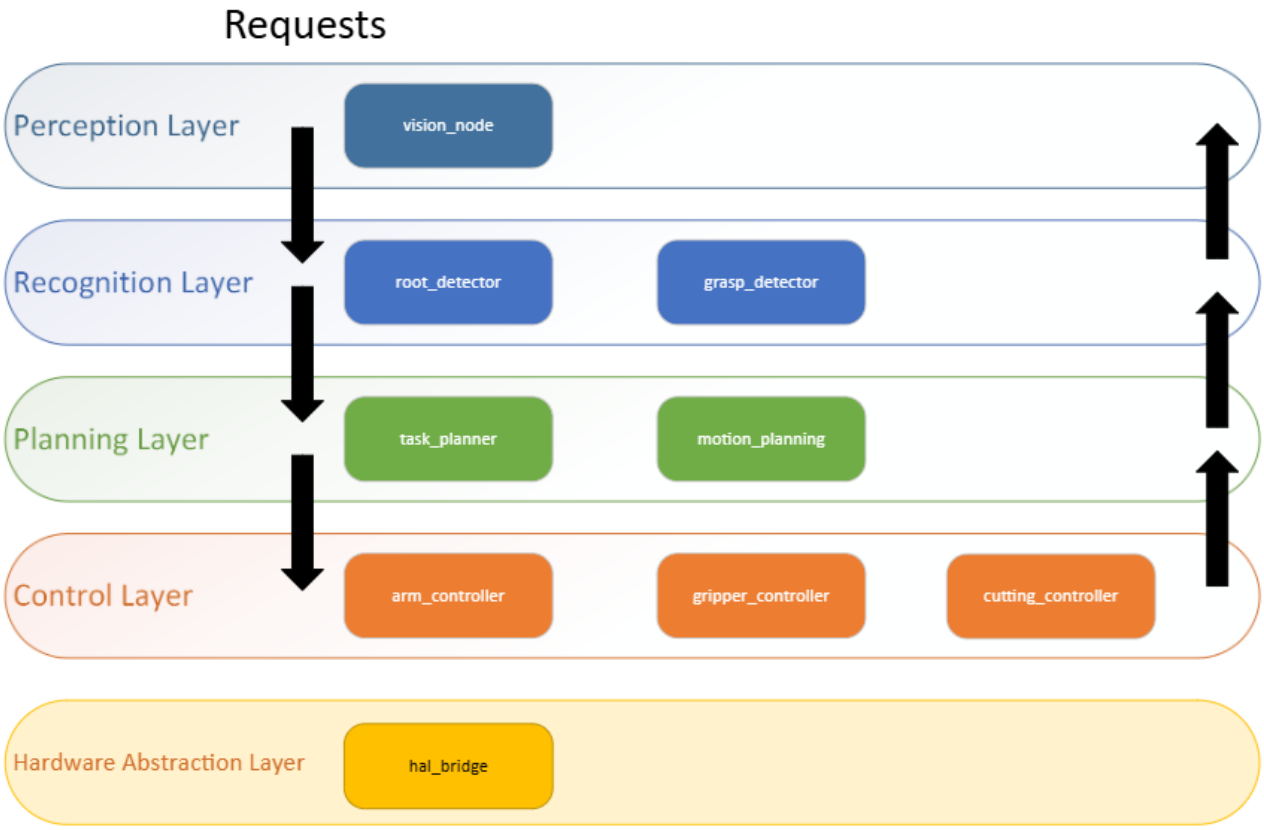


Figure B.5: Early architectural design

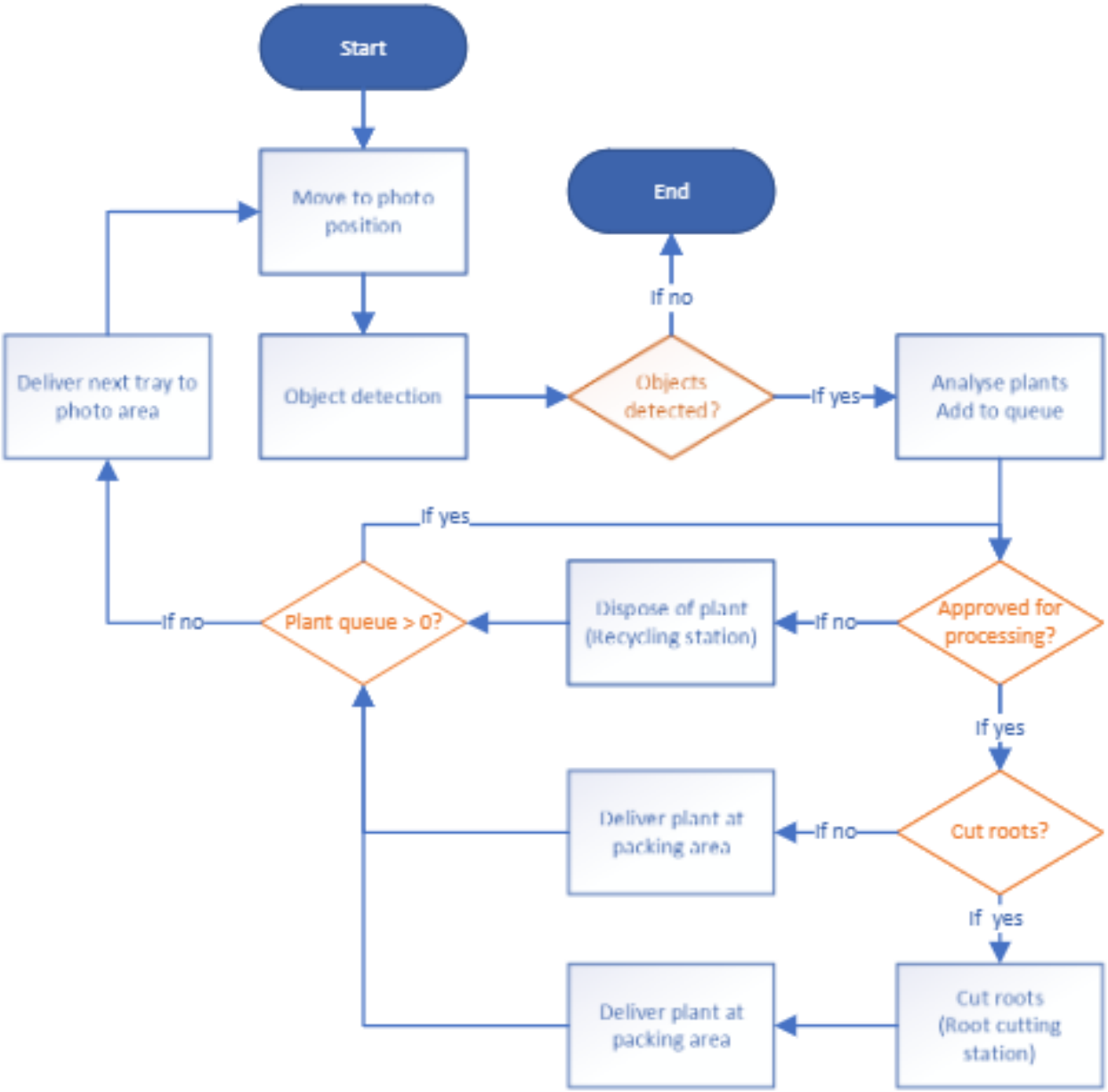


Figure B.6: Early attempts at mapping processes

MQTT Communication

Topic	Direction	Publisher	Subscriber	Purpose	Example Payload
leafy_automation/motion	ROS2 → Arduino	ROS2 arm_control_node	Arduino CommandManager	Move joints J0–J4	MOVE 1000 2000 1500 500 250
leafy_automation/gripper	ROS2 → Arduino	ROS2 gripper_control_node	Arduino CommandManager	Open/close gripper	GRIP 1
leafy_automation/calibrate	ROS2 → Arduino	ROS2 gripper_control_node	Arduino CommandManager	Run calibration phase	CALIBRATE
leafy_automation/status/command_received	Arduino → ROS2	Arduino CommandManager	ROS2 arm_control_node gripper_control_node	Confirm that any incoming command (MOVE, GRIP, CALIBRATE) was received and understood	COMMAND RECEIVED: MOVE
leafy_automation/status/motion	Arduino → ROS2	Arduino CommandManager	ROS2 Node arm_control_node	Report MOVE command completed	MOVE DONE
leafy_automation/status/gripper	Arduino → ROS2	Arduino CommandManager	ROS2 Node gripper_control_node	Report GRIPPER action completed	GRIPPER DONE
leafy_automation/status/calibration_done	Arduino → ROS2	Arduino CommandManager	ROS2 Node arm_control_node	Report CALIBRATE phase completed	CALIBRATION DONE
leafy_automation/status/heartbeat	Arduino → ROS2	Arduino CommandManager	ROS2 Node / Monitor Node	Periodic alive signal	alive?

Layer	Node(s)	Libraries / Tools	Notes
Perception	camera_driver_n	usb_cam, cv_camera, image_transport, opencv-python, rclpy, depth	Use existing camera drivers.
Recognition	plant_classifier_n	PyTorch, TensorFlow, ONNX Runtime, OpenCV, vision_msgs, geometr	AI inference works well in Python. Use standard ROS2 message types.
Planning	task_planner_no	MoveIt2, rclcpp, behaviortree_cpp_v3, yaml-cpp	Use C++ for motion planning. Integrate with MoveIt and behavior trees.
Control	trajectory_contro	ros2_control, control_msgs, joint_trajectory_controller, hardware_in	Requires real-time safety. Integrates ROS2 control stack.
HAL	arduino_interface	rclpy, pyserial, paho-mqtt, micro_ros_agent	Python for prototyping. C++ for performance. Wraps hardware access.

Figure B.7: Enter Caption

Layer	Node	Message Type / Custom Msg	Topic / Service / Action	Type	Notes
Perception	camera_node	sensor_msgs/Image	/camera/image_raw	Topic	Publishes raw camera images to be processed by recognition lay
Recognition	plant_classifier_node	vision_msgs/ObjectHypothesis	/plant_class	Topic	Publishes classification result of plant
Recognition	grip_point_estimator_node	geometry_msgs/PoseStamped	/grip_pose	Topic	Publishes estimated gripping pose
Planning	task_planner_node	leafy_msgs/TaskGoal (custom)	/next_task	Service	Receives task requests, returns task details
Planning	motion_planner_node	trajectory_msgs/JointTrajectory	/planned_trajectory	Action	Executes robot motion via trajectory action server
Control	trajectory_controller_node	std_msgs/Float64MultiArray	/cmd_joint_positions	Topic	Low-level joint command array
Control	joint_command_translator_node	control_msgs/JointTrajectoryControllerState	/joint_states	Topic	Reports joint states for feedback
HAL	arduino_interface_node	leafy_msgs/MotorCommand (custom)	/motor_commands	Topic	Low-level motor control to Arduino
HAL	gripper_driver_node	leafy_msgs/GripperStatus (custom)	/gripper_status	Topic	Reports gripper open/close state and force feedback

Figure B.8: Thoughts for future work.

5 Design and Website

5.1 Design

JCDH | -

The design will be used for presentations, website, and the thesis. Keeping a consistent look across all these items is important to maintain a professional look.

The design philosophy was taken by combining parts of nature (plants) and modernity. This fusion creates a clean and natural look. Figure B.19 shows the color palette we landed on.

Figure B.9: Our project color palette

5.1.1 Project logo design

SME | *JCDH*

The visual identity of our project is represented by the logo shown in Figure B.10. The design was primarily inspired by the project title, Leafy Automation, and reflects both the nature of our work and the agricultural domain in which the project operates. Given that the primary function of the robotic system is to harvest leafy greens, the choice of a green color palette and the inclusion of a leaf motif in the logo was both symbolic and appropriate.

The leaf element in the logo not only refers to the crops our system is designed to handle but also establishes a visual link to our project partner, Hydroplant Technologies AS, whose branding similarly incorporates a leaf. This further reinforces the connection between our work and the broader vision of sustainable and modern horticultural practices.

The design process began with a hand-drawn sketch in a digital notebook. The initial concept was iteratively refined to strike a balance between organic shapes and professional aesthetics. Ultimately, the final version was created by combining freehand elements with typefaces that complemented the natural curves of the leaf. The fonts used were (Codigra and BirdsofPar-

adise), chosen for their readability and visual harmony with the drawing.

After finalizing the layout and structure, the logo was imported into Canva for final adjustments. The background was removed to ensure versatility across multiple applications, including presentations, promotional materials, and project accessories. As an example, the logo was featured on the custom name tags designed for the team, as illustrated in Section 5.1.2.

Figure B.10: Project logo

5.1.2 Name tags

SME | -

Figure B.11: Name tag: Beatrix Rimestad

Figure B.12: Name tag: Daniels Blomnieks

Figure B.13: Name tag: Elin Gravningen

Figure B.14: Name tag: Jim Christian Haukvik

Figure B.15: Name tag: Vetle Myhre Nilsen

Figure B.16: Name tag: Sunniva Myrvang Eineteig

5.1.3 Recruitment ad

JCDH, EG | -

Figure B.17: Recruitment ad

All pictures used in the recruitment ad have permissive and free-to-use licenses from unsplash.com which don't require attribution.

5.2 Website

JCDH | -

The website is hosted on USN's servers and can be found at https://itfag.usn.no/grupper/D08-

25/ which is a subdirectory on an Apache instance.

Although the server includes support for both PHP and a MySQL (including PHPmyadmin), there is no requirement to use these technologies, and reduced complexity often causes less issues down the road.

The website is written using the Bootstrap Toolkit which is a CSS toolkit that simplifies the process of website development and prototyping.

Git was considered for the website, but it was decided that because of it's simplicity and in order to consolidate resources on the engineering project itself, it was not needed. The group agreed upon not using too much time or resources on developing the website.

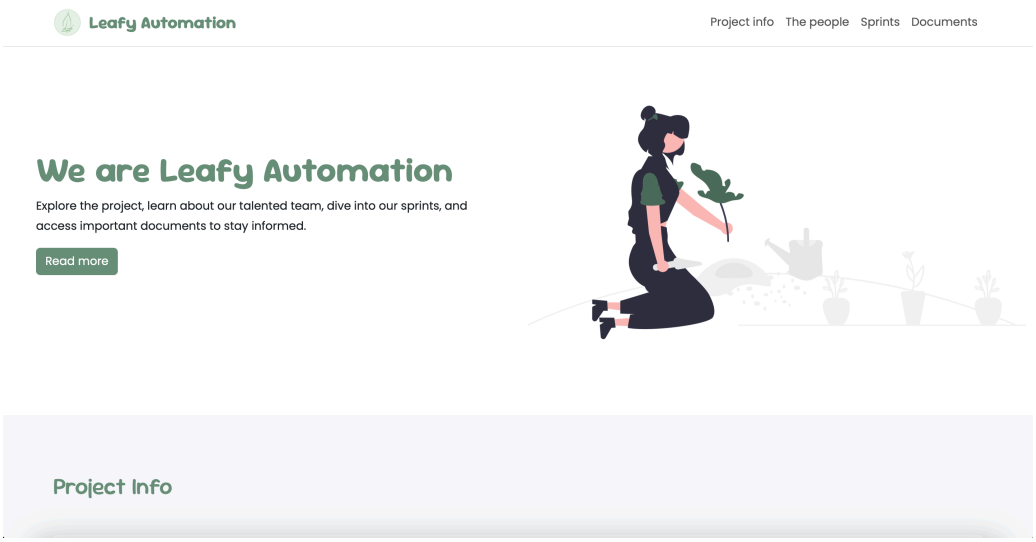


Figure B.18: Website, iteration 1

5.2.1 Security considerations

JCDH | -

When developing websites, there are in most cases a great many security considerations to keep in mind. A good approach is to keep this list of can-be security issues as small as possible. Therefore, the choice to drop technologies like PHP and MySQL - and only focusing on pure HTML documents - keeps our website stable and secure for the foreseeable future.

A curated list of potential security issues regarding PHP and MySQL follows:

- Out-of-date PHP version

- MySQL injections

5.3 Website source code

JCDH | -

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="Explore the project, learn about
    our talented team, dive into our sprints, and access important documents
    to stay informed.">
        <link rel="icon" href="assets/img/logo.png">
        <title>Leafy Automation</title>

        <!-- Open Graph -->
        <meta property="og:title" content="Leafy Automation">
        <meta property="og:description" content="Explore the project, learn
    about our talented team, dive into our sprints, and access important
    documents to stay informed.">
        <meta property="og:image" content="https://itfag.usn.no/grupper/D08
    -25/demo2/assets/img/logo.jpg">

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/
    bootstrap.min.css" rel="stylesheet" integrity="sha384-
    QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
    crossorigin="anonymous">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
    font-awesome/6.7.2/css/all.min.css" integrity="sha512-Evv84Mr4kqVGRNSgIGL
    /F/aIDqQb7xQ2vcrdIwxfjThSH8CSR7PBEakCr51Ck+w+/U6swU2Im1vVX0SVk9ABhg=="
    crossorigin="anonymous" referrerpolicy="no-referrer" />
        <link href="assets/css/style.css" rel="stylesheet">
    </head>
    <body id="top">
        <nav class="navbar navbar-expand-lg">
            <div class="container">
                <a class="navbar-brand" href="#top"><span>Leafy Automation</
    span></a>

                <button class="navbar-toggler border-0" type="button" data-
    bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav
    " aria-expanded="false" aria-label="Toggle navigation">
                    <i class="fa-solid fa-ellipsis"></i>
                </button>

                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav ms-auto">
```

```
                    <li class="nav-item">
                        <a class="nav-link" href="#project-info">Project
 info</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#people">The people</a
>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#sprints">Sprints</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#documents">Documents<
/a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <!-- Hero Section -->
    <div class="container-fluid pb-5 pt-5 hero">
        <div class="row">
            <div class="col-md-6 d-flex align-items-center ps-md-5 pb-5"
>
                <div class="hero-info">
                    <h1>We are Leafy Automation</h1>
                    <p>Explore the project, learn about our talented
team, dive into our sprints, and access important documents to stay
informed.</p>
                    <a class="btn btn-plant" href="#project-info">Read
more</a>
                </div>
            </div>

            <div class="col-md-6">
                <img src="assets/img/logo.jpg" class="img-fluid hero-img
 my-md-2 floating-image" alt="Hero Image">
            </div>
        </div>
    </div>

    <!-- Project Info Section -->
    <div id="project-info" class="section bg-light">
        <div class="container">
            <h2 class="mb-5">Project Info</h2>

            <div class="row">
                <div class="col-md-12 mb-4">
```

```
                    <div class="card">
                        <div class="card-body">
                            <h4>Description</h4>
                            <p>Hydroplant Technologies seeks to develop
an autonomous system for efficient lettuce harvesting and handling in
vertical farming. The goal is to automate the process from harvesting to
further processing in a sustainable, scalable, and cost-effective way.</p
>
                        </div>
                    </div>
                </div>

                <div class="col-md-12 mb-4">
                    <div class="card">
                        <div class="card-body">
                            <h4>Objectives</h4>
                            <ul>
                                <li>Recognize and handle different types
 of lettuce.</li>
                                <li>Transport lettuce between system
areas safely.</li>
                                <li>Position accurately for optimal
operation.</li>
                            </ul>
                        </div>
                    </div>
                </div>

                <div class="col-md-12 mb-4">
                    <div class="card">
                        <div class="card-body">
                            <h4>Key Requirements</h4>
                            <ul>
                                <li><strong>Recognition:</strong>
Identify lettuce types using sensors.</li>
                                <li><strong>Movement:</strong> Precise,
safe transport of lettuce.</li>
                                <li><strong>Positioning:</strong>
Accurate location within the system.</li>
                                <li><strong>Quality Check:</strong>
Identify and remove bad leaves.</li>
                            </ul>
                        </div>
                    </div>
                </div>

                <div class="col-md-12 mb-4">
                    <div class="card">
                        <div class="card-body">
```

```
                            <h4>Expected Deliverables</h4>
                            <ul>
                                <li>Prototype or simulation of the
system.</li>
                                <li>Design and technical documentation.<
/li>
                                <li>Testing and evaluation reports.</li>
                            </ul>
                        </div>
                    </div>
                </div>

                <div class="col-md-12 mb-4">
                    <div class="card">
                        <div class="card-body">
                            <h4>Additional Info</h4>
                            <p>This project supports Hydroplant
Technologies' vision to enhance vertical farming's sustainability and
cost-efficiency.</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- People Section -->
    <div id="people" class="section">
        <div class="container">
            <h2 class="mb-5">The People</h2>

            <div class="row">
                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/sunniva.jpg" class="
card-img-top" alt="Person 1">
                        <div class="card-body">
                            <h5 class="card-title">Sunniva Myrvang
Eineteig</h5>
                            <p class="card-text"><strong>Machine
engineer</strong> <br> External contact  Instagram</p>
                        </div>
                    </div>
                </div>

                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/jim.jpg" class="card
-img-top" alt="Person 2">
```

```
                        <div class="card-body">
                            <h5 class="card-title">Jim Christian Dale
Haukvik</h5>
                            <p class="card-text"><strong>Computer
engineer</strong> <br> AI  Network-protocol  Website  ClickUp</p>
                        </div>
                    </div>
                </div>

                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/beatrix.jpg" class="
card-img-top" alt="Person 3">
                        <div class="card-body">
                            <h5 class="card-title">Beatrix Møller
Rimestad</h5>
                            <p class="card-text"><strong>Machine
engineer</strong> <br> Internal contact</p>
                        </div>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/elin.jpg" class="
card-img-top" alt="Person 4">
                        <div class="card-body">
                            <h5 class="card-title">Elin Gravningen</h5>
                            <p class="card-text"><strong>Computer
engineer</strong> <br> Robotics  Risk-analysis  SCRUM</p>
                        </div>
                    </div>
                </div>

                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/daniels.jpg" class="
card-img-top" alt="Person 5">
                        <div class="card-body">
                            <h5 class="card-title">Daniels Aleksandrs
Blomnieks</h5>
                            <p class="card-text"><strong>Machine
engineer</strong> <br> LaTeX  Risk-analysis</p>
                        </div>
                    </div>
                </div>
```

```
                <div class="col-md-4 mb-4">
                    <div class="card">
                        <img src="assets/img/people/vetle.jpg" class="
card-img-top" alt="Person 6">
                        <div class="card-body">
                            <h5 class="card-title">Vetle Myhre Nilsen</
h5>
                            <p class="card-text"><strong>Electronics
engineer</strong> <br> LaTeX  Group-environment</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <!-- Sprints Section -->
        <!-- https://undraw.co -->
        <div id="sprints" class="section bg-light">
            <div class="container">
                <h2 class="mb-5">Sprints</h2>

                <!-- Sprint 1 -->
                <div class="card mb-4 sprint">
                    <div class="row align-items-center">
                        <div class="col-md-4">
                            <img src="assets/img/illustrations/team.svg"
class="img-fluid sprint-img" alt="Sprint 1">
                        </div>

                        <div class="col-md-8">
                            <div class="card-body">
                                <h3 class="card-title">Sprint 1</h3>
                                <p class="card-text">In the first sprint we
working on gripper concepts, robotics concepts, motor driver, networking
and camera functionality.</p>
                            </div>
                        </div>
                    </div>
                </div>

                <!-- Sprint 2 -->
                <div class="card mb-4 sprint">
                    <div class="row align-items-center flex-md-row-reverse">

                    <div class="col-md-4">
                        <img src="assets/img/illustrations/team.svg" class="
img-fluid sprint-img" alt="Sprint 2">
                    </div>
```

```
                <div class="col-md-8">
                    <div class="card-body">
                        <h3 class="card-title">Sprint 2</h3>
                        <p class="card-text">In sprint 2 we worked on
CAD and hardware, AI model training and ROS2. We now have a functional
mechanical base, fine tuned object detection models, and a defined
communication structure.</p>
                    </div>
                </div>
                </div>
            </div>

            <!-- Sprint 3 -->
            <div class="card mb-4 sprint">
                <div class="row align-items-center">
                    <div class="col-md-4">
                        <img src="assets/img/illustrations/team.svg"
class="img-fluid sprint-img" alt="Sprint 3">
                    </div>

                    <div class="col-md-8">
                        <div class="card-body">
                            <h3 class="card-title">Sprint 3</h3>
                            <p class="card-text">Sprint 3 included
completion of mechanical assembly, Arduino modules (ROS2 and motor
control), HMI and AI salad detection.</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Documents Section -->
    <div id="documents" class="section bg-dark text-light">
        <div class="container">
            <h2 class="mb-5">Documents</h2>

            <!-- Bachelor Thesis -->
            <div class="card text-dark mb-3">
                <div class="card-body">
                    <h5 class="card-title">Bachelor Thesis</h5>
                    <p class="card-text">Coming soon</p>
                    <!--<a href="#" class="btn btn-plant">Download</a>--
>
                </div>
            </div>
        </div>
```

```
        </div>

        <!-- group image -->
        <div id="group">
            <img src="assets/img/group.jpg" class="img-fluid group-img"
    style="object-fit: cover;" alt="Group Image">
        </div>

        <!-- Footer Section -->
        <div class="container">
            <footer class="d-flex flex-wrap justify-content-between align-
    items-center py-3 my-4">
                <div class="col-md-4 d-flex align-items-center">
                    <span class="mb-3 mb-md-0 text-body-secondary">Ⅼ 2025
    Leafy Automation</span>
                </div>

                <ul class="nav col-md-4 justify-content-end list-unstyled d-
    flex">
                    <li class="ms-3"><a class="text-body-secondary" href="
    https://www.instagram.com/leafyautomation"><i class="fa-brands fa-
    instagram"></i></a></li>
                </ul>
            </footer>
        </div>

        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/
    bootstrap.bundle.min.js" integrity="sha384-
    YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
    crossorigin="anonymous"></script>
        <script src="https://code.jquery.com/jquery-3.7.1.min.js" integrity=
    "sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="
    anonymous"></script>
        <script src="assets/js/main.js"></script>
     </body>
</html>
```

Listing B.1: index.html

```
@import url('https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0
    ,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;
    display=swap');
@import url('https://fonts.googleapis.com/css2?family=Sour+Gummy:ital,wght@0
    ,100..900;1,100..900&display=swap');

body {
    font-family: 'Poppins', sans-serif;
}
```

```
h1, h2 {
    font-family: 'Poppins', sans-serif;
}

p, ul li {
    line-height: 1.8em;
}

ul {
    padding: 0;
}

.card-body p:last-child,
.card-body ul:last-child {
    margin-bottom: 0;
}

#project-info ul {
    list-style: none;
}

#project-info ul li::before {
    content: '\1F331';
    margin-right: 10px;
}

.hero .hero-info {
    opacity: 0;
    animation: fadeIn 1s ease-in-out forwards;
}

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(20px);
    }

    to {
        opacity: 1;
        transform: translateY(0);
    }
}

.floating-image {
    display: block;
    margin: 0 auto;
    animation: float 3s ease-in-out infinite;
}
```

```
@keyframes float {
    0% {
        transform: translateY(0);
    }
    50% {
        transform: translateY(-10px);
    }
    100% {
        transform: translateY(0);
    }
}

nav.navbar {
    box-shadow: none;
}

.navbar {
    position: fixed;
    width: 100%;
    top: 0;
    left: 0;
    z-index: 999;
    background: #fff;
    border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}

.navbar .navbar-brand img {
    height: 40px;
    width: 40px;
    border-radius: 50%;
    object-fit: cover;
}

.navbar .navbar-brand span {
    font-family: "Poppins", sans-serif;
    font-size: 1.5rem;
    font-weight: bold;
    transition: 0.2s;
    vertical-align: middle;
    padding-left: 5px;
}

.navbar .navbar-brand span:hover {
    color:#4d6e5c;
}

.navbar .container-fluid {
    padding-left: 3rem;
    padding-right: 3rem;
```

```
}

.navbar-scroll {
    background: #fff;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.hero {
    margin-top: 63px;
}

.hero h1 {
    font-size: 3rem;
    font-weight: bold;
    color: #668d76;
}

.hero-img {
    border-radius: 5px;
    width: 100%;
    object-fit: cover;
}

.section {
    padding: 5rem 0;
}

.navbar-brand, h2 {
    color: #668d76;
}

#people .card:hover {
    transform: scale(1.02);
    transition: transform 0.3s ease-in-out;
}

#people img {
    height: 500px;
    width: 100%;
    object-fit: cover;
    filter: grayscale(100%);
}

#sprints .sprint {
    margin: 50px auto;
}

#sprints .card img {
    padding: 10px;
```

```
    border-radius: 5px;
}

#group img {
    filter: grayscale(100%);
}

.btn-plant {
    background-color: #668d76;
    color: #fff;
}

.btn-plant:hover {
    background-color: #4d6e5c;
    color: #fff;
}
```

Listing B.2: style.css

```
jQuery(function($) {
    $(window).scroll(function() {
        if ($(window).scrollTop() > 0) {
            $(".navbar").addClass("navbar-scroll");
        } else {
            $(".navbar").removeClass("navbar-scroll");
        }
    });
});
```

Listing B.3: main.js

6 Scrum Presentation

EG | -

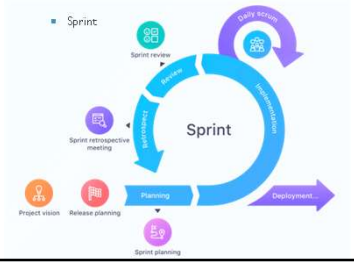
SCRUM: EN AGIL TILNÆRMING TIL PROSJEKTARBEID

HVORDAN SAMARBEIDE OG LEVEREVERDI GJENNOM ITERATIV UTVIKLING

1

VI SER PÅ

- Tradisjonell opp mot moderne prosjektarbeid
- Hovedprinsipper og verdier i SCRUM
- Forberedelse
  - Rollene
  - Visjon og Story board mapping
  - The Product Backlog – Backlog grooming
  - "The definition of Done"
  - Product Roadmap
- Scrum-struktur
  - Planning
  - Daily scrum
  - Sprint review
  - Retrospektive
- Sprint



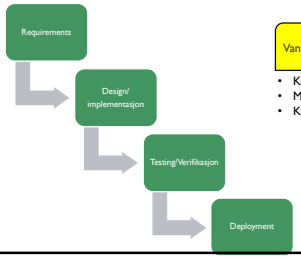
2

HVA ER UTFORDRINGEN?

Hvordan kan vi levere et knallbra prosjekt med ulik fagbakgrunn, komplekse utfordringer, og stramme tidsfrister?

3

TRADISJONELL MATE Å JOBBE PÅ

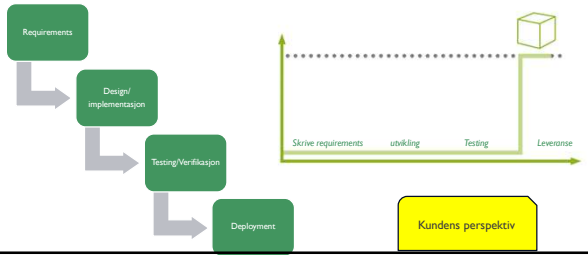


Vannfallsmetoden

- Kunden gir et oppdrag
- Magic occurs
- Kunden får produktet sitt til avtalt tid

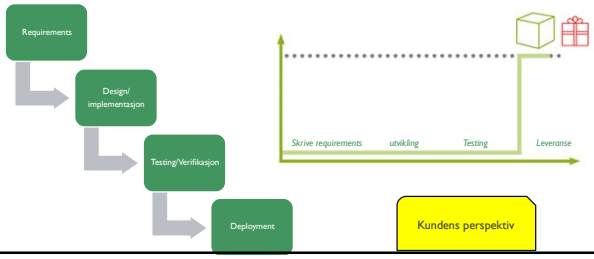
4

TRADISJONELL MÅTE Å JOBBE PÅ



5

TRADISJONELL MÅTE Å JOBBE PÅ



6

VEL OG BRA – I DEN IDEELLE VERDEN

Men, shit quite frequently happens:

- **Endring i krav eller spesifikasjoner**
  - Eks: Et team jobber med å designe en prototyp for en ny maskin, men halvveis i prosjektet endres spesifikasjonene fordi kunden ønsker å inkludere en ny funksjon basert på markedsundersøkelser.
- **Forsinket oppdagelse av feil**
  - En utvikler leverer et design som videreføres til produksjonsfasen, men under testfasen oppdages at en komponent ikke oppfyller ytelseskravene.
- **Manglende kundeinvolvering**
  - Et team utvikler en programvareløsning, men kunden er kun involvert i starten og slutten av prosjektet. Når løsningen leveres, oppdager kunden at produktet ikke oppfyller deres faktiske behov.
- **Uventede teknologiske endringer**
  - Et team jobber med å utvikle et nytt produkt basert på en bestemt teknologi, men halvveis i prosjektet lanseres en ny, mer effektiv teknologi som gjør den opprinnelige løsningen utdatert.

7

VEL OG BRA – I DEN IDEELLE VERDEN

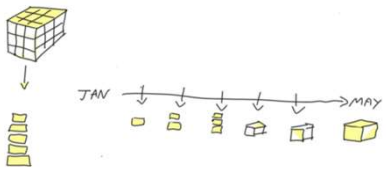
- Vannfallsmetoden gir lav fleksibilitet og tilpasningsevne
- Problemer oppdages sent, som ofte krever enorme ressurser å korrigere
- Legger ikke opp til jevnlig tilbakemelding fra kunde (som ikke nødvendigvis kan så mye teknisk)
- Planlegging og diskusjon rundt utviklingsoppgaver skjer primært i begynnelsen av prosjektet

En studie publisert i *International Journal of Project Management* fant at prosjekter som benyttet agile metoder hadde **28% høyere sannsynlighet for å levere på tid sammenlignet med de som brukte vannfallsmetoden.** Videre viste en rapport fra *Standish Group* at **agile prosjekter hadde en suksessrate på 42%, mens tradisjonelle vannfallsprosjekter lå på 14%**

8

AGILT PROSJEKTARBEID

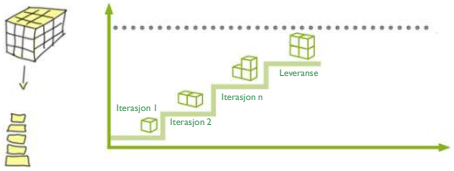
Stort prosjekt → Bryter opp I biter → Bitene utvikles (i iterasjoner) → Deler av produktet leveres underveis "releases" → Hele produktet ferdigstilles **til slutt.**



9

AGILT PROSJEKTARBEID

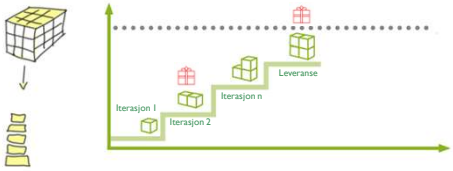
Handler om å bryte opp et stort prosjekt til mange små komponenter, og bygge dem opp bit for bit gjennom en strukturert, iterativ prosess.



10

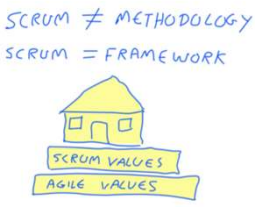
AGILT PROSJEKTARBEID

Handler om å bryte opp et stort prosjekt til mange små komponenter, og bygge dem opp bit for bit gjennom en strukturert, iterativ prosess.



11

AGILT….. SCRUM?

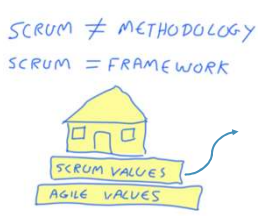


12

AGILT….. SCRUM?

13

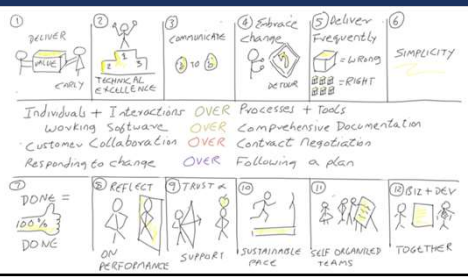
AGILT….. SCRUM?



- **Fokus** – Teamet fokuserer på sprintets mål og oppgavene som gir mest verdi for produktet.
- **Åpenhet** – Teamet har en transparent arbeidskultur hvor det deles åpent om fremdrift, utfordringer, og beslutninger.
- **Engasjement** – Teamet forplikter seg til å oppnå sprintets mål og jobber helhjertet for å levere verdi. Dedikasjon til både oppgaver og samarbeidet.
- **Respekt** – Teamet respekterer hverandres roller, ferdigheter og perspektiver.
- **Mot** – Teamet har mot til å ta utfordrende beslutninger, gi ærlige tilbakemeldinger, og adressere problemer direkte.

14

SCRUM MANIFESTO



15

OK, OK, OK….. NOK ABSTRAKTE GREIER

Over til noe litt mer håndfast og praktisk

16

EKSEMPEL CASE

Oppdrag:

Lage en interaktiv treningssykkel.

Development team

3x maskiningeniører 2x dataingeniører

17

SCRUM TEAM

Development team:

3x maskiningeniører 2x dataingeniører

I SCRUM har vi tre "roller"

Developer – Alle utviklerne på teamet

Product Owner - Product Owner er ansvarlig for å gi teamet klarhet i hva som skal bygges, og sikre at det reflekterer kundenes og interessentenes krav.

Scrum Master – Scrum Master hjelper teamet med å følge Scrum-prosessen, fjerner hindringer og sørger for at teamet kan jobbe effektivt. Altså ikke en «sjef», men en fasilitator som server teamet.

18

SCRUM TEAM

I SCRUM har vi tre "roller"

Developer – Alle utviklerne på teamet

Product Owner - Product Owner er ansvarlig for å gi teamet klarhet i hva som skal bygges, og sikre at det reflekterer kundenes og interessentenes krav.

Scrum Master – Scrum Master hjelper teamet med å følge Scrum-prosessen, fjerner hindringer og sørger for at teamet kan jobbe effektivt. Altså ikke en «sjef», men en fasilitator som server teamet.

Development team:

3x maskiningeniører 2x dataingeniører

19

VISJON - HØYNIVÅ BESKRIVELSE AV PRODUKTET

Funksjonelle krav

- **User Stories**
  - Fokus på brukerens behov og opplevelse (funksjonelle krav).
  - Formulert i et enkelt språk for å fremme god kommunikasjon mellom utviklere og ikke tekniske interessenter

Tekniske krav

- Fokus på de spesifikke tekniske egenskapene
- Presist, teknisk språk som beskriver hvordan produktet skal fungere.
- Spesifiserer målbare parametere, som ytelse, sikkerhet, toleranser, materialvalg, osv.

20

EKSEMPEL

Standardmal for User Stories:

Som [brukerrolle] vil jeg [mål/behov] slik at [fordel/verdi].

Funksjonelt krav

- Som en bruker vil jeg ha en treningssykkel som måler pedalfrekvens og motstand, slik at jeg kan holde oversikt over treningsøktene mine.

Tekniske krav

- Sykkelen skal være utstyrt med en magnetisk sensor som kan måle pedalfrekvens i området 20–150 omdreininger per minutt (RPM) med en nøyaktighet på ±2 RPM.
- Motstandssystemet skal være basert på et elektromagnetisk bremsesystem som kan generere belastning fra 10 til 500 watt, med en justerbarhet på minst 1 watt per trinn.

21

EKSEMPEL

Standardmal for User Stories:

Som [brukerrolle] vil jeg [mål/behov] slik at [fordel/verdi].

Funksjonelt krav

- Som en bruker vil jeg ha en treningssykkel som måler pedalfrekvens og motstand, slik at jeg kan holde oversikt over treningsøktene mine.

Tekniske krav

- Sykkelen skal være utstyrt med en magnetisk sensor som kan måle pedalfrekvens i området 20–150 omdreininger per minutt (RPM) med en nøyaktighet på ±2 RPM.
- Motstandssystemet skal være basert på et elektromagnetisk bremsesystem som kan generere belastning fra 10 til 500 watt, med en justerbarhet på minst 1 watt per trinn.

22

EKSEMPEL

Standardmal for User Stories:

Som [brukerrolle] vil jeg [mål/behov] slik at [fordel/verdi].

Funksjonelt krav

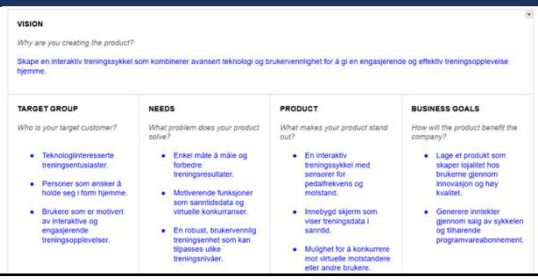
- Som en bruker vil jeg ha en treningssykkel som måler pedalfrekvens og motstand, slik at jeg kan holde oversikt over treningsøktene mine.

Tekniske krav

- Sykkelen skal være utstyrt med en magnetisk sensor som kan måle pedalfrekvens i området 20–150 omdreininger per minutt (RPM) med en nøyaktighet på ±2 RPM.
- Motstandssystemet skal være basert på et elektromagnetisk bremsesystem som kan generere belastning fra 10 til 500 watt, med en justerbarhet på minst 1 watt per trinn.

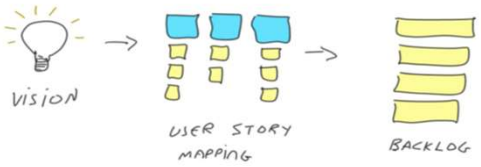
23

VISION BOARD



24

FRA VISJON TIL BACKLOG



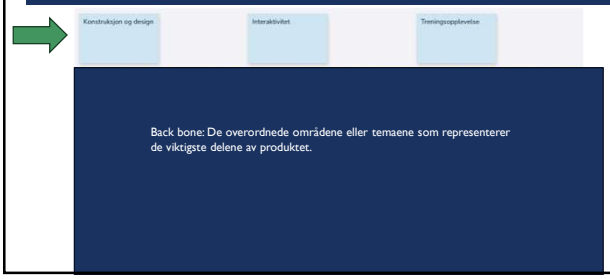
25

STORYBOARD MAPPING

…eller digital, f.eks med www.cardboardit.com

26

STORYBOARD MAPPING



Back bone: De overordnede områdene eller temaene som representerer de viktigste delene av produktet.

27

STORYBOARD MAPPING

"Epics": Dette er store funksjonelle blokker eller funksjonsområder innenfor hvert backbone item.

28

STORYBOARD MAPPING

Stories: Dette er de minste og mest detaljerte komponentene. De beskriver enkeltstående funksjoner eller bruksscenarioer som kan utvikles og testes separat

29

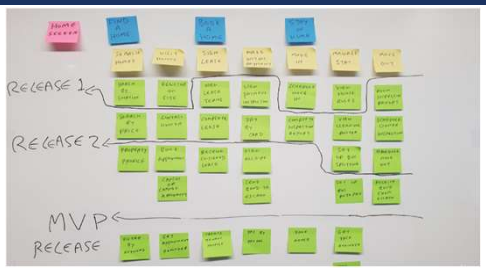
STORYBOARD MAPPING

30

STORYBOARD MAPPING

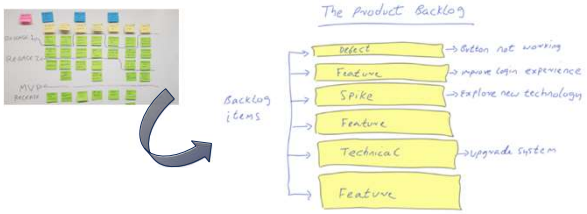
31

STORYBOARD MAPPING



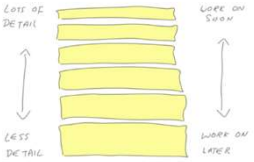
32

THE PRODUCT BACKLOG



33

BACKLOGGEN



Struktur
- Øverst: Mest prioriterte og detaljerte oppgaver, klare for utvikling.
- Nederst: Mindre detaljert, ideer og fremtidige planer.

Fleksibilitet
- Backloggen justeres etter behov, krav og innsikt.

Ansvar
- Product Owner administrerer prioriteringer.
- Teamet bidrar med estimering og detaljering.

Mål
- Fokuserer alltid på å levere mest verdi først.

34

BACKLOG GROOMING/REFINEMENT

Backlog grooming foregår regelmessig, og varer typisk 30-60 min.

De vanligste oppgavene

- At Backloggen er prioritert
- Estimert arbeid per story
- Oppdatere eller fjerne irrelevante stories
- Sikre at stories er "klare"
  - Kontrollere at de øverste oppgavene i backloggen har klare akseptansekriterier og er forståelige for teamet
- Definition of «Done»

35

MEN HVA BETYR AT EN STORY ER "KLAR"?

36

EKSEMPEL PÅ EN "KLAR" USER STORY

Tittel: Justerbart sete for ergonomisk tilpasning

Beskrivelse:
Som en bruker ønsker jeg å kunne justere setet opp og ned samt frem og tilbake, slik at jeg kan tilpasse sykkelen til min kroppsstørrelse for en komfortabel og trygg treningsopplevelse.

Akseptansekriterier:

- Setet skal kunne justeres vertikalt i minst 5 nivåer (3 cm intervaller).
- Setet skal kunne skyves horisontalt (minst 10 cm justeringsområde).
- Mekanismen skal være enkel å betjene med én hånd.

Tekniske detaljer:

- Bruk en låsemekanisme med fjærbelastning for enkel justering.
- Materialer skal tåle en vektbelastning på opptil 150 kg.
- Test for minst 10 000 sykluser for slitestyrke.

Estimat:
3 dager arbeidstid (inkludert testing).

"Klar" betyr at den er godt definert, forståelig og arbeidsmengde er estimert.

37

MEN HVA BETYR AT EN STORY ER "KLAR"?

Ja, ja, ja. Hva med når den er "Ferdig" da?

38

EKSEMPELER PÅ "DEFINITION OF DONE"

For en software story:

- Koden er skrevet og sjekket inn i versjonskontrollsystemet.
- Enhetstester er skrevet og bestått.
- Funksjonen er manuelt testet og validert mot akseptansekriterier.
- Ingen kjente feil knyttet til oppgaven.
- Dokumentasjonen er oppdatert (brukerveiledning og tekniske spesifikasjoner).

For et fysisk produkt:

- Mekanismen fungerer som spesifisert under realistiske forhold.
- Komponenter har bestått holdbarhetstesting (f.eks. stress- eller brukssimulering).
- Brukermanual er oppdatert.
- Produktet er montert og klar for kundevalidering.

39

NÅ ER VI STRAKS KLARE!

DET SISTE VI TRENGER ER EN TIDSLINJE – ET PRODUCT ROADMAP

40

PRODUCT ROADMAP

DATE	1ST QUARTER	2ND QUARTER	3RD QUARTER
NAME	Release 1	Release 2	MVP Release
GOAL	**Acquisition**: Onboarding initial 100 home seekers for user testing "Find a Home" experience	**Acquisition**: User testing for "Book a Home" experience	**Activation**: User testing for "Full experience"
STORIES	1. Register on site 2. Filter by location 3. Filter by price 4. View property profile 5. View lease terms 6. Complete lease 7. View house rules 8. Contact owner	1. Book appointment 2. Cancel or change appointment 3. Receive co-signed lease 4. View payment instructions 5. Pay by credit card 6. View/print/email receipt 7. Send bond to escrow 8. Schedule move in	1. Complete inspection report 2. View cleaning roster 3. Set up bill splitting 4. Set up auto pay for bills 5. Complete room inspection report 6. Schedule inspection by owner 7. Schedule move out 8. Receive bond from escrow
METRICS	100 registered users % registration completions % searches with owner contacted % lease completions Average # of profiles viewed per session	% sessions with appointment bookings % of cancelled bookings % of changed appointments Daily active users Return users Net promoter score	1,000 active users by end of 3rd quarter % of tenants who complete inspection % of tenants who set up bill splitting % of tenants who set up auto-pay

Link

41

KLAR FOR SPRINT PLANNING

Nå har vi:

- Story board map ✓
- Groomed Product Backlog ✓
- Definition of Done ✓
- Product Roadmap ✓

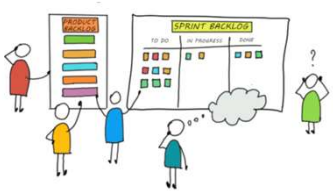
= masse forarbeid i boks

…Endelig kan vi snakke om sprinter

42

SPRINT

- **Time boxing**
  - Sprints har fast varighet
- **Sprint-mål**
  - Et klart og overordnet mål for hva teamet skal oppnå.
- **Inkrement**
  - Resultatet er et fungerende produkt eller funksjonalitet som kan vurderes.
- **The sprint board**



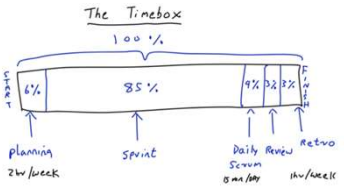
43

NOEN FASTSATTE MØTER

- **Sprint planning**
  - Planlegge arbeidet som skal løses
- **Daily Scrum**
  - Et standup-møte på max 15 minutter hver dag
- **Sprint Review**
  - Gjennomgang og demonstrasjon av fullførte leveranser til kunde
- **Sprint retrospektive**
  - Evaluering av hva som fungerte bra og hva som kan forbedres i neste sprint

44

NOEN FASTSATTE MØTER

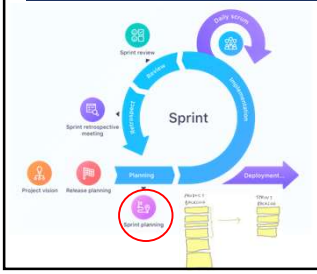


- **Sprint planning**
  - Planlegge arbeidet som skal løses
- **Daily Scrum**
  - Et standup-møte på max 15 minutter hver dag
- **Sprint Review**
  - Gjennomgang og demonstrasjon av fullførte leveranser til kunde
- **Sprint retrospektive**
  - Evaluering av hva som fungerte bra og hva som kan forbedres i neste sprint

…ser kanskje mye ut, men er faktisk veldig tideffektivt!

45

SPRINT PLANNING



Sprint Planning

Formål: Bestemme hva som skal oppnås og hvordan arbeidet skal utføres i sprinten.

Deltakere: Scrum Master, Product Owner og Utviklingsteamet.

Innhold:

- Velge stories fra backloggen som skal fullføres i sprinten.
- Planlegge oppgaver og arbeidsflyt for å levere sprintmålet.

Resultat: Et klart Sprint-mål og en *Sprint Backlog*.

46

DAILY SCRUM

Daily Scrum

Formål: Sikre fremdrift mot sprintmålet og identifisere hindringer.

Tid: Maks 15 minutter – hold møtet kort og fokusert.

Deltakere: Hele Scrum-teamet.

Struktur:

- Hva har jeg gjort siden sist?
- Hva skal jeg gjøre i dag?
- Er det noen hindringer som står i veien?

Resultat: Fremmer kommunikasjon, synlighet og ansvarlighet.

47

SPRINT REVIEW

Sprint Review

Formål: Evaluere og demonstrere arbeidet som er fullført i sprinten.

Deltakere: Scrum-teamet og interessenter (stakeholders).

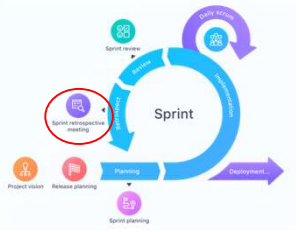
Innhold:

- Demonstrasjon av det fungerende produktinkrementet.
- Diskusjon om hva som er oppnådd, og hva som gjenstår.
- Innsamling av tilbakemeldinger fra interessenter.

Resultat: Oppdatering av Backloggen basert på tilbakemeldinger.

48

SPRINT REVIEW



Sprint Retrospective

Formålet: Reflektere over sprinten og finne forbedringsområder.

Deltakere: Kun Scrum-teamet (trygt rom for ærlig diskusjon).

Innhold:

- Hva fungerte bra?
- Hva kan forbedres?
- Hvilke tiltak kan vi iverksette for å bli bedre?

Resultat: Konkrete forbedringspunkter for neste sprint.

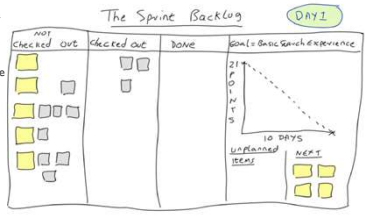
49

LEGG OPP TIL SUKSESS

50

SCRUM BOARD

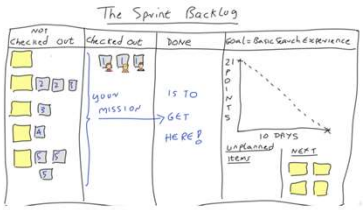
- Visualisere fremdrift, oppgaver og status på arbeidet i sprinten.
- Kan være fysisk (whiteboard) eller digitalt.
- Mange måter, men veldig vanlig å kombinere med Kanban
- Brukes aktivt i møter og for å holde synlig oversikt



51

SCRUM BOARD

- Visualisere fremdrift, oppgaver og status på arbeidet i sprinten.
- Kan være fysisk (whiteboard) eller digitalt.
- Mange måter, men veldig vanlig å kombinere med Kanban
- Brukes aktivt i møter og for å holde synlig oversikt



52

TO ORD OM MULTITASKING

Oh no….A "fun, little exercise" time!

Lettevs	Numbevs	Roman Numevals
1	A	I
2	B	II
3	C	III
4	D	IV
5	E	V
6	F	VI
7	G	VII
8	H	VIII
9	I	IX
10	J	X

53

TO ORD OM MULTITASKING

Oh no….A "fun, little exercise" time!

Lettevs	Numbevs	Roman Numevals
1	A	I
2	B	II
3	C	III
4	D	IV
5	E	V
6	F	VI
7	G	VII
8	H	VIII
9	I	IX
10	J	X

54

TO ORD OM MULTITASKING

Oh no….A "fun, little exercise" time!

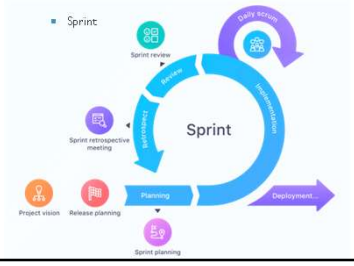
Er det effektivt å multitaske?

Letters	Numbers	Roman numerals
1	A	i
2	B	ii
3	C	iii
4	D	iv
5	E	v
6	F	vi
7	G	vii
8	H	viii
9	I	ix
10	J	x

55

OPPSUMMERING

- Tradisjonell opp mot moderne prosjektarbeid
- Hovedprinsipper og verdier i SCRUM
- Forberedelse
  - Rollene
  - Visjon, User Stories, Story board mapping
  - The Product Backlog – Backlog grooming
  - "The definition of Done"
  - Product Roadmap
- Gjennomføring - Sprints
  - Planning
  - Daily scrum
  - Sprint review
  - Retrospektive
- Sprint



56

WE'RE DONE!

57

7 ClickUp sprints and backlog

7.1 Sprints

ClickUp

Sprint 1

COMPLETE

Task ID	Task Name	Assignee
8698pc5rk	pulleys og belte bestilling til shahin	Daniels Blomnieks
8698kzt8e	Complete the HMI authentication system and user database	Jim Christian Dale Haukvik
8698kd74y	Setup ROS2 Workspace and Repository	Elin Gravningen

IN PROGRESS

Task ID	Task Name	Assignee
8698kzwhn	Choose components	Vetle Myhre Nilsen
8698kzu5t	Draw schematic	Vetle Myhre Nilsen
8698kx0kz	Bli enige om interface specifications	Elin Gravningen, Jim Christian Dale Haukvik, Vetle Myhre Nilsen
8698kzt2m	Define requirements for base	Beatrix Rimestad
8698kztmz	Cad tegning base (V1, V2, V3...)	Beatrix Rimestad
8698mf98y	Joint 3	Daniels Blomnieks
8698kvzx3	CAD Prototype 1	Sunniva Eineteig

TO DO

Task ID	Task Name	Assignee
8698kwzub	Define All ROS2 Interfaces (msg/srv/action)	Elin Gravningen
8698kx1ef	Sette opp nodene som skal utvikles	Elin Gravningen
86987kz27	Implement some kind of AI model which can give information about	Jim Christian Dale Haukvik
8698kzqq7	Complete Object Detection AI model implementation (no training).	Jim Christian Dale Haukvik
8698kw191	Choose soft touch for the gripper	Sunniva Eineteig

Figure B.19: Sprint1

Sprint 2

Name	Assigne	Status
Joint 2	Daniels	COMPLETE
Cad tegning base (V1, V2, V3 ...)	Beatrix	COMPLETE
Produkesjon av enheter til base	Beatrix	COMPLETE
Complete object detection AI model	Jim	COMPLETE
Train Object Detection AI model with open-source dataset from Hugging Face and deduce viability.	Jim	COMPLETE
CAD Prototype (gripper)	Sunniva	COMPLETE
Choose soft touch for the gripper	Sunniva	COMPLETE
Define all ROS2 Interfaces (msg / srv / action)	Elin	COMPLETE

Sprint 3

Name	Assigne	Status
Sette opp nodene som skal utvikles	Elin	Complete
create cable routing clips to frame	Daniels	Complete
Joint 1	Daniels	Complete
Arduino - CommunicationManager	Elin	Complete
Arduino - MotorControl	Elin	Complete
Implement some kind of AI model which can give information about the salads size / dimensions	Jim	Complete
Arduino - GripperControler	Elin	Complete
Define requirements for base	Beatrix	Complete
V5 - Base	Beatrix	Complete
Bli enig om interface specification	Jim, Elin, Vetle	IN PROGRESS
finne en løsning på 20mm shaft	Daniels	IN PROGRESS
CAD model with soft touch	Sunniva	IN PROGRESS
3D-print soft touch (TPU)	Sunniva	IN PROGRESS
Choose components	Vetle	TO DO
> MOSFET simulation	Vetle	TO DO
Design an interchangable gripper mechanism	Sunniva	TO DO
PCB Layout	Vetle	TO DO
> PCB art	Vetle	TO DO
> ECM	Vetle	TO DO
Draw schematic	Vetle	TO DO
> EMC checks and ...	Vetle	TO DO
> Simulation of RC filters for limit switches	Vetle	TO DO
mount for camera frame	Daniels	TO DO

x

7.2 Backlog

Backlog

Name	Assigne	Status	List
Vurdere Github repo for nettside	Jim, Elin	TO DO	Web
LaTeX kurs for gruppen	Daniels, Vetle	TO DO	Felles arbeid
Rydde opp i rapport - mapper	Daniels, Vetle	TO DO	Felles arbeid
Robotics core	Elin		
> Define interfaces between core and motors, including sensors.	Elin, Vetle	TO DO	Dataingeniør arbeid
> Define interfaces between core and central	Elin, Jim	TO DO	Dataingeniør arbeid
> Single stepper motor control	Elin	TO DO	Dataingeniør arbeid
>> Rotate motor forwards and backwards		TO DO	Dataingeniør arbeid
>> Move motor at different speeds (needs to be defined)		TO DO	Dataingeniør arbeid
>> Test that kill switch stops operations but holds position		TO DO	Dataingeniør arbeid
> Expand code to operate 2 motors	Elin	TO DO	Dataingeniør arbeid
>> Each motor works independently		TO DO	Dataingeniør arbeid
>> Both motors can operate independently at the same time		TO DO	Dataingeniør arbeid
> Implement synchronized movement	Elin	TO DO	Dataingeniør arbeid
>> Both motors move together at the same speed		TO DO	Dataingeniør arbeid
> Make motor control library	Elin	TO DO	Dataingeniør arbeid

>> Reusable functions for moving, stopping, speed, changing direction, kill switch. (?)		TO DO	Dataingeniør arbeid
> Implement limit switches	Elin, Vetle	TO DO	Dataingeniør arbeid
>> Movement stops when limit is reached		TO DO	Dataingeniør arbeid
>> The system logs and reports the occurrence to operator/farmer?		TO DO	Dataingeniør arbeid
> Implement emergency stop	Elin, Vetle	TO DO	Dataingeniør arbeid
>> Pressing the button immediately halts all movement.		TO DO	Dataingeniør arbeid
>> Resume button that resumes action		TO DO	Dataingeniør arbeid
>> Reset button sending robots back to starting positions?			
> Position tracking?	Elin	TO DO	Dataingeniør arbeid
> Automate movement sequences	Elin	TO DO	Dataingeniør arbeid
>> Set a "home" position		TO DO	Dataingeniør arbeid
> System integration	Elin	TO DO	Dataingeniør arbeid
> Implement gripper pressure sensor	Elin, Vetle	TO DO	Dataingeniør arbeid
> Implement motor currency sensors	Elin, Vetle	TO DO	Dataingeniør arbeid
Choosing motor	Vetle	TO DO	Elektronikkingeniør arbeid
Stepper motor driver development	Elin, Vetle	TO DO	Elektronikkingeniør arbeid
> Current sensing		TO DO	Elektronikkingeniør arbeid
> Stallguard		TO DO	Elektronikkingeniør arbeid
> Limit switches		TO DO	Elektronikkingeniør arbeid
> Rotary encoder		TO DO	Elektronikkingeniør arbeid
Power delivery	Vetle	TO DO	Elektronikkingeniør arbeid
Choosing interface for motors	Vetle	TO DO	Elektronikkingeniør arbeid

PCB?	Vetle	TO DO	Elektronikkingeniør arbeid
Force sensor		TO DO	Elektronikkingeniør arbeid
Bilder opp fra bedriftsbesøk på Insta	Sunniva	TO DO	SoMe
Based on results from open-source object detection dataset, decide if we need to compile our own dataset.	Jim	TO DO	AI
Compile a dataset with associated labels and bounding boxes from images of the lettuce.	Jim	TO DO	AI
Train the object detection AI model with the dataset	Jim	TO DO	AI
Build standalone Python script for live detection	Jim	TO DO	AI
Refactor AI for ROS2 compatibility	Jim	TO DO	AI
Finalize AI models	Jim	TO DO	AI
Opprette kommunikasjon mellom nodene i ROS2		TO DO	Robotics
ROS2 Utvikle arm_controller		TO DO	Robotics
ROS2 Utvikle FSM (task planner)		TO DO	Robotics
ROS2 Innlemme og interface AI --> task planner		TO DO	Robotics
ROS2 Implementere motion_planner (Enkel)		TO DO	Robotics
ROS2 Implementere FSM		TO DO	Robotics
Lage felles ROS2 launch-fil		TO DO	Robotics
Full Hardware/software-test		TO DO	Robotics
Legge til kallibreringsrutine		TO DO	Robotics
Sette opp MQTT Broker & ROS2		TO DO	Robotics
Complete the http protocol class	Jim	TO DO	Misc

Finalize first iteration of API layer between central and core	Jim	TO DO	Misc
Check if citations are correct according to style guidelines	Jim	TO DO	Misc
Write about login table and login screen in thesis	Jim	TO DO	Misc
Write a section about ClickUp		TO DO	Misc
Regenerate Doxygen docs for code and put in the thesis before delivering	Jim	TO DO	Misc
Explain the code I wrote in the thesis	Jim	TO DO	Misc
Lisens på Powerpoint og Visio stock images. Kan de brukes i bachelor rapport?	Jim	TO DO	Misc
Make sure variable names and product names are italic or bold text in thesis	Jim	TO DO	Misc
Implement token based security for the API	Jim	TO DO	Misc
Implement a class abstraction for interacting with the camera	Jim	TO DO	Camera
Refactor Camera for ROS2 compatibility	Jim	TO DO	Camera
Create a login screen and users database for HMI	Jim	TO DO	HMI
Build a live detection demo	Jim	TO DO	HMI
Design basic HMI layout	Jim	TO DO	HMI
Develop basic functional HMI	Jim	TO DO	HMI
Finalize HMI	Jim	TO DO	HMI

Fine tune and test the Green Percentage Image segmentation algorithm	Jim	TO DO	ML
Complete database for HMI, Camera and AI	Jim	TO DO	Network
Prepare integration adapter for ROS2 (future-ready)	Jim	TO DO	Network
Authorization checks for api	Jim	TO DO	Network
Document AI pipeline and UI behavior	Jim	TO DO	General
Finalize software, document architecture, and record test logs.	Jim	TO DO	General
CAD simuleringer	Beatrix	IN PROGRESS	Robot base
Valg av materialer	Beatrix	IN PROGRESS	Robot base
Cad tegning kasse kontrollenheter	Beatrix	TO DO	Robot base
Produksjon av elementer til kasse kontrollenheter	Beatrix	TO DO	Robot base
Dokumentere base og design begrunnelser		TO DO	Robot base
development of arm prototype	Daniels	IN PROGRESS	Robot arm
> joint belt drive	Daniels	IN PROGRESS	Robot arm
>> SW simulations	Daniels	IN PROGRESS	Robot arm
> joint direct drive	Daniels	IN PROGRESS	Robot arm
>> SW simulations	Daniels	IN PROGRESS	Robot arm
> simulation of assembly	Daniels	IN PROGRESS	Robot arm
physical prototype	Daniels	TO DO	Robot arm
3D print	Daniels	TO DO	Robot arm
Gripper/end effector research og konseptforslag	Sunniva	IN PROGRESS	Gripper / end effector

> Design for pressure/force sensor or mechanical stop	Sunniva	IN PROGRESS	Gripper / end effector
> Soft touch på end effectoren, hva funker best?	Sunniva	IN PROGRESS	Gripper / end effector
> Sensor touch? Eller mekanisk	Sunniva	IN PROGRESS	Gripper / end effector
> Kamera plassering på griper eller håndledd på arm?	Sunniva, Jim	IN PROGRESS	Gripper / end effector
> Motor til griper, hvilken type?	Sunniva, Vetle	IN PROGRESS	Gripper / end effector
> Kobling fra arm til griper	Sunniva, Daniels	IN PROGRESS	Gripper / end effector
gripe arm konsepter	Sunniva	IN PROGRESS	Gripper / end effector
> kanskje solidworks modell?	Daniels	IN PROGRESS	Gripper / end effector
Requirements: Calculations (weight, force, torque, etc.)	Sunniva	TO DO	Gripper / end effector
FEM analysis in SW of gripper	Sunniva	TO DO	Gripper / end effector
3D-Print gripper parts	Sunniva	TO DO	Gripper / end effector
connect gripper parts after print	Sunniva	TO DO	Gripper / end effector
Install motor/servo for testing	Sunniva, Vetle	TO DO	Gripper / end effector
Perform grip test	Sunniva	TO DO	Gripper / end effector
Connect gripper/end effector to robot arm	Sunniva, Daniels	TO DO	Gripper / end effector
Final testing	Jim, Elin, Beatrix, Sunniva, Daniels, Vetle	TO DO	Gripper / end effector
Final documentation	Sunniva	TO DO	Gripper / end effector
Teste assembly for dynamiske belastninger i forskjellige konfigurasjoner (FEM)	Beatrix	TO DO	Structural integrity
Order PCB	Vetle	TO DO	Custom PCB
Order components	Vetle	TO DO	Order components

Appendix C

mechanical

1 belts and pulleys

DAB |

just a more detailed explanation. all parts are from maedler website [28]

1.0.1 Pulleys

DAB |

pulley choice was made based on what bearings we had available based on its internal diameter. But pulley for joint 1 had to be also sized so that the pulley itself is not too big and comes in the way of maneuverability of the robot.
the list and info of pulley is in belt drive subsection 9. These pulleys do not have set screws, its a clamp type where its held by the clamping force. one motor shaft was to short to have the belt close to the frame as possible, so we decided to add 2 set screw M3 in pulley groove to fasten it see in figure C.1.

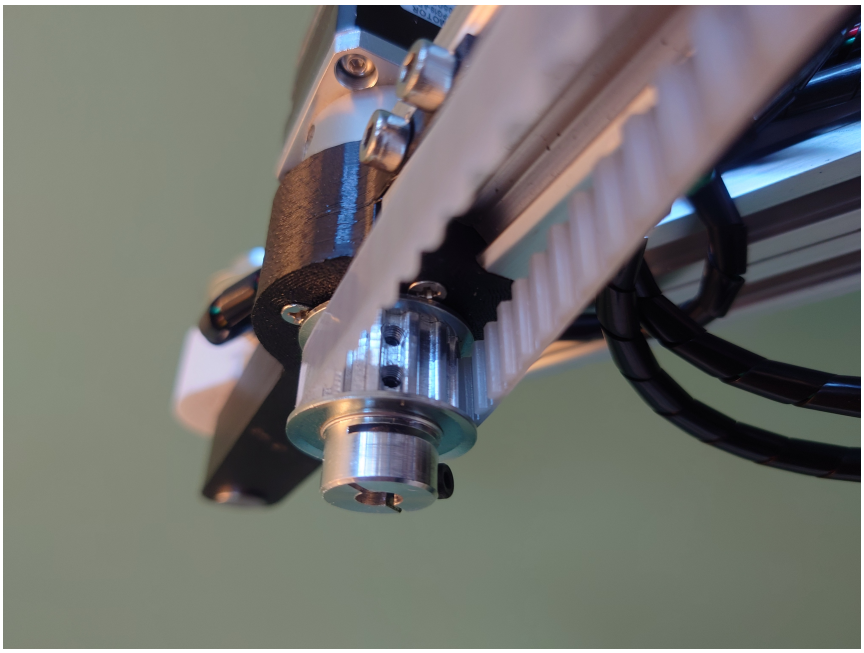


Figure C.1: pulley joint 3 motor side

There is also a pulley for joint 2 motor side, the shaft is almost to short so a good idea would be to also add 2 set screws to it instead of relying on the clamp with minimal contact.
CAD models for pulleys are all available to download from maedler supplier. use of Their cad files are for internal use only and not for commercial use. since they just redirect to a government site abut the general use of Cad files.

1.0.2 Belts

DAB |

Belt type is T5 10mm belt. the five means the distance from center of the peak to center of the peak is 5mm that is called pitch, and width of 10mm. these belts are timing belts
Belt lengths were chosen trough meddler calculator. There you can input pulley teeth amount and the distance from the pulleys center to center, and also select what kind of belt width and type. then it gives the correct belt length.
after that we look at what is the closest size they sell and chose a little bit over size a few mm to be on the safe side and we will add a belt tensioner which will almost eliminate all the sag in the belt.

2 Robot Gripper Concepts

SME |

A gripper is the mechanical component of a robot's end effector designed to grasp, hold, or transport objects. Performing as the "hand" of a robotic arm, the gripper enables the robot to interact with physical objects efficiently.

The most common types of grippers include vacuum, pneumatic, hydraulic, and electric grippers. In our project, we prioritize high flexibility and the use of materials that do not damage plants. Through research, I identified various types of grippers, some of which are more suitable for our application than others.

2.0.1 Jaw gripper

SME |

One type we examined is the jaw gripper, a parallel gripper that is among the most versatile and widely used in robotics. This gripper features two opposing jaws that move parallel to each other, enabling it to grasp objects of different shapes and sizes. The jaw gripper is well suited for pick-and-place operations due to its precision and ability to handle both small and large irregularly shaped objects. Additionally, by controlling the gripping force and incorporating soft-touch materials on the jaws, this type of gripper can safely manage fragile objects or materials requiring a delicate touch.

Figure C.2: Jaw gripper
[65]

2.0.2 Finger gripper

SME |

Another end effector that we explored is the finger gripper, which offers greater flexibility than

the jaw gripper. This type is particularly effective for handling objects with irregular shapes or intricate geometries. Finger grippers provide a highly adaptable solution for robotics, as they consist of multiple fingers that conform to an object's shape during grasping. The fingers can be made from rigid materials for stability or soft materials to protect delicate items, depending on the specific application. A combination of both materials can also be used to balance gripping force with a gentle touch.

Figure C.3: Finger gripper
[65]

2.0.3 Soft gripper

SME |

Lastly, we investigated soft grippers, which include several gripper types. These grippers are highly adaptable and applicable across various industries. In agricultural harvesting, soft-touch technology has recently gained attention due to its ability to handle delicate crops, cost-effectiveness, and potential for automated harvesting. Soft grippers can consist of one or multiple fingers and are often constructed from rubber or silicone, making them suitable for diverse applications.

Figure C.4: Soft gripper
[66]

Soft grippers are primarily powered by pneumatic or hydraulic systems, utilizing compressed air or pressurized fluids to generate mechanical movement. This energy conversion mechanism allows for smooth and controlled gripper operation.

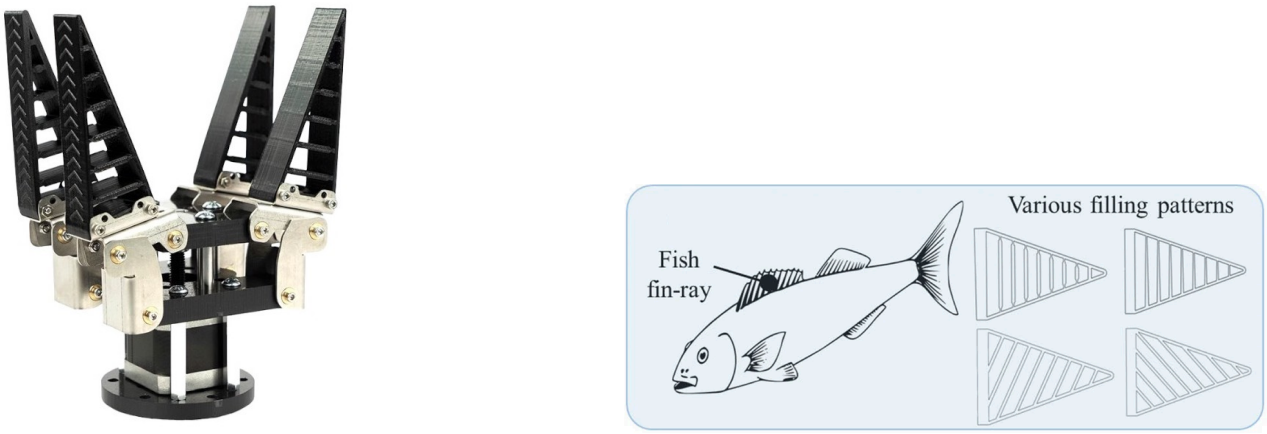
2.0.4 Fin-ray grippers

SME |

Fin-Ray grippers are a type of soft robotic gripper designed for grasping complex and deformable objects. They are inspired by the fin-ray effect, a natural phenomenon observed in the fins of fish. This principle has been adapted for robotics, enabling grippers to adapt to an object's shape without requiring extensive actuation systems.

The structure of Fin-Ray grippers is characterized by a triangular shape, where the fins consist of two flexible elements arranged in a V-shaped configuration. When an external force is applied to the fins, the structure deforms in a way that enhances the gripping capability, allowing the gripper to wrap around objects gently and securely.

One of the key advantages of Fin-Ray grippers is their ease of design and manufacturing. The fin-ray effect can be efficiently modeled using CAD software and is well-suited for additive manufacturing. These grippers can be 3D-printed, making them a cost-effective and accessible solution for various robotic applications. [67]



(a) Fin-ray Gripper

(b) Fish fin-ray effect

Figure C.5: Fin-Ray concept
[68]

2.1 Soft Touch in Agricultural Robotics

SME |

In agricultural robotics, particularly in environments where robots handle delicate food products such as salad greens, a soft touch feature is usually incorporated to prevent damage to

the plants. Traditional robotic grippers, which are often designed for industrial applications, may apply excessive force on fragile products, leading to reduced quality or waste. Therefore, integrating soft touch technology in robotic grippers enhances their ability to handle sensitive agricultural products effectively.

The materials commonly used to achieve a soft touch include silicone and rubber, both of which provide flexibility and cushioning to minimize pressure on delicate items. However, when selecting materials for agricultural applications, it is essential to ensure compliance with food-contact safety regulations. Regulatory standards dictate that all materials in direct contact with edible products must be non-toxic, non-reactive, and free from harmful substances to maintain food safety and quality.

Soft touch technology is typically implemented in the fingers of the robotic gripper, as these components make direct contact with plants or products. One of the most effective methods for achieving a soft touch is silicone molding, a process that allows for precise shaping of the gripper's contact surfaces. Silicone molding offers several advantages, including customizability, durability, and biocompatibility, making it a suitable choice for agricultural applications. By tailoring the shape and flexibility of the gripper fingers, robots can handle a wide variety of plants while ensuring minimal stress on the harvested items.

The integration of soft touch technology in robotic grippers represents a significant advancement in automated harvesting and food processing. With the increasing demand for automation in agriculture, soft touch solutions not only improve efficiency but also contribute to reducing food waste by ensuring gentler handling of crops. As research and technology evolve, further innovations in material science and robotic control mechanisms will likely enhance the effectiveness and adaptability of soft touch grippers in agricultural applications. [69]

3 3D-Printing for gripper development

Additive Manufacturing Overview

3D printing, also known as additive manufacturing, is a fabrication process that creates objects by depositing material layer by layer, directly from a digital model. This technique contrasts with traditional subtractive methods such as milling or turning, where material is removed from a solid block.

For prototyping purposes, 3D printing offers significant advantages in terms of speed, cost-efficiency, and design flexibilityespecially for iterative development and rapid testing cycles.

Materials and Techniques

The gripper components in this project were manufactured using FDM, one of the most accessible and widely used 3D printing methods. This technology melts thermoplastic filaments and extrudes them layer by layer to form the final geometry.

Two primary materials were used:

- PLA: Chosen for its ease of printing, rigidity, and availability. PLA was used for all structural components where stiffness and dimensional accuracy were required.

- TPU: Selected for the Fin-Ray fingers in Prototype 2, TPU offers the flexibility and elasticity necessary for soft-touch gripping. It allows the fingers to deform around the object being handled, reducing the need for sensors while enhancing adaptability.

Design Considerations

Because 3D printed parts generally have lower mechanical strength compared to machined metal parts, the designs had to account for stress distribution, layer adhesion, and print orientation. The geometry was optimized to avoid weak points, and features such as ribbing, filleting, and strategic wall thickness were incorporated to enhance structural integrity.

Moreover, modularity was prioritized to facilitate reprinting and replacement of individual components without needing to manufacture the entire assembly again. This modular approach was especially valuable for testing and educational environments with limited budgets and access to industrial manufacturing tools.

Benefits and Limitations

The primary advantage of using 3D printing in this context was the ability to rapidly iterate on the design at minimal cost. Changes could be made in CAD and quickly verified with a physical prototype. However, 3D printing is not ideal for all application components that experience high mechanical loads or require long-term durability, and may eventually need to be replaced with machined metal versions. These considerations are explored in more detail in chapter 15.1.

Appendix D

Mechanical design

1 Forces acting on base

BMR | —

When designing and choosing the different elements for the base, it has been important to know what forces the base will be subjected to, both in static equilibrium and during operation. In addition to the combined weight of the whole arm, the base also needs to handle the moment caused by off center mass, see fig. D.1.

To lower the strain on the motor shaft and reduce the power wasted on sliding friction between rotating parts, we use bearings to support and control both the axial load and the moment forces caused by off center load.

The axial load comes from the weight of the complete arm with payload (around 80 N). The radial load comes from the moment of the off center mass of the outstretched arm, and the magnitude of this force to be transferred through the bearings is dependent on the distance and arrangement of the bearings.

The motor needs to be strong enough to rotate the combined weight of the arm with poayload at the specified speed.

To find the exact values for this, a dynamic analysis with the inertia of each element in the arm would be the correct way. This is a quite complex calculation, and is left out for this thesis.

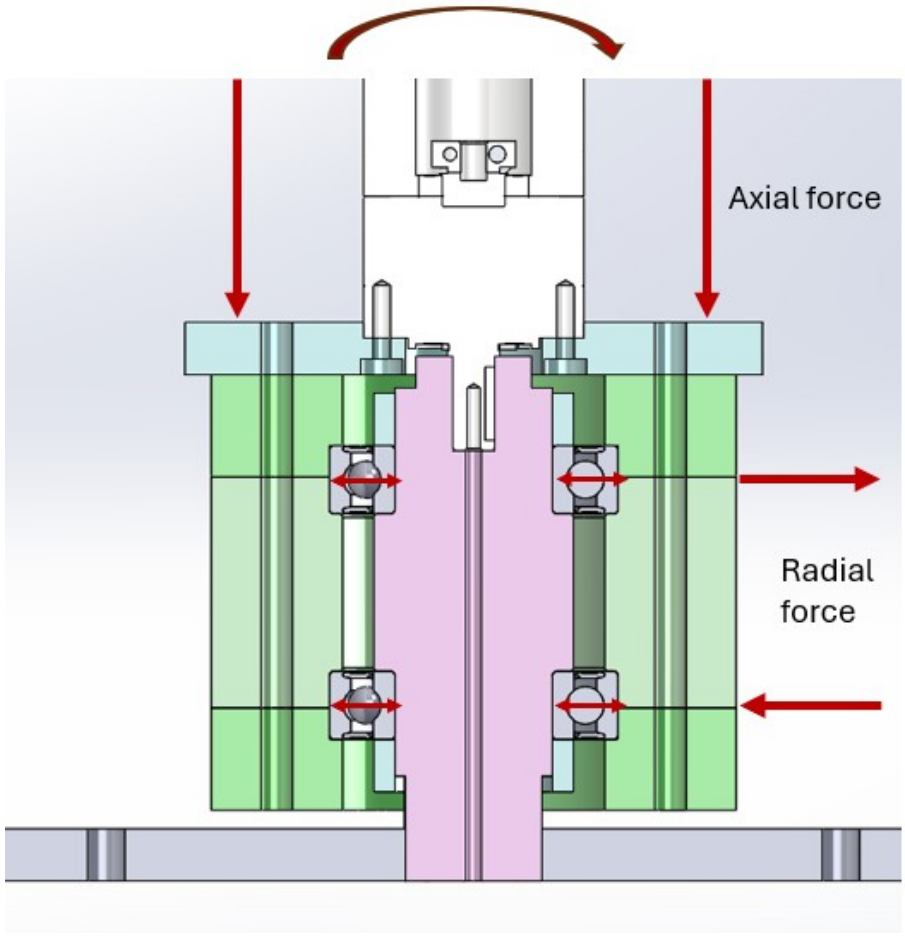


Figure D.1: Forces acting on base and bearings

2 base HPT interface

BMR | —

For mounting of the robot arm, the base flange needs to be secured to a fixed surface by 4 X M8 bolts (see fig.). This could be directly on a working table, on a separate mounting table or for example a rail system (see fig.).

Important to consider - calibrate - put in parameter for the height (difference working table/-mounting table).

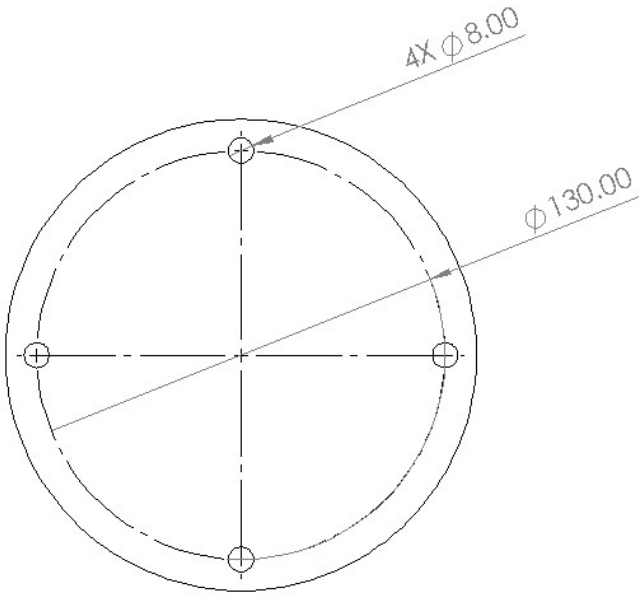


Figure D.2: Interface for base flange

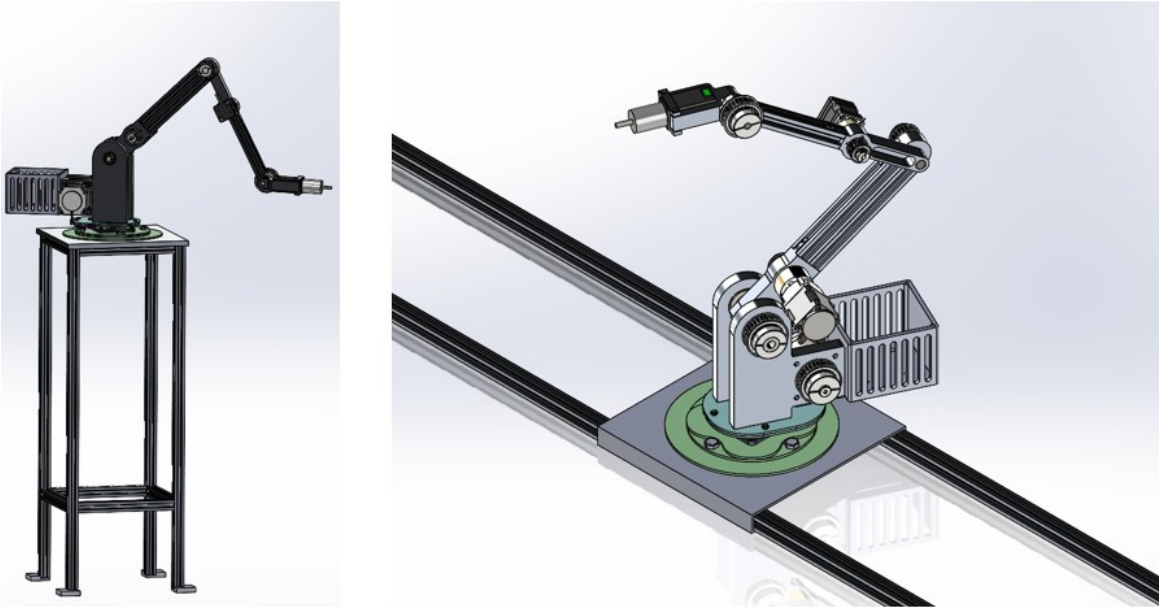


Figure D.3: Interface future possibilities

3 full scale model

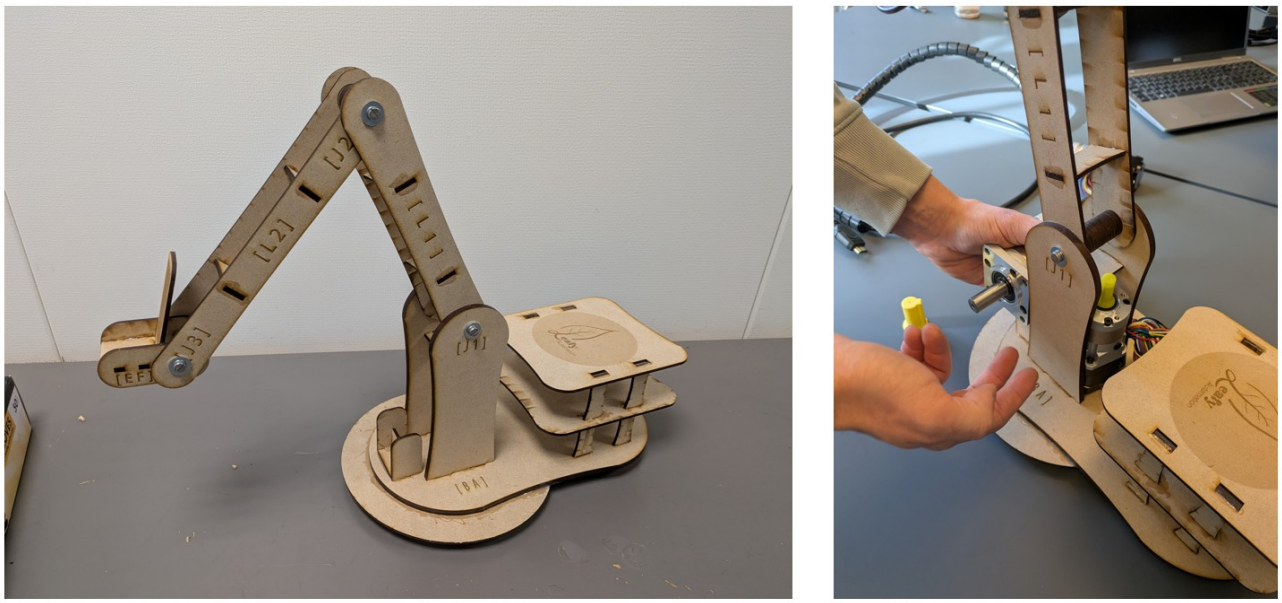


Figure D.4: Full scale model

4 Further work on base

This section contains some thoughts around the future development of the base that has not been implemented yet due to time constraints.

FEA and shaft layout

Initial analysis on the shaft has given a better understanding into where the stress concentrations and problematic areas are situated and which applied loads affect the shaft the most. Running the studies with different shaft lengths and different distances/loads between the two bearings gave some insights. Figure D.5 shows the positioning of the applied loads.

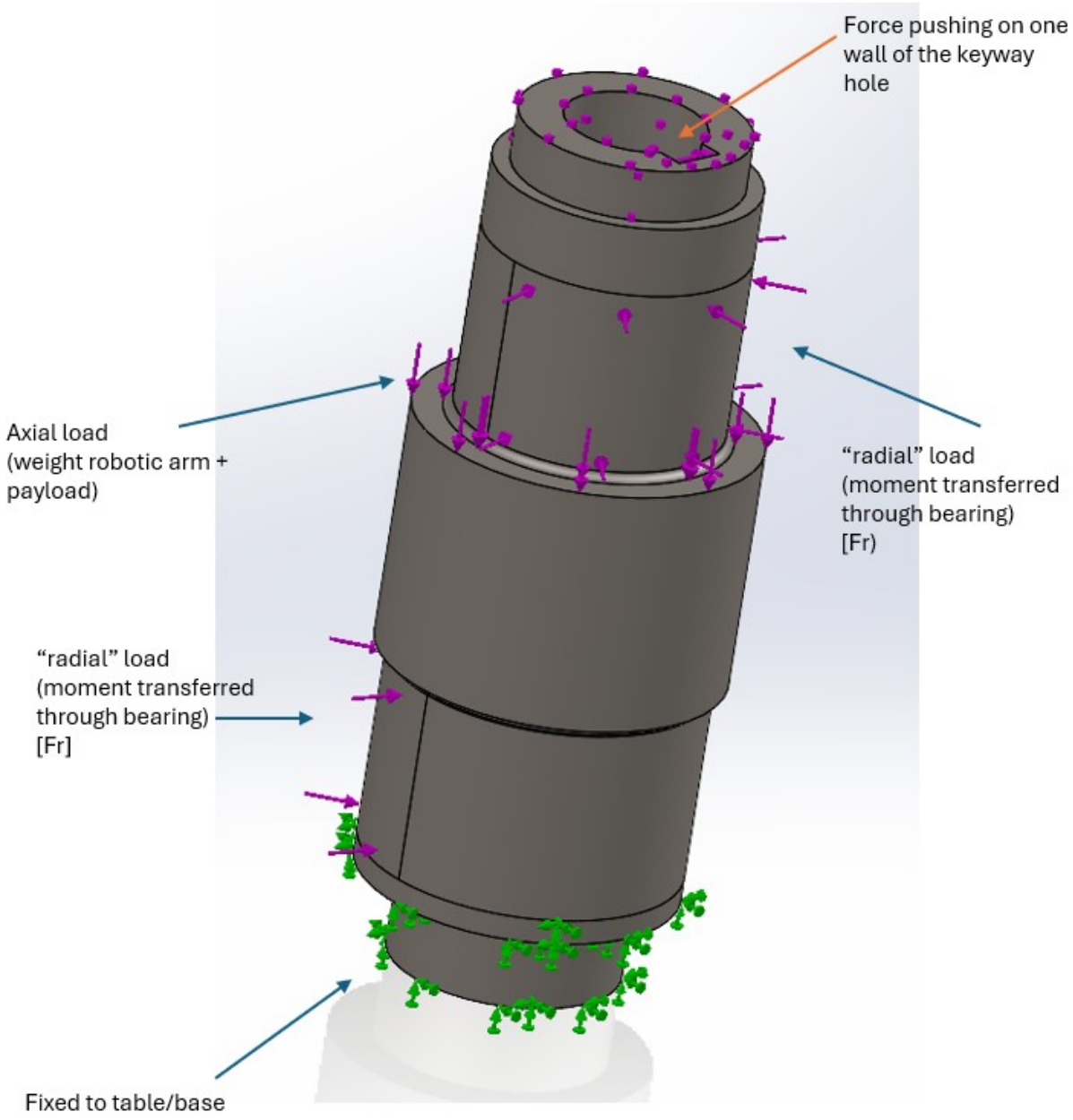


Figure D.5: Load applied in FEA

Axial load (from weight of arm + payload)

The axial load (80 N) is trivial and can be disregarded.

Radial load (from moment created by offset weight of arm + payload))

The radial load is affected by the distance between the two bearings. A longer distance between the two bearings makes the radial loads they shall transfer smaller (using values from hand calculations for the moment load). These loads are placed on 1/4 of the shaft circumference to simulate the actual force distribution from the bearing. Even when shortening the distance (down to 20 mm) to create higher loads, these do not seem to create problem areas.

Load from motor shaft

This load is set to act on one side of the key-slot wall, and is set to 2500 N (input: maximum permissable torque of the motor of 10 Nm divided by the distance of 4 mm to the shaft axis). The longer the shaft is, it creates the highest stress concentrations from the load of the rotating motor shaft at the fixed bottom part. Considerations about how the shaft shall be fixed to the table-base flange should be made. Also a shorter shaft is preferred, this will also reduce effects of bending and deflection [25, p. 377].

One such simulation can be seen as a stress plot and a FOS plot in figure D.6. This shaft has a bore diameter for the bearings of 25 mm and the shaft is 32 mm. The distance of the bearings is 22 mm and the radial load set to 900 N.

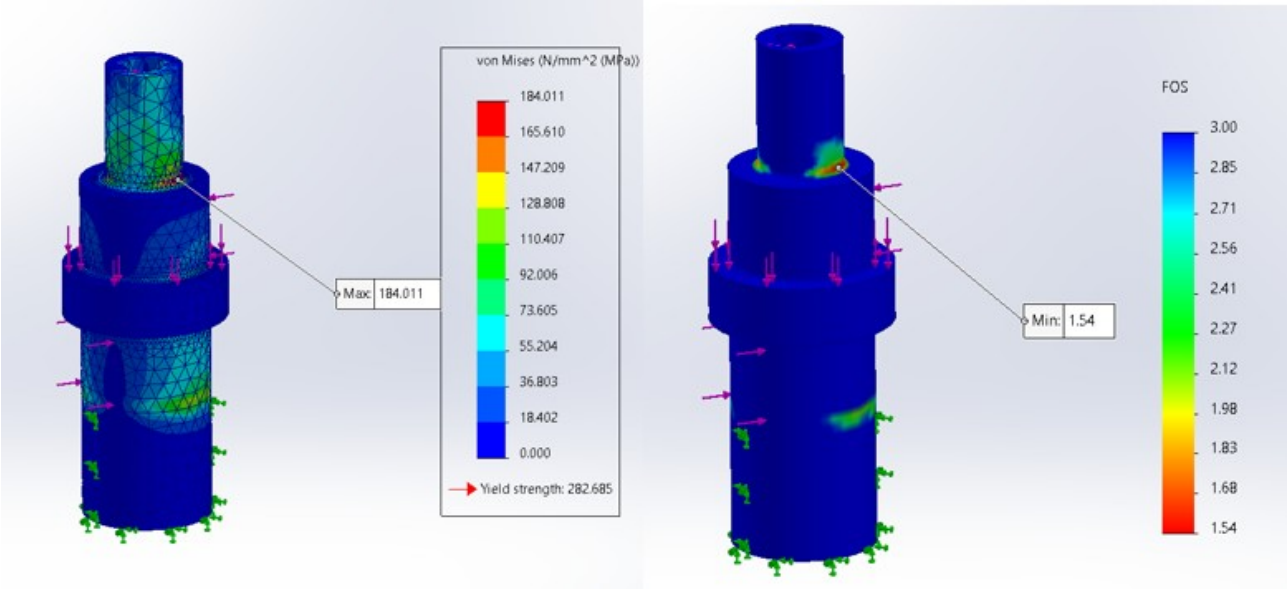


Figure D.6: stress plot for shaft

Machining

The shaft itself can be turned easily, but the hole for the shaft of the motor with it's key can hardly be machined directly (as the design of the prototype). An idea is to design the upper part of the shaft with an open slit for the keyway for the ease of machining, and place a sleeve over the shafts to reduce stress at the keyway slit (see fig. D.7). The key slit should not be directly above the shaft shoulder, since these both will have stress concentrations that can combine [25, p 405].

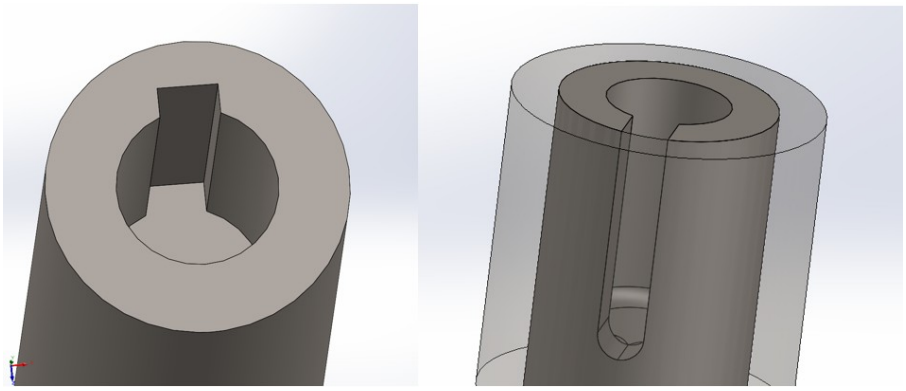


Figure D.7: The left side is hard to machine, right side with sleeve as an alternativ

Bearings and bearing arrangement

Bearings are a "wear item", meaning that they need to be exchanged after a certain time/revolutions. Choosing the right bearings for the application and ensuring proper mounting, facilitates a longer bearing life (as specified by the manufacturer).

The bearings for the base needs to take both axial and moment loads, whereof the moment loads will produce radial loads also. There are many specialized bearings for combined load situations, but they are often quite expensive. Angular contact ball bearings or tapered roller bearings are both suitable choices at a more reasonable price.

Studying some bearings (from the manufacturer SKF) with a bore diameter > 20 mm, and comparing their load ratings with the loads working on the base, shows that they will all be oversized. The axial/thrust load is so low compared to the load ratings that the bearings minimum load requirements is not fulfilled, and it would be necessary to apply a pre-load to the bearings.

Many manufacturers of bearings (like SKF) have very good online resources like calculators for bearing arrangements etc. When a bearing is chosen, they also provide information of corresponding abutment dimensions, geometrical tolerances and CAD models to integrate into an assembly.

5 Further work on arms/joints

DAB |

5.1 FEA on parts

Initial analysis to have a better insight of where the stresses are located to make correct adjustments to improve part strength and design.

5.1.1 Shaft of each joint

joint 1

tested firstly bending forces on the shaft end diameter of 20mm and middle diameter where arm bolts up is 24mm. bearing support where bearing sits on shaft dark blue points and with a force of 40.1N the purple arrows in figure D.8.with one fixture at the end where pulley sits.

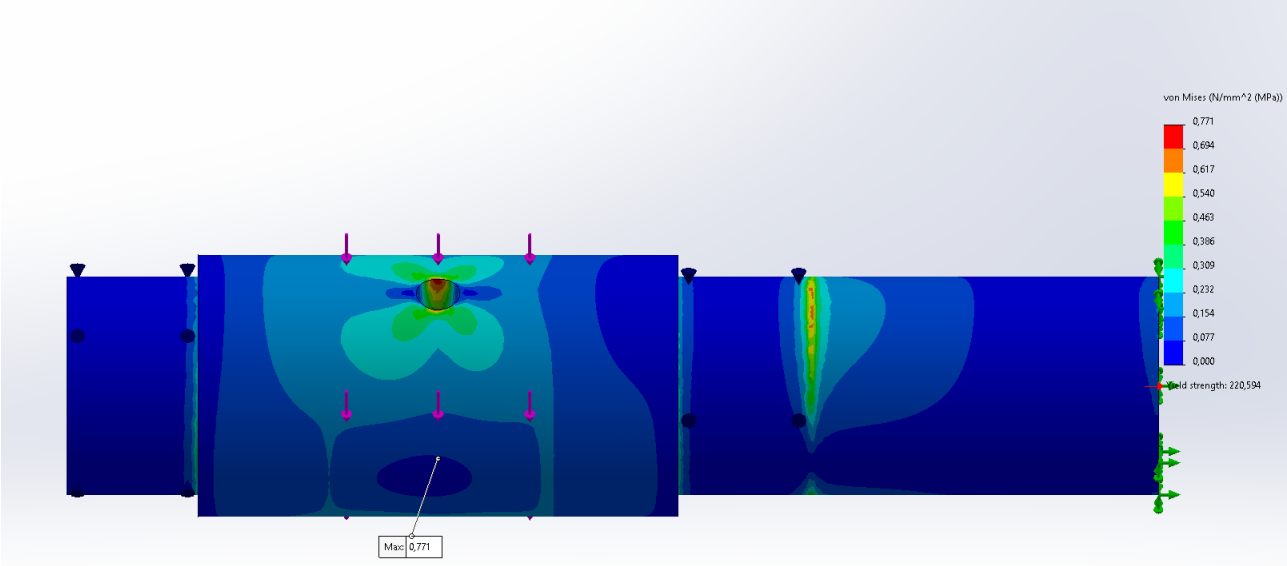
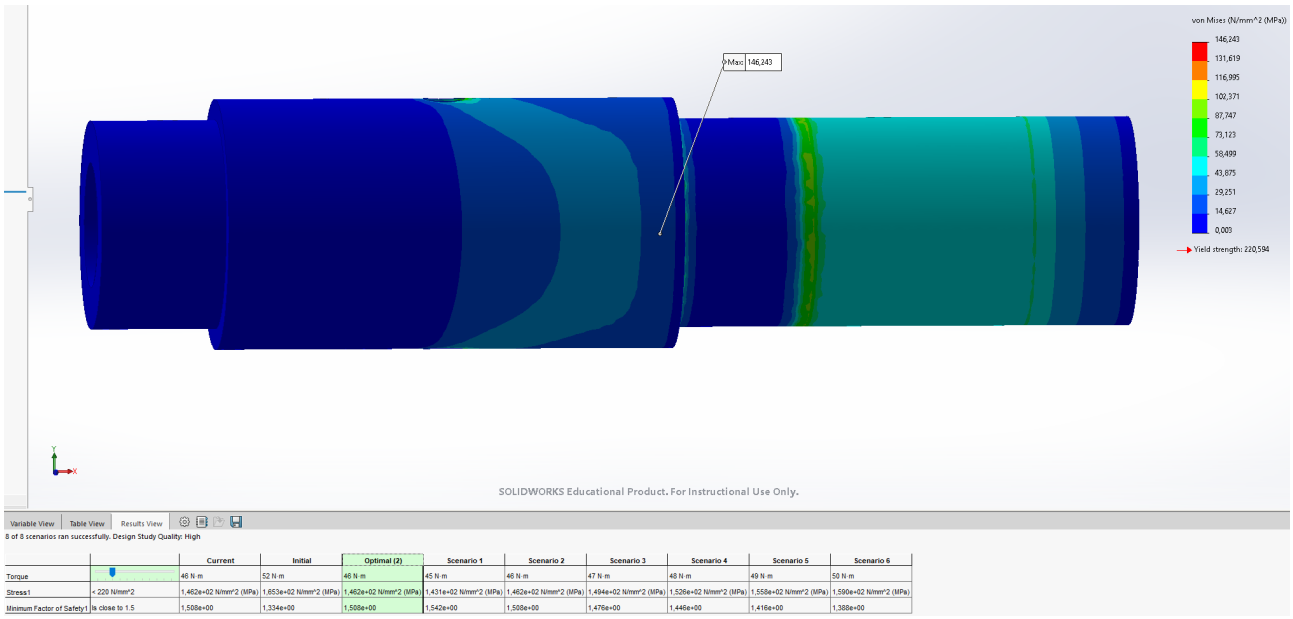


Figure D.8: shaft bending forces

In this design study see figure D.9 try to find permissible torque when factor of safety is 1.5. and we can see it is at 46Nm at n=1.5 factor of safety.



		Current	Initial	Optimal (2)	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Torque		46 N·m	52 N·m	46 N·m	45 N·m	46 N·m	47 N·m	48 N·m	49 N·m	50 N·m
Stress1	< 220 N/mm^2	1,462e+02 N/mm^2 (MPa)	1,653e+02 N/mm^2 (MPa)	1,462e+02 N/mm^2 (MPa)	1,431e+02 N/mm^2 (MPa)	1,462e+02 N/mm^2 (MPa)	1,494e+02 N/mm^2 (MPa)	1,526e+02 N/mm^2 (MPa)	1,558e+02 N/mm^2 (MPa)	1,590e+02 N/mm^2 (MPa)
Minimum Factor of Safety1	Is close to 1.5	1,508e+00	1,334e+00	1,508e+00	1,542e+00	1,508e+00	1,476e+00	1,446e+00	1,416e+00	1,388e+00

Figure D.9: shaft torque forces design study

joint 2

had problems doing design studdy, discovered that increasing mesh quality made the factor of safety show 0 but when you check in static it shows perfectly fine. becouse of this didnt have time to put in.

joint 3

applying Bending forces for joint 3 shaft that is 8mm see figure D.10. purple arrows are 19.62N force downwards. bearing support dark blue points and also fixture in the shaft pin hole.

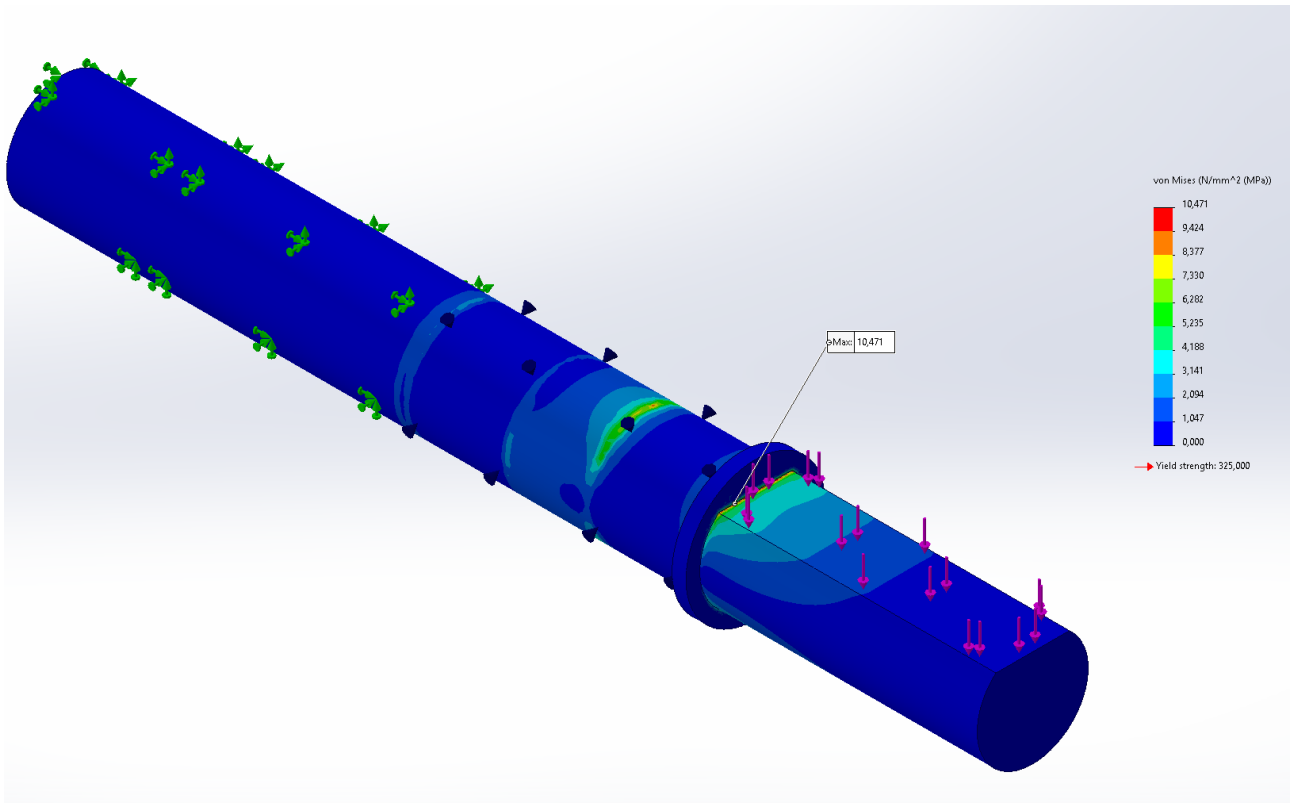
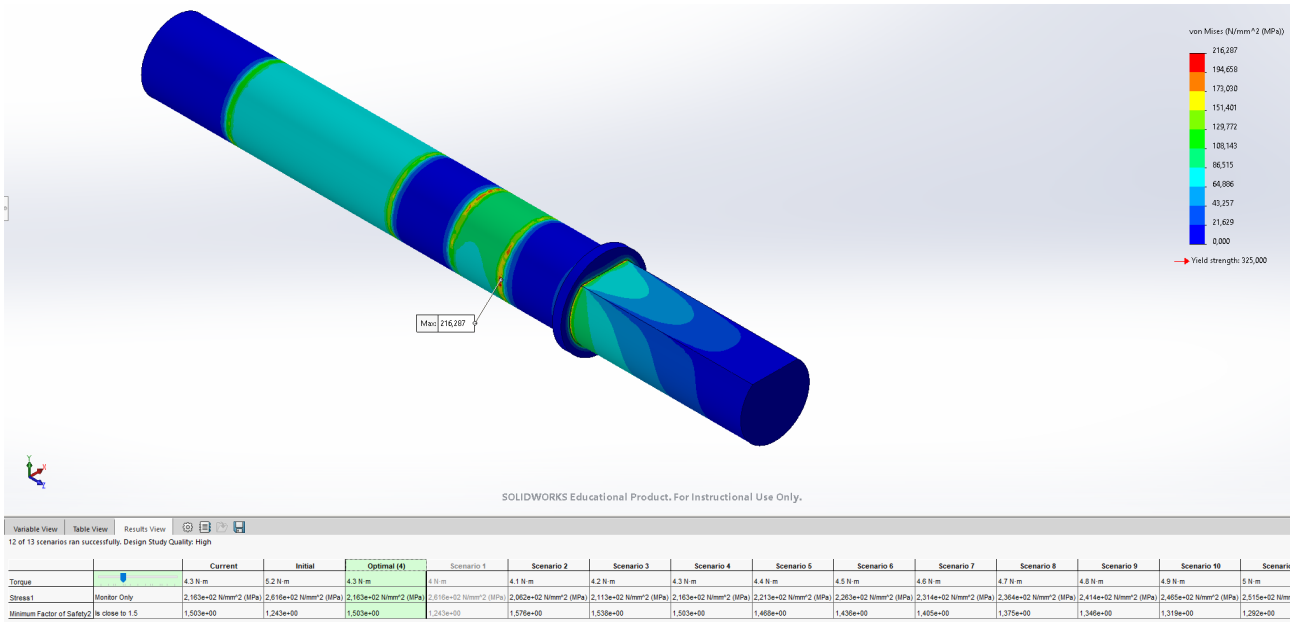


Figure D.10: shaft bending forces

design study figure D.11 shows that the shaft can endure 4.6Nm before dropping below factor of safety of n=1.5.



		Current	Initial	Optimal (4)	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6	Scenario 7	Scenario 8	Scenario 9	Scenario 10	Scenari
Torque		4.3 N·m	5.2 N·m	4.3 N·m	4 N·m	4.1 N·m	4.2 N·m	4.3 N·m	4.4 N·m	4.5 N·m	4.6 N·m	4.7 N·m	4.8 N·m	4.9 N·m	5 N·m
Stress1	Monitor Only	2,163e+02 N/mm^2 (MPa)	2,616e+02 N/mm^2 (MPa)	2,163e+02 N/mm^2 (MPa)	2,616e+02 N/mm^2 (MPa)	2,062e+02 N/mm^2 (MPa)	2,113e+02 N/mm^2 (MPa)	2,163e+02 N/mm^2 (MPa)	2,213e+02 N/mm^2 (MPa)	2,263e+02 N/mm^2 (MPa)	2,314e+02 N/mm^2 (MPa)	2,364e+02 N/mm^2 (MPa)	2,414e+02 N/mm^2 (MPa)	2,465e+02 N/mm^2 (MPa)	2,515e+02 N/m
Minimum Factor of Safety2	Is close to 1.5	1,503e+00	1,243e+00	1,503e+00	1,243e+00	1,576e+00	1,538e+00	1,503e+00	1,468e+00	1,436e+00	1,405e+00	1,375e+00	1,346e+00	1,319e+00	1,292e+00

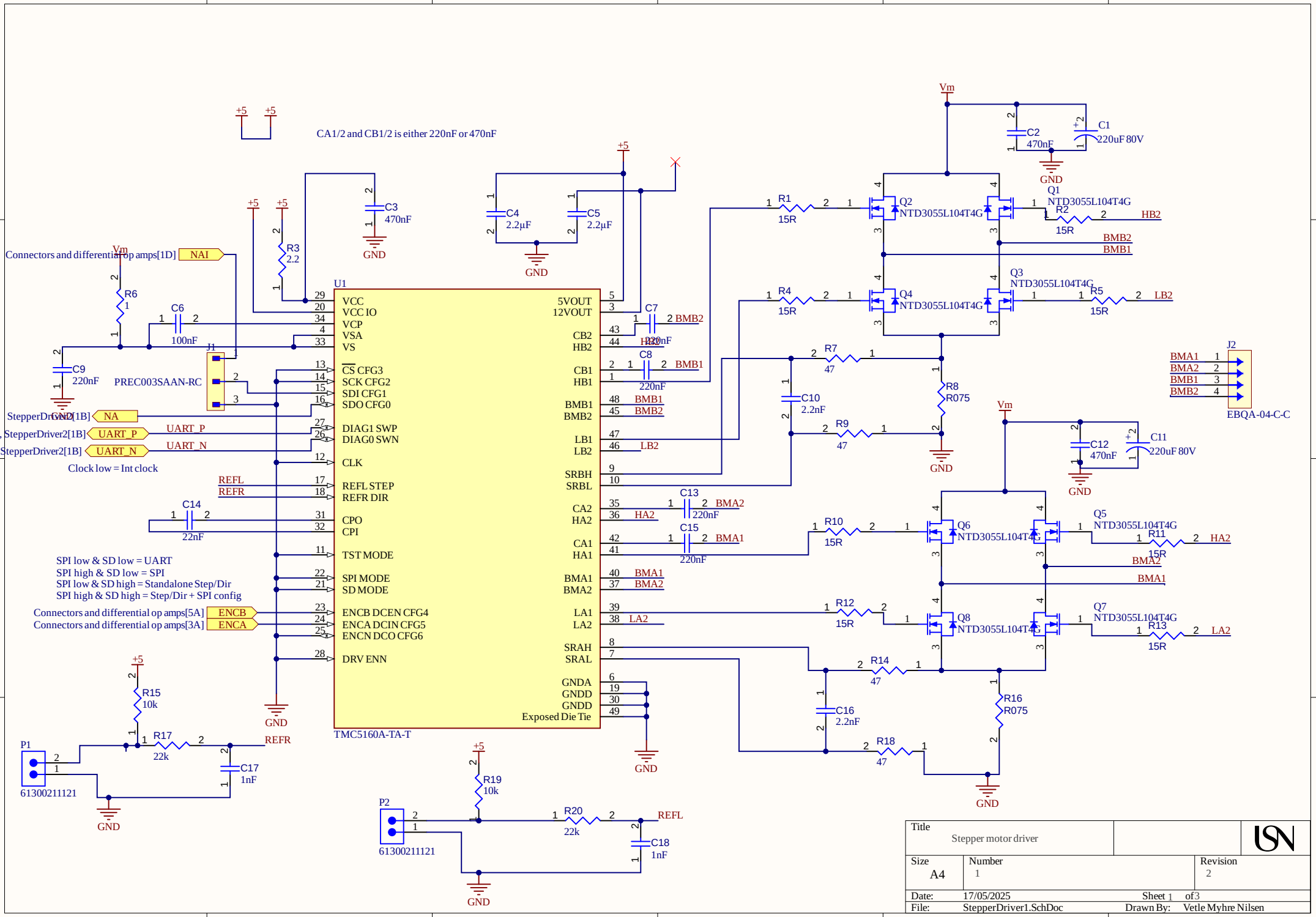
Figure D.11: shaft torque forces design study

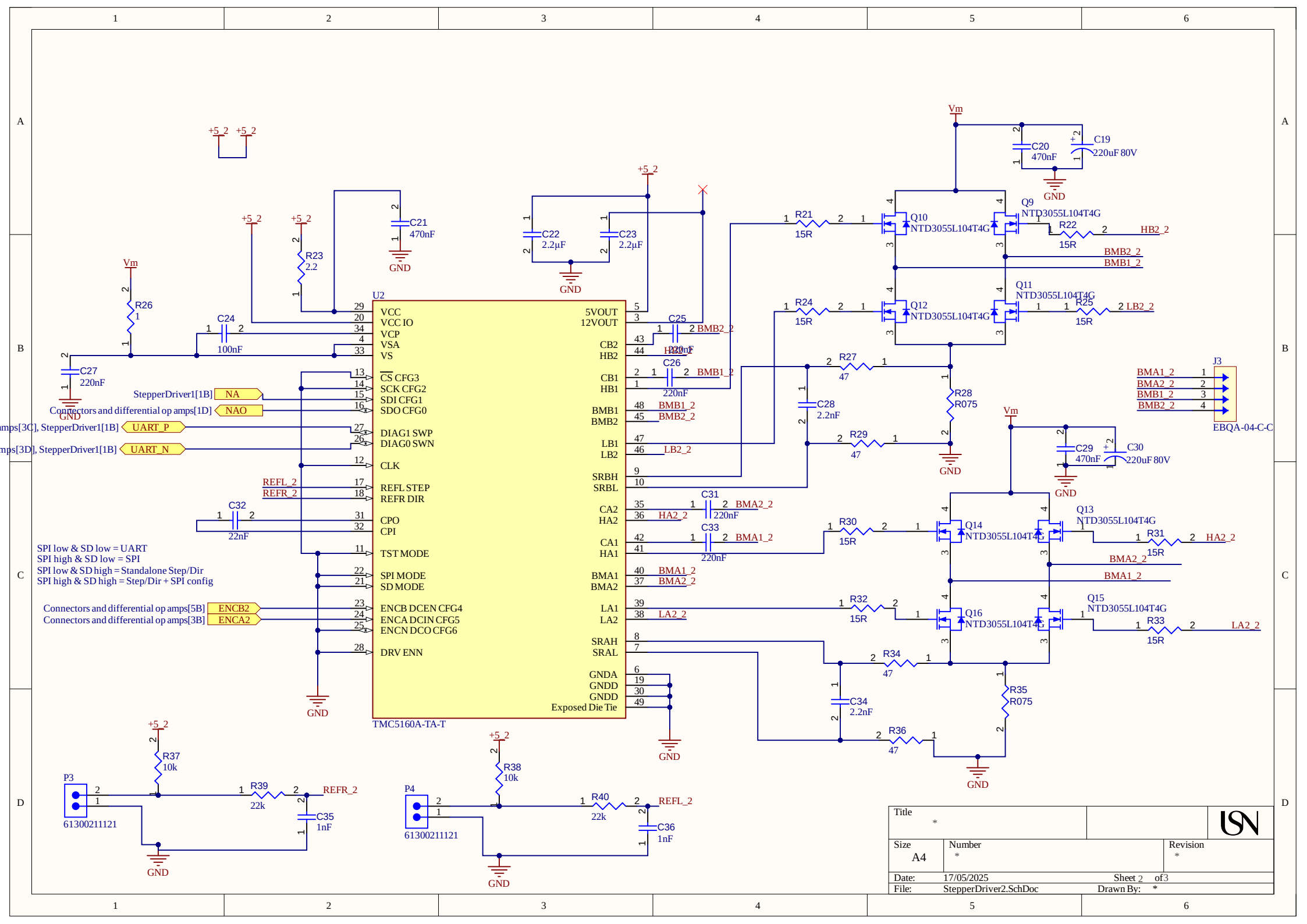
Appendix E

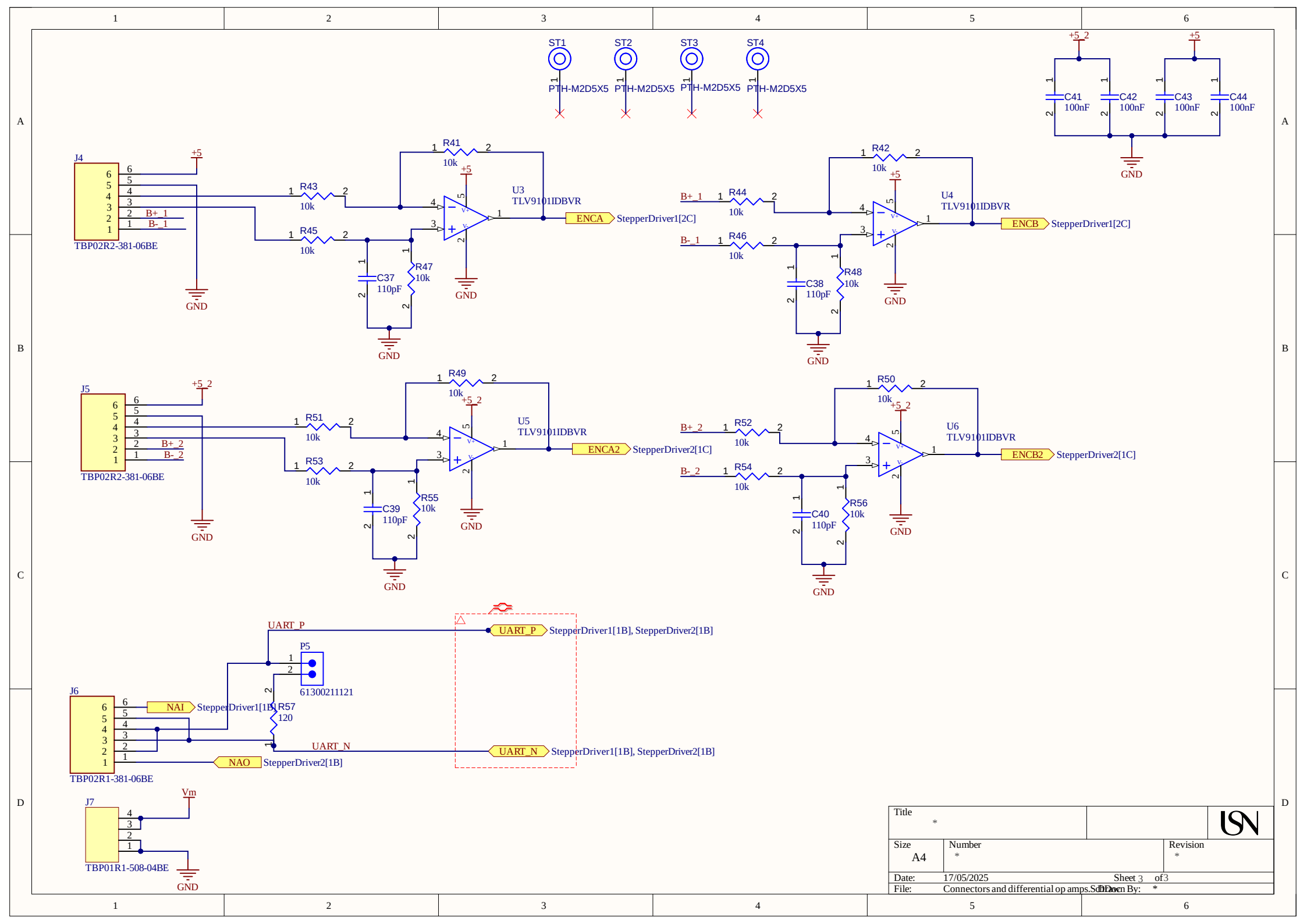
Electronics

1 Schematic

VMN | -


CA1/2 and CB1/2 is either 220nF or 470nF
U1
TMC5160A-TA-T
VCC
VCC IO
VCP
VSA
VS
CS CFG3
SCK CFG2
SDI CFG1
SDO CFG0
DIAG1 SWP
DIAG0 SWN
CLK
REFL STEP
REFR DIR
CPO
CPI
TST MODE
SPI MODE
SD MODE
ENCB DCEN CFG4
ENCA DCIN CFG5
ENCN DCO CFG6
DRV ENN
5VOUT
12VOUT
CB2
HB2
CB1
HB1
BMB1
BMB2
LB1
LB2
SRBH
SRBL
CA2
HA2
CA1
HA1
BMA1
BMA2
LA1
LA2
SRAH
SRAL
GNDA
GNDD
GNDD
Exposed Die Tie
Vm
+5
GND
R1 15R
R2 15R
R4 15R
R5 15R
R10 15R
R11 15R
R12 15R
R13 15R
R7 47
R9 47
R14 47
R18 47
R8 R075
R16 R075
C1 220uF 80V
C2 470nF
C3 470nF
C4 2.2µF
C5 2.2µF
C6 100nF
C7 220nF
C8 220nF
C9 220nF
C10 2.2nF
C11 220uF 80V
C12 470nF
C13 220nF
C15 220nF
C16 2.2nF
C14 22nF
C17 1nF
C18 1nF
R3 2.2
R6 1
R15 10k
R17 22k
R19 10k
R20 22k
Q1 NTD3055L104T4G
Q2 NTD3055L104T4G
Q3 NTD3055L104T4G
Q4 NTD3055L104T4G
Q5 NTD3055L104T4G
Q6 NTD3055L104T4G
Q7 NTD3055L104T4G
Q8 NTD3055L104T4G
J1 PREC003SAAN-RC
J2 EBQA-04-C-C
P1 61300211121
P2 61300211121
Connectors and differential op amps[1D] NAI
StepperDriver2[1B] NA
UART_P
UART_N
Clock low = Int clock
SPI low & SD low = UART
SPI high & SD low = SPI
SPI low & SD high = Standalone Step/Dir
SPI high & SD high = Step/Dir + SPI config
Connectors and differential op amps[5A] ENCB
Connectors and differential op amps[3A] ENCA
REFL
REFR
HB2
BMB2
BMB1
LB2
BMA2
BMA1
HA2
LA2
BMA1
BMA2
BMB1
BMB2
Title
Stepper motor driver
Size
A4
Number
1
Revision
2
Date: 17/05/2025
Sheet 1 of 3
File: StepperDriver1.SchDoc
Drawn By: Vetle Myhre Nilsen

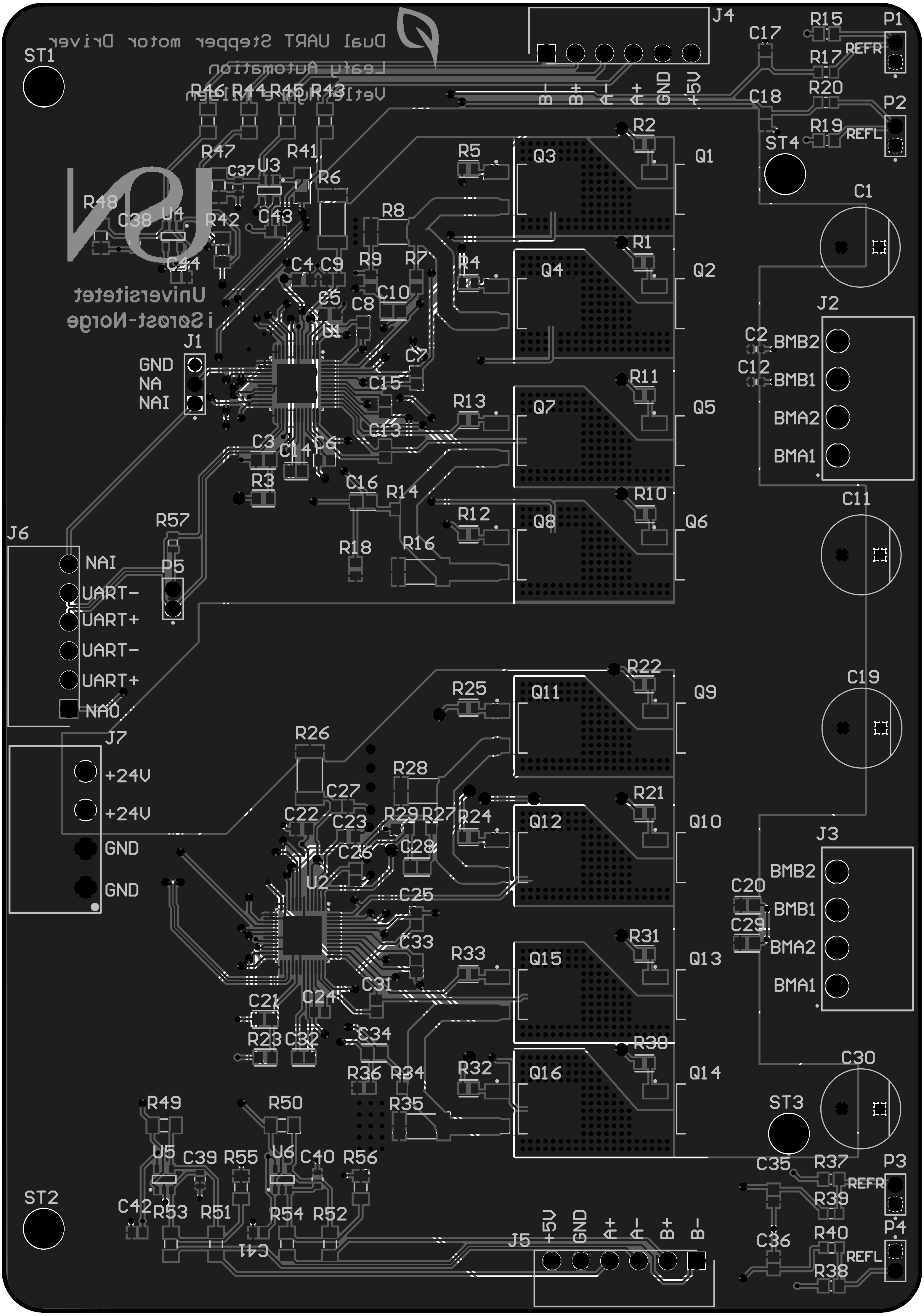

U2
TMC5160A-TA-T
VCC
VCC IO
VCP
VSA
VS
CS CFG3
SCK CFG2
SDI CFG1
SDO CFG0
DIAG1 SWP
DIAG0 SWN
CLK
REFL STEP
REFR DIR
CPO
CPI
TST MODE
SPI MODE
SD MODE
ENCB DCEN CFG4
ENCA DCIN CFG5
ENCN DCO CFG6
DRV ENN
5VOUT
12VOUT
CB2
HB2
CB1
HB1
BMB1
BMB2
LB1
LB2
SRBH
SRBL
CA2
HA2
CA1
HA1
BMA1
BMA2
LA1
LA2
SRAH
SRAL
GNDA
GNDD
GNDD
Exposed Die Tie
SPI low & SD low = UART
SPI high & SD low = SPI
SPI low & SD high = Standalone Step/Dir
SPI high & SD high = Step/Dir + SPI config
StepperDriver1[1B]
Connectors and differential op amps[1D]
Connectors and differential op amps[3C], StepperDriver1[1B]
Connectors and differential op amps[3D], StepperDriver1[1B]
Connectors and differential op amps[5B]
Connectors and differential op amps[3B]
NA
NAO
UART_P
UART_N
ENCB2
ENCA2
REFL_2
REFR_2
HA2_2
HB2_2
BMA1_2
BMA2_2
BMB1_2
BMB2_2
LA2_2
LB2_2
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16
NTD3055L104T4G
J3
EBQA-04-C-C
P3
P4
61300211121
Title
Size
A4
Number
Revision
Date:
17/05/2025
Sheet 2 of 3
File:
StepperDriver2.SchDoc
Drawn By:



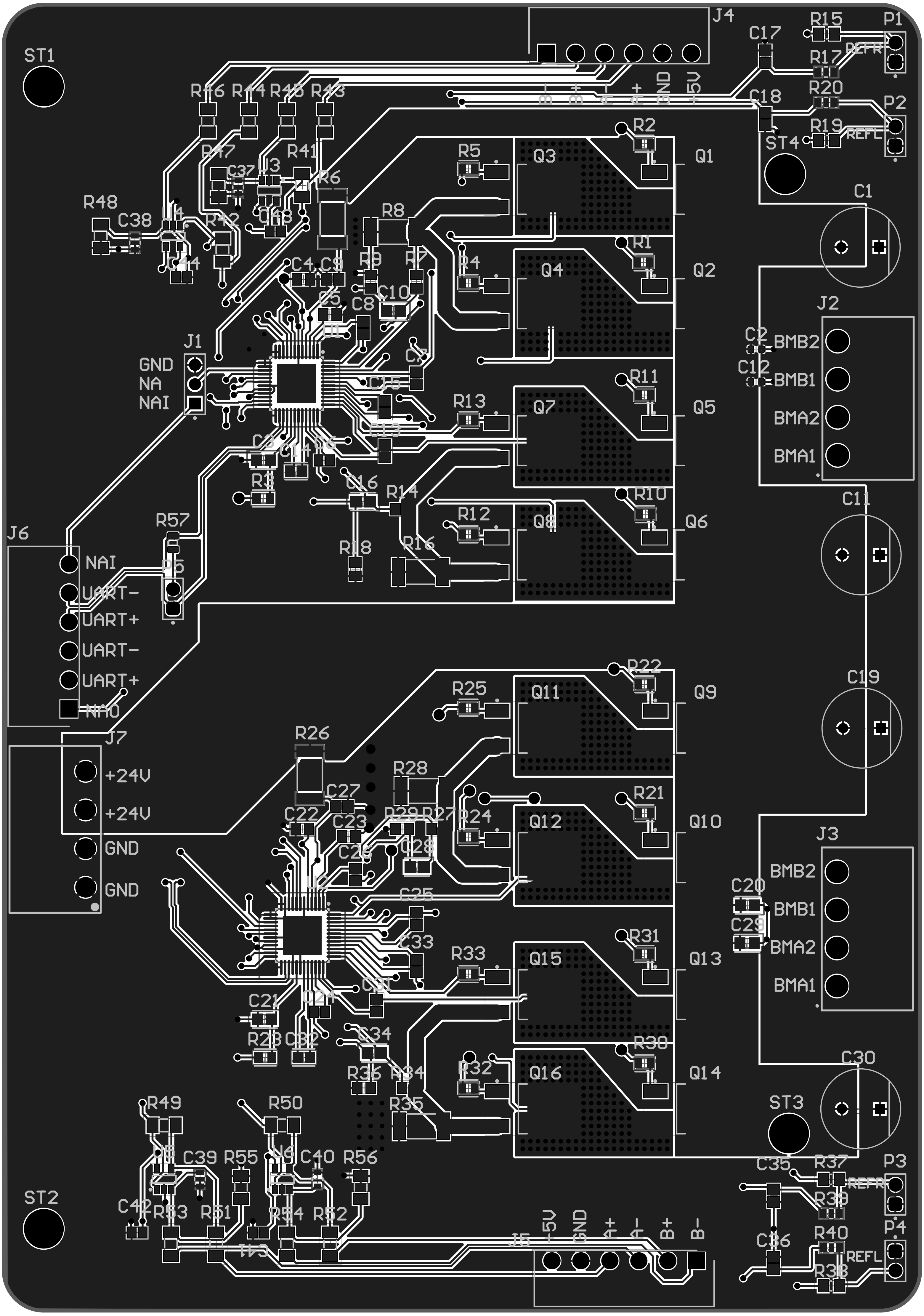
Title *		
Size A4	Number *	Revision *
Date: 17/05/2025	Sheet 3 of 3	
File: Connectors and differential op amps.SchDoc	Drawn By: *	

2 PCB layers

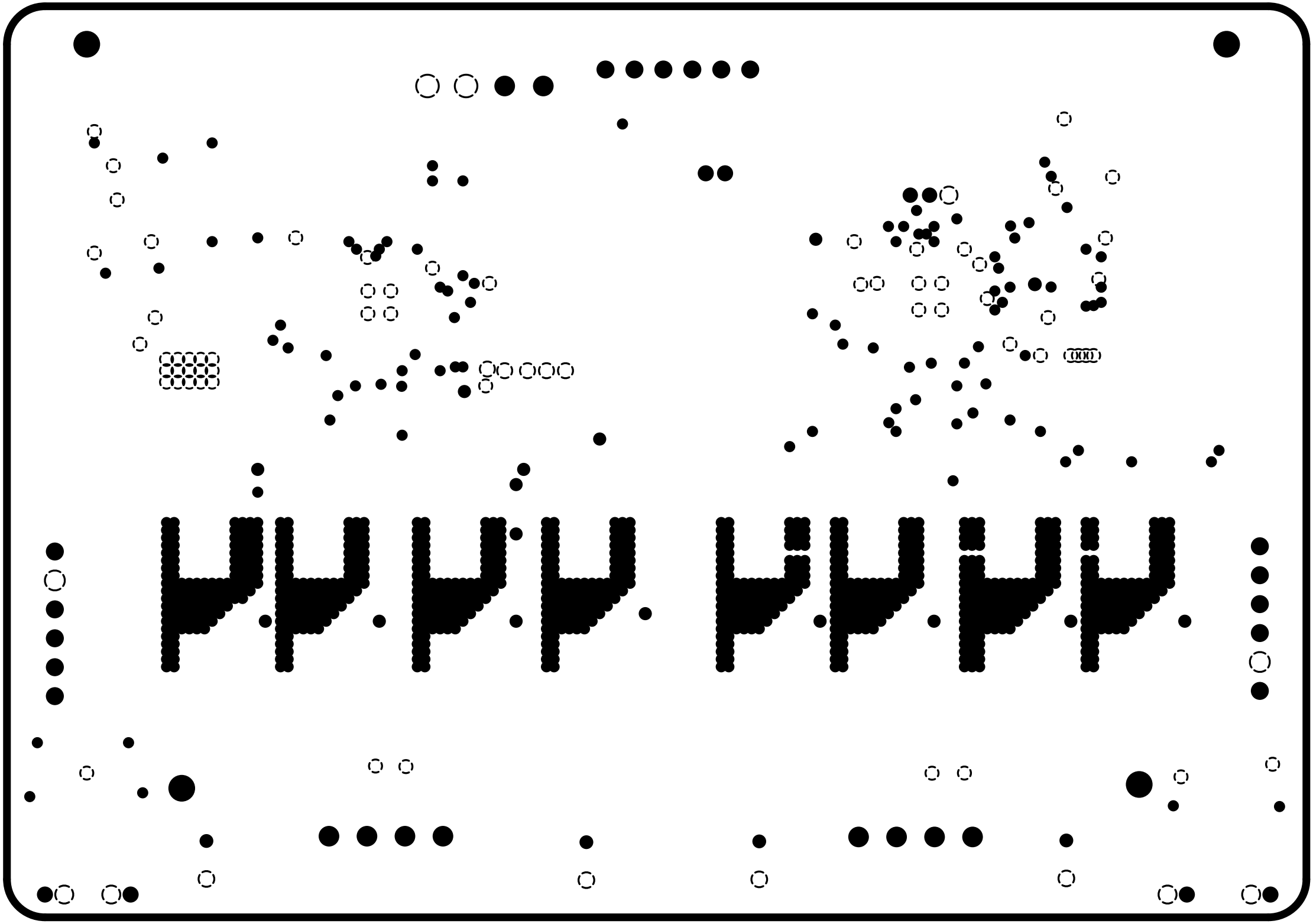
VMN | —

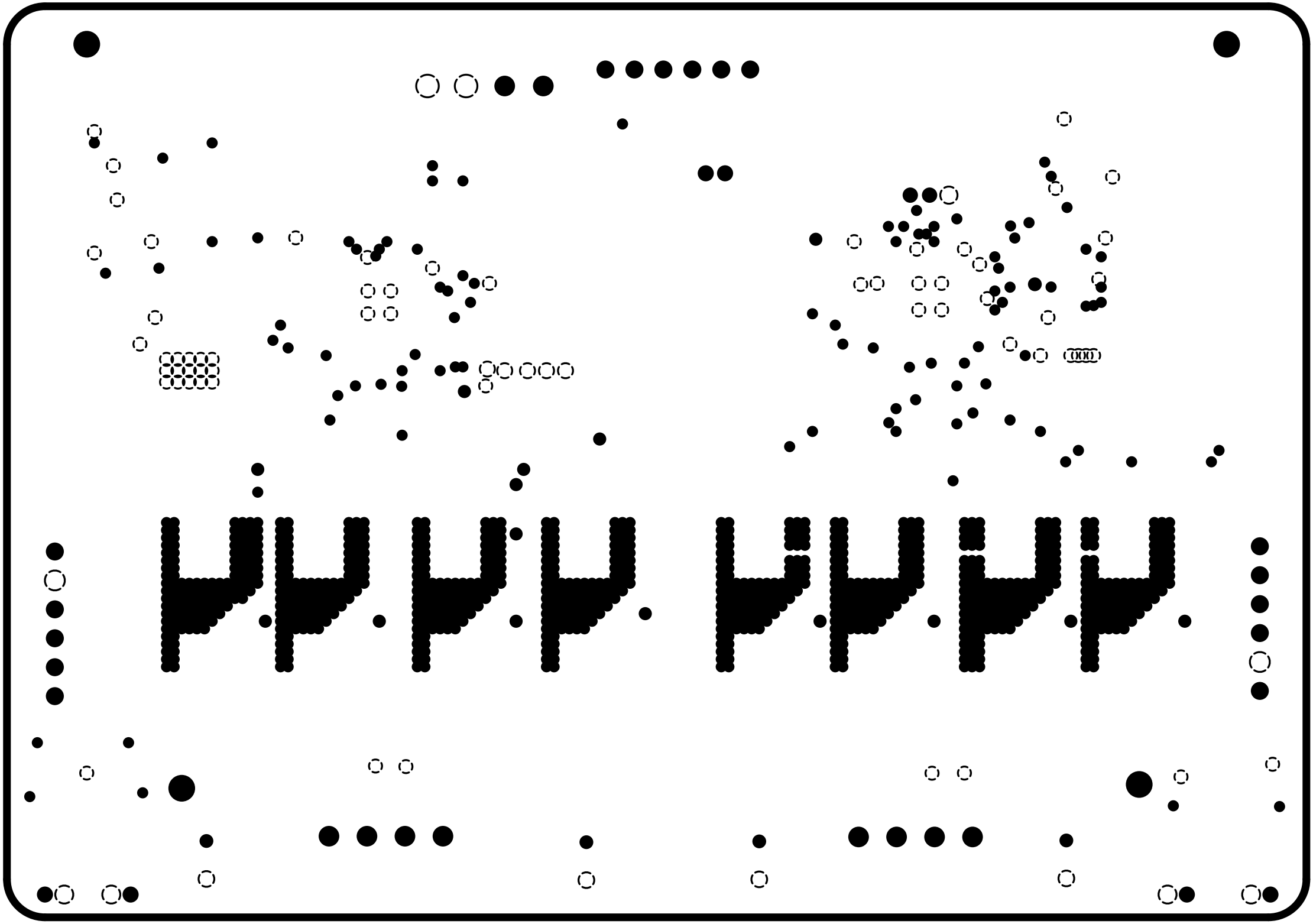


J4
B- B+ A- A+ GND +5V
ST1
ST2
ST3
ST4
J1
GND
NA
NAI
J6
NAI
UART-
UART+
UART-
UART+
NAO
J7
+24V
+24V
GND
GND
J2
BMB2
BMB1
BMA2
BMA1
J3
BMB2
BMB1
BMA2
BMA1
J5
+5V GND A+ A- B+ B-
P1
P2
P3
P4
REFR
REFL
REFR
REFL
U2
U3
U4
U5
U6



J4
-5V
GND
A+
A-
B+
B-
ST1
ST2
ST3
ST4
P1
P2
P3
P4
REFR
REFL
C1
C11
C19
C30
J1
GND
NA
NAI
J2
BMB2
BMB1
BMA2
BMA1
J3
BMB2
BMB1
BMA2
BMA1
J5
J6
NAI
UART-
UART+
UART-
UART+
NAO
J7
+24V
+24V
GND
GND
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16





Dual UART Stepper motor Driver
Leafy Automation
Vetle Myhre Nilsen

Universitetet i Sørøst-Norge

3 PCB BOM

VMN | -

Name	Description	Designator	Quantity	Manufacturer 1	Manufacturer Part Number 1	Supplier Uni	Supplier Subtotal per Board 1
220uF 80V	Cap Aluminum 220uF 80V 20% Radial Aluminum Cylindrical Can 5mm 1120mA 10000 hr 105°C Bulk	C1, C11, C19, C30	4	Panasonic	EEU-FS1K221	1.25	5
Capacitor 470nF +/-20% 25V 0603	Chip Capacitor, 470nF +/-20%, 25V, 0603, Thickness 1 mm	C2, C12	2	Murata	GRM188R71E474KA12D	0.13	0.26
GRM21BR72A474KA73L	Ceramic Capacitor, Multilayer, Ceramic, 100V, 10% +Tol, 10% -Tol, X7R, 15% TC, 0.47uF, Surface Mount, 0805	C3, C20, C21, C29	4	Murata	GRM21BR72A474KA73L	0.084	0.336
C2012X7R1C225K125AB	Ceramic Capacitor, Multilayer, Ceramic, 16V, 10% +Tol, 10% -Tol, X7R, 15% TC, 2.2uF, Surface Mount, 0805	C4, C5, C22, C23	4	TDK	C2012X7R1C225K125AB	0.17	0.68
Capacitor 100 nF +/-10% 50 V 0805	Chip Capacitor, 100 nF, +/- 10%, 50 V, 0805 (2012 Metric)	C6, C24, C41, C42, C43, C44	6	Yageo	CC0805KRX7R9BB104	0.027	0.162
C0805C224K1RACTU	Ceramic Capacitor, Multilayer, Ceramic, 100V, 10% +Tol, 10% -Tol, X7R, 15% TC, 0.22uF, Surface Mount, 0805	C7, C8, C9, C13, C15, C25, C26, C27, C31, C33	10	KEMET	C0805C224K1RACTU	0.173	1.73
C0805C222K5RACTU	Ceramic Capacitor, Multilayer, Ceramic, 50V, 10% +Tol, 10% -Tol, X7R, 15% TC, 0.0022uF, Surface Mount, 0805	C10, C16, C28, C34	4	KEMET	C0805C222K5RACTU	0.029	0.116
08051C223KAT2A	Ceramic Capacitor, Multilayer, Ceramic, 100V, 10% +Tol, 10% -Tol, X7R, 15% TC, 0.022uF, Surface Mount, 0805	C14, C32	2	Kyocera AVX	08051C223KAT2A	0.01	0.02
08055C102KAT2A	Ceramic Capacitor, Multilayer, Ceramic, 50V, 10% +Tol, 10% -Tol, X7R, 15% TC, 0.001uF, Surface Mount, 0805	C17, C18, C35, C36	4	Kyocera AVX	08055C102KAT2A	0.015	0.06
GQM2195C2E111JB12D	Ceramic Capacitor, Multilayer, Ceramic, 250V, 5% +Tol, 5% -Tol, C0G, 30ppm/Cel TC, 0.00011uF, Surface Mount, 0805	C37, C38, C39, C40	4	Murata	GQM2195C2E111JB12D		
PREC003SAAN-RC	Board Connector, 3 Contact(s), 1 Row(s), Male, Straight, 0.1 inch Pitch, Solder Terminal	J1	1	Sullins	PREC003SAAN-RC	0.31	0.31
EBQA-04-C-C	4 Pin terminal block male pins	J2, J3	2	Adam Equipment	EBQA-04-C-C	0.6	1.2
TBP02R2-381-06BE	Connector	J4, J5	2				
TBP02R1-381-06BE	2~24 Poles	J6	1				
TBP01R1-508-04BE	Strip Terminal Block	J7	1	Same Sky	TBP01R1-508-04BE	0.48	0.48
61300211121	Board Connector, 2 Contact(s), 1 Row(s), Male, Straight, 0.1 inch Pitch, Solder Terminal, Locking, Black Insulator, Plug	P1, P2, P3, P4, P5	5	Wurth Electronics	61300211121	0.12	0.6

Name	Description	Designator	Quantity	Manufacturer 1	Manufacturer Part Number 1	Supplier Uni	Supplier Subtotal per Board 1
NTD3055L104T4G	Power Field-Effect Transistor, 12A I(D), 60V, 0.104ohm, 1-Element, N-Channel, Silicon, Metal-oxide Semiconductor FET	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16	16	ON Semiconductor	NTD3055L104T4G	0.945	15.12
15R	Fixed Resistor, Metal Glaze/thick Film, 0.125W, 15ohm, 150V, 1% +/-Tol, 200ppm/Cel, Surface Mount, 0805	R1, R2, R4, R5, R10, R11, R12, R13, R21, R22, R24, R25, R30, R31, R32, R33	16	TE Connectivity	CRGCQ0805F15R	0.036	0.576
CRCW08052R20JNEAHP	Fixed Resistor, Metal Glaze/thick Film, 0.5W, 2.2ohm, 150V, 5% +/-Tol, 200ppm/Cel, Surface Mount, 0805	R3, R23	2	Vishay Dale	CRCW08052R20JNEAHP	0.013	0.026
CRCW25121R00FKEG	Fixed Resistor, Metal Glaze/thick Film, 1W, 1ohm, 500V, 1% +/-Tol, 100ppm/Cel, Surface Mount, 2512	R6, R26	2	Vishay Dale	CRCW25121R00FKEG	0.3133	0.6266
ERJ-6ENF47R0V	Fixed Resistor, Metal Glaze/thick Film, 0.125W, 47ohm, 150V, 1% +/-Tol, 100ppm/Cel, Surface Mount, 0805	R7, R9, R14, R18, R27, R29, R34, R36	8	Panasonic	ERJ-6ENF47R0V	0.052	0.416
R075	RES SHUNT, 2512, 0.075 ohm, 1%, 3W	R8, R16, R28, R35	4	Stackpole Electronics	CSNL2512FT75L0	0.51	2.04
Resistor 10k +/-1% 0805 125 mW	Chip Resistor, 10 KOhm, +/- 1%, 125 mW, -55 to 155 degC, 0805 (2012 Metric)	R15, R19, R37, R38	4	Vishay	CRCW080510K0FKEA	0.026	0.104
RT0805FRE0722KL	Fixed Resistor, Thin Film, 0.125W, 22000ohm, 150V, 1% +/-Tol, 50ppm/Cel, Surface Mount, 0805	R17, R20, R39, R40	4	Yageo	RT0805FRE0722KL	0.0129	0.0516
Resistor 10k +/-1% 1206 250 mW	Chip Resistor, 10 KOhm, +/- 1%, 0.25 W, -55 to 155 degC, 1206 (3216 Metric)	R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52, R53, R54, R55, R56	16	Yageo	RC1206FR-0710KL	0.018	0.288
CRG0805F120R	Fixed Resistor, Metal Glaze/thick Film, 0.125W, 120ohm, 150V, 1% +/-Tol, 100ppm/Cel, Surface Mount, 0805	R57	1	TE Connectivity	CRG0805F120R	0.006	0.006
PTH-M2D5X5	M2 PTH (2.5mm), 5mm pad	ST1, ST2, ST3, ST4	4				
TMC5160A-TA-T	Stepper Motor Controller, CMOS, PQFP48	U1, U2	2	Trinamic	TMC5160A-TA-T	8.28	16.56
TLV9101IDBVR	General Purpose Opertational Amplifier 1 Circuit Rail-to-Rail SOT-23-5	U3, U4, U5, U6	4	Texas Instruments	TLV9101IDBVR	0.61	2.44

Appendix F

Code Documentation

This chapter contains documentation which was generated from the doc comment in the source code. The application used for this task is called Doxygen.

1 Leafy Automation Central

Leafy Automation Central

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 access_levels Namespace Reference

Classes

- class AccessLevel

5.2 api Namespace Reference

Functions

- home ()
- status ()
- stats_route ()
- log_route ()
- get_camera_feed ()
- visual_get_mask ()
- visual_get_geometry ()
- visual_get_depth ()
- capture_image_route ()
- classify_image_route ()

Variables

- routes = Blueprint("api_v1", __name__, url_prefix="/api/v1")
- dict stats

5.2.1 Function Documentation

5.2.1.1 capture_image_route()

```
api.capture_image_route ()
```

```
00094 def capture_image_route():
00095     if not request.data:
00096         return jsonify({"msg": "No image data provided"}), 400
00097 
00098     with open("./central/cache/camera01.jpg", "wb") as f:
00099         f.write(request.data)
00100 
00101     return jsonify({"msg": "Image saved"})
00102 
00103 @routes.route("/classify-image", methods=["POST"])
```

5.2.1.2 classify_image_route()

api.classify_image_route ()

```
00104 def classify_image_route():
00105     image = request.data # Raw image data (binary)
00106     #masks = generate_mask(image)
00107
00108     res = {
00109         "class": classify_image(image),
00110         #"mask": masks
00111     }
00112
00113     return jsonify(res)
```

5.2.1.3 get_camera_feed()

api.get_camera_feed ()

```
00055 def get_camera_feed():
00056     return visual_get_geometry()
00057
```

5.2.1.4 home()

api.home ()

```
00022 def home():
00023     return jsonify({"msg": "Hello from Leafy Automation Central!"})
00024
00025 @routes.route("/status", methods=["GET"])
```

5.2.1.5 log_route()

api.log_route ()

```
00049 def log_route():
00050     log = Log(request.args.get("msg"))
00051     log.save()
00052
00053     return jsonify({})
00054
```

5.2.1.6 stats_route()

api.stats_route ()

```
00042 def stats_route():
00043     stats["image-capture-time"] = request.args.get("image-capture-time")
00044     stats["image-capture-req-time"] = request.args.get("image-capture-req-time")
00045
00046     return jsonify({"msg": "Data captured"})
00047
00048 @routes.route("/log", methods=["GET"])
```

5.2.1.7 status()

api.status ()

```
00026 def status():
00027     image = Image.open("./central/cache/camera01.jpg")
00028     image_classification = classify_image(image)
00029
00030     json = {
00031         "status": "Online",
00032         "img-capture-time": stats["image-capture-time"],
00033         "img-capture-req-time": stats["image-capture-req-time"],
00034         "image-classification": max(image_classification, key=lambda x: x["score"])["label"],
00035         "green-percentage": estimate_green_percentage(cv2.imread("./central/cache/camera01.jpg")),
00036         "log": Log.load()
00037     }
00038
00039     return jsonify(json)
00040
00041 @routes.route("/log-stats", methods=["GET"])
```

5.2.1.8 visual_get_depth()

```
api.visual_get_depth ()
```

```
00082 def visual_get_depth():
00083     image = Image.open("./central/cache/camera01.jpg")
00084
00085     depth = estimate_depth(image)
00086
00087     buffer = io.BytesIO()
00088     depth["depth"].save(buffer, format="PNG")
00089     buffer.seek(0)
00090
00091     return base64.b64encode(buffer.getvalue()).decode("utf-8")
00092
00093 @routes.route("/capture-image", methods=["POST"])
```

5.2.1.9 visual_get_geometry()

```
api.visual_get_geometry ()
```

```
00070 def visual_get_geometry():
00071     image = cv2.imread("./central/cache/camera01.jpg")
00072
00073     plant_manager = PlantManager(image)
00074     bounding_boxes, contours, centroids = plant_manager.detect_plants()
00075     plant_manager.draw_geometry(bounding_boxes, contours, centroids)
00076     image_raw = plant_manager.get_binary_image()
00077
00078     base64_image = base64.b64encode(image_raw.getbuffer()).decode("utf-8")
00079
00080     return base64_image
00081
```

5.2.1.10 visual_get_mask()

```
api.visual_get_mask ()
```

```
00058 def visual_get_mask():
00059     image = cv2.imread("./central/cache/camera01.jpg")
00060
00061     mask = generate_mask(image)
00062
00063     _, img_encoded = cv2.imencode(".jpg", mask)
00064     img_jpg = io.BytesIO(img_encoded.tobytes())
00065
00066     base64_image = base64.b64encode(img_jpg.getbuffer()).decode("utf-8")
00067
00068     return base64_image
00069
```

5.2.2 Variable Documentation

5.2.2.1 routes

```
api.routes = Blueprint("api_v1", __name__, url_prefix="/api/v1")
```

5.2.2.2 stats

```
dict api.stats
```

Initial value:

```
00001 =  {
00002     "image-capture-time": 0,
00003     "image-capture-req-time": 0
00004 }
```

5.3 benchmark Namespace Reference

Classes

- class Benchmark

5.4 camera_feed Namespace Reference

5.5 chessboard Namespace Reference

Variables

- img = cv2.imread("./central/cache/camera01.jpg")
- gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- tuple pattern_size = (5, 8)
- ret
- corners

5.5.1 Variable Documentation

5.5.1.1 corners

```
chessboard.corners
```

5.5.1.2 gray

```
chessboard.gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

5.5.1.3 img

```
chessboard.img = cv2.imread("./central/cache/camera01.jpg")
```

5.5.1.4 pattern_size

```
tuple chessboard.pattern_size = (5, 8)
```

5.5.1.5 ret

```
chessboard.ret
```

5.6 db Namespace Reference

Classes

- class DB

  Class for managing SQLite database connection.

5.7 depth_estimation Namespace Reference

Functions

- estimate_depth (image, fast=True)

Variables

- estimator_fast = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Metric-Indoor-Small-hf")
- estimator_slow = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf")

5.7.1 Function Documentation

5.7.1.1 estimate_depth()

```
depth_estimation.estimate_depth (
              image,
              fast = True)
```

```
Estimate the depth of an image using a depth estimation model.
@param image: PIL image.
@param fast: If True, use a faster model with lower accuracy.
@return: Depth estimation result.
```

```
00006 def estimate_depth(image, fast=True):
00007     """
00008     Estimate the depth of an image using a depth estimation model.
00009     @param image: PIL image.
00010     @param fast: If True, use a faster model with lower accuracy.
00011     @return: Depth estimation result.
00012     """
00013
00014     if fast:
00015         return estimator_fast(image)
00016     else:
00017         return estimator_slow(image)
```

5.7.2 Variable Documentation

5.7.2.1 estimator_fast

```
depth_estimation.estimator_fast = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Me
```

5.7.2.2 estimator_slow

```
depth_estimation.estimator_slow = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Me
```

5.8 green_percentage Namespace Reference

Functions

- estimate_green_percentage (image)

5.8.1 Function Documentation

5.8.1.1 estimate_green_percentage()

```
green_percentage.estimate_green_percentage (
              image)
```

```
Estimate the percentage of green pixels in an image.
@param image: OpenCV image.
@return: Percentage of green pixels in the image.
```

```
00004 def estimate_green_percentage(image):
00005     """
00006     Estimate the percentage of green pixels in an image.
00007     @param image: OpenCV image.
00008     @return: Percentage of green pixels in the image.
00009     """
00010
00011     image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
00012
00013     lower_green = np.array([35, 40, 40])
00014     upper_green = np.array([85, 255, 255])
00015
00016     mask = cv2.inRange(image_hsv, lower_green, upper_green)
00017
00018     green_pixels = np.count_nonzero(mask)
00019     total_pixels = image.shape[0] * image.shape[1]
00020     green_percentage = (green_pixels / total_pixels) * 100
00021
00022     return round(green_percentage, 2)
```

5.9 grip_point Namespace Reference

Functions

- get_grip_point (image)

5.9.1 Function Documentation

5.9.1.1 get_grip_point()

```
grip_point.get_grip_point (
              image)
```

This function uses AI models and CV techniques to calculate the grip point of the point of interest, which is a lettuce in this case.
@param image: PIL image object
@return: grip point vector (x, y, z)

```
00007 def get_grip_point(image):
00008     """
00009     This function uses AI models and CV techniques to calculate the grip point of the point of
      interest,
00010     which is a lettuce in this case.
00011     @param image: PIL image object
00012     @return: grip point vector (x, y, z)
00013     """
00014
00015     image_opencv = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
00016
00017     depth = estimate_depth(image, fast=False)
00018     mask = generate_mask(image_opencv)
00019     bounding_boxes, contours, centroids = generate_geometry_from_mask(image_opencv, mask)
00020
00021     centroid = centroids[0]
00022
00023     x = centroid[0]
00024     y = centroid[1]
00025     z = depth["predicted_depth"][y][x].item()
00026
00027     width, height = image.size
00028     cx = width / 2
00029     cy = height / 2
00030     fx = fy = 500 # Estimate of focal length.
00031
00032     X = (x - cx) * z / fx
00033     Y = (y - cy) * z / fy
00034     Z = z
00035
00036     lettuce_coord_cam_perspective = np.array([X, Y, Z])
00037     cam_vec = np.array([0, 0.5, 0.2]) # Camera position from the robot base center. Must be adjusted
      and calibrated.
00038
00039     return lettuce_coord_cam_perspective - cam_vec
```

5.10 hmi Namespace Reference

Functions

- dashboard ()
- login ()
- logout ()

Variables

- routes = Blueprint("hmi", __name__)

5.10.1 Function Documentation

5.10.1.1 dashboard()

hmi.dashboard ()

```
00007 def dashboard():
00008     if not "user_id" in session:
00009         return redirect("/login")
00010
00011     user = User.get_user(session["user_id"])
00012
00013     modules = [
00014         {
00015             "name": "Leafy Automation Core",
00016             "description": "The core of Leafy Automation (Arduino) controls motors.",
00017             "status": "Online"
00018         },
00019         {
00020             "name": "Camera Module #1",
00021             "description": "The camera module (esp32-cam).",
00022             "status": "Online"
00023         }
00024     ]
00025
00026     return render_template("index.html", modules=modules, user=user)
00027
00028 @routes.route("/login", methods=["GET", "POST"])
```

5.10.1.2 login()

hmi.login ()

```
00029 def login():
00030     if request.method == "POST":
00031         username: str = request.form.get("username")
00032         password: str = request.form.get("password")
00033
00034         if User.auth(username, password):
00035             user1 = User.get_user(username)
00036             session["user_id"] = user1.username
00037             session["username"] = user1.username
00038             session["email"] = user1.email
00039             session["first_name"] = user1.first_name
00040             session["last_name"] = user1.last_name
00041             session["access_level_id"] = user1.access_level_id
00042
00043             return redirect("/")
00044         else:
00045             return redirect("/login?error=login_failed")
00046     elif request.method == "GET":
00047         return render_template("login.html")
00048
00049 @routes.route("/logout", methods=["GET"])
```

5.10.1.3 logout()

hmi.logout ()

```
00050 def logout():
00051     session.clear()
00052     return redirect("/login")
```

5.10.2 Variable Documentation

5.10.2.1 routes

```
hmi.routes = Blueprint("hmi", __name__)
```

5.11 image_analysis Namespace Reference

Classes

- class ImageAnalysis

5.12 image_classification Namespace Reference

Functions

- classify_image (image)

Variables

- classifier = pipeline("zero-shot-image-classification", model="openai/clip-vit-base-patch32")

5.12.1 Function Documentation

5.12.1.1 classify_image()

```
image_classification.classify_image (
              image)
```

```
Classify an image using a pretrained model.
@return: The class of the image.
```

```
00005 def classify_image(image):
00006     """
00007     Classify an image using a pretrained model.
00008     @return: The class of the image.
00009     """
00010
00011     return classifier(image=image, candidate_labels=["crispy_lettuce", "arugula"])
```

5.12.2 Variable Documentation

5.12.2.1 classifier

```
image_classification.classifier = pipeline("zero-shot-image-classification", model="openai/clip-vit-base-patch
```

5.13 log Namespace Reference

Classes

- class Log

5.14 main Namespace Reference

Functions

- get_frame ()

Variables

- app = Flask(__name__)
- socketio = SocketIO(app, cors_allowed_origins="*")
- debug
- True
- host
- model = YOLO("yolo11n.pt")
- results = model.train(data="training/datasets/leafy-automation/data.yaml", epochs=100, imgsz=640)
- metrics = model.val()
- path = model.export(format="onnx")

5.14.1 Function Documentation

5.14.1.1 get_frame()

main.get_frame ()

```
00046 def get_frame():
00047     #benchmark = Benchmark("Object Detection Benchmark facebook detr-resnet-50", 100)
00048
00049     while True:
00050         #benchmark.start_lap()
00051         image_data = api.get_camera_feed()
00052         #benchmark.end_lap()
00053
00054         socketio.emit("camera_frame", image_data)
00055         socketio.sleep(0.1)
00056
```

5.14.2 Variable Documentation

5.14.2.1 app

main.app = Flask(__name__)

5.14.2.2 debug

main.debug

5.14.2.3 host

main.host

5.14.2.4 metrics

```
main.metrics = model.val()
```

5.14.2.5 model

```
main.model = YOLO("yolo11n.pt")
```

5.14.2.6 path

```
main.path = model.export(format="onnx")
```

5.14.2.7 results

```
main.results = model.train(data="training/datasets/leafy-automation/data.yaml", epochs=100,
imgsz=640)
```

5.14.2.8 socketio

```
main.socketio = SocketIO(app, cors_allowed_origins="*")
```

5.14.2.9 True

```
main.True
```

5.15 mask_generation Namespace Reference

Functions

- generate_mask (image)
- generate_geometry_from_mask (image, mask)

5.15.1 Function Documentation

5.15.1.1 generate_geometry_from_mask()

```
mask_generation.generate_geometry_from_mask (
              image,
              mask)
```

Generate geometry from a mask (bounding boxes, centroids and contours).
@param image: OpenCV image.

```
00023 def generate_geometry_from_mask(image, mask):
00024     """
00025     Generate geometry from a mask (bounding boxes, centroids and contours).
00026     @param image: OpenCV image.
00027     """
00028
00029     labels = pcv.watershed_segmentation(image, mask, 50)
00030
00031     bounding_boxes = []
00032     centroids = []
00033     all_contours = []
00034
00035     for label in np.unique(labels):
00036         # Skip background
00037         if label == 0:
00038             continue
00039
00040         object_mask = np.equal(labels, label).astype(np.uint8)
00041
00042         contours, _ = cv2.findContours(object_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
00043
00044         for contour in contours:
00045             x, y, w, h = cv2.boundingRect(contour)
00046             bounding_boxes.append((x, y, x + w, y + h))
00047
00048             moments = cv2.moments(contour)
00049
00050             if moments["m00"] != 0:
00051                 centroids.append(
00052                     (int(moments["m10"] / moments["m00"]), int(moments["m01"] / moments["m00"]))
00053                 )
00054
00055         all_contours.append(contour)
00056
00057     return (bounding_boxes, all_contours, centroids)
```

5.15.1.2 generate_mask()

```
mask_generation.generate_mask (
              image)
```

Generate a mask for an image.
@param image: OpenCV image.
@return Image mask.

```
00005 def generate_mask(image):
00006     """
00007     Generate a mask for an image.
00008     @param image: OpenCV image.
00009     @return Image mask.
00010     """
00011
00012     image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
00013     lower_green = np.array([35, 40, 40])
00014     upper_green = np.array([85, 255, 255])
00015
00016     mask = cv2.inRange(image_hsv, lower_green, upper_green)
00017
00018     img_fill = pcv.fill(bin_img=mask, size=100)
00019     img_fill_holes = pcv.fill_holes(img_fill)
00020
00021     return img_fill_holes
00022
```

5.16 misc Namespace Reference

Functions

- get_device ()
- img_base64 (image)

5.16.1 Function Documentation

5.16.1.1 get_device()

```
misc.get_device ()
```

Get the device to be used for AI models.

```
00005 def get_device():
00006     """
00007     Get the device to be used for AI models.
00008     """
00009 
00010     if torch.cuda.is_available():
00011         return "cuda"
00012     elif torch.backends.mps.is_available():
00013         return "mps"
00014     else:
00015         return "cpu"
00016 
```

5.16.1.2 img_base64()

```
misc.img_base64 (
               image)
```

Convert an image to base64 format.
@param image: OpenCV image.
@return: Base64 encoded string of the image.

```
00017 def img_base64(image):
00018     """
00019     Convert an image to base64 format.
00020     @param image: OpenCV image.
00021     @return: Base64 encoded string of the image.
00022     """
00023 
00024     _, img_encoded = cv2.imencode(".jpg", image)
00025     return base64.b64encode(img_encoded).decode("utf-8")
```

5.17 object_detection Namespace Reference

Functions

- object_detection (image)

Variables

- model = YOLO("leafy-ai-obj-detection.pt")

5.17.1 Function Documentation

5.17.1.1 object_detection()

```
object_detection.object_detection (
              image)
```

Performs object detection on an image using a custom-trained YOLOv11 model.
@param image: PIL image.

```
00005 def object_detection(image):
00006     """
00007     Performs object detection on an image using a custom-trained YOLOv11 model.
00008     @param image: PIL image.
00009     """
00010
00011     return model(image)
```

5.17.2 Variable Documentation

5.17.2.1 model

```
object_detection.model = YOLO("leafy-ai-obj-detection.pt")
```

5.18 plant_manager Namespace Reference

Classes

- class PlantManager

5.19 routes Namespace Reference

Functions

- home ()
- classify_image_route ()

Variables

- routes = Blueprint("routes", __name__)

5.19.1 Function Documentation

5.19.1.1 classify_image_route()

```
routes.classify_image_route ()
```

```
00010 def classify_image_route():
00011     from ai.classify_image import classify_image
00012
00013     image = request.data # Raw image data (binary)
00014
00015     res = {
00016         "class": classify_image(image)
00017     }
00018
00019     return jsonify(res)
```

5.19.1.2 home()

routes.home ()

```
00006 def home():
00007     return jsonify({"msg": "Hello from Leafy Automation Central!"})
00008
00009 @routes.route("/classify-image", methods=["POST"])
```

5.19.2 Variable Documentation

5.19.2.1 routes

routes.routes = Blueprint("routes", __name__)

5.20 user Namespace Reference

Classes

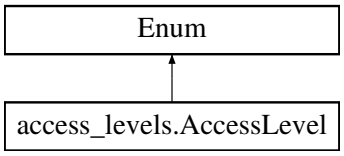
- class User

Chapter 6

Class Documentation

6.1 access_levels.AccessLevel Class Reference

Inheritance diagram for access_levels.AccessLevel:



Static Public Attributes

- int ADMIN = 1
- int SPECTATOR = 2

6.1.1 Detailed Description

```
Enum for access levels.
```

6.1.2 Member Data Documentation

6.1.2.1 ADMIN

```
int access_levels.AccessLevel.ADMIN = 1  [static]
```

6.1.2.2 SPECTATOR

```
int access_levels.AccessLevel.SPECTATOR = 2  [static]
```

The documentation for this class was generated from the following file:

- central/common/access_levels.py

6.2 benchmark.Benchmark Class Reference

Public Member Functions

- __init__ (self, str title, int max_laps=100, tuple labels=("Lap", "Time"))
- start_lap (self)
- end_lap (self)
- avg (self)
- standard_deviation (self)
- save (self)

Public Attributes

- title = title
- start_time = None
- list times = [ ]
- max_laps = max_laps
- bool done = False
- xlabel = labels[0]
- ylabel = labels[1]

6.2.1 Detailed Description

```
A class for handling benchmark related tasks.
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
benchmark.Benchmark.__init__ (
              self,
             str title,
             int  max_laps = 100,
             tuple  labels = ("Lap", "Time"))
```

```
Benchmark constructor.
@param title: Title of the benchmark.
```

```
00011     def __init__(self, title: str, max_laps: int = 100, labels: tuple = ("Lap", "Time")):
00012         """
00013         Benchmark constructor.
00014         @param title: Title of the benchmark.
00015         """
00016 
00017         self.title = title
00018         self.start_time = None
00019         self.times = []
00020         self.max_laps = max_laps
00021         self.done = False
00022         self.xlabel = labels[0]
00023         self.ylabel = labels[1]
00024 
```

6.2.3 Member Function Documentation

6.2.3.1 avg()

```
benchmark.Benchmark.avg (
              self)
```

Returns the average time of the benchmark.
@return: Average time.

```
00050     def avg(self):
00051         """
00052         Returns the average time of the benchmark.
00053         @return: Average time.
00054         """
00055 
00056         return sum(self.times) / len(self.times)
00057 
```

6.2.3.2 end_lap()

```
benchmark.Benchmark.end_lap (
              self)
```

Ends the current lap.

```
00032     def end_lap(self):
00033         """
00034         Ends the current lap.
00035         """
00036 
00037         if self.done:
00038             return
00039 
00040         if len(self.times) == self.max_laps:
00041             print("Benchmark done. Saving results ...")
00042             self.done = True
00043             self.save()
00044         else:
00045             self.times.append(time.perf_counter() - self.start_time)
00046             self.start_time = None
00047 
00048             print(f"Current lap ({len(self.times)} / {self.max_laps})")
00049 
```

6.2.3.3 save()

```
benchmark.Benchmark.save (
              self)
```

Plots the benchmark results.

```
00066     def save(self):
00067         """
00068         Plots the benchmark results.
00069         """
00070 
00071         matplotlib.use('Agg') # Use a non-interactive backend. Prevents thread issues.
00072 
00073         plt.figure()
00074         plt.plot(self.times)
00075         plt.title(self.title)
00076         plt.xlabel(self.xlabel)
00077         plt.ylabel(self.ylabel)
00078         plt.grid()
00079         plt.savefig(f"{self.title.replace(' ', '_')}-{int(time.time() * 1000)}.png")
00080         plt.close()
00081 
00082         print(f"Benchmark average time: {self.avg()} seconds, standard deviation:
      {self.standard_deviation()} seconds")
```

6.2.3.4 standard_deviation()

```
benchmark.Benchmark.standard_deviation (
              self)
```

Returns the standard deviation of the benchmark times.
@return: Standard deviation.

```
00058     def standard_deviation(self):
00059         """
00060         Returns the standard deviation of the benchmark times.
00061         @return: Standard deviation.
00062         """
00063 
00064         return np.std(self.times)
00065 
```

6.2.3.5 start_lap()

```
benchmark.Benchmark.start_lap (
              self)
```

Starts a new lap.

```
00025     def start_lap(self):
00026         """
00027         Starts a new lap.
00028         """
00029 
00030         self.start_time = time.perf_counter()
00031 
```

6.2.4 Member Data Documentation

6.2.4.1 done

```
bool benchmark.Benchmark.done = False
```

6.2.4.2 max_laps

```
benchmark.Benchmark.max_laps = max_laps
```

6.2.4.3 start_time

```
benchmark.Benchmark.start_time = None
```

6.2.4.4 times

```
benchmark.Benchmark.times = []
```

6.2.4.5 title

```
benchmark.Benchmark.title = title
```

6.2.4.6 xlabel

```
benchmark.Benchmark.xlabel = labels[0]
```

6.2.4.7 ylabel

```
benchmark.Benchmark.ylabel = labels[1]
```

The documentation for this class was generated from the following file:

- central/util/benchmark.py

6.3 db.DB Class Reference

Class for managing SQLite database connection.

Static Public Member Functions

- get_connection ()
- query (str sql, tuple args=(), bool commit=False)
- table_is_empty (table_name)
- migrations_populate ()
- migrations ()

6.3.1 Detailed Description

Class for managing SQLite database connection.

6.3.2 Member Function Documentation

6.3.2.1 get_connection()

```
db.DB.get_connection ()  [static]
```

```
Get a new database connection.
@return: SQLite connection object.
```

```
00011     def get_connection():
00012         """
00013         Get a new database connection.
00014         @return: SQLite connection object.
00015         """
00016 
00017         connection = sqlite3.connect("central.db")
00018         connection.row_factory = sqlite3.Row
00019 
00020         return connection
00021 
```

6.3.2.2 migrations()

db.DB.migrations () [static]

Run database migrations.

```
00087     def migrations():
00088         """
00089         Run database migrations.
00090         """
00091 
00092         connection = DB.get_connection()
00093         cursor = connection.cursor()
00094 
00095         # Create users table
00096         cursor.execute("""
00097             CREATE TABLE IF NOT EXISTS users (
00098                 id INTEGER PRIMARY KEY AUTOINCREMENT,
00099                 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
00100                 username TEXT NOT NULL UNIQUE,
00101                 password TEXT NOT NULL,
00102                 email TEXT NOT NULL UNIQUE,
00103                 first_name TEXT NOT NULL,
00104                 last_name TEXT NOT NULL,
00105                 access_level_id INTEGER NOT NULL,
00106                 FOREIGN KEY (access_level_id) REFERENCES access_levels(id)
00107             )
00108         """)
00109 
00110         # Create access_levels table
00111         cursor.execute("""
00112             CREATE TABLE IF NOT EXISTS access_levels (
00113                 id INTEGER PRIMARY KEY AUTOINCREMENT,
00114                 name TEXT NOT NULL UNIQUE
00115             )
00116         """)
00117 
00118         # Create logs table
00119         cursor.execute("""
00120             CREATE TABLE IF NOT EXISTS logs (
00121                 id INTEGER PRIMARY KEY AUTOINCREMENT,
00122                 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
00123                 message TEXT NOT NULL
00124             )
00125         """)
00126 
00127         # Create image_analysis table
00128         cursor.execute("""
00129             CREATE TABLE IF NOT EXISTS image_analysis (
00130                 id INTEGER PRIMARY KEY AUTOINCREMENT,
00131                 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
00132                 classification TEXT NOT NULL,
00133                 green_percentage REAL NOT NULL,
00134                 label TEXT NOT NULL,
00135                 score REAL NOT NULL
00136             )
00137         """)
00138 
00139         # Create bounding_boxes table
00140         cursor.execute("""
00141             CREATE TABLE IF NOT EXISTS bounding_boxes (
00142                 id INTEGER PRIMARY KEY AUTOINCREMENT,
00143                 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
00144                 xmin INTEGER NOT NULL,
00145                 ymin INTEGER NOT NULL,
00146                 xmax INTEGER NOT NULL,
00147                 ymax INTEGER NOT NULL,
00148                 image_analysis_id INTEGER NOT NULL,
00149                 FOREIGN KEY (image_analysis_id) REFERENCES image_analysis(id)
00150             )
00151         """)
00152 
00153         connection.commit()
00154         connection.close()
00155 
00156         DB.migrations_populate()
```

6.3.2.3 migrations_populate()

db.DB.migrations_populate () [static]

Populate the database with initial data.

```
00058     def migrations_populate():
00059         """
00060         Populate the database with initial data.
00061         """
00062 
00063         connection = DB.get_connection()
00064         cursor = connection.cursor()
00065 
00066         # Insert default access levels
00067         if DB.table_is_empty("access_levels"):
00068             cursor.executemany("INSERT INTO access_levels (name) VALUES (?)", [
00069                 ("admin",),
00070                 ("spectator",)
00071             ])
00072 
00073         # Insert default users
00074         if DB.table_is_empty("users"):
00075             user1 = user.User("admin", os.getenv("USER1_PASSWORD"), "admin@example.com", "John",
      "Green", 1)
00076             user2 = user.User("spectator", os.getenv("USER2_PASSWORD"), "spectator@example.com",
      "Leafy", "Green", 2)
00077 
00078             cursor.executemany("INSERT INTO users (username, password, email, first_name, last_name,
      access_level_id) VALUES (?, ?, ?, ?, ?, ?)", [
00079                 (user1.username, user1.password, user1.email, user1.first_name, user1.last_name,
      user1.access_level_id),
00080                 (user2.username, user2.password, user2.email, user2.first_name, user2.last_name,
      user2.access_level_id)
00081             ])
00082 
00083         connection.commit()
00084         connection.close()
00085 
```

6.3.2.4 query()

```
db.DB.query (
            str sql,
            tuple  args = (),
            bool  commit = False)  [static]
```

Execute a SQL query.
@param sql: SQL query string.
@param args: Arguments for the SQL query.
@param commit: Whether to commit the transaction.
@return: Result of the query (sqlite3.Row object).

```
00023     def query(sql: str, args: tuple = (), commit: bool = False):
00024         """
00025         Execute a SQL query.
00026         @param sql: SQL query string.
00027         @param args: Arguments for the SQL query.
00028         @param commit: Whether to commit the transaction.
00029         @return: Result of the query (sqlite3.Row object).
00030         """
00031 
00032         connection = DB.get_connection()
00033         cursor = connection.cursor()
00034         cursor.execute(sql, args)
00035 
00036         if commit:
00037             connection.commit()
00038 
00039         if cursor.description:
00040             result = cursor.fetchall()
00041 
00042         connection.close()
00043 
00044         return result
00045 
```

6.3.2.5 table_is_empty()

```
db.DB.table_is_empty (
              table_name)  [static]
```

Check if a table is empty.
@param table_name: Name of the table to check.
@return: True if the table is empty, False otherwise.

```
00047     def table_is_empty(table_name):
00048         """
00049         Check if a table is empty.
00050         @param table_name: Name of the table to check.
00051         @return: True if the table is empty, False otherwise.
00052         """
00053 
00054         res = DB.query(f"SELECT COUNT(*) FROM {table_name}")
00055         return res[0][0] == 0
00056 
```

The documentation for this class was generated from the following file:

- central/common/db.py

6.4 image_analysis.ImageAnalysis Class Reference

Public Member Functions

- __init__ (self, str classification, float green_percentage, float score)

Public Attributes

- classification = classification
- green_percentage = green_percentage
- score = score

6.4.1 Detailed Description

@brief Class for storing image analysis results in db.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

```
image_analysis.ImageAnalysis.__init__ (
              self,
             str classification,
             float green_percentage,
             float score)
```

```
00008     def __init__(self, classification: str, green_percentage: float, score: float):
00009         self.classification = classification
00010         self.green_percentage = green_percentage
00011         self.score = score
```

6.4.3 Member Data Documentation

6.4.3.1 classification

```
image_analysis.ImageAnalysis.classification = classification
```

6.4.3.2 green_percentage

```
image_analysis.ImageAnalysis.green_percentage = green_percentage
```

6.4.3.3 score

```
image_analysis.ImageAnalysis.score = score
```

The documentation for this class was generated from the following file:

- central/common/image_analysis.py

6.5 log.Log Class Reference

Public Member Functions

- __init__ (self, str message)
- save (self)

Static Public Member Functions

- load ()

Public Attributes

- message = message

6.5.1 Detailed Description

```
@brief Class for logging messages. Also handles saving logs to the database.
```

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
log.Log.__init__ (
              self,
             str message)
```

```
00008     def __init__(self, message: str):
00009         self.message = message
00010
```

6.5.3 Member Function Documentation

6.5.3.1 load()

```
log.Log.load ()  [static]
```

Load all log messages from database.

```
00012     def load():
00013         """
00014         Load all log messages from database.
00015         """
00016 
00017         logs = db.DB.query("SELECT * FROM logs")
00018         return logs
00019 
```

6.5.3.2 save()

```
log.Log.save (
               self)
```

Save the log message to the database.

```
00020     def save(self):
00021         """
00022         Save the log message to the database.
00023         """
00024 
00025         db.DB.query("INSERT INTO logs (message) VALUES (?)", (self.message,), commit=True)
```

6.5.4 Member Data Documentation

6.5.4.1 message

```
log.Log.message = message
```

The documentation for this class was generated from the following file:

- central/common/log.py

6.6 plant_manager.PlantManager Class Reference

Public Member Functions

- __init__ (self, image)
- detect_plants (self)
- world_coordinates (self, x, y, z)
- draw_geometry (self, bounding_boxes, contours, centroids)
- draw_hud (self, bounding_boxes, contours, centroids)
- get_binary_image (self)

Public Attributes

- image = image
- list plants = [ ]

6.6.1 Constructor & Destructor Documentation

6.6.1.1 __init__()

```
plant_manager.PlantManager.__init__ (
               self,
               image)
```

```
00006     def __init__(self, image):
00007         self.image = image
00008         self.plants = []
00009
```

6.6.2 Member Function Documentation

6.6.2.1 detect_plants()

```
plant_manager.PlantManager.detect_plants (
               self)
```

```
Detects plants in the image using the CV mask_generation task.
@return: A tuple containing bounding boxes, contours, and centroids of detected plants.
```

```
00010     def detect_plants(self):
00011         """
00012         Detects plants in the image using the CV mask_generation task.
00013         @return: A tuple containing bounding boxes, contours, and centroids of detected plants.
00014         """
00015         mask = generate_mask(self.image)
00016         return generate_geometry_from_mask(self.image, mask)
00017
```

6.6.2.2 draw_geometry()

```
plant_manager.PlantManager.draw_geometry (
               self,
               bounding_boxes,
               contours,
               centroids)
```

```
Draws geometry (bounding boxes, contours, and centroids) on the image.
@param bounding_boxes: List of bounding boxes.
@param contours: List of contours.
@param centroids: List of centroids.
@return: Image with geometry drawn on.
```

```
00029     def draw_geometry(self, bounding_boxes, contours, centroids):
00030         """
00031         Draws geometry (bounding boxes, contours, and centroids) on the image.
00032         @param bounding_boxes: List of bounding boxes.
00033         @param contours: List of contours.
00034         @param centroids: List of centroids.
00035         @return: Image with geometry drawn on.
00036         """
00037 
00038         for i, bbox in enumerate(bounding_boxes):
00039             xmin, ymin, xmax, ymax = bbox
00040             cv2.rectangle(self.image, (xmin, ymin), (xmax, ymax), (255, 0, 0), 2)
00041 
00042         for contour in contours:
00043             cv2.drawContours(self.image, [contour], -1, (0, 0, 255), 2)
00044 
00045         for centroid in centroids:
00046             cv2.circle(self.image, centroid, 10, (0, 255, 0), -1)
00047 
00048         return self.image
00049 
```

6.6.2.3 draw_hud()

```
plant_manager.PlantManager.draw_hud (
              self,
              bounding_boxes,
              contours,
              centroids)
```

```
Draws a HUD on the image with useful debugging info.
@param bounding_boxes: List of bounding boxes.
@param contours: List of contours.
@param centroids: List of centroids.
@return: Image with HUD drawn on.
```

```
00050     def draw_hud(self, bounding_boxes, contours, centroids):
00051         """
00052         Draws a HUD on the image with useful debugging info.
00053         @param bounding_boxes: List of bounding boxes.
00054         @param contours: List of contours.
00055         @param centroids: List of centroids.
00056         @return: Image with HUD drawn on.
00057         """
00058 
00059         image_size_text = f"Image Size: {self.image.shape[1]}x{self.image.shape[0]}"
00060         cv2.putText(self.image, image_size_text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 255,
      255), 1, cv2.LINE_AA)
00061 
00062         for i, bbox in enumerate(bounding_boxes):
00063             xmin, ymin, xmax, ymax = bbox
00064 
00065             cv2.putText(self.image, f"ID: {i}", (xmin - 300, ymin + 80), cv2.FONT_HERSHEY_PLAIN, 1,
     (0, 0, 0), 1)
00066             #coords
00067             cv2.putText(self.image, f"COORDS: ({xmin}, {ymin}) ({xmax}, {ymax})", (xmin - 300, ymin +
      100), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 0), 1)
00068             #world coords
00069             world_coords = self.world_coordinates(xmin, ymin, 0)
00070             cv2.putText(self.image, f"WORLD COORDS: ({world_coords[0]:.2f}, {world_coords[1]:.2f},
      {world_coords[2]:.2f})", (xmin - 300, ymin + 120), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 0), 1)
00071             #grab point
00072             grab_point = self.world_coordinates(centroids[i][0], centroids[i][1], 0)
00073             cv2.putText(self.image, f"GRAB POINT: ({grab_point[0]:.2f}, {grab_point[1]:.2f},
      {grab_point[2]:.2f})", (xmin - 300, ymin + 140), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 0), 1)
00074 
00075         return self.image
00076 
```

6.6.2.4 get_binary_image()

```
plant_manager.PlantManager.get_binary_image (
              self)
```

```
Returns the binary image of the detected plants.
@return: Binary image of the detected plants.
```

```
00077     def get_binary_image(self):
00078         """
00079         Returns the binary image of the detected plants.
00080         @return: Binary image of the detected plants.
00081         """
00082 
00083         _, img_encoded = cv2.imencode(".jpg", self.image)
00084         return io.BytesIO(img_encoded.tobytes())
```

6.6.2.5 world_coordinates()

```
plant_manager.PlantManager.world_coordinates (
               self,
               x,
               y,
               z)
```

```
Returns world coordinates (not yet implemented).
@param x: x coordinate.
@param y: y coordinate.
@param z: z coordinate.
@return: World coordinates (x, y, z).
```

```
00018     def world_coordinates(self, x, y, z):
00019         """
00020         Returns world coordinates (not yet implemented).
00021         @param x: x coordinate.
00022         @param y: y coordinate.
00023         @param z: z coordinate.
00024         @return: World coordinates (x, y, z).
00025         """
00026 
00027         return (0, 0, 0)
00028 
```

6.6.3 Member Data Documentation

6.6.3.1 image

```
plant_manager.PlantManager.image = image
```

6.6.3.2 plants

```
list plant_manager.PlantManager.plants = []
```

The documentation for this class was generated from the following file:

- central/ai/plant_manager.py

6.7 user.User Class Reference

Public Member Functions

- __init__ (self, str username, str password, str email, str first_name, str last_name, int access_level_id)
- int get_access_level (self)
- bool is_admin (self)
- bool is_spectator (self)
- __iter__ (self)

Static Public Member Functions

- bool auth (str username, str password)
- "User" get_user (identifier)

Public Attributes

- username = username
- email = email
- first_name = first_name
- last_name = last_name
- access_level_id = access_level_id
- password = generate_password_hash(password, method="scrypt")

6.7.1 Constructor & Destructor Documentation

6.7.1.1 __init__()

```
user.User.__init__ (
               self,
              str username,
              str password,
              str email,
              str first_name,
              str last_name,
              int access_level_id)
```

```
00006     def __init__(self, username: str, password: str, email: str, first_name: str, last_name: str,
      access_level_id: int):
00007         self.username = username
00008         self.email = email
00009         self.first_name = first_name
00010         self.last_name = last_name
00011         self.access_level_id = access_level_id
00012 
00013         # Make sure to not hash the password if we're just loading an existing user.
00014         if password != "":
00015             self.password = generate_password_hash(password, method="scrypt")
00016 
```

6.7.2 Member Function Documentation

6.7.2.1 __iter__()

```
user.User.__iter__ (
               self)
```

```
00078     def __iter__(self):
00079         return iter((self.username, self.password, self.email, self.first_name, self.last_name,
      self.access_level_id))
```

6.7.2.2 auth()

```
bool user.User.auth (
            str username,
            str password)  [static]
```

Authenticate user with username and password.
@param username: Username of the user.
@param password: Password of the user.
@return: True if authenticated, False otherwise.

```
00018     def auth(username: str, password: str) -> bool:
00019         """
00020         Authenticate user with username and password.
00021         @param username: Username of the user.
00022         @param password: Password of the user.
00023         @return: True if authenticated, False otherwise.
00024         """
00025
00026         result = db.DB.query("SELECT password FROM users WHERE username = ?", (username,))
00027
00028         if check_password_hash(result[0]["password"], password):
00029             return True
00030
00031         return False
00032
```

6.7.2.3 get_access_level()

```
int user.User.get_access_level (
             self)
```

Get user access level.
@return: Access level name.

```
00052     def get_access_level(self) -> int:
00053         """
00054         Get user access level.
00055         @return: Access level name.
00056         """
00057         result = db.DB.query("SELECT * FROM access_levels WHERE id = ?", (self.access_level_id,))
00058
00059         if result:
00060             return result[0]["name"]
00061
00062         return None
00063
```

6.7.2.4 get_user()

```
"User" user.User.get_user (
             identifier)  [static]
```

Get user by username or id.
@param identifier: User ID or username.
@return: User object.

```
00034     def get_user(identifier) -> "User":
00035         """
00036         Get user by username or id.
00037         @param identifier: User ID or username.
00038         @return: User object.
00039         """
00040
00041         if isinstance(identifier, int):
00042             result = db.DB.query("SELECT * FROM users WHERE id = ?", (identifier,))
00043         else:
00044             result = db.DB.query("SELECT * FROM users WHERE username = ?", (identifier,))
00045
00046         if result:
00047             result = result[0]
00048             return User(result["username"], "", result["email"], result["first_name"],
      result["last_name"], result["access_level_id"])
00049
00050         return None
00051
```

6.7.2.5 is_admin()

```
bool user.User.is_admin (
              self)
```

Check if user is admin.
@return: True if admin, False otherwise.

```
00064     def is_admin(self) -> bool:
00065         """
00066         Check if user is admin.
00067         @return: True if admin, False otherwise.
00068         """
00069         return self.access_level_id == AccessLevel.ADMIN.value
00070
```

6.7.2.6 is_spectator()

```
bool user.User.is_spectator (
              self)
```

Check if user is spectator.
@return: True if spectator, False otherwise.

```
00071     def is_spectator(self) -> bool:
00072         """
00073         Check if user is spectator.
00074         @return: True if spectator, False otherwise.
00075         """
00076         return self.access_level_id == AccessLevel.SPECTATOR.value
00077
```

6.7.3 Member Data Documentation

6.7.3.1 access_level_id

```
user.User.access_level_id = access_level_id
```

6.7.3.2 email

```
user.User.email = email
```

6.7.3.3 first_name

```
user.User.first_name = first_name
```

6.7.3.4 last_name

```
user.User.last_name = last_name
```

6.7.3.5 password

```
user.User.password = generate_password_hash(password, method="scrypt")
```

6.7.3.6 username

```
user.User.username = username
```

The documentation for this class was generated from the following file:

- central/common/user.py

Chapter 7

File Documentation

7.1 central/ai/chessboard.py File Reference

Namespaces

- namespace chessboard

Variables

- chessboard.img = cv2.imread("./central/cache/camera01.jpg")
- chessboard.gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- tuple chessboard.pattern_size = (5, 8)
- chessboard.ret
- chessboard.corners

7.2 central/ai/nodes/camera_feed.py File Reference

Namespaces

- namespace camera_feed

7.3 central/ai/nodes/grip_point.py File Reference

Namespaces

- namespace grip_point

Functions

- grip_point.get_grip_point (image)

7.4 central/ai/plant_manager.py File Reference

Classes

- class plant_manager.PlantManager

Namespaces

- namespace plant_manager

7.5 central/ai/tasks/depth_estimation.py File Reference

Namespaces

- namespace depth_estimation

Functions

- depth_estimation.estimate_depth (image, fast=True)

Variables

- depth_estimation.estimator_fast = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Metric-Indoor-Small-hf")
- depth_estimation.estimator_slow = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf")

7.6 central/ai/tasks/green_percentage.py File Reference

Namespaces

- namespace green_percentage

Functions

- green_percentage.estimate_green_percentage (image)

7.7 central/ai/tasks/image_classification.py File Reference

Namespaces

- namespace image_classification

Functions

- image_classification.classify_image (image)

Variables

- image_classification.classifier = pipeline("zero-shot-image-classification", model="openai/clip-vit-base-patch32")

7.8 central/ai/tasks/mask_generation.py File Reference

Namespaces

- namespace mask_generation

Functions

- mask_generation.generate_mask (image)
- mask_generation.generate_geometry_from_mask (image, mask)

7.9 central/ai/tasks/object_detection.py File Reference

Namespaces

- namespace object_detection

Functions

- object_detection.object_detection (image)

Variables

- object_detection.model = YOLO("leafy-ai-obj-detection.pt")

7.10 central/api.py File Reference

Namespaces

- namespace api

Functions

- api.home ()
- api.status ()
- api.stats_route ()
- api.log_route ()
- api.get_camera_feed ()
- api.visual_get_mask ()
- api.visual_get_geometry ()
- api.visual_get_depth ()
- api.capture_image_route ()
- api.classify_image_route ()

Variables

- api.routes = Blueprint("api_v1", __name__, url_prefix="/api/v1")
- dict api.stats

7.11 central/common/access_levels.py File Reference

Classes

- class access_levels.AccessLevel

Namespaces

- namespace access_levels

7.12 central/common/db.py File Reference

Classes

- class db.DB

  Class for managing SQLite database connection.

Namespaces

- namespace db

7.13 central/common/image_analysis.py File Reference

Classes

- class image_analysis.ImageAnalysis

Namespaces

- namespace image_analysis

7.14 central/common/log.py File Reference

Classes

- class log.Log

Namespaces

- namespace log

7.15 central/common/user.py File Reference

Classes

- class user.User

Namespaces

- namespace user

7.16 central/hmi.py File Reference

Namespaces

- namespace hmi

Functions

- hmi.dashboard ()
- hmi.login ()
- hmi.logout ()

Variables

- hmi.routes = Blueprint("hmi", __name__)

7.17 central/main.py File Reference

Namespaces

- namespace main

Functions

- main.get_frame ()

Variables

- main.app = Flask(__name__)
- main.socketio = SocketIO(app, cors_allowed_origins="*")
- main.debug
- main.True
- main.host

7.18 training/main.py File Reference

Namespaces

- namespace main

Variables

- main.model = YOLO("yolo11n.pt")
- main.results = model.train(data="training/datasets/leafy-automation/data.yaml", epochs=100, imgsz=640)
- main.metrics = model.val()
- main.path = model.export(format="onnx")

7.19 central/routes.py File Reference

Namespaces

- namespace routes

Functions

- routes.home ()
- routes.classify_image_route ()

Variables

- routes.routes = Blueprint("routes", __name__)

7.20 central/static/js/controllers/CameraController.js File Reference

7.21 central/static/js/controllers/StatusController.js File Reference

7.22 central/static/js/main.js File Reference

7.23 central/static/js/models/CameraModel.js File Reference

7.24 central/static/js/models/StatusModel.js File Reference

7.25 central/static/js/views/CameraView.js File Reference

7.26 central/static/js/views/StatusView.js File Reference

7.27 central/util/benchmark.py File Reference

Classes

- class benchmark.Benchmark

Namespaces

- namespace benchmark

7.28 central/util/misc.py File Reference

Namespaces

- namespace misc

Functions

- misc.get_device ()
- misc.img_base64 (image)

Index

2 Leafy Automation Core

Leafy Automation Core

Generated by Doxygen 1.13.2

Chapter 1

Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Topic Documentation

4.1 Configuration

Variables

- static constexpr uint8_t ARM_JOINTS = 5

  Number of stepper-driven joints (J0. . . J4).
- static constexpr uint8_t STEP_PIN [ARM_JOINTS]

  STEP pin mapping for joints J0. . . J4.
- static constexpr uint8_t DIR_PIN [ARM_JOINTS]

  DIR pin mapping for joints J0. . . J4.
- static constexpr uint16_t MICROSTEPS = 200

  Microsteps per full revolution.
- static constexpr float GEAR_RATIO [ARM_JOINTS]

  Gear ratio for each joint.
- static constexpr float MAX_OUTPUT_RPM [ARM_JOINTS]

  Max output RPM per joint.
- static constexpr uint8_t LIMIT_LEFT_PINS [ARM_JOINTS]

  Array of digital input pins connected to each joint's left limit switch (Active LOW)
- static constexpr uint8_t LIMIT_RIGHT_PINS [ARM_JOINTS]

  Array of digital input pins connected to each joint's right limit switch (Active LOW)
- static constexpr uint8_t GRIPPER_SERVO_PIN

  PWM pin for the servo controlling the gripper (End Effector, EF).
- static constexpr unsigned long GRIP_MOVE_TIME_MS = 500

  Allocated time in milliseconds for the gripper to open/close.
- static constexpr uint8_t GRIPPER_CLOSED_ANGLE = 0

  Open closed (degrees) for the gripper servo.
- static constexpr uint8_t GRIPPER_OPEN_ANGLE = 90

  Open angle (degrees) for the gripper servo.

4.1.1 Detailed Description

4.1.2 Variable Documentation

4.1.2.1 ARM_JOINTS

```
uint8_t ARM_JOINTS = 5  [static], [constexpr]
```

Number of stepper-driven joints (J0. . . J4).

Definition at line 21 of file config.h.

4.1.2.2 DIR_PIN

```
uint8_t DIR_PIN[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
    1,
    1,
    1,
    1,
    1}
```

DIR pin mapping for joints J0...J4.

Definition at line 35 of file config.h.

```
00035                                                {
00036     /* J0 */ 1, // TO DO: Replace 1s with actual pin numbers
00037     /* J1 */ 1,
00038     /* J2 */ 1,
00039     /* J3 */ 1,
00040     /* J4 */ 1};
```

4.1.2.3 GEAR_RATIO

```
float GEAR_RATIO[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
    1.0f / 10.0f,
    1.0f / 50.0f,
    1.0f / 50.0f,
    1.0f / 19.0f,
    1.0f / 16.0f}
```

Gear ratio for each joint.

Definition at line 48 of file config.h.

```
00048                                                 {
00049     /* J0 */ 1.0f / 10.0f,
00050     /* J1 */ 1.0f / 50.0f,
00051     /* J2 */ 1.0f / 50.0f,
00052     /* J3 */ 1.0f / 19.0f,
00053     /* J4 */ 1.0f / 16.0f};
```

4.1.2.4 GRIP_MOVE_TIME_MS

```
unsigned long GRIP_MOVE_TIME_MS = 500  [static], [constexpr]
```

Allocated time in milliseconds for the gripper to open/close.

Definition at line 85 of file config.h.

4.1.2.5 GRIPPER_CLOSED_ANGLE

```
uint8_t GRIPPER_CLOSED_ANGLE = 0  [static], [constexpr]
```

Open closed (degrees) for the gripper servo.

Definition at line 89 of file config.h.

4.1.2.6 GRIPPER_OPEN_ANGLE

```
uint8_t GRIPPER_OPEN_ANGLE = 90  [static], [constexpr]
```

Open angle (degrees) for the gripper servo.

Definition at line 93 of file config.h.

4.1.2.7 GRIPPER_SERVO_PIN

```
uint8_t GRIPPER_SERVO_PIN  [static], [constexpr]
```

Initial value:

```
=
    1
```

PWM pin for the servo controlling the gripper (End Effector, EF).

Definition at line 80 of file config.h.

4.1.2.8 LIMIT_LEFT_PINS

```
uint8_t LIMIT_LEFT_PINS[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
    xx, xx, xx, xx, xx}
```

Array of digital input pins connected to each joint's left limit switch (Active LOW)

Definition at line 68 of file config.h.

```
00068                                                   {
00069     xx, xx, xx, xx, xx}; // Replace with pin numbers
```

4.1.2.9 LIMIT_RIGHT_PINS

```
uint8_t LIMIT_RIGHT_PINS[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
    xx, xx, xx, xx, xx}
```

Array of digital input pins connected to each joint's right limit switch (Active LOW)

Definition at line 75 of file config.h.

```
00075                                                    {
00076     xx, xx, xx, xx, xx}; // Replace with pin numbers
```

4.1.2.10 MAX_OUTPUT_RPM

```
float MAX_OUTPUT_RPM[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
     30.0f,
     20.0f,
     20.0f,
     25.0f,
     25.0f}
```

Max output RPM per joint.

Definition at line 57 of file config.h.

```
00057                                                 {
00058     /* J0 */ 30.0f,
00059     /* J1 */ 20.0f,
00060     /* J2 */ 20.0f,
00061     /* J3 */ 25.0f,
00062     /* J4 */ 25.0f};
```

4.1.2.11 MICROSTEPS

```
uint16_t MICROSTEPS = 200  [static], [constexpr]
```

Microsteps per full revolution.

Definition at line 44 of file config.h.

4.1.2.12 STEP_PIN

```
uint8_t STEP_PIN[ARM_JOINTS]  [static], [constexpr]
```

Initial value:

```
= {
     1,
     1,
     1,
     1,
     1}
```

STEP pin mapping for joints J0. . . J4.

Precondition

STEP_PIN size must equal ARM_JOINTS

Definition at line 26 of file config.h.

```
00026                                             {
00027     /* J0 */ 1, // TO DO: Replace 1s with actual pin numbers
00028     /* J1 */ 1,
00029     /* J2 */ 1,
00030     /* J3 */ 1,
00031     /* J4 */ 1};
```

4.2 Communication Manager

Files

- file communication_manager.cpp

  Implementation of the Communication Manager module.

Functions

- void handleIncomingCommand (const String &command)

  Handle an incoming command and route it to the appropriate module.
- void checkActionStatus ()

  Checks in-progress flags and publish DONE messages.

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 checkActionStatus()

```
void checkActionStatus ()
```

Checks in-progress flags and publish DONE messages.

Checks in-progress flags and publishes DONE messages.

Note

Must be called each loop to detect action completion promptly.

Returns

void

Checks the movementInProgress, gripperInProgress, and calibrationInProgress flags. For each flag that is set, it calls the corresponding completion test:

- allJointsDone() for MOVE
- gripperDone() for GRIP
- calibrationDone() for CALIBRATE If the test returns true, it publishes the respective “DONE” status via publishStatus() and clears the in-progress flag.

  Returns

  void

  Note

  Must be called in every main loop to catch completions promptly.

Definition at line 112 of file communication_manager.cpp.

```
00112                         {
00113
00114   if (calibrationInProgress && calibrationDone()) {
00115     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATION DONE");
00116     calibrationInProgress = false;
00117   }
00118   if (movementInProgress && allJointsDone()) {
00119     publishStatus(MQTT_TOPIC_STATUS_MOTION, "MOVE DONE");
00120     movementInProgress = false;
00121   }
00122
00123   if (gripperInProgress && gripperDone()) {
00124     publishStatus(MQTT_TOPIC_STATUS_GRIPPER, "GRIPPER DONE");
00125     gripperInProgress = false;
00126   }
00127 }
```

4.2.2.2 handleIncomingCommand()

```
void handleIncomingCommand (
            const String & command)
```

Handle an incoming command and route it to the appropriate module.

Decode and route a received command string.

Parameters

command	Text like "MOVE 100 200 300 400 500" or "GRIP 1".

Returns

void

Note

Is called from the MQTT Client's callback.

Definition at line 30 of file communication_manager.cpp.

```
00030                                                 {
00031   String trimmed = command;
00032   trimmed.trim(); // Removing leading/trailing whitespaces
00033
00034   if (trimmed.startsWith("MOVE")) {
00035     handleMoveCommand(trimmed);
00036   } else if (trimmed.startsWith("GRIP")) {
00037     handleGripperCommand(trimmed);
00038   } else if (trimmed.startsWith("CALIBRATE")) {
00039     handleCalibrationCommand(trimmed);
00040   } else {
00041     Serial.print("[CommunicationManager] Unknown command: ");
00042     Serial.println(trimmed);
00043   }
00044 }
```

4.3 Gripper Driver

Functions

- void initGripper ()

  Initialise the gripper servo on its PWM pin and open it.
- void moveGripper (int state)

  Command the gripper to open (state=0) or close (state=1).
- void updateGripper ()

  Update the gripper; clear the moving flag after the configured move time.
- bool gripperDone ()

  Returns true if the gripper has completed its movement.

Variables

- static Servo gripperServo
- static bool moving = false
- static unsigned long moveStartTime = 0

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 gripperDone()

```
bool gripperDone ()
```

Returns true if the gripper has completed its movement.

Check whether the gripper has completed its action.

Returns

true if no gripper motion is in progress.

Definition at line 52 of file gripper_driver.cpp.

```
00052 { return !moving; }
```

4.3.2.2 initGripper()

```
void initGripper ()
```

Initialise the gripper servo on its PWM pin and open it.

Initialise the gripper servo and set to open position.

Precondition

GRIPPER_SERVO_PIN must be defined in config.h.

Postcondition

Servo is attached and moved to open angle.

Returns

void

Definition at line 22 of file gripper_driver.cpp.

```
00022                   {
00023   gripperServo.attach(GRIPPER_SERVO_PIN);
00024   gripperServo.write(GRIPPER_OPEN_ANGLE);
00025   moving = false;
00026 }
```

4.3.2.3 moveGripper()

```
void moveGripper (
            int state)
```

Command the gripper to open (state=0) or close (state=1).

Command the gripper to open or close.

Parameters

state	0 = open, 1 = close.

Returns

void

Definition at line 30 of file gripper_driver.cpp.

```
00030                             {
00031   uint8_t angle;
00032   if (state) {
00033     angle = GRIPPER_CLOSED_ANGLE;
00034   } else {
00035     angle = GRIPPER_OPEN_ANGLE;
00036   }
00037   gripperServo.write(angle);
00038   moveStartTime = millis();
00039   moving = true;
00040 }
```

4.3.2.4 updateGripper()

```
void updateGripper ()
```

Update the gripper; clear the moving flag after the configured move time.

Must be called frequently to update the gripper motion state.

Returns

void

Definition at line 44 of file gripper_driver.cpp.

```
00044                         {
00045   if (moving && (millis() - moveStartTime >= GRIP_MOVE_TIME_MS)) {
00046     moving = false;
00047   }
00048 }
```

4.3.3 Variable Documentation

4.3.3.1 gripperServo

```
Servo gripperServo  [static]
```

Definition at line 14 of file gripper_driver.cpp.

4.3.3.2 moveStartTime

```
unsigned long moveStartTime = 0  [static]
```

Definition at line 18 of file gripper_driver.cpp.

4.3.3.3 moving

```
bool moving = false  [static]
```

Definition at line 17 of file gripper_driver.cpp.

4.4 Motor Driver

Files

- file motor_driver.cpp

  Implementation of the Motor Driver for joints J0–J4.

Functions

- void initMotors ()

  Initialise all stepper drivers' speed & acceleration per config.h settings.
- void moveJoint (uint8_t jointIndex, int32_t stepCount)

  Queue a relative microstep move for a specific joint.
- void updateMotors ()

  Must be called every loop in order to advance the stepper motors.
- void calibrateAllJoints ()

  Perform a blocking homing routine (the calibration routine) of all joints with timeout and debounce.
- bool calibrationDone ()

  Check if calibration has completed.
- bool allJointsDone ()

  Check if all steppers have reached their targets.
- int32_t getJointPosition (uint8_t jointIndex)

  Get the current microstep position of a joint.
- void stopAllJoints ()

  Stop all motor motion immediately by clearing queued moves.

4.4.1 Detailed Description

4.4.2 Function Documentation

4.4.2.1 allJointsDone()

```
bool allJointsDone ()
```

Check if all steppers have reached their targets.

Returns

true if every joint's distanceToGo()==0.

Definition at line 116 of file motor_driver.cpp.

```
00116                       {
00117   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00118     if (stepperMotors[j].distanceToGo() != 0)
00119       return false;
00120   }
00121   return true;
00122 }
```

4.4.2.2 calibrateAllJoints()

```
void calibrateAllJoints ()
```

Perform a blocking homing routine (the calibration routine) of all joints with timeout and debounce.

Run a blocking homing (calibration) sequence on all stepper joints.

Precondition

LIMIT_LEFT_PINS[] and LIMIT_RIGHT_PINS[] must be defined in /include/config.h.

Postcondition

After return, currentPosition()==0 for each motor.

Returns

void

Note

This routine blocks until all limit switches have been found.

Definition at line 55 of file motor_driver.cpp.

```
00055                           {
00056   const unsigned long timeoutMs = 5000; // max time per switch
00057   const unsigned int debounceMs = 50;   // debounce delay
00058   homing = true;
00059   homed = false;
00060
00061   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00062     unsigned long startTime;
00063     bool switchState;
00064
00065     // Drive toward left switch
00066     stepperMotors[j].setMaxSpeed(MICROSTEPS * 100.0f);
00067     stepperMotors[j].moveTo(-1000000);
00068     startTime = millis();
00069     while (true) {
00070       stepperMotors[j].run();
00071       switchState = digitalRead(LIMIT_LEFT_PINS[j]) == LOW; // active low
00072       if (switchState) {
00073         delay(debounceMs);
00074         if (digitalRead(LIMIT_LEFT_PINS[j]) == LOW)
00075           break;
00076       }
00077       if (millis() - startTime > timeoutMs)
00078         break;
00079     }
00080     stepperMotors[j].setCurrentPosition(0);
00081
00082     // Drive toward right switch
00083     stepperMotors[j].moveTo(1000000);
00084     startTime = millis();
00085     while (true) {
00086       stepperMotors[j].run();
00087       switchState = digitalRead(LIMIT_RIGHT_PINS[j]) == LOW;
00088       if (switchState) {
00089         delay(debounceMs);
00090         if (digitalRead(LIMIT_RIGHT_PINS[j]) == LOW)
00091           break;
00092       }
00093       if (millis() - startTime > timeoutMs)
00094         break;
00095     }
00096     long maxSteps = stepperMotors[j].currentPosition();
00097
00098     // Return to midpoint
00099     long mid = maxSteps / 2;
00100     stepperMotors[j].setCurrentPosition(0);
00101     stepperMotors[j].moveTo(mid);
00102     while (stepperMotors[j].distanceToGo() != 0) {
00103       stepperMotors[j].run();
00104     }
00105     stepperMotors[j].setCurrentPosition(0);
00106   }
00107
00108   homed = true;
00109   homing = false;
00110 }
```

4.4.2.3 calibrationDone()

```
bool calibrationDone ()
```

Check if calibration has completed.

Returns

true if the last call to calibrateAllJoints() completed.

Definition at line 113 of file motor_driver.cpp.

```
00113 { return homed; }
```

4.4.2.4 getJointPosition()

```
int32_t getJointPosition (
            uint8_t jointIndex)
```

Get the current microstep position of a joint.

Parameters

jointIndex	Index of the joint (0...4).

Returns

Current position in microsteps (zeroed at last calibration).

Definition at line 125 of file motor_driver.cpp.

```
00125                                                {
00126   return stepperMotors[jointIndex].currentPosition();
00127 }
```

4.4.2.5 initMotors()

```
void initMotors ()
```

Initialise all stepper drivers' speed & acceleration per config.h settings.

Initialise stepper parameters(max speed & acceleration).

Precondition

STEP_PIN[], DIR_PIN[], MICROSTEPS, GEAR_RATIO[] and MAX_OUTPUT_RPM[] must be configured via config.h.

Postcondition

Each steppers[j] has its maxSpeed and acceleration set.

Returns

void

Definition at line 29 of file motor_driver.cpp.

```
00029                  {
00030   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00031     // compute max step rate: (RPM/60) * (microsteps/gear_ratio)
00032     float stepsPerSec =
00033         (MAX_OUTPUT_RPM[j] / 60.0f) * (MICROSTEPS / GEAR_RATIO[j]);
00034     stepperMotors[j].setMaxSpeed(stepsPerSec);
00035     stepperMotors[j].setAcceleration(stepsPerSec * 2.0f);
00036   }
00037   homed = false;
00038 }
```

4.4.2.6 moveJoint()

```
void moveJoint (
            uint8_t jointIndex,
            int32_t stepCount)
```

Queue a relative microstep move for a specific joint.

Parameters

jointIndex	Index of the joint (0...4, i.e. J0...J4).
steps	Signed microstep delta (positive=forward, negative=backward).

Returns

void

Definition at line 41 of file motor_driver.cpp.

```
00041                                                  {
00042   stepperMotors[jointIndex].move(stepCount);
00043 }
```

4.4.2.7 stopAllJoints()

```
void stopAllJoints ()
```

Stop all motor motion immediately by clearing queued moves.

Returns

void

Definition at line 130 of file motor_driver.cpp.

```
00130                       {
00131   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00132     stepperMotors[j].stop();
00133   }
00134 }
```

4.4.2.8 updateMotors()

```
void updateMotors ()
```

Must be called every loop in order to advance the stepper motors.

Returns

void

Definition at line 46 of file motor_driver.cpp.

```
00046                       {
00047   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00048     stepperMotors[j].run();
00049   }
00050 }
```

4.5 MQTT Client Module

Functions

- void initMQTT ()

  Initialise MQTT server and set callback.
- bool publishStatus (const char *topic, const String &message)

  Publish a status message and report failure.
- void sendHeartbeat ()

  Register the handler for incoming MQTT messages.
- void setMessageHandler (void(*handler)(const String &msg))

  Registers a callback to handle incoming parsed MQTT messages.
- void mqttLoop ()

  Process incoming messages and reconnect with exponential backoff.

Variables

- constexpr char MQTT_TOPIC_MOTION [ ] = "leafy_automation/motion"
- constexpr char MQTT_TOPIC_GRIPPER [ ] = "leafy_automation/gripper"
- constexpr char MQTT_TOPIC_CALIBRATE [ ] = "leafy_automation/calibrate"
- constexpr char MQTT_TOPIC_STATUS_COMMAND [ ]
- constexpr char MQTT_TOPIC_STATUS_MOTION [ ]
- constexpr char MQTT_TOPIC_STATUS_GRIPPER [ ]
- constexpr char MQTT_TOPIC_STATUS_CALIBRATION [ ]
- constexpr char MQTT_TOPIC_STATUS_HEARTBEAT [ ]

4.5.1 Detailed Description

4.5.2 Function Documentation

4.5.2.1 initMQTT()

```
void initMQTT ()
```

Initialise MQTT server and set callback.

Initialises MQTT connection and subscribes to control topics.

Precondition

WiFi is connected via initWiFi().

Postcondition

Single connect attempt. Further reconnects in mqttLoop().

Returns

void

Precondition

WiFi must already be connected via initWiFi().

Postcondition

Control topics are subscribed and the incoming message callback is set.

Definition at line 62 of file mqtt_client.cpp.

```
00062                 {
00063   mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
00064   mqttClient.setCallback(mqttCallback);
00065
00066   // Establish connection attempt
00067   if (mqttClient.connect(MQTT_CLIENT_ID)) {
00068     subscribeTopics();
00069     Serial.println("MQTT connected.");
00070   } else {
00071     Serial.print("MQTT connect failed, rc=");
00072     Serial.println(mqttClient.state());
00073   }
00074 }
```

4.5.2.2 mqttLoop()

```
void mqttLoop ()
```

Process incoming messages and reconnect with exponential backoff.

Process incoming MQTT traffic and attempt reconnects if needed.

Must be called frequently in loop() to maintain the connection.

Note

Must be called frequently in loop() to maintain connection.

Returns

void

Note

Must be called frequently in loop() to maintain the connection.

Returns

void

Definition at line 132 of file mqtt_client.cpp.

```
00132                 {
00133   unsigned long now = millis();
00134
00135   if (!mqttClient.connected() &&
00136       (now - LAST_ATTEMPT >= backoffInterval(RETRIES))) {
00137     if (mqttClient.connect(MQTT_CLIENT_ID)) {
00138       Serial.println("MQTT reconnected");
00139       subscribeTopics();
00140       RETRIES = 0;
00141     } else {
00142       RETRIES++;
00143       Serial.println("MQTT reconnect failed, will retry");
00144     }
00145     LAST_ATTEMPT = now;
00146   }
00147   mqttClient.loop();
00148 }
```

4.5.2.3 publishStatus()

```
bool publishStatus (
            const char * topic,
            const String & message)
```

Publish a status message and report failure.

Publishes a status message to a given MQTT topic.

Parameters

topic	MQTT topic string.
message	Payload to publish.

Returns

true if publish was accepted; false otherwise.

Parameters

topic	The MQTT topic to publish to.
msg	The payload string.

Returns

true if the message was successfully handed off to the network, otherwise returns false.

Definition at line 83 of file mqtt_client.cpp.

```
00083                                                       {
00084   bool ok = mqttClient.publish(topic, message.c_str());
00085   if (!ok) {
00086     Serial.print("Publish failed to topic: ");
00087     Serial.println(topic);
00088   }
00089   return ok;
00090 }
```

4.5.2.4 sendHeartbeat()

```
void sendHeartbeat ()
```

Register the handler for incoming MQTT messages.

Sends a periodic "alive" signal to the status/heartbeat topic.

Parameters

handler	Function invoked when message is received.

Returns

void

void

Definition at line 98 of file mqtt_client.cpp.

```
00098                     {
00099   static unsigned long lastPing = 0;
00100   if (millis() - lastPing >= 1000) {
00101     publishStatus(MQTT_TOPIC_STATUS_HEARTBEAT, "alive");
00102     lastPing = millis();
00103   }
00104 }
```

4.5.2.5 setMessageHandler()

```
void setMessageHandler (
            void(* handler )(const String &msg))
```

Registers a callback to handle incoming parsed MQTT messages.

To set user defined callback to handle parsed MQTT messages.

Parameters

handler	Function to call with message string.

Returns

void

Parameters

handler	Function to call when a new message arrives.

Note

This function should be called after initMQTT() to set the callback

Definition at line 112 of file mqtt_client.cpp.

```
00112                                                    {
00113   incomingMessageHandler = handler;
00114 }
```

4.5.3 Variable Documentation

4.5.3.1 MQTT_TOPIC_CALIBRATE

```
char MQTT_TOPIC_CALIBRATE[] = "leafy_automation/calibrate"  [inline], [constexpr]
```

Definition at line 22 of file mqtt_client.h.

4.5.3.2 MQTT_TOPIC_GRIPPER

```
char MQTT_TOPIC_GRIPPER[] = "leafy_automation/gripper"  [inline], [constexpr]
```

Definition at line 21 of file mqtt_client.h.

4.5.3.3 MQTT_TOPIC_MOTION

```
char MQTT_TOPIC_MOTION[] = "leafy_automation/motion"  [inline], [constexpr]
```

Definition at line 20 of file mqtt_client.h.

4.5.3.4 MQTT_TOPIC_STATUS_CALIBRATION

```
char MQTT_TOPIC_STATUS_CALIBRATION[]  [inline], [constexpr]
```

Initial value:

```
=
    "leafy_automation/status/calibration"
```

Definition at line 31 of file mqtt_client.h.

4.5.3.5 MQTT_TOPIC_STATUS_COMMAND

```
char MQTT_TOPIC_STATUS_COMMAND[]  [inline], [constexpr]
```

Initial value:

```
=
    "leafy_automation/status/command_received"
```

Definition at line 25 of file mqtt_client.h.

4.5.3.6 MQTT_TOPIC_STATUS_GRIPPER

```
char MQTT_TOPIC_STATUS_GRIPPER[]  [inline], [constexpr]
```

Initial value:

```
=
    "leafy_automation/status/gripper"
```

Definition at line 29 of file mqtt_client.h.

4.5.3.7 MQTT_TOPIC_STATUS_HEARTBEAT

```
char MQTT_TOPIC_STATUS_HEARTBEAT[]  [inline], [constexpr]
```

Initial value:

```
=
    "leafy_automation/status/heartbeat"
```

Definition at line 33 of file mqtt_client.h.

4.5.3.8 MQTT_TOPIC_STATUS_MOTION

```
char MQTT_TOPIC_STATUS_MOTION[]  [inline], [constexpr]
```

Initial value:

```
=
    "leafy_automation/status/motion"
```

Definition at line 27 of file mqtt_client.h.

Chapter 5

Class Documentation

5.1 API Class Reference

High-level API abstraction for interacting with the Leafy Automation Central.

`#include <api.h>`

Public Member Functions

- API (String access_token)
  Constructs an API object.
- bool ping ()
  Pings the Leafy Automation Central. Used to check if the server is online.

Static Public Attributes

- static const String access_token
  The access token for the Leafy Automation Central.

Private Attributes

- String auth_token = LA_SERVER_TOKEN

5.1.1 Detailed Description

High-level API abstraction for interacting with the Leafy Automation Central.

Definition at line 10 of file api.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 API()

```
API::API (
            String access_token)
```

Constructs an API object.

Parameters

access_token	The access token for the Leafy Automation Central.

Definition at line 3 of file api.cpp.

```
00003                             {
00004     this->auth_token = access_token;
00005 }
```

5.1.3 Member Function Documentation

5.1.3.1 ping()

```
bool API::ping ()
```

Pings the Leafy Automation Central. Used to check if the server is online.

This function sends a GET request to the '/' endpoint of the Leafy Automation Central, and checks for a response. Used to check if the server is online.

Returns

Whether the ping was successful.

Definition at line 7 of file api.cpp.

```
00007                 {
00008     return HTTP(LA_SERVER_ADDR)
00009         .get("/")
00010         .fetch()
00011         .text() != "";
00012 }
```

5.1.4 Member Data Documentation

5.1.4.1 access_token

```
const String API::access_token  [static]
```

The access token for the Leafy Automation Central.

Definition at line 15 of file api.h.

5.1.4.2 auth_token

```
String API::auth_token = LA_SERVER_TOKEN  [private]
```

Definition at line 34 of file api.h.

The documentation for this class was generated from the following files:

- include/common/api/api.h
- src/common/api/api.cpp

5.2 HTTP Class Reference

A simple HTTP client abstraction.

`#include <http.h>`

Public Member Functions

- HTTP (String host)

  Constructs a HTTP object.
- HTTP & header (String key, String value)

  Adds a header to the HTTP request.
- HTTP & get (String path)

  Performs an HTTP GET request.
- HTTP & post (String path, uint8_t *data, size_t data_length)

  Performs an HTTP POST request.
- HTTP & fetch ()

  Ends the HTTP request and returns the response.
- String text ()

  Returns the response as plaintext.
- JsonDocument json ()

  Returns the response as a JSON document.

Private Attributes

- String host
- String response
- bool connected = false

Static Private Attributes

- static WiFiClient client

5.2.1 Detailed Description

A simple HTTP client abstraction.

This class provides basic HTTP functionalities such as GET and POST requests using a provided NetCommander instance for network communication.

https://datatracker.ietf.org/doc/html/rfc2616

Definition at line 16 of file http.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 HTTP()

```
HTTP::HTTP (
            String host)
```

Constructs a HTTP object.

Definition at line 5 of file http.cpp.

```
00005                     {
00006     this->host = host;
00007     this->connected = client.connect(host.c_str(), LA_SERVER_PORT);
00008
00009     if (!this->connected) {
00010         Serial.println("HTTP request failed");
00011     }
00012 }
```

5.2.3 Member Function Documentation

5.2.3.1 fetch()

```
HTTP & HTTP::fetch ()
```

Ends the HTTP request and returns the response.

Returns

The HTTP response as a String.

Definition at line 54 of file http.cpp.

```
00054                     {
00055     if (!this->connected) {
00056         return *this;
00057     }
00058
00059     while (client.connected()) {
00060         if (client.available()) {
00061             this->response += (char) client.read();
00062         }
00063     }
00064
00065     client.stop();
00066
00067     return *this;
00068 }
```

5.2.3.2 get()

```
HTTP & HTTP::get (
            String path)
```

Performs an HTTP GET request.

Parameters

host	The hostname or IP address to connect to.
path	The resource path to request.

Returns

The HTTP object instance.

Definition at line 24 of file http.cpp.

```
00024                                {
00025     if (!this->connected) {
00026         return *this;
00027     }
00028
00029     HTTP::client.println("GET " + path + " HTTP/1.1");
00030     HTTP::client.println("Host: " + this->host);
00031     HTTP::client.println("Connection: close");
00032     client.println(); // Double crlf (carriage return line feed) to end the request.
00033
00034     return *this;
00035 }
```

5.2.3.3 header()

```
HTTP & HTTP::header (
            String key,
            String value)
```

Adds a header to the HTTP request.

Parameters

key	The header key.
value	The header value.

Returns

The HTTP object instance.

Definition at line 14 of file http.cpp.

```
00014                                                {
00015     if (!this->connected) {
00016         return *this;
00017     }
00018
00019     HTTP::client.println(key + ": " + value);
00020
00021     return *this;
00022 }
```

5.2.3.4 json()

```
JsonDocument HTTP::json ()
```

Returns the response as a JSON document.

Returns

The JSON document.

Definition at line 74 of file http.cpp.

```
00074                           {
00075     JsonDocument doc;
00076
00077     DeserializationError error = deserializeJson(doc, this->response);
00078
00079     if (error) {
00080         Serial.print("deserializeJson() failed: ");
00081         Serial.println(error.c_str());
00082
00083         doc.clear();
00084     }
00085
00086     return doc;
00087 }
```

5.2.3.5 post()

```
HTTP & HTTP::post (
            String path,
            uint8_t * data,
            size_t data_length)
```

Performs an HTTP POST request.

Parameters

host	The hostname or IP address to connect to.
data	The data to send.
data_length	Length of the data to send.

Returns

The HTTP object instance.

Definition at line 37 of file http.cpp.

```
00037                                                                   {
00038     if (!this->connected) {
00039         return *this;
00040     }
00041
00042     HTTP::client.println("POST " + path + " HTTP/1.1");
00043     HTTP::client.println("Host: " + this->host);
00044     HTTP::client.println("Content-Type: application/octet-stream");
00045     HTTP::client.println("Content-Length: " + String(data_length));
00046     HTTP::client.println("Connection: keep-alive");
00047     client.println(); // Double crlf (carriage return line feed) to end the request.
00048
00049     HTTP::client.write(data, data_length);
00050
00051     return *this;
00052 }
```

5.2.3.6 text()

```
String HTTP::text ()
```

Returns the response as plaintext.

Returns

The response as a String.

Definition at line 70 of file http.cpp.

```
00070                   {
00071     return this->response;
00072 }
```

5.2.4 Member Data Documentation

5.2.4.1 client

```
WiFiClient HTTP::client  [static], [private]
```

Definition at line 76 of file http.h.

5.2.4.2 connected

```
bool HTTP::connected = false  [private]
```

Definition at line 74 of file http.h.

5.2.4.3 host

```
String HTTP::host  [private]
```

Definition at line 72 of file http.h.

5.2.4.4 response

```
String HTTP::response  [private]
```

Definition at line 73 of file http.h.

The documentation for this class was generated from the following files:

- include/common/net/http.h
- src/common/net/http.cpp

5.3 NetCommander Class Reference

Provides a simple interface for connecting to the internet.

```
#include <net_commander.h>
```

Public Member Functions

- void connect (String ssid, String password)
  Connects to a Wi-Fi network.
- void disconnect ()
  Disconnects from the Wi-Fi network.

5.3.1 Detailed Description

Provides a simple interface for connecting to the internet.

NetCommander wraps the WiFiS3 library to simplify Wi-Fi connection handling. It manages network credentials and provides a `WiFiClient` instance for communication.

Definition at line 20 of file net_commander.h.

5.3.2 Member Function Documentation

5.3.2.1 connect()

```
void NetCommander::connect (
            String ssid,
            String password)
```

Connects to a Wi-Fi network.

Parameters

ssid	The SSID (name) of the Wi-Fi network.
password	The password for the Wi-Fi network.

Definition at line 3 of file net_commander.cpp.

```
00003                                                                   {
00004     Serial.println("Connecting to WiFi...");
00005
00006     WiFi.begin(ssid.c_str(), password.c_str());
00007
00008     while (WiFi.status() != WL_CONNECTED) {
00009         delay(1000);
00010         Serial.println(".");
00011     }
00012
00013     Serial.println("Connected to WiFi!");
00014 }
```

5.3.2.2 disconnect()

```
void NetCommander::disconnect ()
```

Disconnects from the Wi-Fi network.

Definition at line 16 of file net_commander.cpp.

```
00016                                {
00017     WiFi.disconnect();
00018     Serial.println("Disconnected from WiFi.");
00019 }
```

The documentation for this class was generated from the following files:

- include/common/net/net_commander.h
- src/common/net/net_commander.cpp

Chapter 6

File Documentation

6.1 include/common/api/api.h File Reference

```
#include "common/secrets.h"
#include "common/net/http.h"
```

Classes

- class API

  High-level API abstraction for interacting with the Leafy Automation Central.

6.2 api.h

Go to the documentation of this file.

```
00001 /**
00002  * @brief High-level API abstraction for interacting with the Leafy Automation Central.
00003  */
00004
00005 #pragma once
00006
00007 #include "common/secrets.h"
00008 #include "common/net/http.h"
00009
00010 class API {
00011     public:
00012         /**
00013          * @brief The access token for the Leafy Automation Central.
00014          */
00015         static const String access_token;
00016
00017         /**
00018          * @brief Constructs an API object.
00019          *
00020          * @param access_token The access token for the Leafy Automation Central.
00021          */
00022         API(String access_token);
00023
00024         /**
00025          * @brief Pings the Leafy Automation Central. Used to check if the server is online.
00026          *
00027          * @details This function sends a GET request to the '/' endpoint of the Leafy Automation
      Central,
00028          * and checks for a response. Used to check if the server is online.
00029          *
00030          * @return Whether the ping was successful.
00031          */
00032         bool ping();
00033     private:
00034         String auth_token = LA_SERVER_TOKEN;
00035 };
```

6.3 include/common/net/http.h File Reference

```
#include <ArduinoJson.h>
#include "common/net/net_commander.h"
```

Classes

- class HTTP

  A simple HTTP client abstraction.

6.4 http.h

Go to the documentation of this file.

```
00001 /**
00002  * @brief A simple HTTP client abstraction.
00003  *
00004  * @details This class provides basic HTTP functionalities such as GET and POST requests
00005  * using a provided NetCommander instance for network communication.
00006  *
00007  * https://datatracker.ietf.org/doc/html/rfc2616
00008  */
00009
00010 #pragma once
00011
00012 #include <ArduinoJson.h>
00013
00014 #include "common/net/net_commander.h"
00015
00016 class HTTP {
00017     public:
00018         /**
00019          * @brief Constructs a HTTP object.
00020          */
00021         HTTP(String host);
00022
00023         /**
00024          * @brief Adds a header to the HTTP request.
00025          *
00026          * @param key The header key.
00027          * @param value The header value.
00028          * @return The HTTP object instance.
00029          */
00030         HTTP& header(String key, String value);
00031
00032         /**
00033          * @brief Performs an HTTP GET request.
00034          *
00035          * @param host The hostname or IP address to connect to.
00036          * @param path The resource path to request.
00037          * @return The HTTP object instance.
00038          */
00039         HTTP& get(String path);
00040
00041         /**
00042          * @brief Performs an HTTP POST request.
00043          *
00044          * @param host The hostname or IP address to connect to.
00045          * @param data The data to send.
00046          * @param data_length Length of the data to send.
00047          * @return The HTTP object instance.
00048          */
00049         HTTP& post(String path, uint8_t* data, size_t data_length);
00050
00051         /**
00052          * @brief Ends the HTTP request and returns the response.
00053          *
00054          * @return The HTTP response as a String.
00055          */
00056         HTTP& fetch();
00057
00058         /**
00059          * @brief Returns the response as plaintext.
```

```
00060          *
00061          * @return The response as a String.
00062          */
00063         String text();
00064
00065         /**
00066          * @brief Returns the response as a JSON document.
00067          *
00068          * @return The JSON document.
00069          */
00070         JsonDocument json();
00071     private:
00072         String host;
00073         String response;
00074         bool connected = false;
00075
00076         static WiFiClient client;
00077 };
```

6.5 include/common/net/net_commander.h File Reference

```
#include <Arduino.h>
#include "common/secrets.h"
```

Classes

- class NetCommander

  Provides a simple interface for connecting to the internet.

6.6 net_commander.h

Go to the documentation of this file.

```
00001 /**
00002  * @brief Provides a simple interface for connecting to the internet.
00003  *
00004  * @details NetCommander wraps the WiFiS3 library to simplify Wi-Fi connection handling.
00005  * It manages network credentials and provides a `WiFiClient` instance for communication.
00006  */
00007
00008 #pragma once
00009
00010 #include <Arduino.h>
00011
00012 #ifdef PLATFORMIO_ENV_UNO_R4_WIFI
00013 #include "WiFiS3.h"
00014 #elif PLATFORMIO_ENV_ESP32CAM
00015 #include <WiFi.h>
00016 #endif
00017
00018 #include "common/secrets.h"
00019
00020 class NetCommander {
00021     public:
00022         /**
00023          * @brief Connects to a Wi-Fi network.
00024          *
00025          * @param ssid The SSID (name) of the Wi-Fi network.
00026          * @param password The password for the Wi-Fi network.
00027          */
00028         void connect(String ssid, String password);
00029
00030         /**
00031          * @brief Disconnects from the Wi-Fi network.
00032          */
00033         void disconnect();
00034 };
```

6.7 include/common/secrets.h File Reference

Network and MQTT credentials for Leafy Automation firmware.

Macros

- #define WIFI_SSID "your-ssid"
- #define WIFI_PASSWORD "your-password"
- #define MQTT_SERVER "your-broker-ip"
- #define MQTT_PORT 1883
- #define MQTT_CLIENT_ID "LeafyAutomationClient"

6.7.1 Detailed Description

Network and MQTT credentials for Leafy Automation firmware.

Definition in file secrets.h.

6.7.2 Macro Definition Documentation

6.7.2.1 MQTT_CLIENT_ID

```
#define MQTT_CLIENT_ID "LeafyAutomationClient"
```

Definition at line 17 of file secrets.h.

6.7.2.2 MQTT_PORT

```
#define MQTT_PORT 1883
```

Definition at line 16 of file secrets.h.

6.7.2.3 MQTT_SERVER

```
#define MQTT_SERVER "your-broker-ip"
```

Definition at line 15 of file secrets.h.

6.7.2.4 WIFI_PASSWORD

```
#define WIFI_PASSWORD "your-password"
```

Definition at line 12 of file secrets.h.

6.7.2.5 WIFI_SSID

```
#define WIFI_SSID "your-ssid"
```

Definition at line 11 of file secrets.h.

6.8 secrets.h

Go to the documentation of this file.

```
00001 /**
00002  * @file secrets.h
00003  * @brief Network and MQTT credentials for Leafy Automation firmware.
00004  *
00005  */
00006
00007  #ifndef SECRETS_H
00008  #define SECRETS_H
00009
00010  // WiFi credentials
00011  #define WIFI_SSID         "your-ssid"
00012  #define WIFI_PASSWORD     "your-password"
00013
00014  // MQTT broker settings
00015  #define MQTT_SERVER       "your-broker-ip"
00016  #define MQTT_PORT         1883
00017  #define MQTT_CLIENT_ID    "LeafyAutomationClient"
00018  // Not sure if this will be required yet, keeping for now
00019  // #define MQTT_USERNAME   "your-mqtt-username"
00020  // #define MQTT_PASSWORD   "your-mqtt-password"
00021
00022  #endif // SECRETS_H
00023
```

6.9 include/common/secrets.sample.h File Reference

Macros

- #define WIFI_SSID "your-ssid"
- #define WIFI_PASSWORD "your-password"
- #define LA_SERVER_ADDR "ip-addr"
- #define LA_SERVER_PORT 5000
- #define LA_SERVER_TOKEN "your-token"

6.9.1 Macro Definition Documentation

6.9.1.1 LA_SERVER_ADDR

```
#define LA_SERVER_ADDR "ip-addr"
```

Definition at line 9 of file secrets.sample.h.

6.9.1.2 LA_SERVER_PORT

```
#define LA_SERVER_PORT 5000
```

Definition at line 10 of file secrets.sample.h.

6.9.1.3 LA_SERVER_TOKEN

```
#define LA_SERVER_TOKEN "your-token"
```

Definition at line 11 of file secrets.sample.h.

6.9.1.4 WIFI_PASSWORD

```
#define WIFI_PASSWORD "your-password"
```

Definition at line 8 of file secrets.sample.h.

6.9.1.5 WIFI_SSID

```
#define WIFI_SSID "your-ssid"
```

Copy this file to secrets.h and fill in the values

Definition at line 7 of file secrets.sample.h.

6.10 secrets.sample.h

Go to the documentation of this file.

```
00001 /**
00002  * Copy this file to secrets.h and fill in the values
00003  */
00004
00005 #pragma once
00006
00007 #define WIFI_SSID "your-ssid"
00008 #define WIFI_PASSWORD "your-password"
00009 #define LA_SERVER_ADDR "ip-addr"
00010 #define LA_SERVER_PORT 5000
00011 #define LA_SERVER_TOKEN "your-token"
```

6.11 include/common/util/logger.h File Reference

```
#include "Arduino.h"
#include "common/net/http.h"
```

Functions

- void logger_print_line (String msg)
  Simple logger which outputs to a REST endpoint.

6.11.1 Function Documentation

6.11.1.1 logger_print_line()

```
void logger_print_line (
             String msg)
```

Simple logger which outputs to a REST endpoint.

Parameters

msg	The message to log.

Definition at line 11 of file logger.h.

```
00011                                       {
00012     String res = HTTP(LA_SERVER_ADDR)
00013         .get("/api/v1/log?msg=" + msg)
00014         .fetch()
00015         .text();
00016 }
```

6.12 logger.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Arduino.h"
00004
00005 #include "common/net/http.h"
00006
00007 /**
00008  * @brief Simple logger which outputs to a REST endpoint.
00009  * @param msg The message to log.
00010  */
00011 void logger_print_line(String msg) {
00012     String res = HTTP(LA_SERVER_ADDR)
00013         .get("/api/v1/log?msg=" + msg)
00014         .fetch()
00015         .text();
00016 }
```

6.13 include/config.h File Reference

Project-wide configuration constants (pins, timings, ratios).

```
#include <Arduino.h>
```

Variables

- static constexpr uint8_t ARM_JOINTS = 5

  Number of stepper-driven joints (J0...J4).
- static constexpr uint8_t STEP_PIN [ARM_JOINTS]

  STEP pin mapping for joints J0...J4.
- static constexpr uint8_t DIR_PIN [ARM_JOINTS]

  DIR pin mapping for joints J0...J4.
- static constexpr uint16_t MICROSTEPS = 200

  Microsteps per full revolution.
- static constexpr float GEAR_RATIO [ARM_JOINTS]

  Gear ratio for each joint.
- static constexpr float MAX_OUTPUT_RPM [ARM_JOINTS]

  Max output RPM per joint.
- static constexpr uint8_t LIMIT_LEFT_PINS [ARM_JOINTS]

  Array of digital input pins connected to each joint's left limit switch (Active LOW)

- static constexpr uint8_t LIMIT_RIGHT_PINS [ARM_JOINTS]

  Array of digital input pins connected to each joint's right limit switch (Active LOW)
- static constexpr uint8_t GRIPPER_SERVO_PIN

  PWM pin for the servo controlling the gripper (End Effector, EF).
- static constexpr unsigned long GRIP_MOVE_TIME_MS = 500

  Allocated time in milliseconds for the gripper to open/close.
- static constexpr uint8_t GRIPPER_CLOSED_ANGLE = 0

  Open closed (degrees) for the gripper servo.
- static constexpr uint8_t GRIPPER_OPEN_ANGLE = 90

  Open angle (degrees) for the gripper servo.

6.13.1 Detailed Description

Project-wide configuration constants (pins, timings, ratios).

Author

Elin Gravningen

Central place for hardware mapping and motion parameters that may vary as the project develops (such as adapting gripping according to plant type).

Definition in file config.h.

6.14 config.h

Go to the documentation of this file.

```
00001 // config.h
00002 #ifndef CONFIG_H
00003 #define CONFIG_H
00004
00005 #include <Arduino.h>
00006
00007 /**
00008  * @file config.h
00009  * @author Elin Gravningen
00010  * @brief Project-wide configuration constants (pins, timings, ratios).
00011  * @details Central place for hardware mapping and motion parameters that
00012  *  may vary as the project develops (such as adapting gripping according to
00013  * plant type).
00014  *
00015  * @defgroup Configuration Configuration
00016  * @{
00017  */
00018
00019 /// @var ARM_JOINTS
00020 /// Number of stepper-driven joints (J0...J4).
00021 static constexpr uint8_t ARM_JOINTS = 5;
00022
00023 /// @var STEP_PIN
00024 /// STEP pin mapping for joints J0...J4.
00025 /// @pre STEP_PIN size must equal ARM_JOINTS
00026 static constexpr uint8_t STEP_PIN[ARM_JOINTS] = {
00027     /* J0 */ 1, // TO DO: Replace 1s with actual pin numbers
00028     /* J1 */ 1,
00029     /* J2 */ 1,
00030     /* J3 */ 1,
00031     /* J4 */ 1};
00032
00033 /// @var DIR_PIN
00034 /// DIR pin mapping for joints J0...J4.
00035 static constexpr uint8_t DIR_PIN[ARM_JOINTS] = {
00036     /* J0 */ 1, // TO DO: Replace 1s with actual pin numbers
00037     /* J1 */ 1,
```

```
00038     /* J2 */ 1,
00039     /* J3 */ 1,
00040     /* J4 */ 1};
00041
00042 /// @var MICROSTEPS
00043 /// Microsteps per full revolution
00044 static constexpr uint16_t MICROSTEPS = 200;
00045
00046 /// @var GEAR_RATIO
00047 /// Gear ratio for each joint
00048 static constexpr float GEAR_RATIO[ARM_JOINTS] = {
00049     /* J0 */ 1.0f / 10.0f,
00050     /* J1 */ 1.0f / 50.0f,
00051     /* J2 */ 1.0f / 50.0f,
00052     /* J3 */ 1.0f / 19.0f,
00053     /* J4 */ 1.0f / 16.0f};
00054
00055 /// @var MAX_OUTPUT_RPM
00056 /// Max output RPM per joint
00057 static constexpr float MAX_OUTPUT_RPM[ARM_JOINTS] = {
00058     /* J0 */ 30.0f,
00059     /* J1 */ 20.0f,
00060     /* J2 */ 20.0f,
00061     /* J3 */ 25.0f,
00062     /* J4 */ 25.0f};
00063
00064 /// @var LIMIT_LEFT_PINS
00065 /// Array of digital input pins connected to each joint's left limit switch
00066 /// (Active LOW)
00067 /// @ingroup Configuration
00068 static constexpr uint8_t LIMIT_LEFT_PINS[ARM_JOINTS] = {
00069     xx, xx, xx, xx, xx}; // Replace with pin numbers
00070
00071 /// @var LIMIT_RIGHT_PINS
00072 /// Array of digital input pins connected to each joint's right limit switch
00073 /// (Active LOW)
00074 /// @ingroup Configuration
00075 static constexpr uint8_t LIMIT_RIGHT_PINS[ARM_JOINTS] = {
00076     xx, xx, xx, xx, xx}; // Replace with pin numbers
00077
00078 /// @var GRIPPER_SERVO_PIN
00079 /// PWM pin for the servo controlling the gripper (End Effector, EF).
00080 static constexpr uint8_t GRIPPER_SERVO_PIN =
00081     1; // TO DO: Replace 1s with actual pin numbers
00082
00083 /// @var GRIP_MOVE_TIME_MS
00084 /// Allocated time in milliseconds for the gripper to open/close.
00085 static constexpr unsigned long GRIP_MOVE_TIME_MS = 500;
00086
00087 /// @var GRIPPER_CLOSED_ANGLE
00088 /// Open closed (degrees) for the gripper servo.
00089 static constexpr uint8_t GRIPPER_CLOSED_ANGLE = 0;
00090
00091 /// @var GRIPPER_OPEN_ANGLE
00092 /// Open angle (degrees) for the gripper servo.
00093 static constexpr uint8_t GRIPPER_OPEN_ANGLE = 90;
00094
00095 #endif // CONFIG_H
00096 /** @} */ // end of Configuration
```

6.15 include/modules/base/main_base.h File Reference

Functions

- void main_base_setup ()

  Defines the code paths for the Arduino (base system).
- void main_base_loop ()

6.15.1 Function Documentation

6.15.1.1 main_base_loop()

```
void main_base_loop ()
```

6.15.1.2 main_base_setup()

```
void main_base_setup ()
```

Defines the code paths for the Arduino (base system).

6.16 main_base.h

Go to the documentation of this file.

```
00001 /**
00002  * @brief Defines the code paths for the Arduino (base system).
00003  */
00004
00005 #pragma once
00006
00007 void main_base_setup();
00008
00009 void main_base_loop();
```

6.17 include/modules/cam/esp32-cam-gpio.h File Reference

This file is based on the esp32-cam CameraWebServer example from the arduino-esp32 repository. https://github.com/espressif/arduino-esp32.

```
#include "esp_camera.h"
```

Macros

- #define PWDN_GPIO_NUM 32
- #define RESET_GPIO_NUM -1
- #define XCLK_GPIO_NUM 0
- #define SIOD_GPIO_NUM 26
- #define SIOC_GPIO_NUM 27
- #define Y9_GPIO_NUM 35
- #define Y8_GPIO_NUM 34
- #define Y7_GPIO_NUM 39
- #define Y6_GPIO_NUM 36
- #define Y5_GPIO_NUM 21
- #define Y4_GPIO_NUM 19
- #define Y3_GPIO_NUM 18
- #define Y2_GPIO_NUM 5
- #define VSYNC_GPIO_NUM 25
- #define HREF_GPIO_NUM 23
- #define PCLK_GPIO_NUM 22
- #define LED_GPIO_NUM 4

Functions

- camera_config_t setupCameraConfig ()

6.17.1 Detailed Description

This file is based on the esp32-cam CameraWebServer example from the arduino-esp32 repository. https://github.com/espressif/arduino-esp32.

Definition in file esp32-cam-gpio.h.

6.17.2 Macro Definition Documentation

6.17.2.1 HREF_GPIO_NUM

```
#define HREF_GPIO_NUM 23
```

Definition at line 42 of file esp32-cam-gpio.h.

6.17.2.2 LED_GPIO_NUM

```
#define LED_GPIO_NUM 4
```

Definition at line 46 of file esp32-cam-gpio.h.

6.17.2.3 PCLK_GPIO_NUM

```
#define PCLK_GPIO_NUM 22
```

Definition at line 43 of file esp32-cam-gpio.h.

6.17.2.4 PWDN_GPIO_NUM

```
#define PWDN_GPIO_NUM 32
```

Definition at line 27 of file esp32-cam-gpio.h.

6.17.2.5 RESET_GPIO_NUM

```
#define RESET_GPIO_NUM -1
```

Definition at line 28 of file esp32-cam-gpio.h.

6.17.2.6 SIOC_GPIO_NUM

```
#define SIOC_GPIO_NUM 27
```

Definition at line 31 of file esp32-cam-gpio.h.

6.17.2.7 SIOD_GPIO_NUM

```
#define SIOD_GPIO_NUM 26
```

Definition at line 30 of file esp32-cam-gpio.h.

6.17.2.8 VSYNC_GPIO_NUM

```
#define VSYNC_GPIO_NUM 25
```

Definition at line 41 of file esp32-cam-gpio.h.

6.17.2.9 XCLK_GPIO_NUM

```
#define XCLK_GPIO_NUM 0
```

Definition at line 29 of file esp32-cam-gpio.h.

6.17.2.10 Y2_GPIO_NUM

```
#define Y2_GPIO_NUM 5
```

Definition at line 40 of file esp32-cam-gpio.h.

6.17.2.11 Y3_GPIO_NUM

```
#define Y3_GPIO_NUM 18
```

Definition at line 39 of file esp32-cam-gpio.h.

6.17.2.12 Y4_GPIO_NUM

```
#define Y4_GPIO_NUM 19
```

Definition at line 38 of file esp32-cam-gpio.h.

6.17.2.13 Y5_GPIO_NUM

```
#define Y5_GPIO_NUM 21
```

Definition at line 37 of file esp32-cam-gpio.h.

6.17.2.14 Y6_GPIO_NUM

```
#define Y6_GPIO_NUM 36
```

Definition at line 36 of file esp32-cam-gpio.h.

6.17.2.15 Y7_GPIO_NUM

```
#define Y7_GPIO_NUM 39
```

Definition at line 35 of file esp32-cam-gpio.h.

6.17.2.16 Y8_GPIO_NUM

```
#define Y8_GPIO_NUM 34
```

Definition at line 34 of file esp32-cam-gpio.h.

6.17.2.17 Y9_GPIO_NUM

```
#define Y9_GPIO_NUM 35
```

Definition at line 33 of file esp32-cam-gpio.h.

6.17.3 Function Documentation

6.17.3.1 setupCameraConfig()

```
camera_config_t setupCameraConfig ()
```

Definition at line 48 of file esp32-cam-gpio.h.

```
00048                                   {
00049     camera_config_t config;
00050     config.ledc_channel = LEDC_CHANNEL_0;
00051     config.ledc_timer = LEDC_TIMER_0;
00052     config.pin_d0 = Y2_GPIO_NUM;
00053     config.pin_d1 = Y3_GPIO_NUM;
00054     config.pin_d2 = Y4_GPIO_NUM;
00055     config.pin_d3 = Y5_GPIO_NUM;
00056     config.pin_d4 = Y6_GPIO_NUM;
00057     config.pin_d5 = Y7_GPIO_NUM;
00058     config.pin_d6 = Y8_GPIO_NUM;
00059     config.pin_d7 = Y9_GPIO_NUM;
00060     config.pin_xclk = XCLK_GPIO_NUM;
00061     config.pin_pclk = PCLK_GPIO_NUM;
00062     config.pin_vsync = VSYNC_GPIO_NUM;
00063     config.pin_href = HREF_GPIO_NUM;
00064     config.pin_sccb_sda = SIOD_GPIO_NUM;
00065     config.pin_sccb_scl = SIOC_GPIO_NUM;
00066     config.pin_pwdn = PWDN_GPIO_NUM;
00067     config.pin_reset = RESET_GPIO_NUM;
00068     config.xclk_freq_hz = 20000000;
00069     config.frame_size = FRAMESIZE_VGA;
00070     config.pixel_format = PIXFORMAT_JPEG;
00071     config.grab_mode = CAMERA_GRAB_LATEST;
00072     config.fb_location = CAMERA_FB_IN_PSRAM;
00073     config.jpeg_quality = 10;
00074     config.fb_count = 2;
00075
00076     return config;
00077 }
```

6.18 esp32-cam-gpio.h

Go to the documentation of this file.

```
00001 /**
00002  * @file
00003  * @brief This file is based on the esp32-cam CameraWebServer example from the arduino-esp32
      repository.
00004  * https://github.com/espressif/arduino-esp32
00005
00006  * Copyright Espressif Systems (Shanghai) PTE LTD
00007
00008  * This library is free software; you can redistribute it and/or
00009  * modify it under the terms of the GNU Lesser General Public
00010  * License as published by the Free Software Foundation; either
00011  * version 2.1 of the License, or (at your option) any later version.
00012
00013  * This library is distributed in the hope that it will be useful,
00014  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00015  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00016  * Lesser General Public License for more details.
00017
00018  * You should have received a copy of the GNU Lesser General Public
00019  * License along with this library; if not, write to the Free Software
00020  * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
00021  */
00022
00023 #pragma once
00024
00025 #include "esp_camera.h"
00026
00027 #define PWDN_GPIO_NUM  32
00028 #define RESET_GPIO_NUM -1
00029 #define XCLK_GPIO_NUM  0
00030 #define SIOD_GPIO_NUM  26
00031 #define SIOC_GPIO_NUM  27
00032
00033 #define Y9_GPIO_NUM    35
00034 #define Y8_GPIO_NUM    34
00035 #define Y7_GPIO_NUM    39
00036 #define Y6_GPIO_NUM    36
00037 #define Y5_GPIO_NUM    21
00038 #define Y4_GPIO_NUM    19
00039 #define Y3_GPIO_NUM    18
00040 #define Y2_GPIO_NUM    5
00041 #define VSYNC_GPIO_NUM 25
00042 #define HREF_GPIO_NUM  23
00043 #define PCLK_GPIO_NUM  22
00044
00045 // 4 for flash led or 33 for normal led
00046 #define LED_GPIO_NUM   4
00047
00048 camera_config_t setupCameraConfig() {
00049     camera_config_t config;
00050     config.ledc_channel = LEDC_CHANNEL_0;
00051     config.ledc_timer = LEDC_TIMER_0;
00052     config.pin_d0 = Y2_GPIO_NUM;
00053     config.pin_d1 = Y3_GPIO_NUM;
00054     config.pin_d2 = Y4_GPIO_NUM;
00055     config.pin_d3 = Y5_GPIO_NUM;
00056     config.pin_d4 = Y6_GPIO_NUM;
00057     config.pin_d5 = Y7_GPIO_NUM;
00058     config.pin_d6 = Y8_GPIO_NUM;
00059     config.pin_d7 = Y9_GPIO_NUM;
00060     config.pin_xclk = XCLK_GPIO_NUM;
00061     config.pin_pclk = PCLK_GPIO_NUM;
00062     config.pin_vsync = VSYNC_GPIO_NUM;
00063     config.pin_href = HREF_GPIO_NUM;
00064     config.pin_sccb_sda = SIOD_GPIO_NUM;
00065     config.pin_sccb_scl = SIOC_GPIO_NUM;
00066     config.pin_pwdn = PWDN_GPIO_NUM;
00067     config.pin_reset = RESET_GPIO_NUM;
00068     config.xclk_freq_hz = 20000000;
00069     config.frame_size = FRAMESIZE_VGA;
00070     config.pixel_format = PIXFORMAT_JPEG;
00071     config.grab_mode = CAMERA_GRAB_LATEST;
00072     config.fb_location = CAMERA_FB_IN_PSRAM;
00073     config.jpeg_quality = 10;
00074     config.fb_count = 2;
00075
00076     return config;
00077 }
```

6.19 include/modules/cam/main_cam.h File Reference

Functions

- void main_cam_setup ()

  Defines the code paths for the Cam (esp32-cam).
- void main_cam_loop ()

6.19.1 Function Documentation

6.19.1.1 main_cam_loop()

```
void main_cam_loop ()
```

6.19.1.2 main_cam_setup()

```
void main_cam_setup ()
```

Defines the code paths for the Cam (esp32-cam).

6.20 main_cam.h

Go to the documentation of this file.

```
00001 /**
00002  * @brief Defines the code paths for the Cam (esp32-cam).
00003  */
00004
00005 #pragma once
00006
00007 void main_cam_setup();
00008
00009 void main_cam_loop();
```

6.21 include/Utilities.h File Reference

6.22 Utilities.h

Go to the documentation of this file.

```
00001
```

6.23 src/base/main_base.cpp File Reference

```
#include "modules/base/main_base.h"
```

6.24 main_base.cpp

Go to the documentation of this file.

```
00001 #include "modules/base/main_base.h"
00002
00003 #ifdef PLATFORMIO_ENV_UNO_R4_WIFI
00004 #include "common/net/net_commander.h"
00005 #include "common/net/http.h"
00006 #include "common/api/api.h"
00007
00008 NetCommander netCommander;
00009 API api(LA_SERVER_TOKEN);
00010
00011 void main_base_setup() {
00012     Serial.begin(9600);
00013
00014     netCommander.connect(WIFI_SSID, WIFI_PASSWORD);
00015
00016     String res = HTTP(LA_SERVER_ADDR)
00017         .get("/api/v1")
00018         .fetch()
00019         .text();
00020
00021     Serial.println(res);
00022
00023     netCommander.disconnect();
00024 }
00025
00026 void main_base_loop() {
00027
00028 }
00029
00030 #endif
```

6.25 src/common/api/api.cpp File Reference

```
#include "common/api/api.h"
```

6.26 api.cpp

Go to the documentation of this file.

```
00001 #include "common/api/api.h"
00002
00003 API::API(String access_token) {
00004     this->auth_token = access_token;
00005 }
00006
00007 bool API::ping() {
00008     return HTTP(LA_SERVER_ADDR)
00009         .get("/")
00010         .fetch()
00011         .text() != "";
00012 }
```

6.27 src/common/net/http.cpp File Reference

```
#include "common/net/http.h"
```

6.28 http.cpp

Go to the documentation of this file.

```
00001 #include "common/net/http.h"
00002
00003 WiFiClient HTTP::client;
00004
00005 HTTP::HTTP(String host) {
00006     this->host = host;
00007     this->connected = client.connect(host.c_str(), LA_SERVER_PORT);
00008
00009     if (!this->connected) {
00010         Serial.println("HTTP request failed");
00011     }
00012 }
00013
00014 HTTP& HTTP::header(String key, String value) {
00015     if (!this->connected) {
00016         return *this;
00017     }
00018
00019     HTTP::client.println(key + ": " + value);
00020
00021     return *this;
00022 }
00023
00024 HTTP& HTTP::get(String path) {
00025     if (!this->connected) {
00026         return *this;
00027     }
00028
00029     HTTP::client.println("GET " + path + " HTTP/1.1");
00030     HTTP::client.println("Host: " + this->host);
00031     HTTP::client.println("Connection: close");
00032     client.println(); // Double crlf (carriage return line feed) to end the request.
00033
00034     return *this;
00035 }
00036
00037 HTTP& HTTP::post(String path, uint8_t* data, size_t data_length) {
00038     if (!this->connected) {
00039         return *this;
00040     }
00041
00042     HTTP::client.println("POST " + path + " HTTP/1.1");
00043     HTTP::client.println("Host: " + this->host);
00044     HTTP::client.println("Content-Type: application/octet-stream");
00045     HTTP::client.println("Content-Length: " + String(data_length));
00046     HTTP::client.println("Connection: keep-alive");
00047     client.println(); // Double crlf (carriage return line feed) to end the request.
00048
00049     HTTP::client.write(data, data_length);
00050
00051     return *this;
00052 }
00053
00054 HTTP& HTTP::fetch() {
00055     if (!this->connected) {
00056         return *this;
00057     }
00058
00059     while (client.connected()) {
00060         if (client.available()) {
00061             this->response += (char) client.read();
00062         }
00063     }
00064
00065     client.stop();
00066
00067     return *this;
00068 }
00069
00070 String HTTP::text() {
00071     return this->response;
00072 }
00073
00074 JsonDocument HTTP::json() {
00075     JsonDocument doc;
00076
00077     DeserializationError error = deserializeJson(doc, this->response);
00078
00079     if (error) {
00080         Serial.print("deserializeJson() failed: ");
00081         Serial.println(error.c_str());
00082
```

```
00083         doc.clear();
00084     }
00085
00086     return doc;
00087 }
```

6.29 src/common/net/net_commander.cpp File Reference

```
#include "common/net/net_commander.h"
```

6.30 net_commander.cpp

Go to the documentation of this file.

```
00001 #include "common/net/net_commander.h"
00002
00003 void NetCommander::connect(String ssid, String password) {
00004     Serial.println("Connecting to WiFi...");
00005
00006     WiFi.begin(ssid.c_str(), password.c_str());
00007
00008     while (WiFi.status() != WL_CONNECTED) {
00009         delay(1000);
00010         Serial.println(".");
00011     }
00012
00013     Serial.println("Connected to WiFi!");
00014 }
00015
00016 void NetCommander::disconnect() {
00017     WiFi.disconnect();
00018     Serial.println("Disconnected from WiFi.");
00019 }
```

6.31 src/communication_manager/communication_manager.cpp File Reference

Implementation of the Communication Manager module.

```
#include "communication_manager.h"
#include "MQTT_client.h"
#include "gripper_driver.h"
#include "motor_driver.h"
```

Functions

- static void handleMoveCommand (const String &incCommand)

  Handle a MOVE command by parsing it into 5 values which are then converted and sent to joint actuators via moveJoint(). Sets movementInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.
- static void handleGripperCommand (const String &incCommand)

  Handle a GRIP command: 0 = open, 1 = close. Sets gripperInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.
- static void handleCalibrationCommand (const String &incCommand)

  Handle a CALIBRATE command: 0 = cancel, 1 = start calibration. Sets calibrationInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.
- void handleIncomingCommand (const String &command)

  Handle an incoming command and route it to the appropriate module.
- void checkActionStatus ()

  Checks in-progress flags and publish DONE messages.

Variables

- static bool calibrationInProgress = false
- static bool movementInProgress = false
- static bool gripperInProgress = false

6.31.1 Detailed Description

Implementation of the Communication Manager module.

Author

Elin Gravningen

Parses and routes incoming MQTT command strings to the driver modules, and monitors action completion to publish DONE events.

Definition in file communication_manager.cpp.

6.31.2 Function Documentation

6.31.2.1 handleCalibrationCommand()

```
static void handleCalibrationCommand (
            const String & command)  [static]
```

Handle a CALIBRATE command: 0 = cancel, 1 = start calibration. Sets calibrationInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.

Definition at line 50 of file communication_manager.cpp.

```
00050                                                        {
00051   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00052
00053   int state = incCommand.substring(incCommand.indexOf(' ') + 1).toInt();
00054   if (state == 1) {
00055     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATE RECEIVED");
00056     calibrationInProgress = true;
00057     calibrateAllJoints();
00058     calibrationInProgress = false;
00059     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATION DONE");
00060   } else {
00061     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION,
00062                   "CALIBRATE CANCELED"); // @todo: Should we implement
00063                                          // calibration cancellation?
00064     calibrationInProgress = false;
00065   }
00066 }
```

6.31.2.2 handleGripperCommand()

```
static void handleGripperCommand (
            const String & command)  [static]
```

Handle a GRIP command: 0 = open, 1 = close. Sets gripperInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.

Definition at line 99 of file communication_manager.cpp.

```
00099                                                        {
00100   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00101   int state = incCommand.substring(incCommand.indexOf(' ') + 1).toInt();
00102   moveGripper(state);
00103   gripperInProgress = true;
00104 }
```

6.31.2.3 handleMoveCommand()

```
static void handleMoveCommand (
            const String & command)  [static]
```

Handle a MOVE command by parsing it into 5 values which are then converted and sent to joint actuators via moveJoint(). Sets movementInProgress = true to indicate motion has started. This flag is later cleared by checkActionStatus() once all joints reach their targets.

Definition at line 72 of file communication_manager.cpp.

```
00072                                                    {
00073   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00074   int32_t jointValues[5];
00075   int index = 0;
00076   int argStart = incCommand.indexOf(' ') + 1;
00077
00078   while (index < 5) {
00079     int argEnd = incCommand.indexOf(' ', argStart);
00080     if (argEnd == -1)
00081       argEnd = incCommand.length();
00082     jointValues[index++] = incCommand.substring(argStart, argEnd).toInt();
00083     argStart = argEnd + 1;
00084   }
00085
00086   moveJoint(0, jointValues[0]); // J0
00087   moveJoint(1, jointValues[1]); // J1
00088   moveJoint(2, jointValues[2]); // J2
00089   moveJoint(3, jointValues[3]); // J3
00090   moveJoint(4, jointValues[4]); // J4
00091
00092   movementInProgress = true;
00093 }
```

6.31.3 Variable Documentation

6.31.3.1 calibrationInProgress

```
bool calibrationInProgress = false  [static]
```

Definition at line 18 of file communication_manager.cpp.

6.31.3.2 gripperInProgress

```
bool gripperInProgress = false  [static]
```

Definition at line 20 of file communication_manager.cpp.

6.31.3.3 movementInProgress

```
bool movementInProgress = false  [static]
```

Definition at line 19 of file communication_manager.cpp.

6.32 communication_manager.cpp

Go to the documentation of this file.

```
00001 /**
00002  * @file communication_manager.cpp
00003  * @author Elin Gravningen
00004  * @brief Implementation of the Communication Manager module.
00005  * @details Parses and routes incoming MQTT command strings to the driver
00006  *          modules, and monitors action completion to publish DONE events.
00007  *
00008  * @ingroup Communication_Manager
00009  */
00010
00011 #include "communication_manager.h"
00012 #include "MQTT_client.h" // for publishStatus()
00013 #include "gripper_driver.h"
00014 #include "motor_driver.h"
00015
00016 // Internal flags to track the state of calibration, movement, and gripper
00017 // actions.
00018 static bool calibrationInProgress = false;
00019 static bool movementInProgress = false;
00020 static bool gripperInProgress = false;
00021
00022 // Internal helper function declarations for parsing and handling specific
00023 // commands
00024 static void handleMoveCommand(const String &command);
00025 static void handleGripperCommand(const String &command);
00026 static void handleCalibrationCommand(const String &command);
00027
00028 /// Handle an incoming command and route it to the appropriate module.
00029 /// @ingroup Communication_Manager
00030 void handleIncomingCommand(const String &command) {
00031   String trimmed = command;
00032   trimmed.trim(); // Removing leading/trailing whitespaces
00033
00034   if (trimmed.startsWith("MOVE")) {
00035     handleMoveCommand(trimmed);
00036   } else if (trimmed.startsWith("GRIP")) {
00037     handleGripperCommand(trimmed);
00038   } else if (trimmed.startsWith("CALIBRATE")) {
00039     handleCalibrationCommand(trimmed);
00040   } else {
00041     Serial.print("[CommunicationManager] Unknown command: ");
00042     Serial.println(trimmed);
00043   }
00044 }
00045
00046 /// Handle a CALIBRATE command: 0 = cancel, 1 = start calibration.
00047 /// Sets calibrationInProgress = true to indicate motion has started.
00048 /// This flag is later cleared by checkActionStatus() once all joints reach
00049 /// their targets.
00050 static void handleCalibrationCommand(const String &incCommand) {
00051   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00052
00053   int state = incCommand.substring(incCommand.indexOf(' ') + 1).toInt();
00054   if (state == 1) {
00055     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATE RECEIVED");
00056     calibrationInProgress = true;
00057     calibrateAllJoints();
00058     calibrationInProgress = false;
00059     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATION DONE");
00060   } else {
00061     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION,
00062                   "CALIBRATE CANCELED"); // @todo: Should we implement
00063                                          // calibration cancellation?
00064     calibrationInProgress = false;
00065   }
00066 }
00067
00068 /// Handle a MOVE command by parsing it into 5 values which are then converted
00069 /// and sent to joint actuators via moveJoint(). Sets movementInProgress = true
00070 /// to indicate motion has started. This flag is later cleared by
00071 /// checkActionStatus() once all joints reach their targets.
00072 static void handleMoveCommand(const String &incCommand) {
00073   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00074   int32_t jointValues[5];
00075   int index = 0;
00076   int argStart = incCommand.indexOf(' ') + 1;
00077
00078   while (index < 5) {
00079     int argEnd = incCommand.indexOf(' ', argStart);
00080     if (argEnd == -1)
00081       argEnd = incCommand.length();
00082     jointValues[index++] = incCommand.substring(argStart, argEnd).toInt();
```

```
00083     argStart = argEnd + 1;
00084   }
00085
00086   moveJoint(0, jointValues[0]); // J0
00087   moveJoint(1, jointValues[1]); // J1
00088   moveJoint(2, jointValues[2]); // J2
00089   moveJoint(3, jointValues[3]); // J3
00090   moveJoint(4, jointValues[4]); // J4
00091
00092   movementInProgress = true;
00093 }
00094
00095 /// Handle a GRIP command: 0 = open, 1 = close.
00096 /// Sets gripperInProgress = true to indicate motion has started.
00097 /// This flag is later cleared by checkActionStatus() once all joints reach
00098 /// their targets.
00099 static void handleGripperCommand(const String &incCommand) {
00100   publishStatus(MQTT_TOPIC_STATUS_COMMAND, incCommand);
00101   int state = incCommand.substring(incCommand.indexOf(' ') + 1).toInt();
00102   moveGripper(state);
00103   gripperInProgress = true;
00104 }
00105
00106 /**
00107  * @brief Checks in-progress flags and publish DONE messages.
00108  * @ingroup Communication_Manager
00109  * @note  Must be called each loop to detect action completion promptly.
00110  * @return void
00111  */
00112 void checkActionStatus() {
00113
00114   if (calibrationInProgress && calibrationDone()) {
00115     publishStatus(MQTT_TOPIC_STATUS_CALIBRATION, "CALIBRATION DONE");
00116     calibrationInProgress = false;
00117   }
00118   if (movementInProgress && allJointsDone()) {
00119     publishStatus(MQTT_TOPIC_STATUS_MOTION, "MOVE DONE");
00120     movementInProgress = false;
00121   }
00122
00123   if (gripperInProgress && gripperDone()) {
00124     publishStatus(MQTT_TOPIC_STATUS_GRIPPER, "GRIPPER DONE");
00125     gripperInProgress = false;
00126   }
00127 }
```

6.33 src/communication_manager/communication_manager.h File Reference

Parses incoming MQTT commands and publishes status updates.

```
#include <Arduino.h>
```

Functions

- void handleIncomingCommand (const String &command)

  Handle an incoming command and route it to the appropriate module.
- void checkActionStatus ()

  Checks in-progress flags and publish DONE messages.

6.33.1 Detailed Description

Parses incoming MQTT commands and publishes status updates.

Author

Elin Gravningen

This module is responsible for interpreting command strings received via MQTT and dispatches the messages to the appropriate actuator logic. It also sends status updates back to the Central Leafy Automation ROS2 system.

Definition in file communication_manager.h.

6.34 communication_manager.h

Go to the documentation of this file.

```
00001 #ifndef COMMUNICATION_MANAGER_H
00002 #define COMMUNICATION_MANAGER_H
00003
00004 #include <Arduino.h>
00005
00006 /**
00007  * @file communication_manager.h
00008  * @author Elin Gravningen
00009  * @brief Parses incoming MQTT commands and publishes status updates.
00010  * @details This module is responsible for interpreting command strings received
00011  * via MQTT and dispatches the messages to the appropriate actuator logic. It
00012  * also sends status updates back to the Central Leafy Automation ROS2 system.
00013  *
00014  * @defgroup Communication_Manager Communication Manager
00015  * @{
00016  */
00017
00018 /**
00019  * @brief Decode and route a received command string.
00020  * @param command Text like "MOVE 100 200 300 400 500" or "GRIP 1".
00021  * @return void
00022  * @ingroup Communication_Manager
00023  * @note Is called from the MQTT Client's callback.
00024  */
00025 void handleIncomingCommand(const String &command);
00026
00027 /**
00028  * @brief Checks in-progress flags and publishes DONE messages.
00029  * @details Checks the movementInProgress, gripperInProgress, and
00030  * calibrationInProgress flags. For each flag that is set, it calls the
00031  * corresponding completion test:
00032  *          - allJointsDone() for MOVE
00033  *          - gripperDone() for GRIP
00034  *          - calibrationDone() for CALIBRATE
00035  *          If the test returns true, it publishes the respective "DONE" status
00036  * via publishStatus() and clears the in-progress flag.
00037  * @return void
00038  * @ingroup Communication_Manager
00039  * @note Must be called in every main loop to catch completions promptly.
00040  */
00041 void checkActionStatus();
00042
00043 #endif // COMMUNICATION_MANAGER_H
00044
00045 /** @} */ // end of Communication_Manager
```

6.35 src/gripper_driver/gripper_driver.cpp File Reference

Controls the gripper servo (End Effector EF).

```
#include "gripper_driver.h"
#include "config.h"
#include <Servo.h>
```

Functions

- void initGripper ()

  Initialise the gripper servo on its PWM pin and open it.
- void moveGripper (int state)

  Command the gripper to open (state=0) or close (state=1).
- void updateGripper ()

  Update the gripper; clear the moving flag after the configured move time.
- bool gripperDone ()

  Returns true if the gripper has completed its movement.

Variables

- static Servo gripperServo
- static bool moving = false
- static unsigned long moveStartTime = 0

6.35.1 Detailed Description

Controls the gripper servo (End Effector EF).

Author

Elin Gravningen

Definition in file gripper_driver.cpp.

6.36 gripper_driver.cpp

Go to the documentation of this file.

```
00001 /**
00002  * @file gripper_driver.cpp
00003  * @author Elin Gravningen
00004  * @brief Controls the gripper servo (End Effector EF).
00005  * @defgroup Gripper_Driver Gripper Driver
00006  * @{
00007  */
00008
00009 #include "gripper_driver.h"
00010 #include "config.h"
00011 #include <Servo.h>
00012
00013 // Servo object for gripper
00014 static Servo gripperServo;
00015
00016 // Movement state
00017 static bool moving = false;
00018 static unsigned long moveStartTime = 0;
00019
00020 /// @ingroup Gripper_Driver
00021 /// Initialise the gripper servo on its PWM pin and open it.
00022 void initGripper() {
00023   gripperServo.attach(GRIPPER_SERVO_PIN);
00024   gripperServo.write(GRIPPER_OPEN_ANGLE);
00025   moving = false;
00026 }
00027
00028 /// @ingroup Gripper_Driver
00029 /// Command the gripper to open (state=0) or close (state=1).
00030 void moveGripper(int state) {
00031   uint8_t angle;
00032   if (state) {
00033     angle = GRIPPER_CLOSED_ANGLE;
00034   } else {
00035     angle = GRIPPER_OPEN_ANGLE;
00036   }
00037   gripperServo.write(angle);
00038   moveStartTime = millis();
00039   moving = true;
00040 }
00041
00042 /// @ingroup Gripper_Driver
00043 /// Update the gripper; clear the moving flag after the configured move time.
00044 void updateGripper() {
00045   if (moving && (millis() - moveStartTime >= GRIP_MOVE_TIME_MS)) {
00046     moving = false;
00047   }
00048 }
00049
00050 /// @ingroup Gripper_Driver
00051 /// Returns true if the gripper has completed its movement.
00052 bool gripperDone() { return !moving; }
00053
00054 /** @} */ // end of Gripper_Driver
```

6.37 src/gripper_driver/gripper_driver.h File Reference

Controls the gripper servo (End Effector EF).

```
#include <Arduino.h>
```

Functions

- void initGripper ()

  Initialise the gripper servo on its PWM pin and open it.
- void moveGripper (int state)

  Command the gripper to open (state=0) or close (state=1).
- void updateGripper ()

  Update the gripper; clear the moving flag after the configured move time.
- bool gripperDone ()

  Returns true if the gripper has completed its movement.

6.37.1 Detailed Description

Controls the gripper servo (End Effector EF).

Author

Elin Gravningen

Provides initialisation, open/close commands, and status checking for the gripper mechanism using the Servo library.

Definition in file gripper_driver.h.

6.38 gripper_driver.h

Go to the documentation of this file.

```
00001 /**
00002  * @file gripper_driver.h
00003  * @author Elin Gravningen
00004  * @brief Controls the gripper servo (End Effector EF).
00005  * @details Provides initialisation, open/close commands, and status checking
00006  *          for the gripper mechanism using the Servo library.
00007  * @defgroup Gripper_Driver Gripper Driver
00008  * @{
00009  */
00010 #ifndef GRIPPER_DRIVER_H
00011 #define GRIPPER_DRIVER_H
00012
00013 #include <Arduino.h>
00014
00015 /**
00016  * @brief Initialise the gripper servo and set to open position.
00017  * @pre   GRIPPER_SERVO_PIN must be defined in config.h.
00018  * @post  Servo is attached and moved to open angle.
00019  * @ingroup Gripper_Driver
00020  * @return void
00021  */
00022 void initGripper();
00023
00024 /**
```

```
00025  * @brief Command the gripper to open or close.
00026  * @param state 0 = open, 1 = close.
00027  * @ingroup Gripper_Driver
00028  * @return void
00029  */
00030 void moveGripper(int state);
00031
00032 /**
00033  * @brief Must be called frequently to update the gripper motion state.
00034  * @ingroup Gripper_Driver
00035  * @return void
00036  */
00037 void updateGripper();
00038
00039 /**
00040  * @brief Check whether the gripper has completed its action.
00041  * @ingroup Gripper_Driver
00042  * @return true if no gripper motion is in progress.
00043  */
00044 bool gripperDone();
00045
00046 #endif // GRIPPER_DRIVER_H
```

6.39 src/main.cpp File Reference

Entry point for the Arduino Uno R4 Wifi, responsible for low level motor control. Coordinates hardware and MQTT handling.

```
#include "common/secrets.h"
#include "communication_manager.h"
#include "gripper_driver.h"
#include "motor_driver.h"
#include "mqtt_client.h"
#include <Arduino.h>
```

Functions

- void setup ()

  Arduino setup() — runs once on power-up or reset.
- void loop ()

  Arduino loop() — runs continuously after setup().

6.39.1 Detailed Description

Entry point for the Arduino Uno R4 Wifi, responsible for low level motor control. Coordinates hardware and MQTT handling.

Author

Elin Gravningen

Initialises serial, network, motor and gripper drivers, and enters the main control loop to service MQTT, motion updates, status checks, and heartbeat.

Definition in file main.cpp.

6.39.2 Function Documentation

6.39.2.1 loop()

```
void loop ()
```

Arduino loop() — runs continuously after setup().

Note

Each iteration services MQTT traffic, steps motors and gripper, publishes completion events, and sends a periodic heartbeat.

Definition at line 48 of file main.cpp.

```
00048             {
00049   mqttLoop();           // Handle MQTT messages & reconnect logic
00050   updateMotors();       // Advance each stepper motor toward its target
00051   updateGripper();      // Update gripper movement timing
00052   checkActionStatus(); // Check if any actions are in progress (e.g., move,
00053                        // gripper, calibration) and publish status
00054   sendHeartbeat();      // Send regular "alive" status
00055 }
```

6.39.2.2 setup()

```
void setup ()
```

Arduino setup() — runs once on power-up or reset.

Note

Initialises Serial, WiFi, MQTT client, and hardware drivers, then registers the command callback for incoming MQTT messages.

Definition at line 23 of file main.cpp.

```
00023             {
00024   Serial.begin(115200);
00025   while (!Serial) {
00026     delay(10);
00027   }
00028
00029   // Networking
00030   initWiFi();                                // Connect to WiFi
00031   initMQTT();                                // Connect to broker
00032   setMessageHandler(handleIncomingCommand); // Setting up the message handler
00033                                              // for incoming commands
00034
00035   // Hardware subsystems
00036   initMotors();  // Initialise stepper drivers (phase 1 drivers)
00037   initGripper(); // Initialise gripper
00038
00039   Serial.println("[System] Leafy Automation Core is ready");
00040 }
```

6.40 main.cpp

Go to the documentation of this file.

```
00001 /**
00002  * @file main.cpp
00003  * @author Elin Gravningen
00004  * @brief Entry point for the Arduino Uno R4 Wifi, responsible for low level
00005  * motor control. Coordinates hardware and MQTT handling.
00006  *  @details Initialises serial, network, motor and gripper drivers, and enters
00007  * the main control loop to service MQTT, motion updates, status checks, and
00008  * heartbeat.
00009  */
00010
00011 #include "common/secrets.h"
00012 #include "communication_manager.h"
00013 #include "gripper_driver.h"
00014 #include "motor_driver.h"
00015 #include "mqtt_client.h"
00016 #include <Arduino.h>
00017
00018 /**
00019  * @brief Arduino setup() -- runs once on power-up or reset.
00020  * @note Initialises Serial, WiFi, MQTT client, and hardware drivers, then
00021  * registers the command callback for incoming MQTT messages.
00022  */
00023 void setup() {
00024   Serial.begin(115200);
00025   while (!Serial) {
00026     delay(10);
00027   }
00028
00029   // Networking
00030   initWiFi();                              // Connect to WiFi
00031   initMQTT();                              // Connect to broker
00032   setMessageHandler(handleIncomingCommand); // Setting up the message handler
00033                                            // for incoming commands
00034
00035   // Hardware subsystems
00036   initMotors();  // Initialise stepper drivers (phase 1 drivers)
00037   initGripper(); // Initialise gripper
00038
00039   Serial.println("[System] Leafy Automation Core is ready");
00040 }
00041
00042 /**
00043  * @brief Arduino loop() -- runs continuously after setup().
00044  * @note Each iteration services MQTT traffic, steps motors and gripper,
00045  *       publishes completion events, and sends a periodic heartbeat.
00046  */
00047
00048 void loop() {
00049   mqttLoop();          // Handle MQTT messages & reconnect logic
00050   updateMotors();      // Advance each stepper motor toward its target
00051   updateGripper();     // Update gripper movement timing
00052   checkActionStatus(); // Check if any actions are in progress (e.g., move,
00053                        // gripper, calibration) and publish status
00054   sendHeartbeat();     // Send regular "alive" status
00055 }
```

6.41 src/motor_driver/motor_driver.cpp File Reference

Implementation of the Motor Driver for joints J0–J4.

```
#include "motor_driver.h"
#include "config.h"
#include <AccelStepper.h>
```

Functions

- void initMotors ()

Initialise all stepper drivers' speed & acceleration per config.h settings.

- void moveJoint (uint8_t jointIndex, int32_t stepCount)

  Queue a relative microstep move for a specific joint.
- void updateMotors ()

  Must be called every loop in order to advance the stepper motors.
- void calibrateAllJoints ()

  Perform a blocking homing routine (the calibration routine) of all joints with timeout and debounce.
- bool calibrationDone ()

  Check if calibration has completed.
- bool allJointsDone ()

  Check if all steppers have reached their targets.
- int32_t getJointPosition (uint8_t jointIndex)

  Get the current microstep position of a joint.
- void stopAllJoints ()

  Stop all motor motion immediately by clearing queued moves.

Variables

- static AccelStepper stepperMotors [ARM_JOINTS]
- static bool homed = false

  true once homing has completed
- static bool homing = false

  true while homing is in progress

6.41.1 Detailed Description

Implementation of the Motor Driver for joints J0–J4.

Author

Elin Gravningen

Uses the AccelStepper library to drive step/dir drivers, perform blocking homing via limit switches, and report motion status.

Definition in file motor_driver.cpp.

6.41.2 Variable Documentation

6.41.2.1 homed

```
bool homed = false  [static]
```

true once homing has completed

Definition at line 24 of file motor_driver.cpp.

6.41.2.2 homing

```
bool homing = false  [static]
```

true while homing is in progress

Definition at line 25 of file motor_driver.cpp.

6.41.2.3 stepperMotors

```
AccelStepper stepperMotors[ARM_JOINTS]  [static]
```

Initial value:

```
= {
    AccelStepper(AccelStepper::DRIVER, STEP_PIN[0], DIR_PIN[0]),
    AccelStepper(AccelStepper::DRIVER, STEP_PIN[1], DIR_PIN[1]),
    AccelStepper(AccelStepper::DRIVER, STEP_PIN[2], DIR_PIN[2]),
    AccelStepper(AccelStepper::DRIVER, STEP_PIN[3], DIR_PIN[3]),
    AccelStepper(AccelStepper::DRIVER, STEP_PIN[4], DIR_PIN[4])}
```

Definition at line 16 of file motor_driver.cpp.

```
00016                                                {
00017     AccelStepper(AccelStepper::DRIVER, STEP_PIN[0], DIR_PIN[0]),
00018     AccelStepper(AccelStepper::DRIVER, STEP_PIN[1], DIR_PIN[1]),
00019     AccelStepper(AccelStepper::DRIVER, STEP_PIN[2], DIR_PIN[2]),
00020     AccelStepper(AccelStepper::DRIVER, STEP_PIN[3], DIR_PIN[3]),
00021     AccelStepper(AccelStepper::DRIVER, STEP_PIN[4], DIR_PIN[4])};
```

6.42 motor_driver.cpp

Go to the documentation of this file.

```
00001 /**
00002  * @file motor_driver.cpp
00003  * @author Elin Gravningen
00004  * @brief Implementation of the Motor Driver for joints J0-J4.
00005  * @details Uses the AccelStepper library to drive step/dir drivers, perform
00006  *          blocking homing via limit switches, and report motion status.
00007  *
00008  * @ingroup Motor_Driver
00009  */
00010
00011 #include "motor_driver.h"
00012 #include "config.h"
00013 #include <AccelStepper.h>
00014
00015 // One AccelStepper instance per joint (J0...J4), using the DRIVER interface.
00016 static AccelStepper stepperMotors[ARM_JOINTS] = {
00017     AccelStepper(AccelStepper::DRIVER, STEP_PIN[0], DIR_PIN[0]),
00018     AccelStepper(AccelStepper::DRIVER, STEP_PIN[1], DIR_PIN[1]),
00019     AccelStepper(AccelStepper::DRIVER, STEP_PIN[2], DIR_PIN[2]),
00020     AccelStepper(AccelStepper::DRIVER, STEP_PIN[3], DIR_PIN[3]),
00021     AccelStepper(AccelStepper::DRIVER, STEP_PIN[4], DIR_PIN[4])};
00022
00023 // Homing state flag
00024 static bool homed = false;  ///< true once homing has completed
00025 static bool homing = false; ///< true while homing is in progress
00026
00027 /// @ingroup Motor_Driver
00028 /// Initialise all stepper drivers' speed & acceleration per config.h settings.
00029 void initMotors() {
00030   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00031     // compute max step rate: (RPM/60) * (microsteps/gear_ratio)
00032     float stepsPerSec =
00033         (MAX_OUTPUT_RPM[j] / 60.0f) * (MICROSTEPS / GEAR_RATIO[j]);
00034     stepperMotors[j].setMaxSpeed(stepsPerSec);
00035     stepperMotors[j].setAcceleration(stepsPerSec * 2.0f);
00036   }
00037   homed = false;
00038 }
```

```
00039
00040 /// @ingroup Motor_Driver
00041 void moveJoint(uint8_t jointIndex, int32_t stepCount) {
00042   stepperMotors[jointIndex].move(stepCount);
00043 }
00044
00045 /// @ingroup Motor_Driver
00046 void updateMotors() {
00047   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00048     stepperMotors[j].run();
00049   }
00050 }
00051
00052 /// @ingroup Motor_Driver
00053 /// Perform a blocking homing routine (the calibration routine) of all joints with timeout and
00054 /// debounce.
00055 void calibrateAllJoints() {
00056   const unsigned long timeoutMs = 5000; // max time per switch
00057   const unsigned int debounceMs = 50;   // debounce delay
00058   homing = true;
00059   homed = false;
00060
00061   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00062     unsigned long startTime;
00063     bool switchState;
00064
00065     // Drive toward left switch
00066     stepperMotors[j].setMaxSpeed(MICROSTEPS * 100.0f);
00067     stepperMotors[j].moveTo(-1000000);
00068     startTime = millis();
00069     while (true) {
00070       stepperMotors[j].run();
00071       switchState = digitalRead(LIMIT_LEFT_PINS[j]) == LOW; // active low
00072       if (switchState) {
00073         delay(debounceMs);
00074         if (digitalRead(LIMIT_LEFT_PINS[j]) == LOW)
00075           break;
00076       }
00077       if (millis() - startTime > timeoutMs)
00078         break;
00079     }
00080     stepperMotors[j].setCurrentPosition(0);
00081
00082     // Drive toward right switch
00083     stepperMotors[j].moveTo(1000000);
00084     startTime = millis();
00085     while (true) {
00086       stepperMotors[j].run();
00087       switchState = digitalRead(LIMIT_RIGHT_PINS[j]) == LOW;
00088       if (switchState) {
00089         delay(debounceMs);
00090         if (digitalRead(LIMIT_RIGHT_PINS[j]) == LOW)
00091           break;
00092       }
00093       if (millis() - startTime > timeoutMs)
00094         break;
00095     }
00096     long maxSteps = stepperMotors[j].currentPosition();
00097
00098     // Return to midpoint
00099     long mid = maxSteps / 2;
00100     stepperMotors[j].setCurrentPosition(0);
00101     stepperMotors[j].moveTo(mid);
00102     while (stepperMotors[j].distanceToGo() != 0) {
00103       stepperMotors[j].run();
00104     }
00105     stepperMotors[j].setCurrentPosition(0);
00106   }
00107
00108   homed = true;
00109   homing = false;
00110 }
00111
00112 /// @ingroup Motor_Driver
00113 bool calibrationDone() { return homed; }
00114
00115 /// @ingroup Motor_Driver
00116 bool allJointsDone() {
00117   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00118     if (stepperMotors[j].distanceToGo() != 0)
00119       return false;
00120   }
00121   return true;
00122 }
00123
00124 /// @ingroup Motor_Driver
00125 int32_t getJointPosition(uint8_t jointIndex) {
```

```
00126   return stepperMotors[jointIndex].currentPosition();
00127 }
00128
00129 /// @ingroup Motor_Driver
00130 void stopAllJoints() {
00131   for (uint8_t j = 0; j < ARM_JOINTS; ++j) {
00132     stepperMotors[j].stop();
00133   }
00134 }
```

6.43 src/motor_driver/motor_driver.h File Reference

Driver for stepper motors J0–J4 (DM332T/DM320T step/dir drivers).

```
#include <Arduino.h>
```

Functions

- void initMotors ()

  Initialise all stepper drivers' speed & acceleration per config.h settings.
- void moveJoint (uint8_t jointIndex, int32_t stepCount)

  Queue a relative microstep move for a specific joint.
- void updateMotors ()

  Must be called every loop in order to advance the stepper motors.
- void calibrateAllJoints ()

  Perform a blocking homing routine (the calibration routine) of all joints with timeout and debounce.
- bool calibrationDone ()

  Check if calibration has completed.
- bool allJointsDone ()

  Check if all steppers have reached their targets.
- int32_t getJointPosition (uint8_t jointIndex)

  Get the current microstep position of a joint.
- void stopAllJoints ()

  Stop all motor motion immediately by clearing queued moves.

6.43.1 Detailed Description

Driver for stepper motors J0–J4 (DM332T/DM320T step/dir drivers).

Author

Elin Gravningen

This module provides initialisation, movement commands, and status checks for each joint motor. Each motor is mapped to a joint index:

- 0 = base (J0)
- 1 = shoulder (J1)
- 2 = elbow (J2)
- 3 = wrist bend (J3)
- 4 = wrist rotation (J4)

Definition in file motor_driver.h.

6.44 motor_driver.h

Go to the documentation of this file.

```
00001 /**
00002  * @file motor_driver.h
00003  * @author Elin Gravningen
00004  * @brief Driver for stepper motors J0-J4 (DM332T/DM320T step/dir drivers).
00005  * @details This module provides initialisation, movement commands, and status
00006  * checks for each joint motor. Each motor is mapped to a joint index:
00007  *   - 0 = base (J0)
00008  *   - 1 = shoulder (J1)
00009  *   - 2 = elbow (J2)
00010  *   - 3 = wrist bend (J3)
00011  *   - 4 = wrist rotation (J4)
00012  *
00013  * @defgroup Motor_Driver Motor Driver
00014  * @{
00015  */
00016
00017 #ifndef MOTOR_DRIVER_H
00018 #define MOTOR_DRIVER_H
00019
00020 #include <Arduino.h>
00021
00022 /**
00023  * @brief Initialise stepper parameters(max speed & acceleration).
00024  * @pre   STEP_PIN[], DIR_PIN[], MICROSTEPS, GEAR_RATIO[] and MAX_OUTPUT_RPM[]
00025  * must be configured via config.h.
00026  * @post  Each steppers[j] has its maxSpeed and acceleration set.
00027  * @ingroup Motor_Driver
00028  * @return void
00029  */
00030 void initMotors();
00031
00032 /**
00033  * @brief Queue a relative microstep move for a specific joint.
00034  * @param jointIndex Index of the joint (0...4, i.e. J0...J4).
00035  * @param steps Signed microstep delta (positive=forward, negative=backward).
00036  * @ingroup Motor_Driver
00037  * @return void
00038  */
00039 void moveJoint(uint8_t jointIndex, int32_t stepCount);
00040
00041 /**
00042  * @brief Must be called every loop in order to advance the stepper motors.
00043  * @ingroup Motor_Driver
00044  * @return void
00045  */
00046 void updateMotors();
00047
00048 /**
00049  * @brief Run a blocking homing (calibration) sequence on all stepper joints.
00050  *
00051  * @pre LIMIT_LEFT_PINS[] and LIMIT_RIGHT_PINS[] must be defined in
00052  * /include/config.h.
00053  * @post After return, currentPosition()==0 for each motor.
00054  * @ingroup Motor_Driver
00055  * @return void
00056  * @note  This routine blocks until all limit switches have been found.
00057  */
00058 void calibrateAllJoints();
00059
00060 /**
00061  * @brief Check if calibration has completed.
00062  * @ingroup Motor_Driver
00063  * @return true if the last call to calibrateAllJoints() completed.
00064  */
00065 bool calibrationDone();
00066
00067 /**
00068  * @brief Check if all steppers have reached their targets.
00069  * @ingroup Motor_Driver
00070  * @return true if every joint's distanceToGo()==0.
00071  */
00072 bool allJointsDone();
00073
00074 /**
00075  * @brief Get the current microstep position of a joint.
00076  * @param jointIndex Index of the joint (0...4).
00077  * @ingroup Motor_Driver
00078  * @return Current position in microsteps (zeroed at last calibration).
00079  */
00080 int32_t getJointPosition(uint8_t jointIndex);
00081
00082 /**
```

```
00083  * @brief Stop all motor motion immediately by clearing queued moves.
00084  * @ingroup Motor_Driver
00085  * @return void
00086  */
00087 void stopAllJoints();
00088
00089 #endif // MOTOR_DRIVER_H
00090 /** @} */ // end of Motor_Driver
```

6.45 src/mqtt_client/mqtt_client.cpp File Reference

```
#include "mqtt_client.h"
#include "common/secrets.h"
#include <PubSubClient.h>
#include <WiFiS3.h>
```

Functions

- static PubSubClient mqttClient (wifiClient)
- static void subscribeTopics ()
- static void mqttCallback (char *topic, byte *payload, unsigned int length)

  Callback from PubSubClient for incoming MQTT messages. Receives the MQTT payload and reserves the required space, and then converts it to a String. It then checks if a user defined handler has been set, and if it has, calls it for the converted message.
- void initMQTT ()

  Initialise MQTT server and set callback.
- bool publishStatus (const char *topic, const String &message)

  Publish a status message and report failure.
- void sendHeartbeat ()

  Register the handler for incoming MQTT messages.
- void setMessageHandler (void(*handler)(const String &msg))

  Registers a callback to handle incoming parsed MQTT messages.
- static unsigned long backoffInterval (uint8_t retries)

  Calculate backoff interval (ms) for reconnection attempts.
- void mqttLoop ()

  Process incoming messages and reconnect with exponential backoff.

Variables

- static unsigned long LAST_ATTEMPT = 0
- static uint8_t RETRIES = 0
- static constexpr unsigned int BASE_INTERVAL_MS = 2000

  Minimum interval between connection attempts (in ms)
- static constexpr unsigned int MAX_BACKOFF_MS = 60000

  Maximum backoff interval (in ms)
- static WiFiClient wifiClient
- static void(* incomingMessageHandler)(const String &msg) = nullptr

  Message handler - used when payload is received. This user-defined function pointer is set to handle incoming messages.

6.45.1 Function Documentation

6.45.1.1 backoffInterval()

```
static unsigned long backoffInterval (
            uint8_t retries)  [static]
```

Calculate backoff interval (ms) for reconnection attempts.

Parameters

retries	Number of attempts made so far.

Returns

Interval in milliseconds, capped at MAX_BACKOFF_MS.

Definition at line 121 of file mqtt_client.cpp.

```
00121                                                 {
00122   unsigned long interval = BASE_INTERVAL_MS « min<uint8_t>(retries, 5);
00123   return min(interval, MAX_BACKOFF_MS);
00124 }
```

6.45.1.2 mqttCallback()

```
static void mqttCallback (
            char * topic,
            byte * payload,
            unsigned int length)  [static]
```

Callback from PubSubClient for incoming MQTT messages. Receives the MQTT payload and reserves the required space, and then converts it to a String. It then checks if a user defined handler has been set, and if it has, calls it for the converted message.

Definition at line 44 of file mqtt_client.cpp.

```
00044                                                                       {
00045   String msg;
00046   msg.reserve(length);
00047   for (unsigned int i = 0; i < length; ++i) {
00048     msg += static_cast<char>(payload[i]);
00049   }
00050   if (incomingMessageHandler)
00051     incomingMessageHandler(msg);
00052 }
```

6.45.1.3 mqttClient()

```
static PubSubClient mqttClient (
            wifiClient )  [static]
```

6.45.1.4 subscribeTopics()

```
static void subscribeTopics ()  [static]
```

Definition at line 30 of file mqtt_client.cpp.

```
00030                           {
00031   mqttClient.subscribe(MQTT_TOPIC_MOTION);
00032   mqttClient.subscribe(MQTT_TOPIC_GRIPPER);
00033   mqttClient.subscribe(MQTT_TOPIC_CALIBRATE);
00034 }
```

6.45.2 Variable Documentation

6.45.2.1 BASE_INTERVAL_MS

```
unsigned int BASE_INTERVAL_MS = 2000  [static], [constexpr]
```

Minimum interval between connection attempts (in ms)

Definition at line 21 of file mqtt_client.cpp.

6.45.2.2 incomingMessageHandler

```
void(* incomingMessageHandler) (const String &msg) (
            const String & msg) = nullptr  [static]
```

Message handler - used when payload is received. This user-defined function pointer is set to handle incoming messages.

Definition at line 38 of file mqtt_client.cpp.

6.45.2.3 LAST_ATTEMPT

```
unsigned long LAST_ATTEMPT = 0  [static]
```

Definition at line 18 of file mqtt_client.cpp.

6.45.2.4 MAX_BACKOFF_MS

```
unsigned int MAX_BACKOFF_MS = 60000  [static], [constexpr]
```

Maximum backoff interval (in ms)

Definition at line 23 of file mqtt_client.cpp.

6.45.2.5 RETRIES

```
uint8_t RETRIES = 0  [static]
```

Definition at line 19 of file mqtt_client.cpp.

6.45.2.6 wifiClient

```
WiFiClient wifiClient  [static]
```

Definition at line 26 of file mqtt_client.cpp.

6.46 mqtt_client.cpp

Go to the documentation of this file.

```
00001 /**
00002  * @file MQTTClient.cpp
00003  * @author Elin Gravningen
00004  * @brief MQTT Client for Leafy Automation Arduino R4 WiFi.
00005  *
00006  * Connects to WiFi and EMQX broker. Subscribes to motion, gripper, and
00007  * calibration topics, and publishes status and heartbeat updates. Allows the
00008  * user to set a custom callback function that will be invoked whenever an MQTT
00009  * message is received.
00010  */
00011
00012 #include "mqtt_client.h"
00013 #include "common/secrets.h" // Defines network credentials and MQTT broker address
00014 #include <PubSubClient.h>
00015 #include <WiFiS3.h>
00016
00017 // Connection attempt settings (for reconnects)
00018 static unsigned long LAST_ATTEMPT = 0;
00019 static uint8_t RETRIES = 0;
00020 /// Minimum interval between connection attempts (in ms)
00021 static constexpr unsigned int BASE_INTERVAL_MS = 2000;
00022 /// Maximum backoff interval (in ms)
00023 static constexpr unsigned int MAX_BACKOFF_MS = 60000;
00024
00025 // MQTT Client settings
00026 static WiFiClient wifiClient;
00027 static PubSubClient mqttClient(wifiClient);
00028
00029 // Helper function to subscribe to MQTT Control topics from ROS2.
00030 static void subscribeTopics() {
00031   mqttClient.subscribe(MQTT_TOPIC_MOTION);
00032   mqttClient.subscribe(MQTT_TOPIC_GRIPPER);
00033   mqttClient.subscribe(MQTT_TOPIC_CALIBRATE);
00034 }
00035
00036 /// Message handler - used when payload is received. This user-defined function
00037 /// pointer is set to handle incoming messages.
00038 static void (*incomingMessageHandler)(const String &msg) = nullptr;
00039
00040 /// Callback from PubSubClient for incoming MQTT messages. Receives the MQTT
00041 /// payload and reserves the required space, and then converts it to a String.
00042 /// It then checks if a user defined handler has been set, and if it has, calls
00043 /// it for the converted message.
00044 static void mqttCallback(char *topic, byte *payload, unsigned int length) {
00045   String msg;
00046   msg.reserve(length);
00047   for (unsigned int i = 0; i < length; ++i) {
00048     msg += static_cast<char>(payload[i]);
00049   }
00050   if (incomingMessageHandler)
00051     incomingMessageHandler(msg);
00052 }
00053
00054 /**
00055  * @ingroup MQTT_Client
00056  * @brief Initialise MQTT server and set callback.
00057  * @pre   WiFi is connected via initWiFi().
00058  * @post  Single connect attempt. Further reconnects in mqttLoop().
00059  * @return void
00060  */
00061
00062 void initMQTT() {
00063   mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
00064   mqttClient.setCallback(mqttCallback);
00065
00066   // Establish connection attempt
00067   if (mqttClient.connect(MQTT_CLIENT_ID)) {
00068     subscribeTopics();
00069     Serial.println("MQTT connected.");
00070   } else {
00071     Serial.print("MQTT connect failed, rc=");
00072     Serial.println(mqttClient.state());
00073   }
00074 }
00075
00076 /**
00077  * @ingroup MQTT_Client
00078  * @brief Publish a status message and report failure.
00079  * @param topic   MQTT topic string.
00080  * @param message Payload to publish.
00081  * @return true if publish was accepted; false otherwise.
00082  */
```

```
00083 bool publishStatus(const char *topic, const String &message) {
00084   bool ok = mqttClient.publish(topic, message.c_str());
00085   if (!ok) {
00086     Serial.print("Publish failed to topic: ");
00087     Serial.println(topic);
00088   }
00089   return ok;
00090 }
00091
00092 /**
00093  * @ingroup MQTT_Client
00094  * @brief Register the handler for incoming MQTT messages.
00095  * @param handler Function invoked when message is received.
00096  * @return void
00097  */
00098 void sendHeartbeat() {
00099   static unsigned long lastPing = 0;
00100   if (millis() - lastPing >= 1000) {
00101     publishStatus(MQTT_TOPIC_STATUS_HEARTBEAT, "alive");
00102     lastPing = millis();
00103   }
00104 }
00105
00106 /**
00107  * @ingroup MQTT_Client
00108  * @brief Registers a callback to handle incoming parsed MQTT messages.
00109  * @param handler Function to call with message string.
00110  * @return void
00111  */
00112 void setMessageHandler(void (*handler)(const String &msg)) {
00113   incomingMessageHandler = handler;
00114 }
00115
00116 /**
00117  * @brief Calculate backoff interval (ms) for reconnection attempts.
00118  * @param retries Number of attempts made so far.
00119  * @return Interval in milliseconds, capped at MAX_BACKOFF_MS.
00120  */
00121 static unsigned long backoffInterval(uint8_t retries) {
00122   unsigned long interval = BASE_INTERVAL_MS << min<uint8_t>(retries, 5);
00123   return min(interval, MAX_BACKOFF_MS);
00124 }
00125
00126 /**
00127  * @ingroup MQTT_Client
00128  * @brief Process incoming messages and reconnect with exponential backoff.
00129  * @note  Must be called frequently in loop() to maintain connection.
00130  * @return void
00131  */
00132 void mqttLoop() {
00133   unsigned long now = millis();
00134
00135   if (!mqttClient.connected() &&
00136       (now - LAST_ATTEMPT >= backoffInterval(RETRIES))) {
00137     if (mqttClient.connect(MQTT_CLIENT_ID)) {
00138       Serial.println("MQTT reconnected");
00139       subscribeTopics();
00140       RETRIES = 0;
00141     } else {
00142       RETRIES++;
00143       Serial.println("MQTT reconnect failed, will retry");
00144     }
00145     LAST_ATTEMPT = now;
00146   }
00147   mqttClient.loop();
00148 }
```

6.47 src/mqtt_client/mqtt_client.h File Reference

Handles MQTT setup, subscriptions, publishing, and heartbeat.

```
#include <Arduino.h>
```

Functions

- void initMQTT ()

Initialise MQTT server and set callback.

- void mqttLoop ()

  Process incoming messages and reconnect with exponential backoff.

- void setMessageHandler (void(∗handler)(const String &msg))

  Registers a callback to handle incoming parsed MQTT messages.

- bool publishStatus (const char ∗topic, const String &message)

  Publish a status message and report failure.

- void sendHeartbeat ()

  Register the handler for incoming MQTT messages.

Variables

- constexpr char MQTT_TOPIC_MOTION [ ] = "leafy_automation/motion"
- constexpr char MQTT_TOPIC_GRIPPER [ ] = "leafy_automation/gripper"
- constexpr char MQTT_TOPIC_CALIBRATE [ ] = "leafy_automation/calibrate"
- constexpr char MQTT_TOPIC_STATUS_COMMAND [ ]
- constexpr char MQTT_TOPIC_STATUS_MOTION [ ]
- constexpr char MQTT_TOPIC_STATUS_GRIPPER [ ]
- constexpr char MQTT_TOPIC_STATUS_CALIBRATION [ ]
- constexpr char MQTT_TOPIC_STATUS_HEARTBEAT [ ]

6.47.1 Detailed Description

Handles MQTT setup, subscriptions, publishing, and heartbeat.

Author

Elin Gravningen

Connects the Arduino R4 WiFi to the EMQX broker, subscribes to control topics (motion, gripper, calibrate), and publishes status updates and periodic heartbeats back to the ROS2 system.

Definition in file mqtt_client.h.

6.48 mqtt_client.h

Go to the documentation of this file.

```
00001 /**
00002  * @file mqtt_client.h
00003  * @author Elin Gravningen
00004  * @brief Handles MQTT setup, subscriptions, publishing, and heartbeat.
00005  * @details Connects the Arduino R4 WiFi to the EMQX broker, subscribes to
00006  *          control topics (motion, gripper, calibrate), and publishes status
00007  *          updates and periodic heartbeats back to the ROS2 system.
00008  *
00009  * @defgroup MQTT_Client MQTT Client Module
00010  * @ingroup LeafyAutomationFirmware
00011  * @{
00012  */
00013
00014 #ifndef MQTT_CLIENT_H
00015 #define MQTT_CLIENT_H
00016
00017 #include <Arduino.h>
00018
00019 // Incoming command topics (from ROS2 to Arduino)
```

```
00020 inline constexpr char MQTT_TOPIC_MOTION[] = "leafy_automation/motion";
00021 inline constexpr char MQTT_TOPIC_GRIPPER[] = "leafy_automation/gripper";
00022 inline constexpr char MQTT_TOPIC_CALIBRATE[] = "leafy_automation/calibrate";
00023
00024 // Outgoing status topics (from Arduino to ROS2)
00025 inline constexpr char MQTT_TOPIC_STATUS_COMMAND[] =
00026     "leafy_automation/status/command_received";
00027 inline constexpr char MQTT_TOPIC_STATUS_MOTION[] =
00028     "leafy_automation/status/motion";
00029 inline constexpr char MQTT_TOPIC_STATUS_GRIPPER[] =
00030     "leafy_automation/status/gripper";
00031 inline constexpr char MQTT_TOPIC_STATUS_CALIBRATION[] =
00032     "leafy_automation/status/calibration";
00033 inline constexpr char MQTT_TOPIC_STATUS_HEARTBEAT[] =
00034     "leafy_automation/status/heartbeat";
00035
00036 /**
00037  * @brief Initialises MQTT connection and subscribes to control topics.
00038  * @ingroup MQTT_Client
00039  * @pre  WiFi must already be connected via initWiFi().
00040  * @post Control topics are subscribed and the incoming message callback is set.
00041  */
00042 void initMQTT();
00043
00044 /**
00045  * @brief Process incoming MQTT traffic and attempt reconnects if needed.
00046  * @ingroup MQTT_Client
00047  * Must be called frequently in loop() to maintain the connection.
00048  * @note Must be called frequently in loop() to maintain the connection.
00049  * @return void
00050  */
00051 void mqttLoop();
00052
00053 /**
00054  * @brief To set user defined callback to handle parsed MQTT messages.
00055  * @ingroup MQTT_Client
00056  * @param handler Function to call when a new message arrives.
00057  * @note This function should be called after initMQTT() to set the callback
00058  */
00059 void setMessageHandler(void (*handler)(const String &msg));
00060
00061 /**
00062  * @brief Publishes a status message to a given MQTT topic.
00063  * @ingroup MQTT_Client
00064  * @param topic   The MQTT topic to publish to.
00065  * @param msg The payload string.
00066  * @return true if the message was successfully handed off to the network,
00067  * otherwise returns false.
00068  */
00069 bool publishStatus(const char *topic, const String &msg);
00070
00071 /**
00072  * @brief Sends a periodic "alive" signal to the status/heartbeat topic.
00073  * @ingroup MQTT_Client
00074  * @return void
00075  */
00076 void sendHeartbeat();
00077
00078 #endif // MQTT_CLIENT_H
00079
00080 /** @} */ // end of MQTT_Client
```

Index

Appendix G

Software

1 Artificial Intelligence Machine Learning tasks in detail

JCDH | -

1.1 Image Classification Task

JCDH | -

"Image classification is the task of assigning a label or class to an entire image. Images are expected to have only one class for each image. Image classification models take an image as input and return a prediction about which class the image belongs to." [70].

In our system, image classification lets us get information about what type of plant we are dealing with. Initial testing showed that current image classification models must be "fine tuned" to give accurate outputs. Figure G.1 shows the general flow of the image classification process.

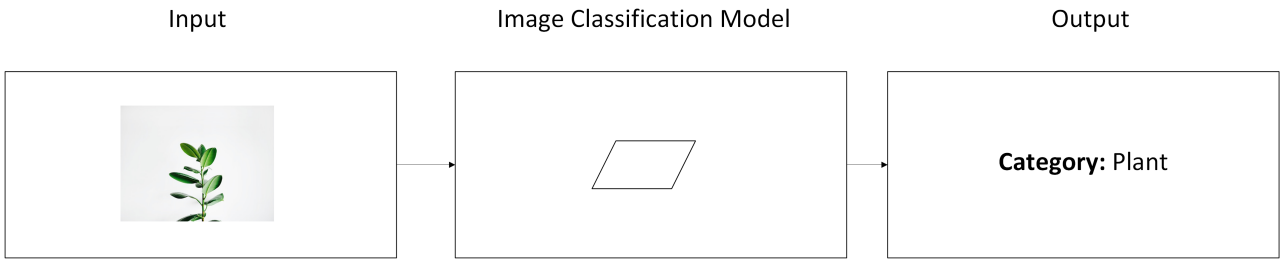


Figure G.1: Image Classification Diagram

1.2 Object Detection Task

JCDH | -

A useful definition of object detection from Hugging Face is the following: "Object Detection models allow users to identify objects of certain defined classes. Object detection models receive an image as input and output the images with bounding boxes and labels on detected objects [71]."

There are a great number of object detection AI models to choose from, so before further work could commence, a comparison of the two most used models in this category had to be compared.

1.2.1 Comparing facebook/detr-resnet-50 and Ultralytics/YOLO11

JCDH | -

Benchmarking was done with the image in Figure G.2, because these models are not yet able to accurately identify salads. This is to be expected as these AI models are made for general use cases like identifying people and objects like cars, and our use-case is highly specific. The image is VGA resolution (640x480).

Figure G.2: AI model input

Image by Bhong Bahala on Unsplash [72]

Once the AI model processes the input, we draw bounding boxes and labels on the image (as seen in Figure G.3).

Figure G.3: AI model output (bounding boxes and labels shown visually)

Image by Bhong Bahala on Unsplash [72]

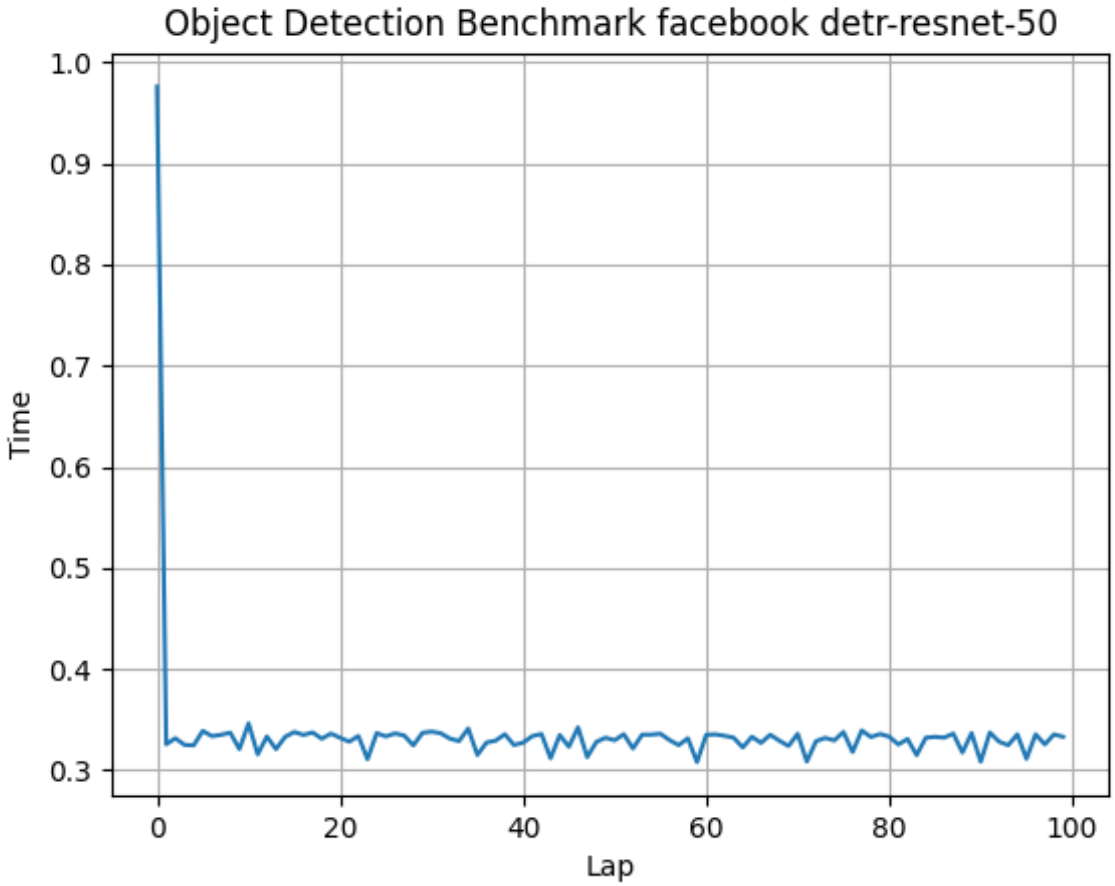


Figure G.4: Object Detection Benchmark: facebook/detr-resnet-50

Average execution time (approx): 0.34 seconds. Standard deviation (approx): 0.06 seconds

facebook/detr-resnet-50: Benchmark average time: 0.33638124000048264 seconds. Standard deviation: 0.0647590208927982 seconds.

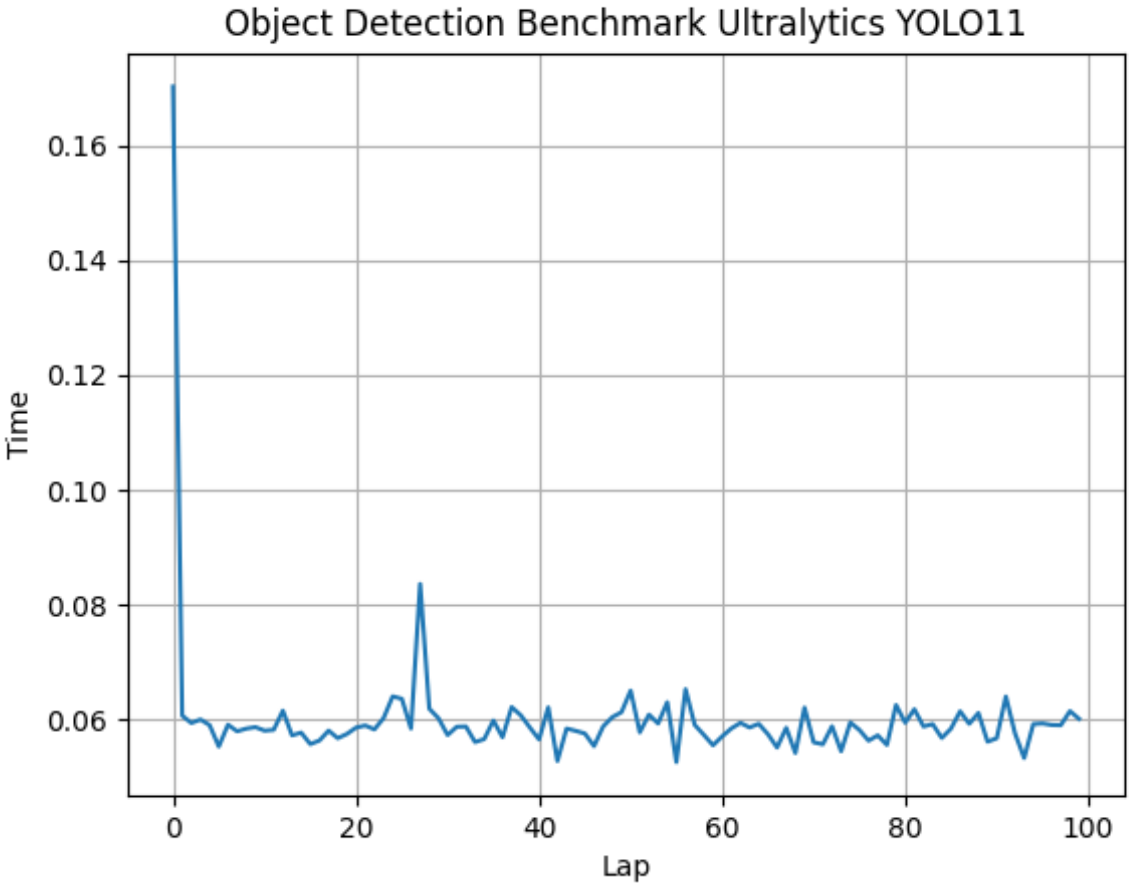


Figure G.5: Object Detection Benchmark: YOLO11
Average execution time (approx): 0.06 seconds. Standard deviation (approx): 0.012 seconds

YOLO11: Benchmark average time: 0.06007338625029661 seconds.
Standard deviation: 0.011615161145226758 seconds.

The Ultralytics/YOLO11 is a promising model for the following reasons: [1]

1. Great performance characteristics.
2. Trained on a smaller and more specific dataset.
3. Easy to use API for training with new datasets.

1.2.2 Bounding boxes and how they work

JCDH | -

[1]The current version of the **ultralytics** library (v8.3.108) which gives access to the YOLO model requires **numpy** v2.1.1 or earlier. Therefore a downgrade had to be made to the **numpy** library (>=2.2.3,<3.0.0 to >=1.23.0,<=2.1.1).

Object detection AI models generate a bounding box around the object(s) of interest, which is a set of points (top left, top right, bottom left, and bottom right) which indicates where in the image the object(s) of interest occurs, as seen in Figure G.6.

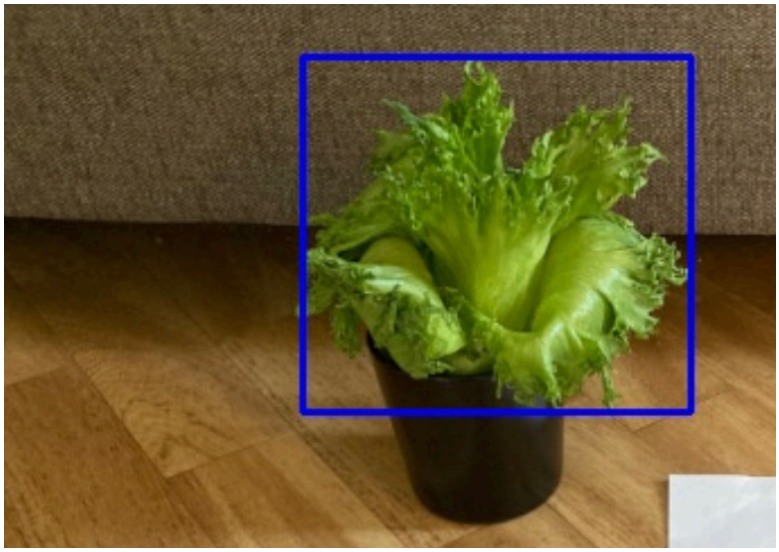


Figure G.6: Example of bounding box

1.2.3 Fine tuning of the object detection model with a open-source dataset

JCDH | -

Fine tuning refers to the process of taking existing AI models and adding additional training data in order to improve its functioning in new areas. An example in our case would be the act of taking an image classification model and adding additional images of lettuces, and its respective labels in order for the model to more accurately recognize sub types of lettuces.

Fine tuning of the object detection model took place using the **Roboflow100/lettuce-pallets** dataset [73], which is hosted on Roboflow and is licensed under "CC BY 4.0" [74]. [2] This dataset is a collection of images of lettuce pallets together with their respective bounding boxes and labels. [3]

Categories in this dataset can be of the following types: **[Ready, empty_pod, germination, pod, young]**.

Fine tuning of the AI model was performed using Google Colab because of resource and time

[2]Fine tuning is also known as training an AI model on a custom dataset.

[3]The dataset is part of an Intel sponsored initiative called RF100 which aims to create a new generalized object detection benchmark [75]

constraints. Normal consumer hardware lacks the necessary resources to fine tune the AI model in a timely fashion.

Data augmentation

When training AI models, you expand your initial dataset by using methods like data augmentation, which means applying augmentation methods to the images. Methods include (but are not limited to) image flipping, cropping, changing contrast and blurring the image. YOLO11 employs these methods by default when using their system to fine-tune their models [76].

Fine tuning results

In this specific fine tuning of the **YOLO11** model with the **Roboflow100/lettuce-pallets** dataset [73], training was run for 5 epochs to assess the results of the process within a reasonable time-frame. Fine-tuning of AI models often runs for more than 100 epochs, but for initial testing, letting the AI model train on the dataset for around 1 hour creates a fine baseline. If results are not satisfactory to our accuracy requirements, fine-tuning can always be continued at a later time.

Please refer to Appendix G.4 for in-depth training results (5 epoch) and Appendix G.5 for 100 epoch results.

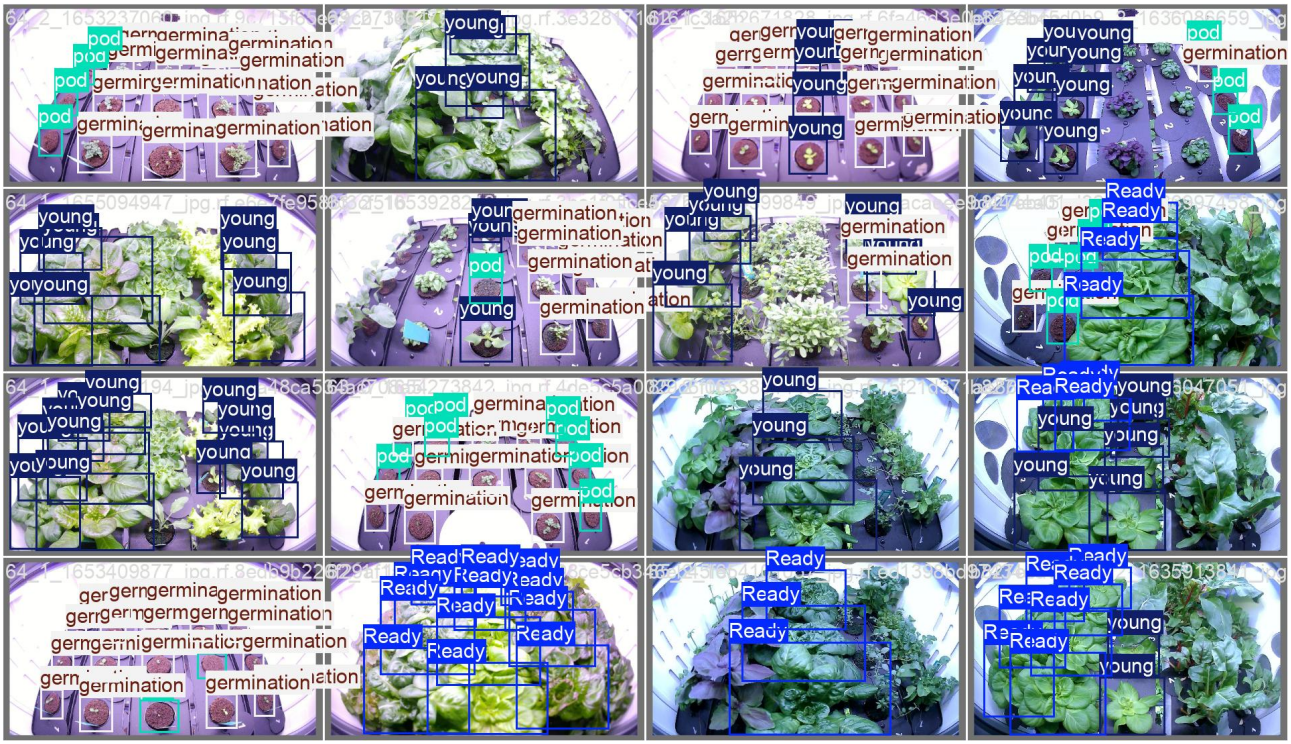


Figure G.7: AI object detection (in-training, batch 2)

Images used for training by Roboflow100/lettuce-pallets are licensed under CC BY 4.0 [73].

5 epoch training results

Taking a look at the results in Figure G.8, we see various metrics pertaining to the training results. Here, the *x-axis* corresponds to the specific training epoch, while the *y-axis* relates to the specific value of the metric.

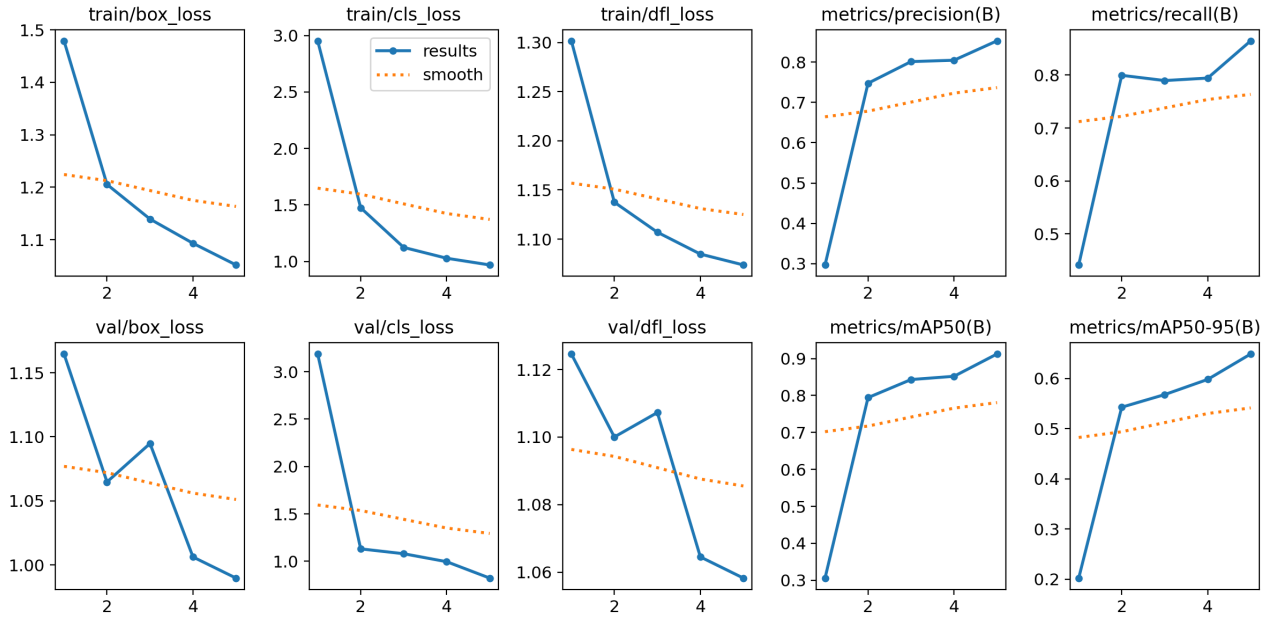


Figure G.8: AI training results

100 epoch training results

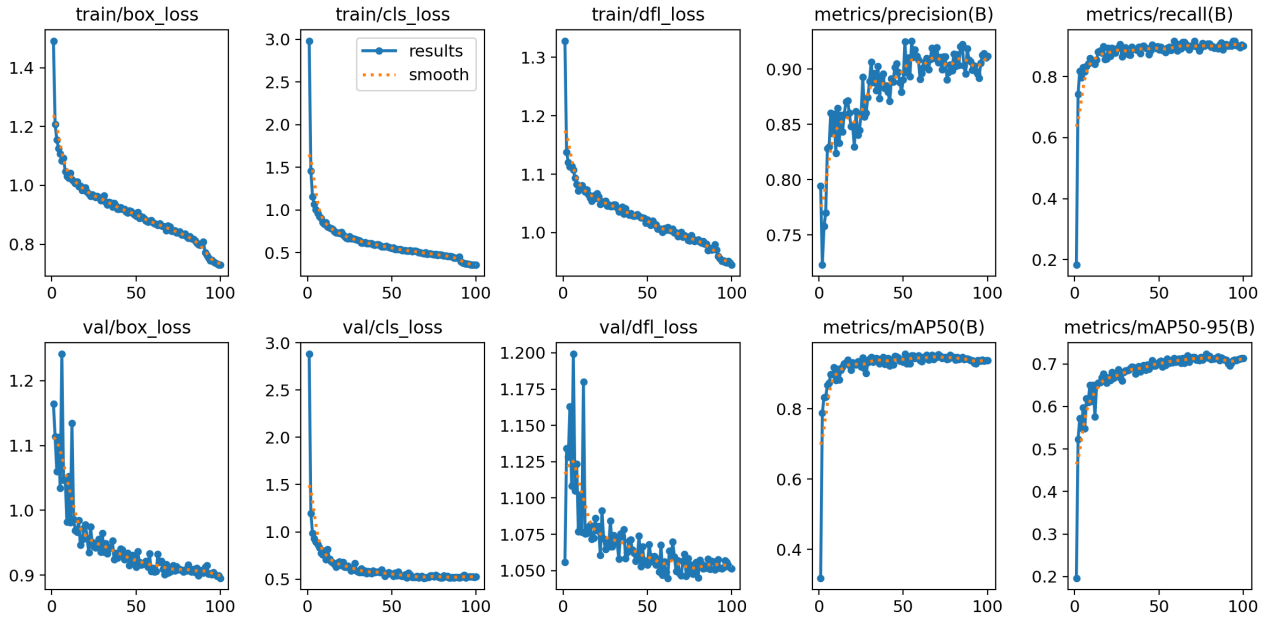


Figure G.9: AI training results (100 epochs)

1.2.4 Evaluating Roboflow100/lettuce-pallets

JCDH | -

The dataset from Roboflow100/lettuce-pallets proved to be insufficient in identifying the types of lettuce we use, and therefore Computer Vision (CV) was explored.

1.3 Depth Estimation Task

As stated by the Hugging Face documentation: "Depth estimation is the task of predicting depth of the objects present in an image." [77].

Figure G.10 shows an example of depth estimation on an image. The information we receive from the depth estimation AI model is the distance (in meters) of each pixel in the the image, or the camera sensor in practice. This data can be translated to a vector responding to a specific point, or points, which gives a real-world representation of distance.

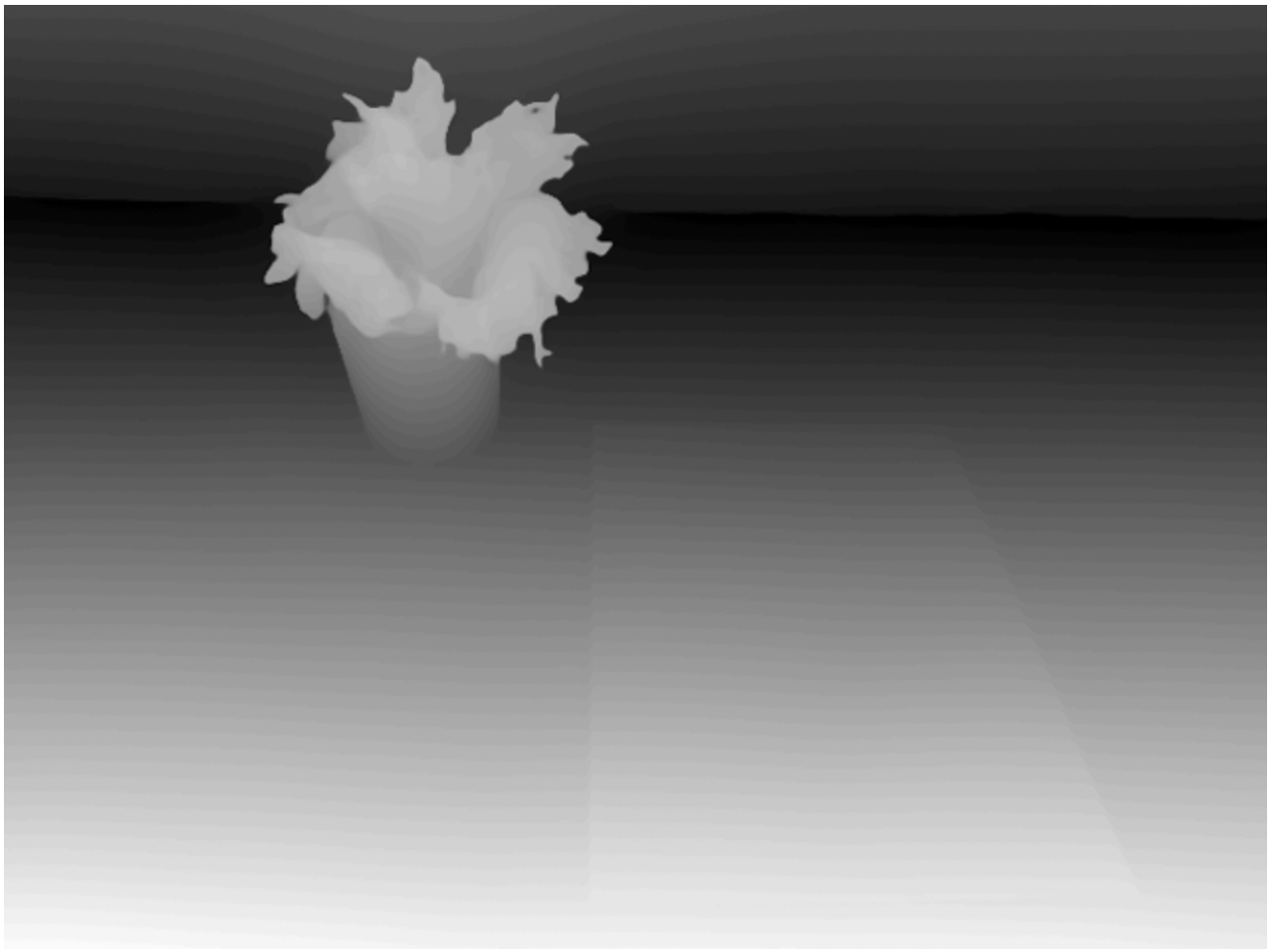


Figure G.10: Depth estimation

Depth Anything delivers three models within the indoor (metric) category (in order of model size) [78]:

- depth-anything/Depth-Anything-V2-Metric-Indoor-Small-hf

- depth-anything/Depth-Anything-V2-Metric-Indoor-Base-hf
- depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf

Larger models should in practice offer more accuracy as they are trained on a larger and more diverse dataset, but larger models have greater processing times. For our system we use the "depth-anything/Depth-Anything-V2-Metric-Indoor-Small-hf" model for visualization purposes, and the "depth-anything/Depth-Anything-V2-Metric-Indoor-Large-hf" for the actual grab point calculations.

2 Artificial Intelligence side notes

JCDH | -

2.1 Performance considerations

JCDH | -

A quick realization which was made was on the topic of the performance issues related to running AI models and ML algorithms. Especially when running on resource constrained hardware like the Raspberry PI or other consumer grade hardware. Thankfully, there exists simpler AI models which are more targeted to our specific use case, as opposed to very general purpose models that meet every conceivable use case. These simpler AI models usually have much lower resource requirements [79]. [4]

2.2 A simple performance improvement to model processing

JCDH | -

While reviewing the code for model processing, it was discovered that the **pipeline** function of the *Hugging Face Transformers* library would load the whole model every time a new image frame was processed (as seen in Figure G.11) [5].

```
def object_detection(image_bytes):
    image = Image.open(io.BytesIO(image_bytes)).convert("RGB")

    model = pipeline("object-detection", model="facebook/detr-resnet-50")

    return model(image)
```

Figure G.11: Python AI processing: Before

A simple solution was to move the initial model loading **pipeline** function outside of the image frame processing function (as seen in Figure G.12) [6].

[4] It was discovered that not all AI models run well on the Macbook Air M1 because Graphical Processing Unit (GPU) support has not yet been implemented.

[5] The aim of this section is to show the performance improvement by improving model loading, and not specifying which model was chosen.

[6] In recent version of *Hugging Face Transformers*, **use_fast** is set to true, which enables a faster Rust-based tokenizer, if available for the chosen model. Some older models used a slower pure Python implementation [80].

```
model = pipeline("object-detection", model="facebook/detr-resnet-50")

def object_detection(image_bytes):
    image = Image.open(io.BytesIO(image_bytes)).convert("RGB")

    return model(image)
```

Figure G.12: Python AI processing: After

In order to measure the gained performance benefits, benchmarking was employed. This was done using the Python **time.perf_counter()** method, and checking the before / after times between the model processing an image [81]. These values where stored in a list, which stored the last 100 frames. An average of these times where then calculated. These benchmarks where done using an image of an iceberg lettuce on a neutral background. The image size was **640 x 480 px**, which is comparable to VGA quality [7].

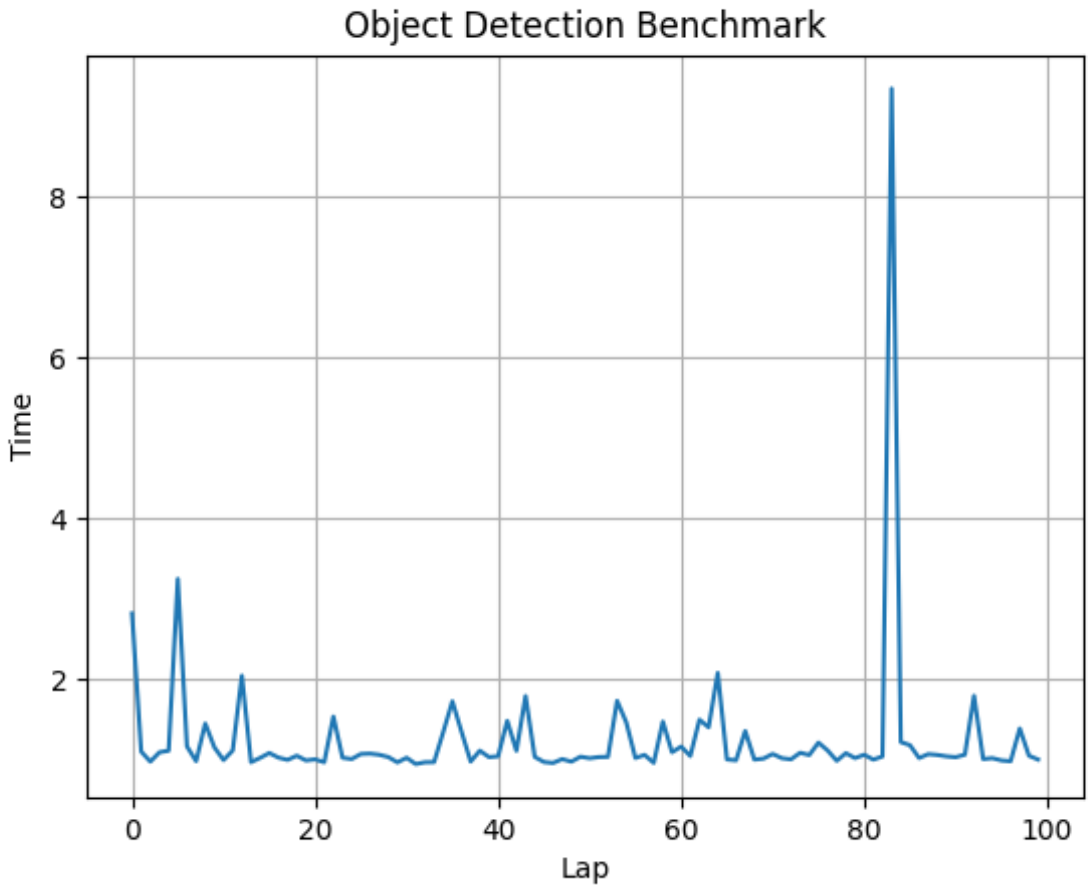


Figure G.13: Object Detection Benchmark (before optimization)
Average execution time (approx): 1.26 seconds. Standard deviation (approx): 0.89 seconds

[7] Benchmarks where done on a MacBook Air M1.

From Figure G.13, we see a very sporadic curve. This is bad because it creates less predictability around how long the processing times for the model will take [8].

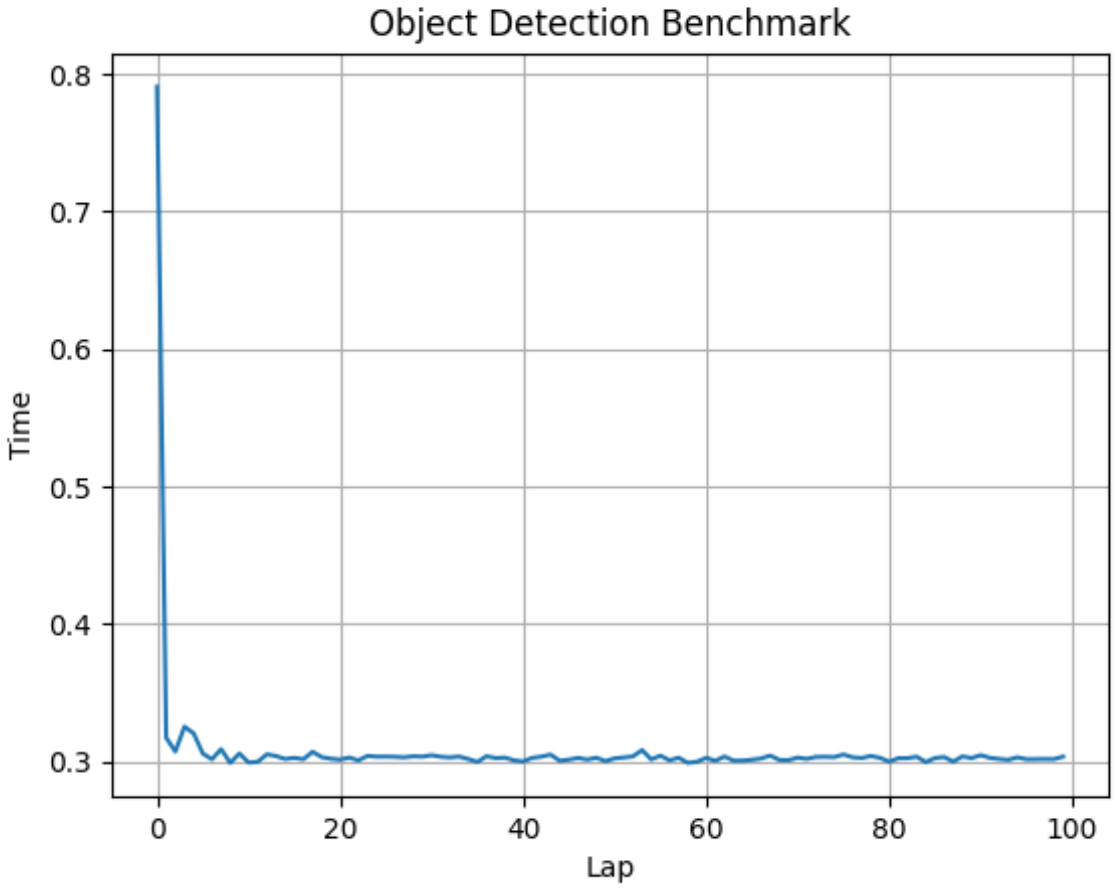


Figure G.14: Object Detection Benchmark (after optimization)
Average execution time (approx): 0.31 seconds. Standard deviation (approx): 0.05 seconds

From the benchmark in Figure G.14, we can see a much smoother curve after the optimizations. We also observe some initial loading times, but after that, the curve looks relatively smooth and stable [9].

These benchmarks where done using a custom developed Benchmark class, which you can read more about in Section 6.

3 AI / ML research phase

JCDH |

[8] Avg. execution time: 1.2581327729400074 seconds. Standard deviation: 0.8867701786412608 seconds.
[9] Avg. execution time: 0.30796559668000556 seconds. Standard deviation: 0.04867672829155695 seconds

Artificial Intelligence (AI) and Machine Learning (ML) are important technologies and critical parts of our system.

3.1 AI models of interest

JCDH |

Research into AI models are still ongoing, but so far the following models seems to be especially useful for our system.

- Image classification
- Mask generation
- Object tracking

3.2 Initial object tracking research

JCDH | -

Initial research began with the Sam2 model by Meta. Based on our workshop meeting with Hydroplant we where recommended this AI object tracking model in particular.

Based on testing using their online demo interface it seemed to work well.

Figure G.15: Initial Sam2 object tracking demo - tracking of a lettuce plant

3.3 Image classification models testing

JCDH | -

Initial testing started with the google/vit-base-patch16-224 model, but further testing of running the model using Hugging Face transformers with python showed that the model performed slowly on a Macbook Air M1. Because of the fact that we would optimally have the model run on a raspberry PI 5th gen, further research into more performant models must be done.

3.4 AI models - specifics

JCDH |

The following is a list of the AI models which our system may need:

- **Image classification:** one possible model is google/vit-base-patch16-224 which is a Vision Transformer (VT) model which was trained on ImageNet-21k that contains 14 million images [82]

4 In-depth AI training results (5 epochs)

JCDH | -

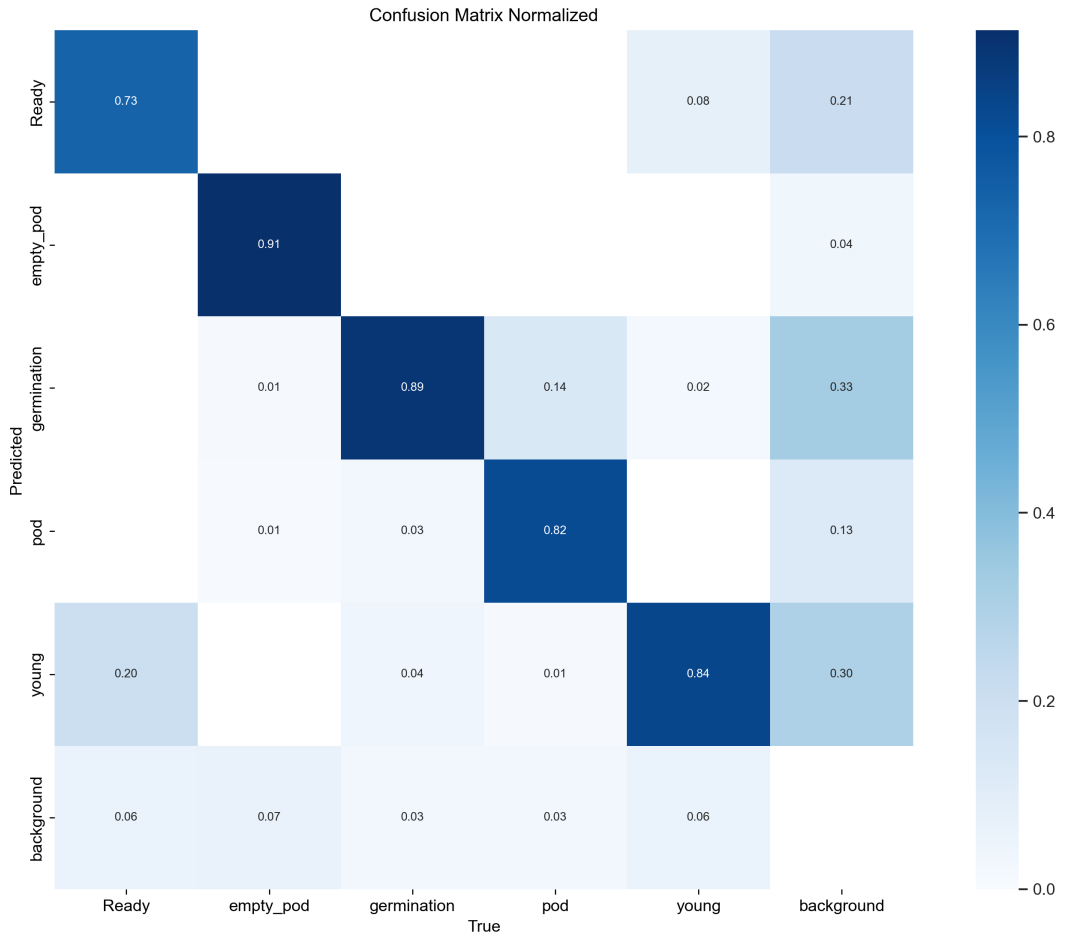


Figure G.16: Confusion matrix - normalized

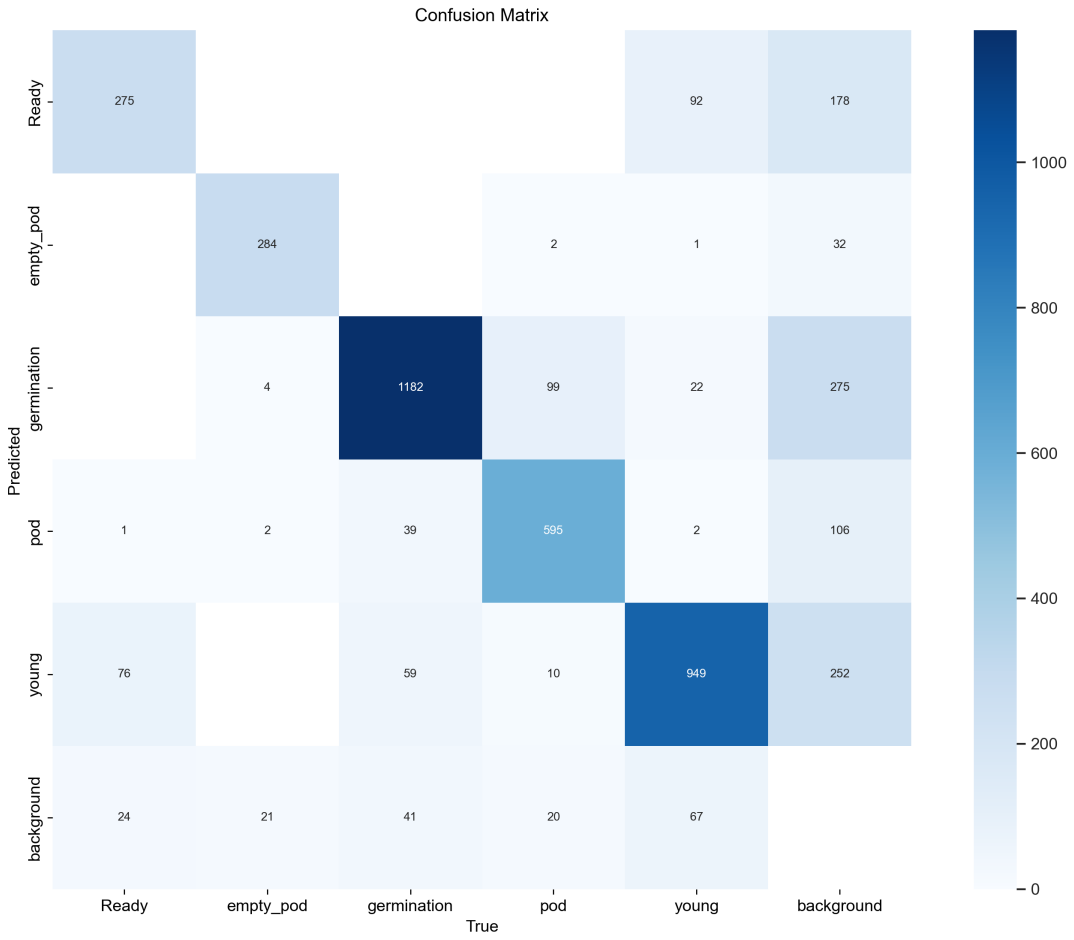


Figure G.17: Confusion matrix

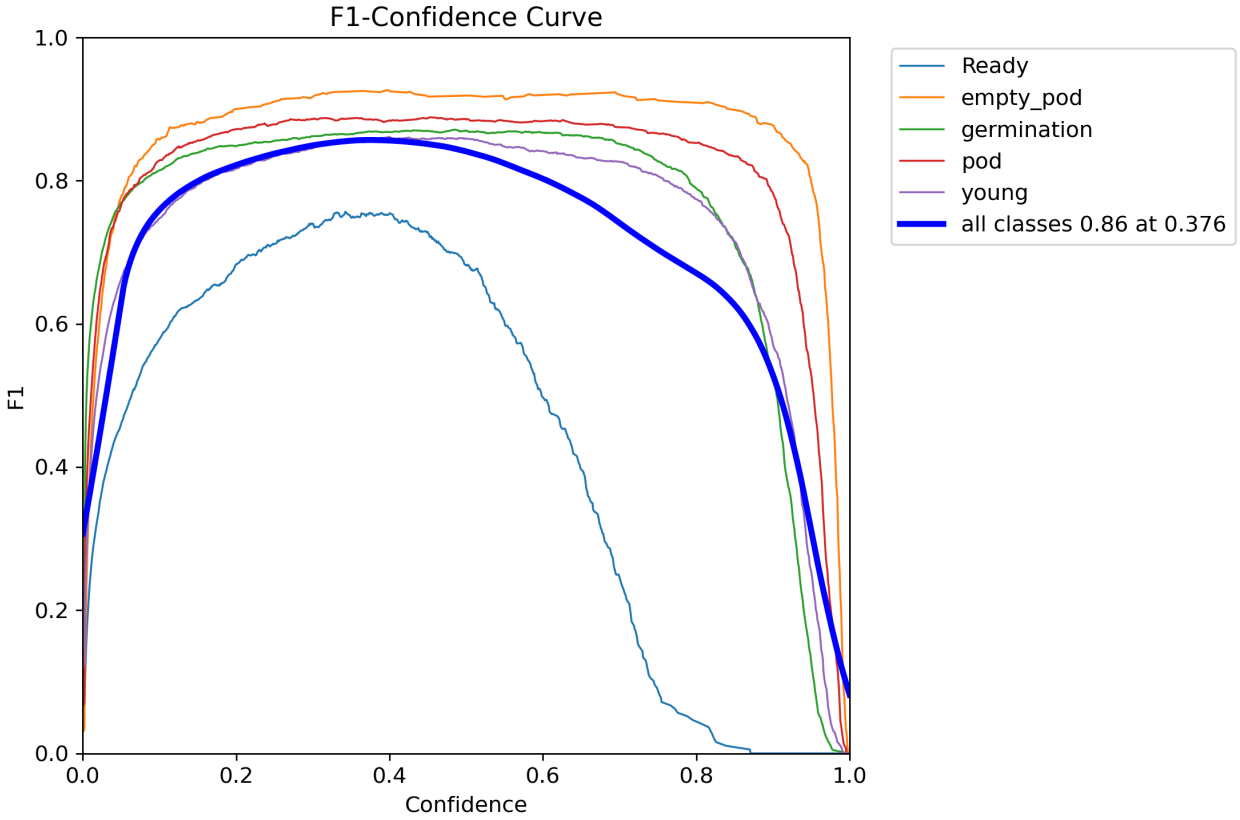


Figure G.18: F1 curve

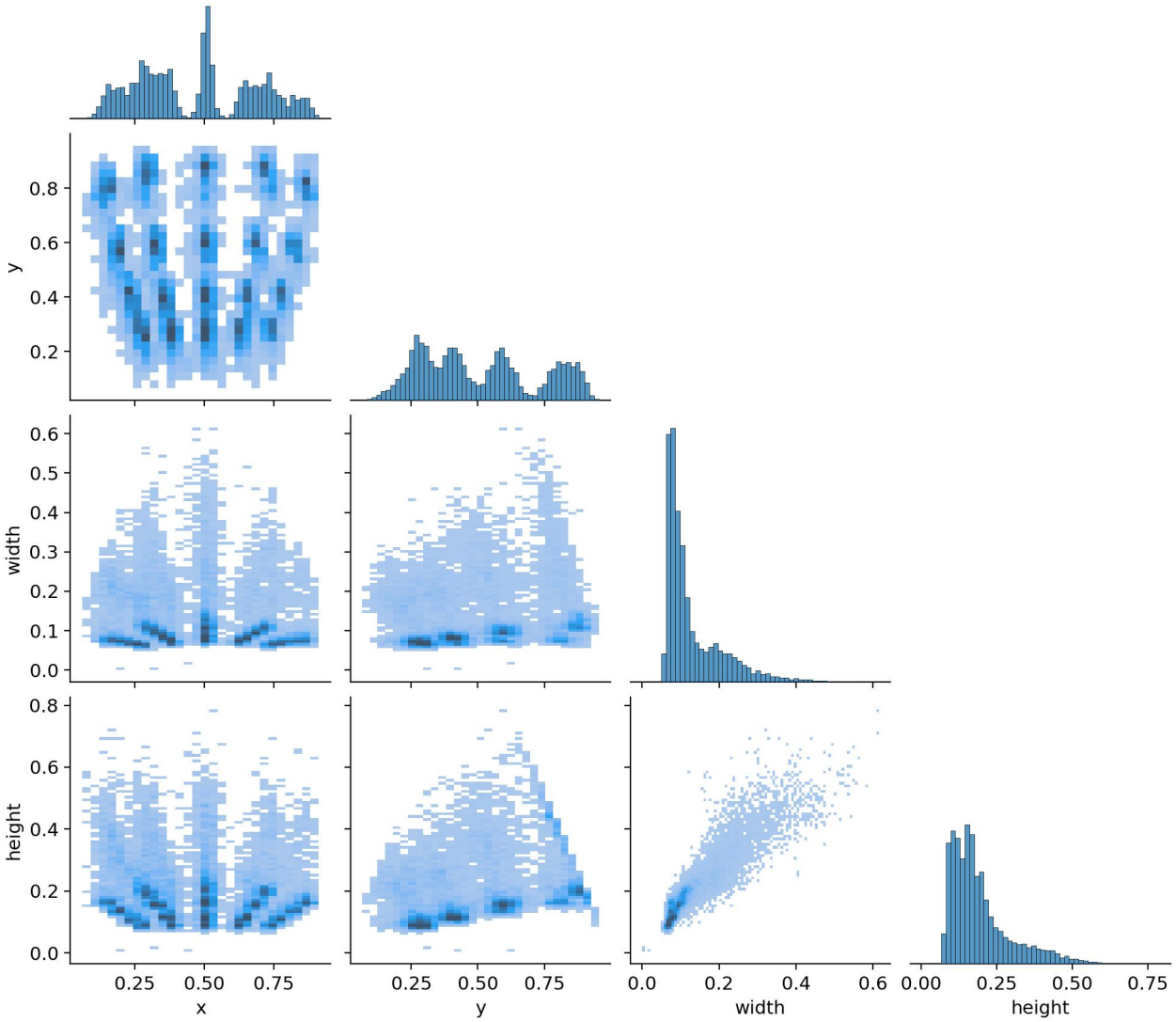


Figure G.19: Labels correlogram

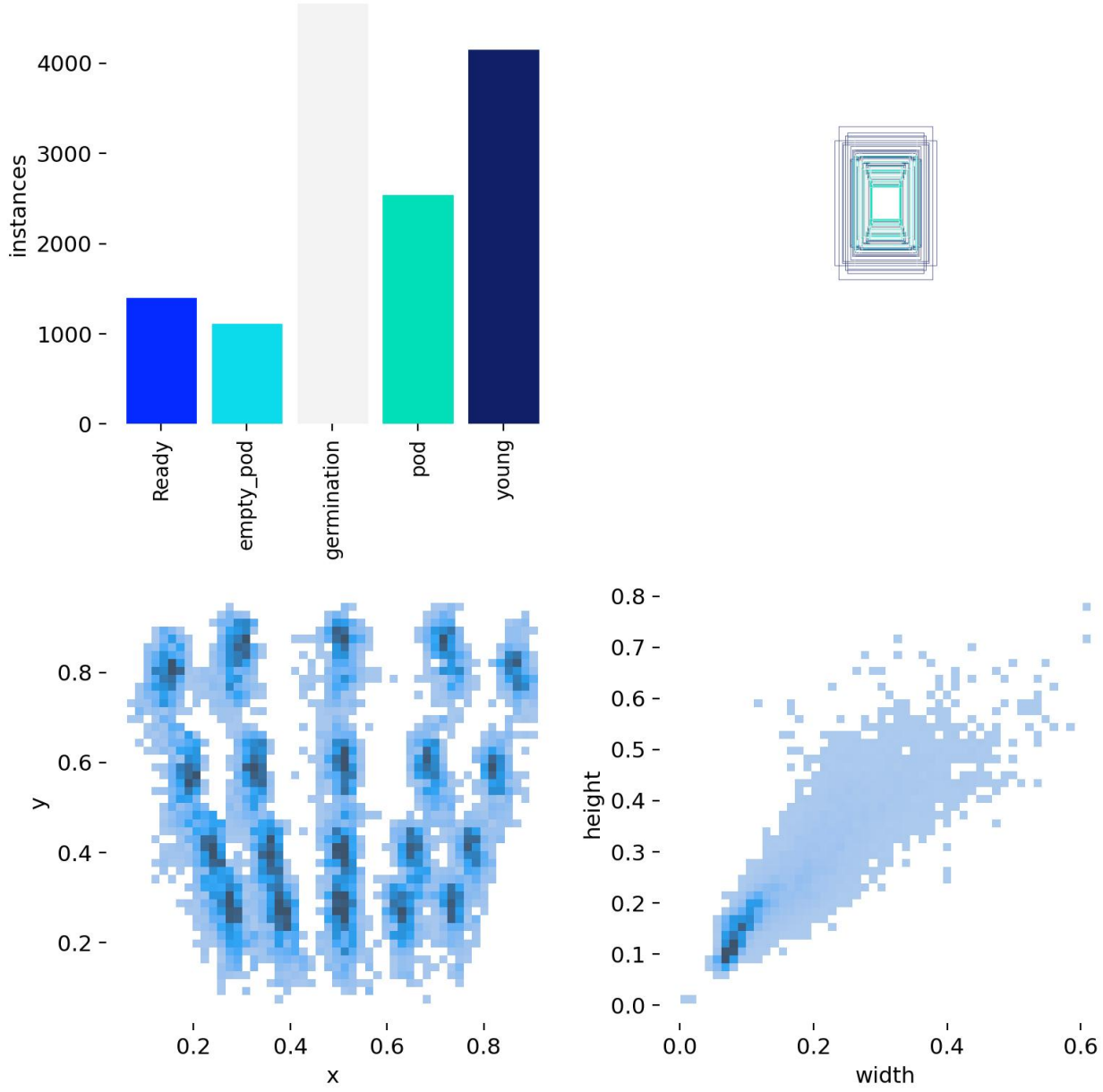


Figure G.20: Labels

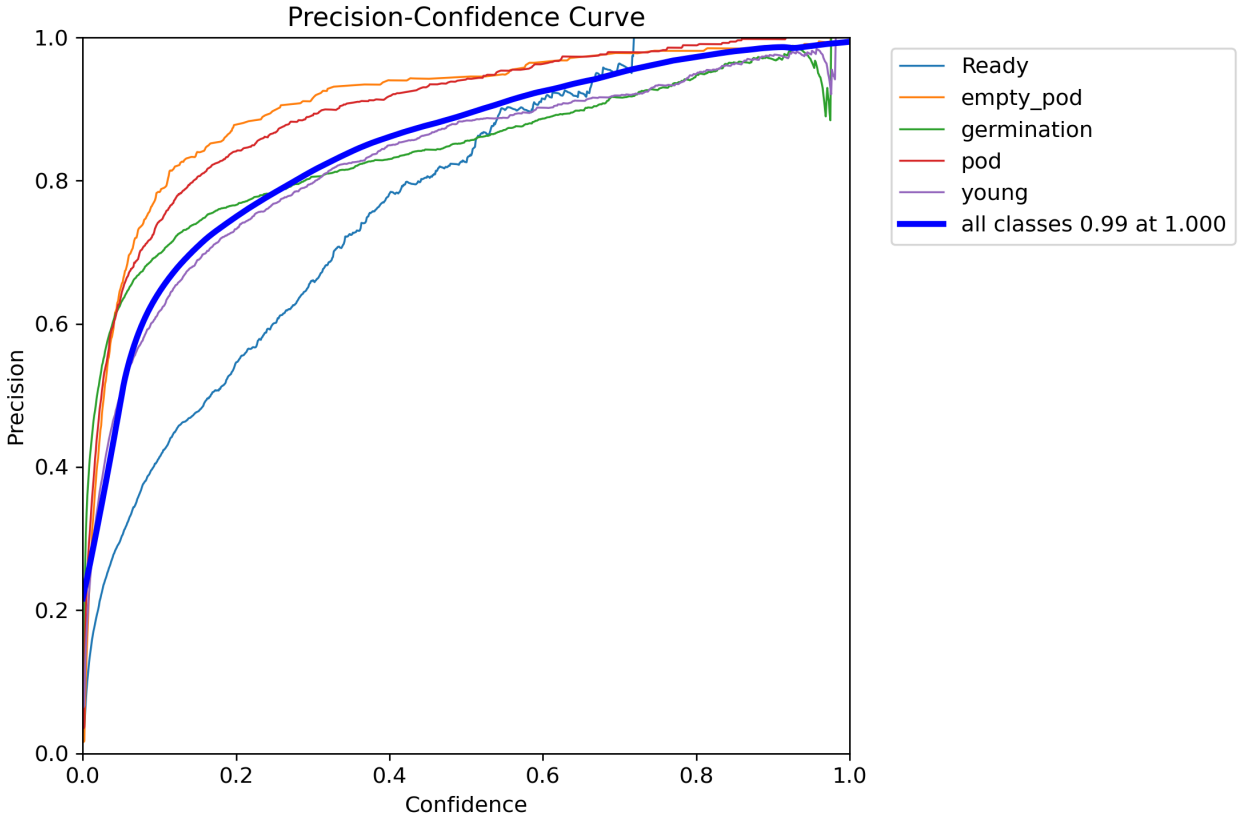


Figure G.21: P curve

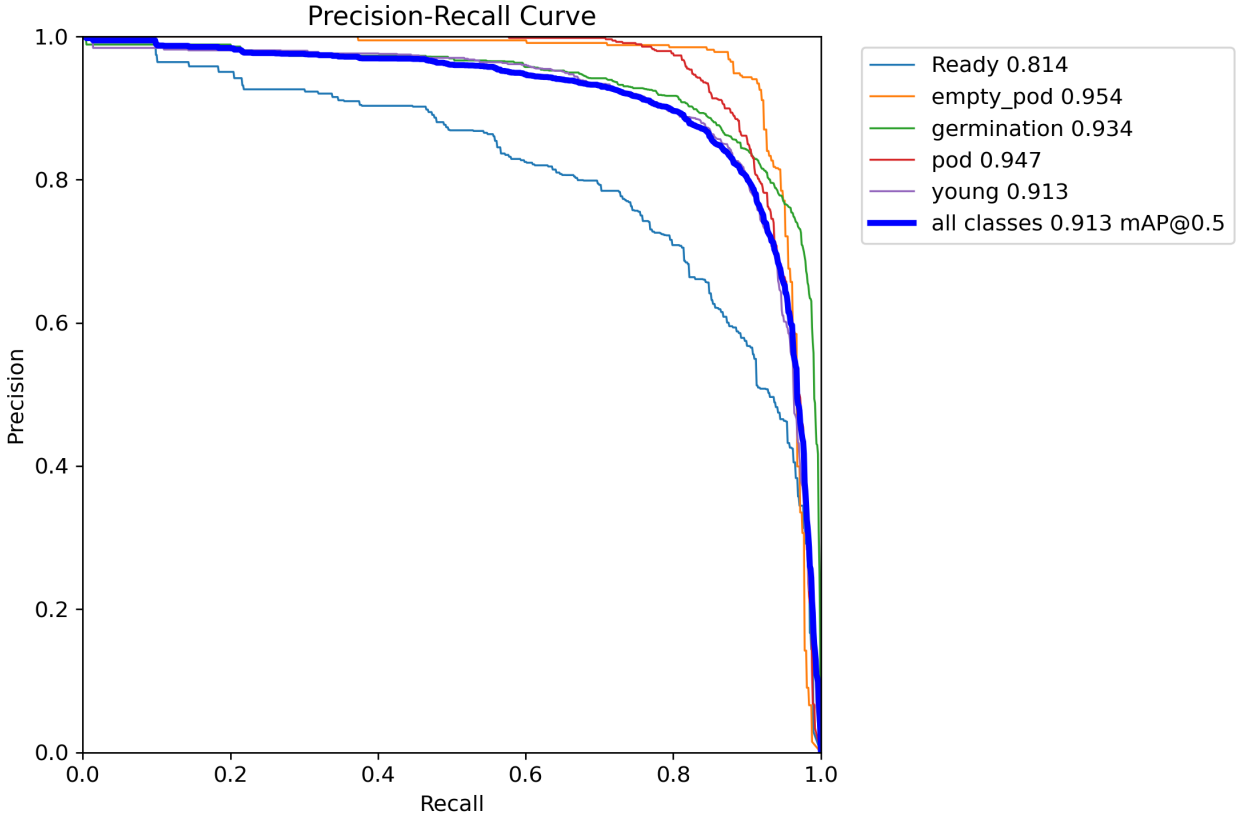


Figure G.22: PR curve

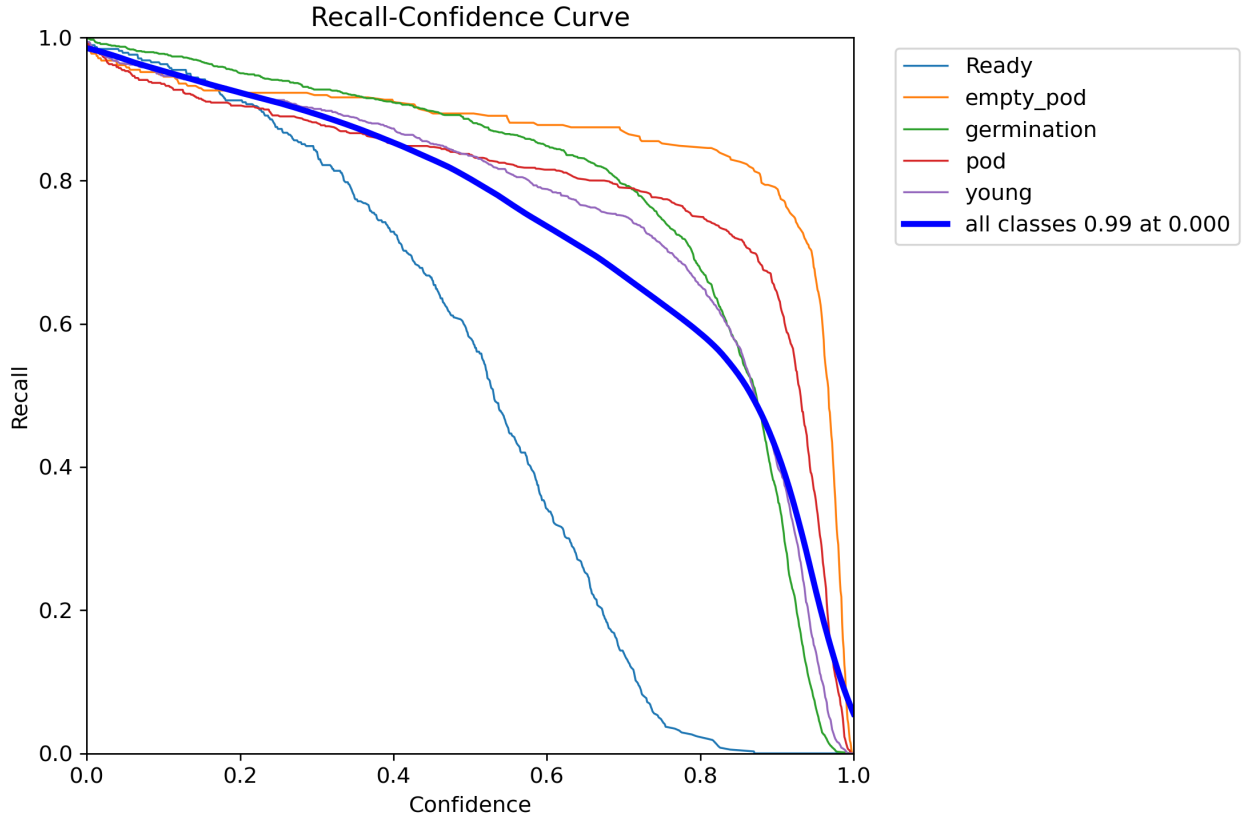


Figure G.23: R curve

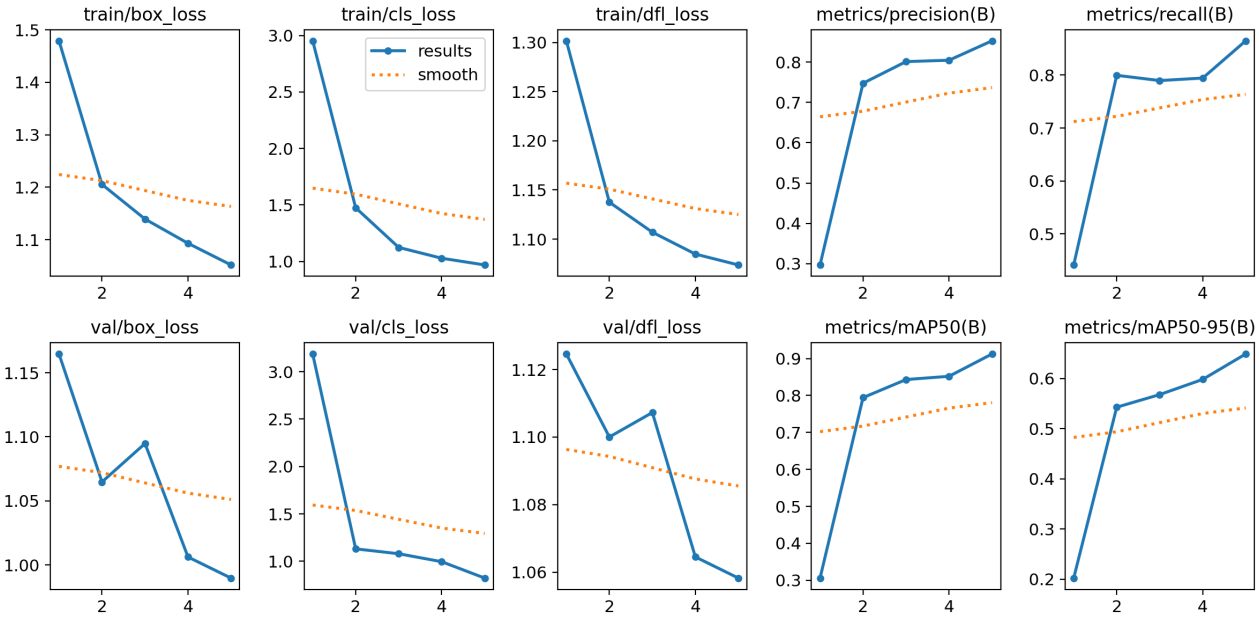


Figure G.24: Results

5 In-depth AI training results (100 epochs)

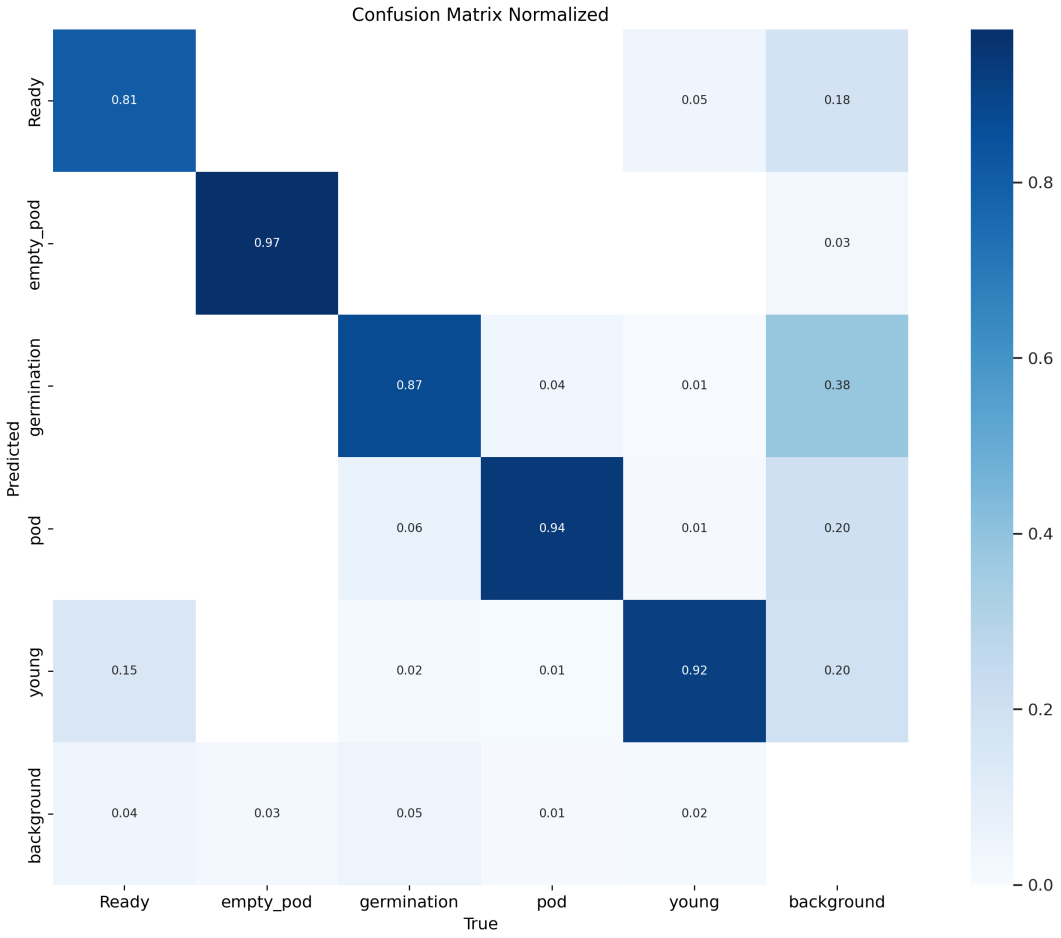


Figure G.25: Confusion matrix - normalized

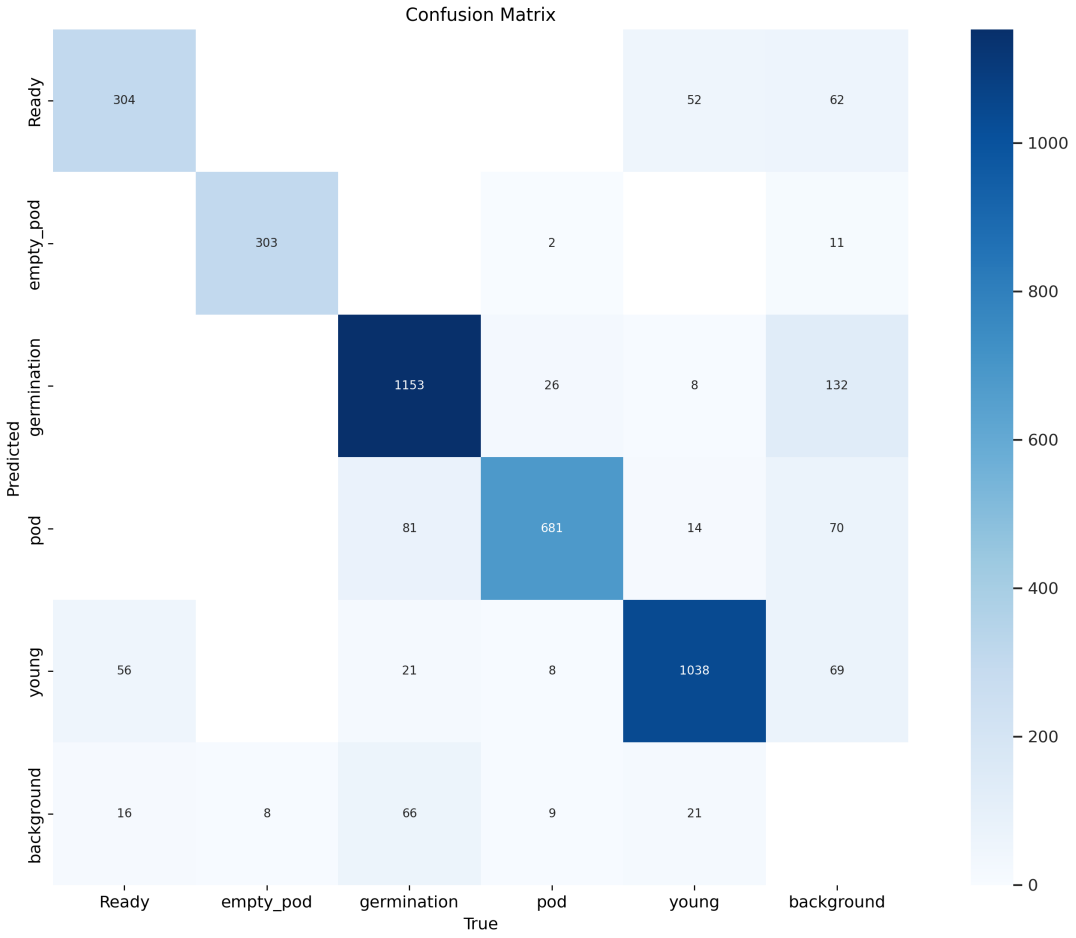


Figure G.26: Confusion matrix

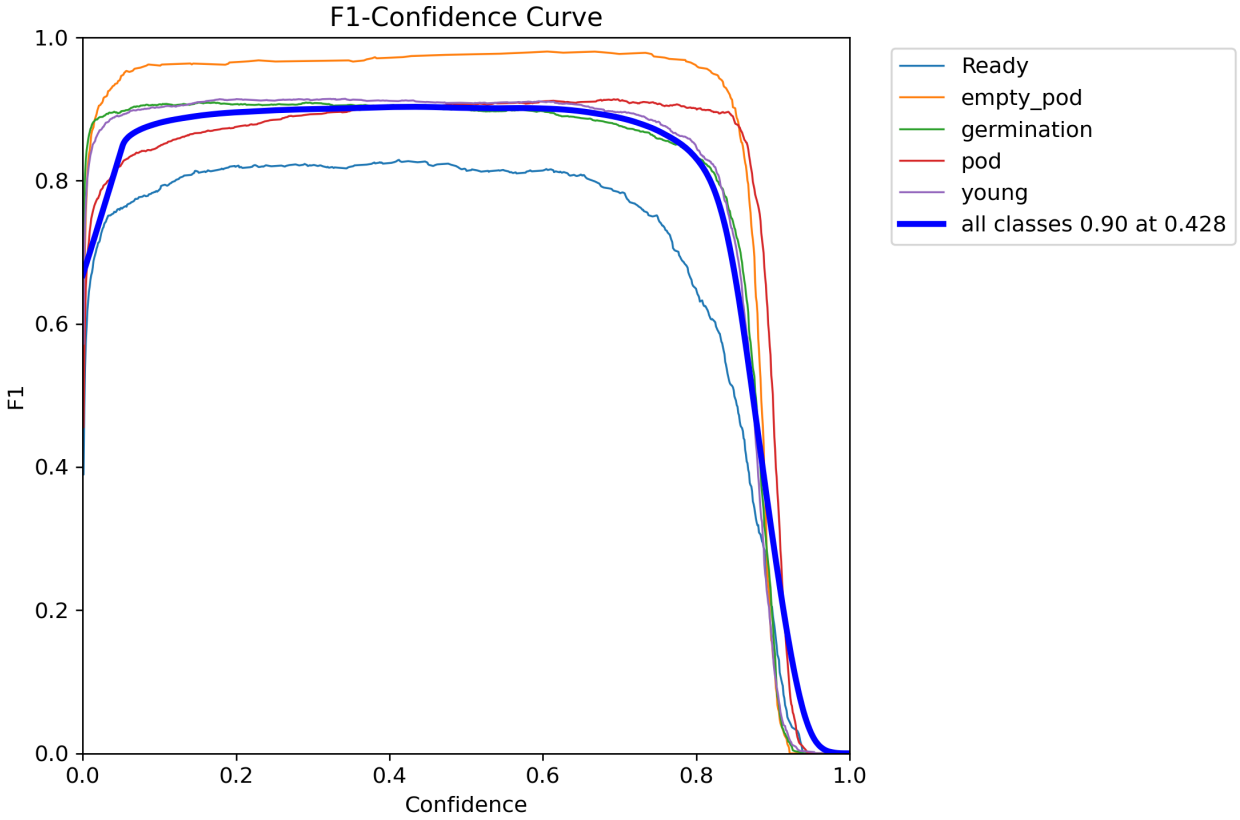


Figure G.27: F1 curve

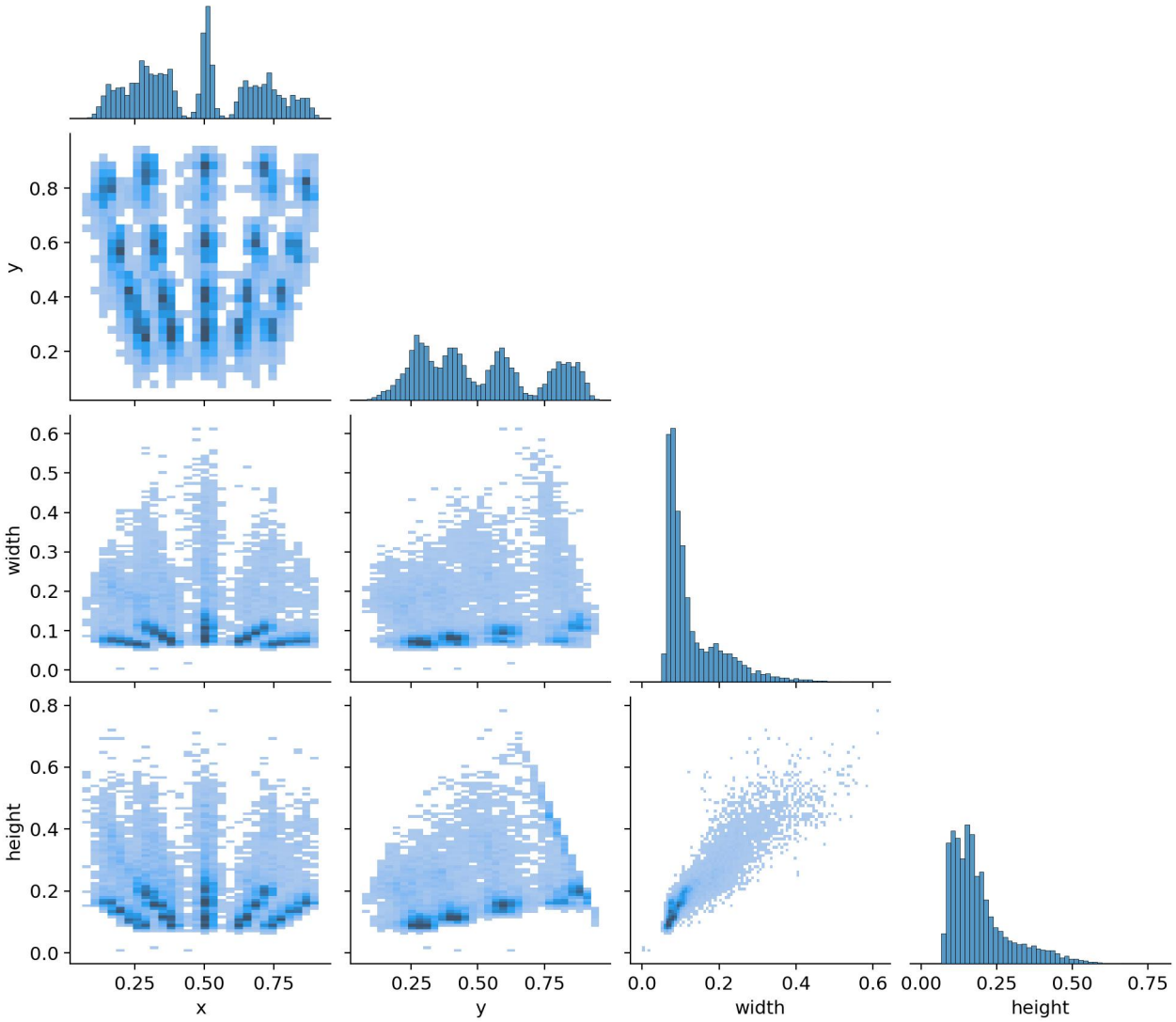


Figure G.28: Labels correlogram

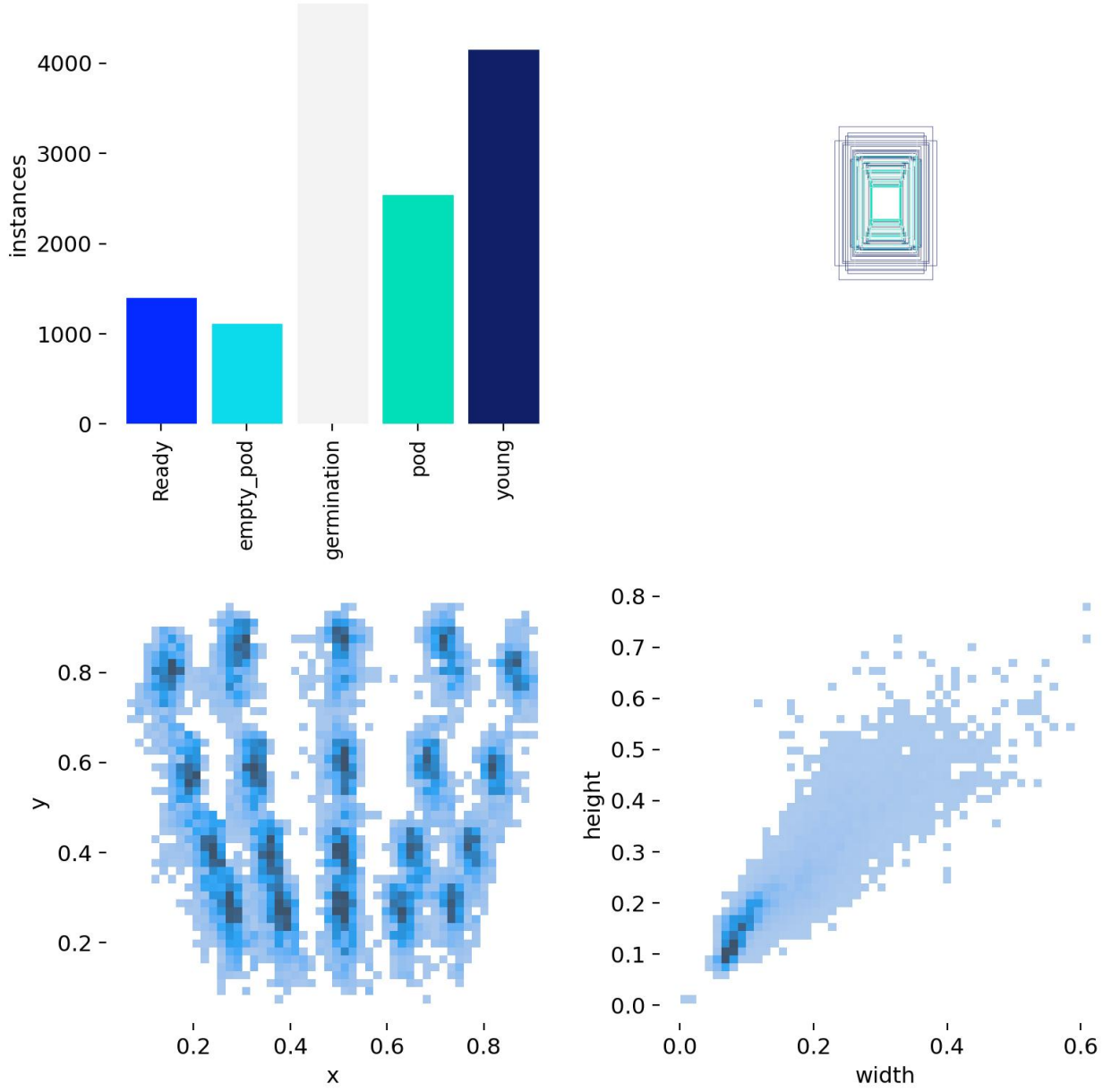


Figure G.29: Labels

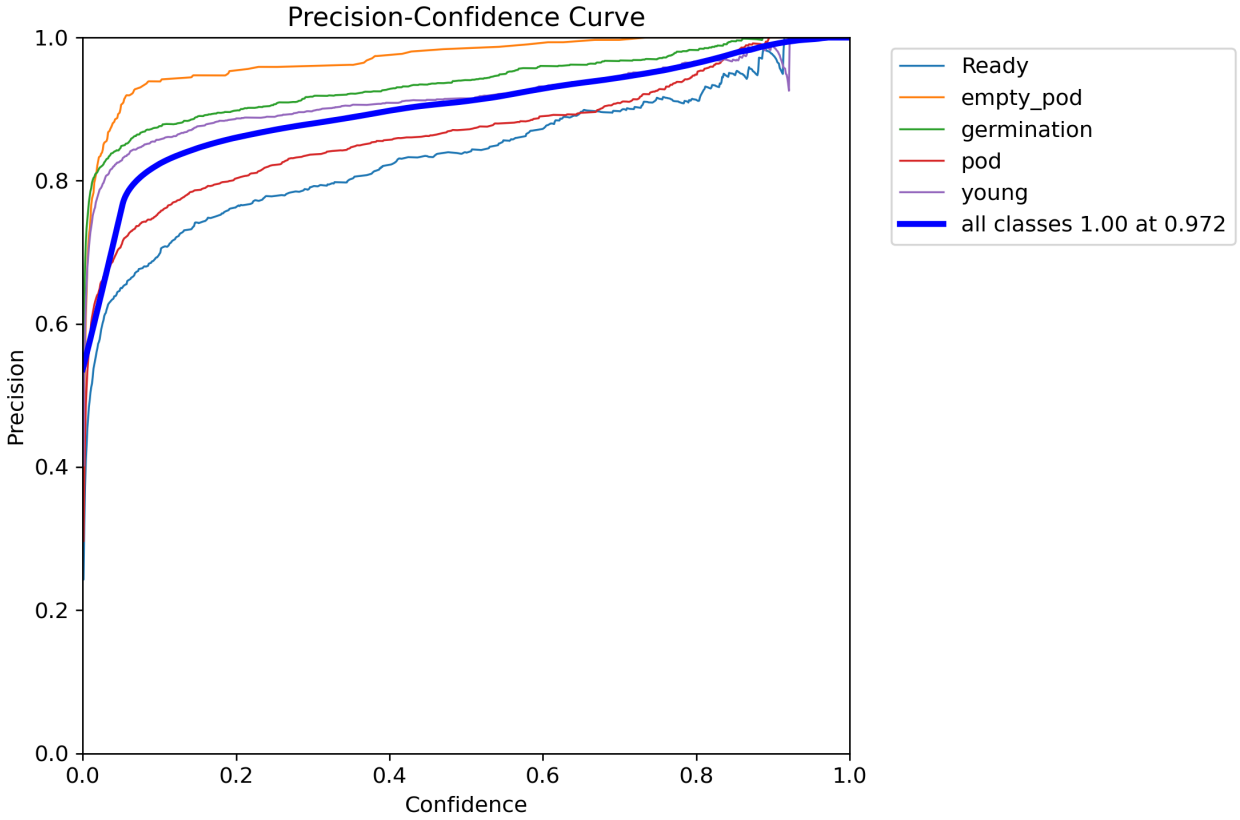


Figure G.30: P curve

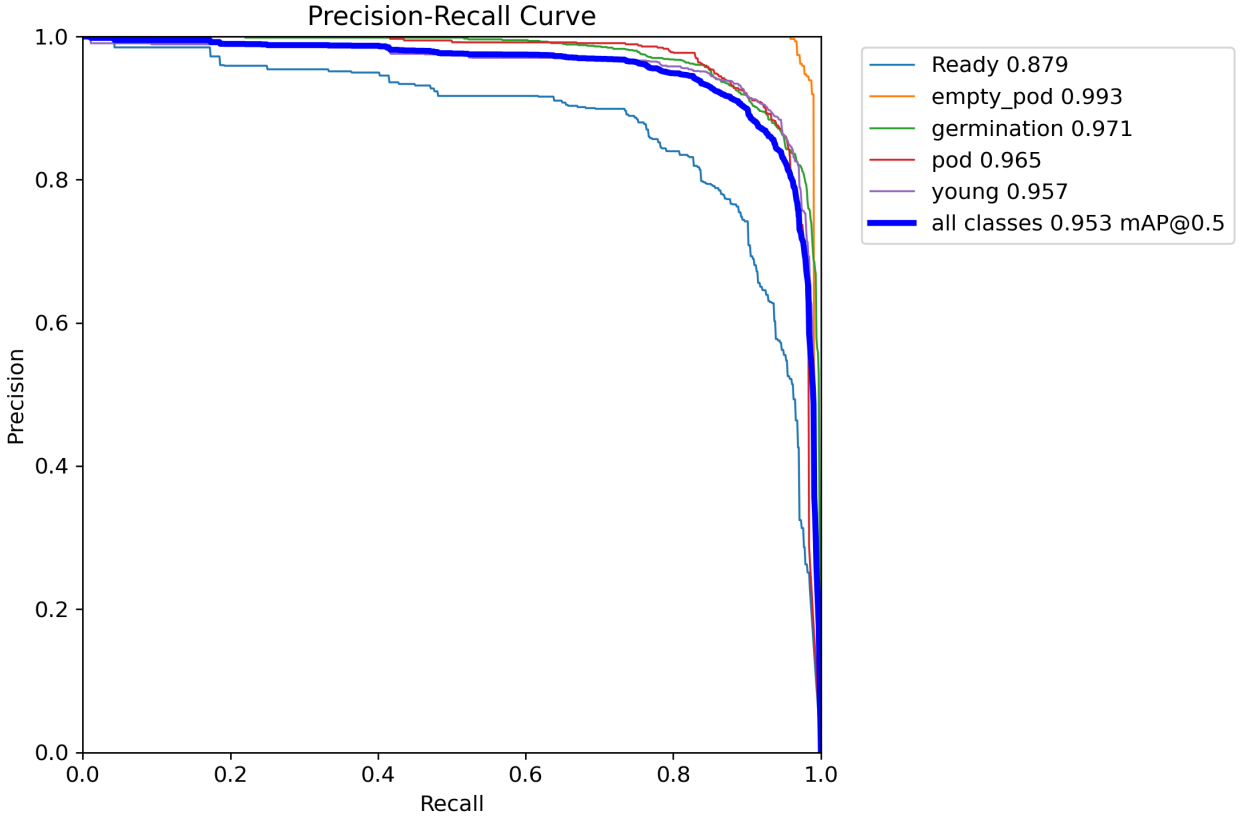


Figure G.31: PR curve

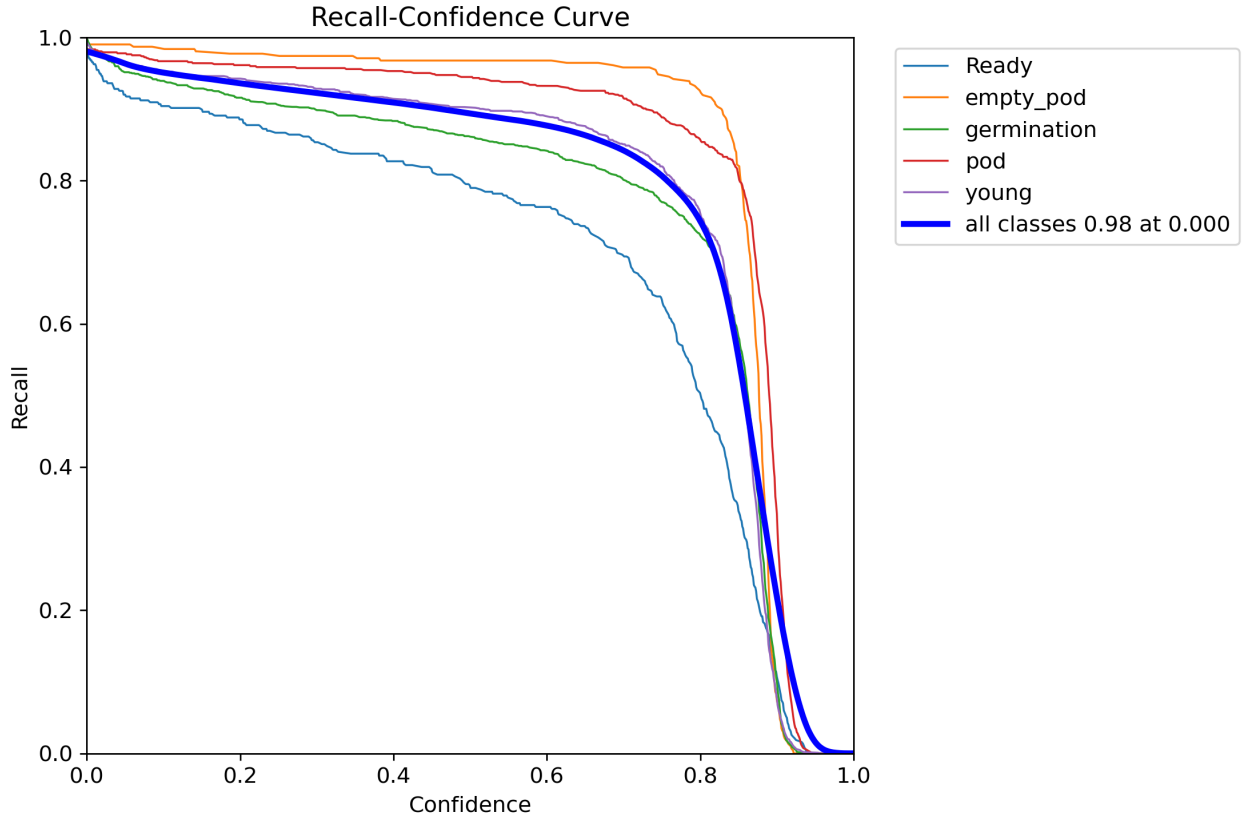


Figure G.32: R curve

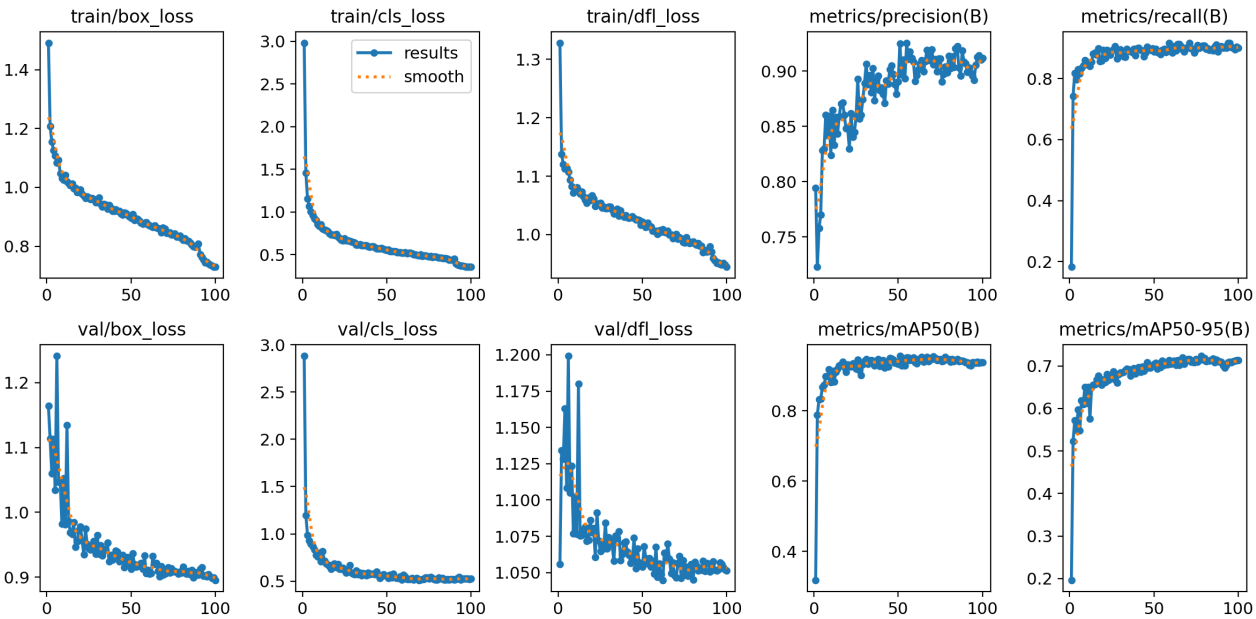


Figure G.33: Results

6 Benchmarking

JCDH | -

Benchmarking is an important concept in software development. We use it to check the time that a task takes, and then use this as a baseline for further improvements to the execution time of the task.

A custom Benchmark class was developed for the Python code, which has a simple Application Programming Interface (API). A simple usage example can be seen in Figure G.34.

```
from util.benchmark import Benchmark

benchmark = Benchmark("Object Detection Benchmark", 100)

benchmark.start_lap()
# The code to benchmark.
benchmark.end_lap()
```

Figure G.34: Python Benchmark Example

Once the requested amounts of iterations (laps) has been achieved, the Benchmark class prints the average execution time of all iterations and stores a MatplotLib plot as an image in the project repo with the format seen in Figure G.35 [10].

```
[normalized title]-[epoch time].png
```

Figure G.35: Python benchmark plot naming format

The average execution time and standard deviation (which gives us an idea of the volatility between each iteration of the benchmark) is calculated using standard formulas.

[10] **normalized title** means replacing spaces with underscores in this case. **epoch time** means seconds since 1. jan. 1970, which is a common way to track time and to ensure an unique filename

7 HTTP / ESP32-Cam benchmarking

JCDH | -

7.1 Benchmarking

JCDH | -

Initial testing shows HTTP image transfer (POST request from the ESP32-CAM) latency to be acceptable, but its incapable of providing a video feed. Although having a video feed is not a hard requirement (as per our requirements in Section **??**), it gives the users of the HMI instant feedback, which could improve the user experience. The following two benchmarks where done using frame size UXGA (1600 x 1200 px), and HTTP Connection: close.

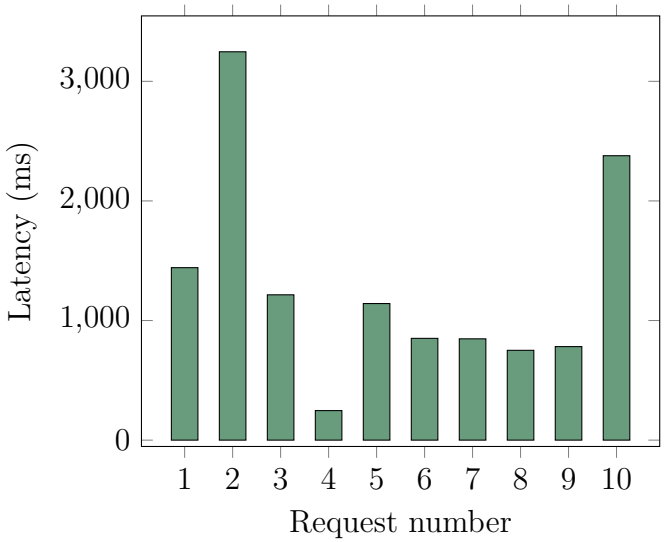


Figure G.36: HTTP request latency measurements

From what can be gathered, the time it takes to send a picture over HTTP can vary greatly. This seems to be related to many different variables like CPU usage, memory allocation etc. The HTTP request latency measurement diagram above shows 10 measurements done in succession with a 10 second delay between each request (this was done to make sure the previous request had time to finish, as this is not a load test. Load testing comes in a later section). If we take the average of these measurements we find that avg. request latency was **1290.2 ms**, or 1.2902 seconds.

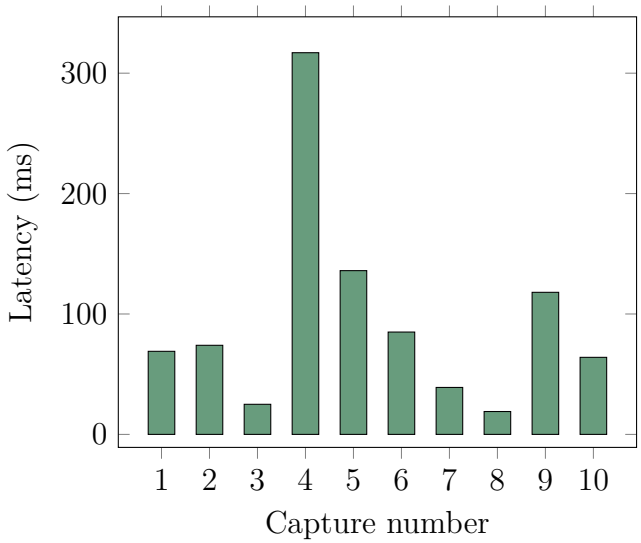


Figure G.37: Image capture latency

The image capture latency diagram above shows us time it takes for the esp32-cam sensor to capture a picture. Assumptions can be made that time it takes will depend upon variables like lighting conditions, etc. This sample of 10 measurements was taken in dim lighting conditions in a normal living room, with a 10 second delay between each measurement. If we take the average of these measurements we find that avg. image capture latency was **94.6 ms**, or 0.0946 seconds.

7.2 Further optimizations

JCDH | -

Further optimizations using was done, which included setting HTTP Connection to keep-alive, waiting 100 ms (instead of 10 seconds) between each sent image and framesize to VGA (640x480) which gave more promising numbers (although image quality and system stability suffered):

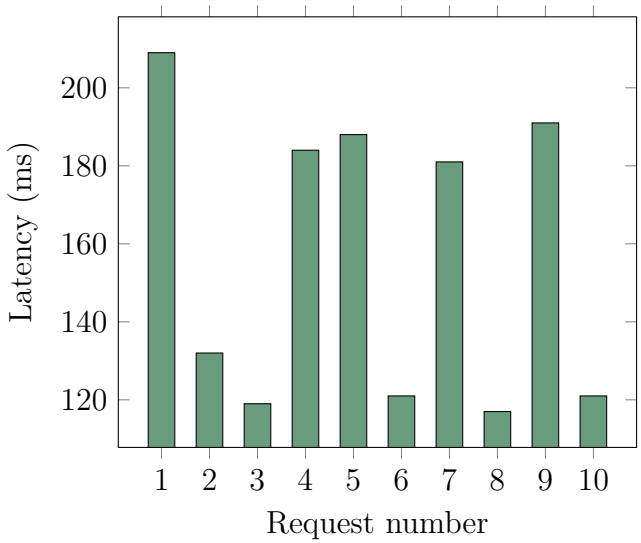


Figure G.38: HTTP request latency measurements (trial #2)

Average latency here was 156.3 ms or 0.1563 seconds.

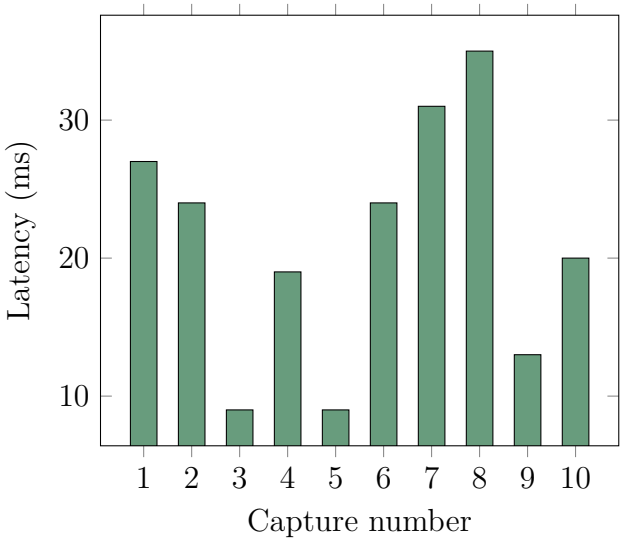


Figure G.39: Image capture latency (trial #2)

Average capture time here was 21.1 ms or 0.0211 seconds.

These averages gives us a baseline to work from when doing further optimizations.

7.3 HTTP load testing

JCDH | -

Initial testing shows no major concerns with the amount of requests being sent over the network. In other words this is not a bottle neck in our system the way its meant to be used.

8 Demo of working HTTP communication between Core and Central

JCDH | -

Further work with Leafy Automation Core was done. A custom “network stack” was created and tested. This communicates with Leafy Automation Central over the network (port 5000 for now) as illustrated in Figure G.40.

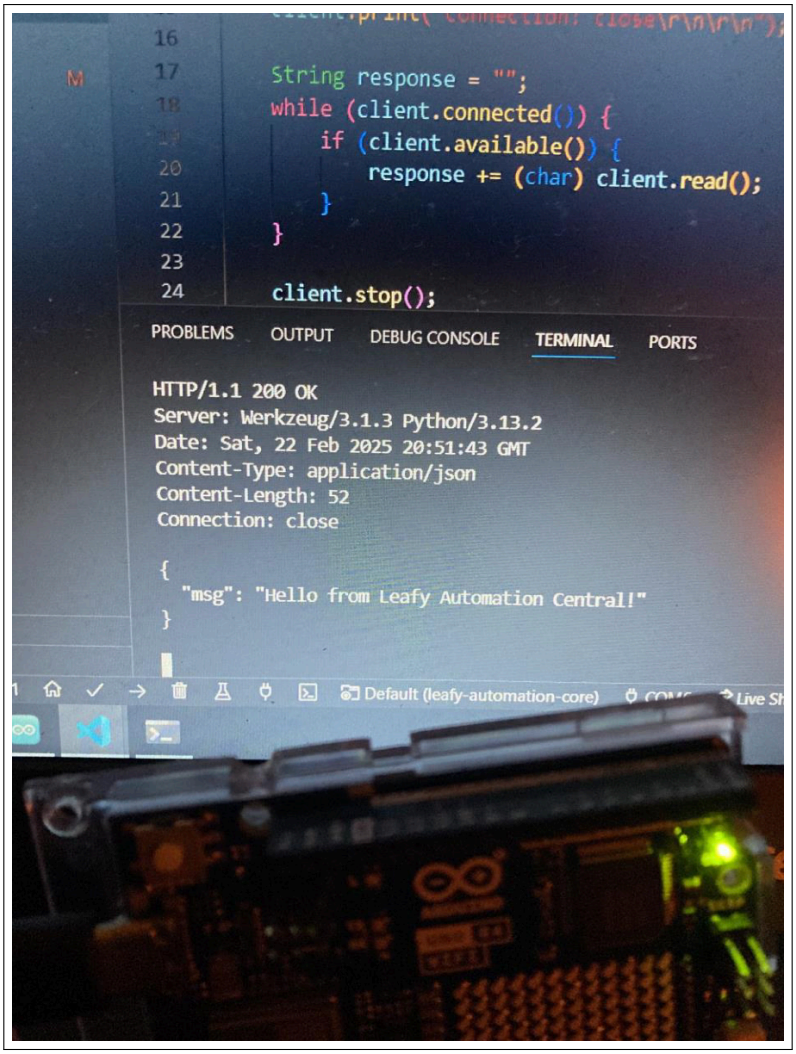


Figure G.40: HTTP initial testing

9 Leafy Automation Core code restructuring

JCDH | -

A change was made to the code structure to more easily facilitate for a modular approach to development. Figure G.41 shows the code structure before, and Figure G.42 shows the code structure after. [11]

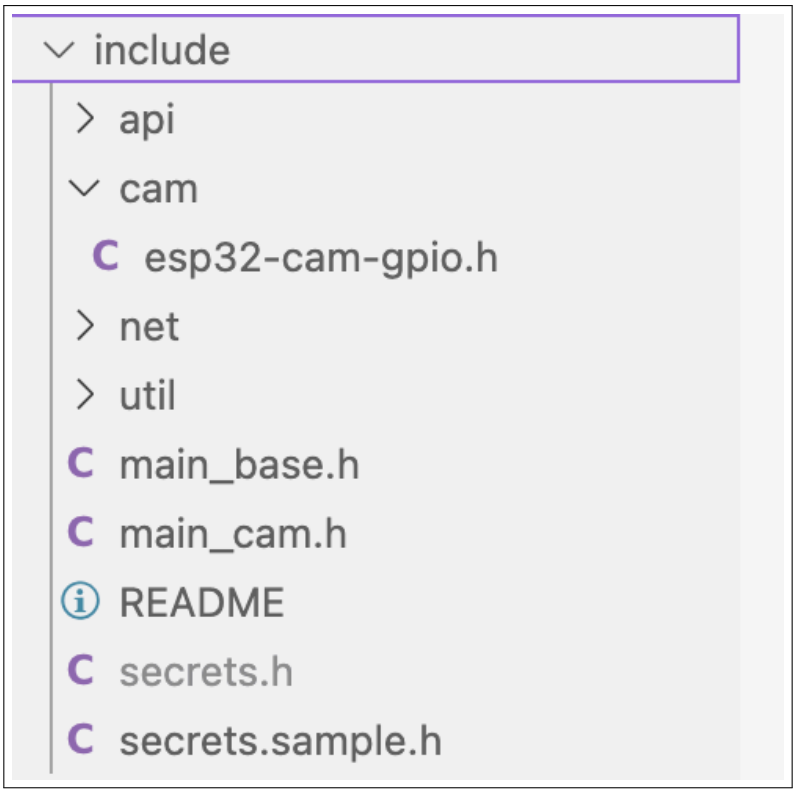


Figure G.41: Leafy Automation Core code restructuring - before

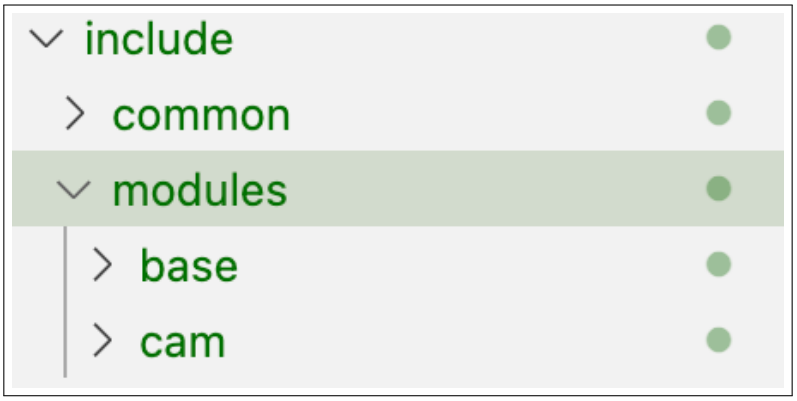


Figure G.42: Leafy Automation Core code restructuring - after

[11] Changes made on 20.03.2025

10 HMI HTTP Camera Feed

JCDH | -

The old HMI HTTP camera feed captures a new frame using a polling technique that runs every 2 seconds. It's a simple, yet effective method to transmit data, but struggles with real-time use cases.

10.1 Frontend

10.2 CameraController.js

```
export class CameraController {
    constructor(model, view) {
        this.model = model;
        this.view = view;

        this.updateFeed();

        setInterval(() => this.updateFeed(), 2000);
    }

    updateFeed() {
        this.view.render(this.model.getFeed());
    }
}
```

Listing G.1: Old HMI HTTP CameraController.js

10.2.1 CameraModel.js

```
export class CameraModel {
    getFeed() {
        return `/api/v1/camera-feed?seed=${new Date().getTime()}`;
    }
}
```

Listing G.2: Old HMI HTTP CameraModel.js

10.2.2 CameraView.js

```
export class CameraView {
    constructor() {
```

```
        this.cameraFeed = $("#camera-feed");
    }

    render(data) {
        this.cameraFeed.attr("src", data);
    }
}
```

Listing G.3: Old HMI HTTP CameraView.js

10.3 Backend

10.4 api.py

```
...
@routes.route("/camera-feed", methods=["GET"])
def camera_feed():
    return send_from_directory("cache", "camera01.jpg")
...
```

Listing G.4: Old HMI HTTP api

11 HMI dashboard v1

JCDH | -

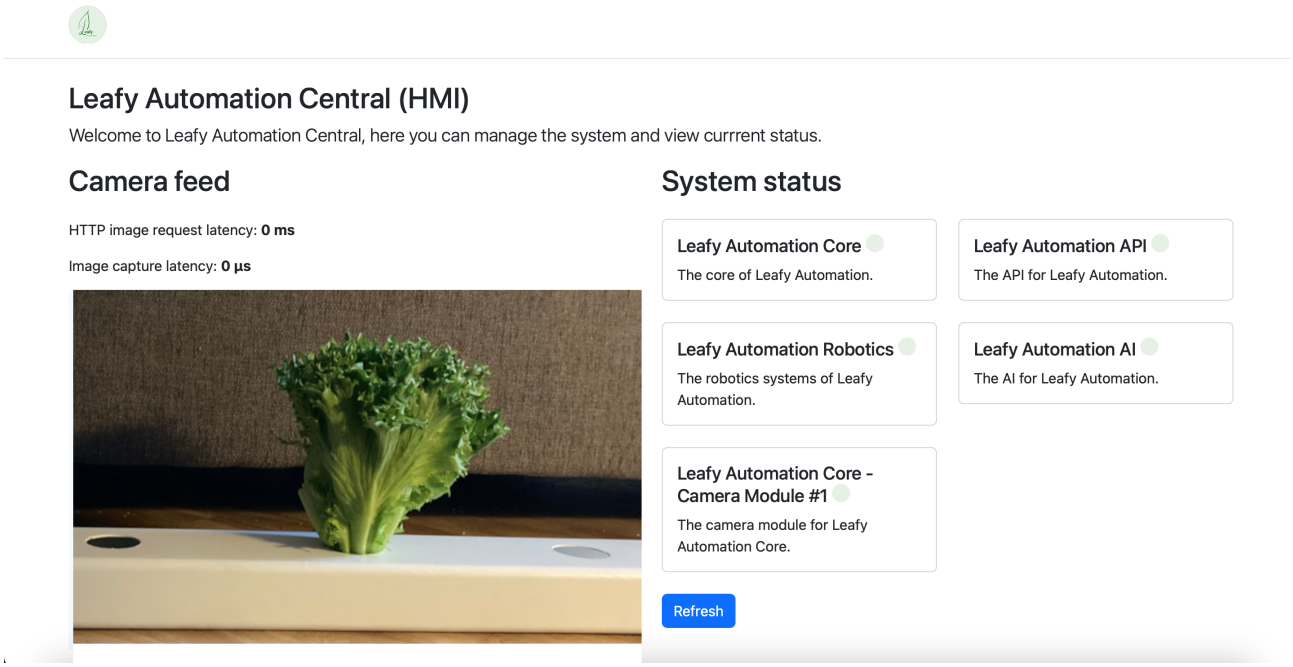


Figure G.43: HMI dashboard v1

12 Database side notes

JCDH | -

How we connect to the database

SQLite stores the database as a file on the file system. This stands in contrast to more complex database application such as MySQL, which runs as a separate server instance and manage databases through traditional client-server communication [83].

As shown in Figure G.44, a connection is established to a specific file on the file system, which represents the database.

```
import sqlite3

class DB:
    """!
    @brief Class for managing SQLite database connection.
    """

    @staticmethod
    def get_connection():
        """
        Get a new database connection.
        @return: SQLite connection object.
        """

        connection = sqlite3.connect("central.db")
        ...

        return connection
    ...
```

Figure G.44: Python DB connection

Migrations

The database migrations are specific queries that run the first time the application starts, or after upgrades to the codebase. They often do tasks like creating new tables or adding / removing columns from existing tables. In the case of Leafy Automation Central, migrations are created for the **users** table and **access_levels** table. Figure G.45 shows how the migration for the "users" table looks like.

```sql
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    access_level_id INTEGER NOT NULL,
    FOREIGN KEY (access_level_id) REFERENCES access_levels(id)
)
```

Figure G.45: SQlite database migration example

Debugging the database

Debugging the SQLite database is easy with the tool we use, which is called *DB Browser for SQLite*. This tools allows you to view all data corresponding to the database tables in a Graphical User Interface (GUI), as seen in Figure G.46 [84].

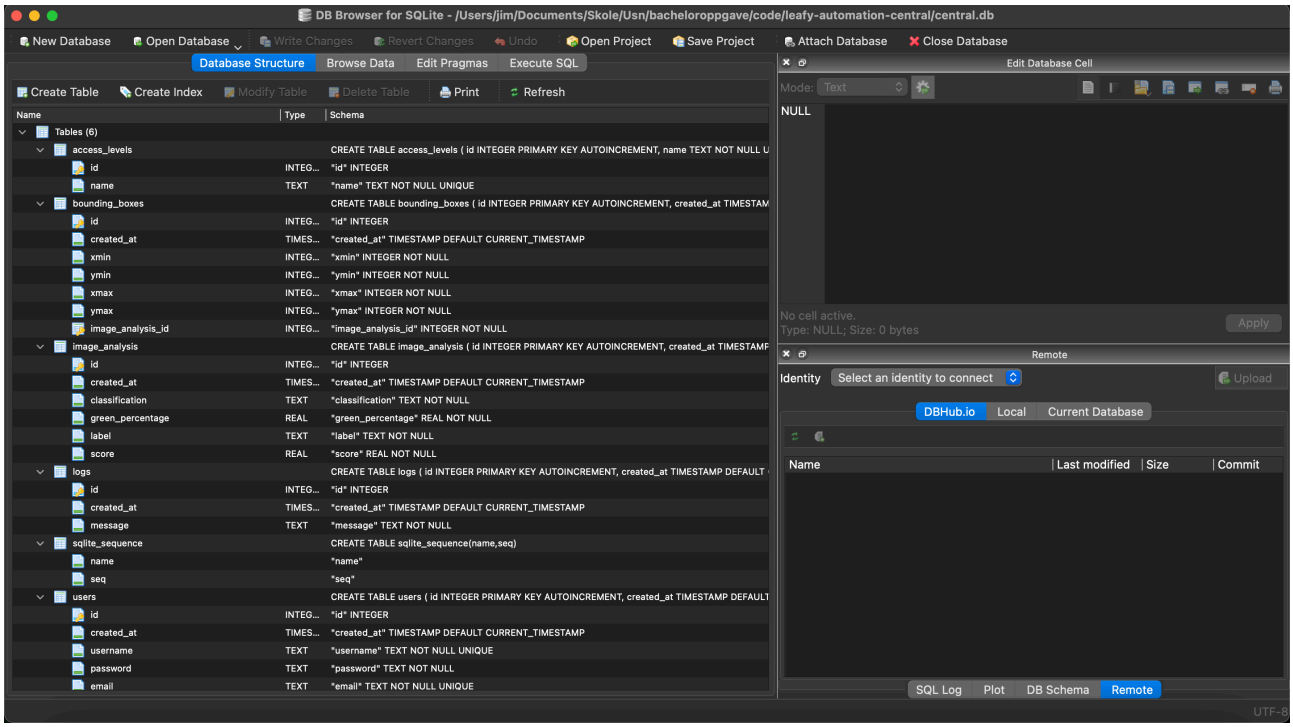


Figure G.46: DB Browser for SQLite example

A note on thread safety

Each request in Python Flask creates a new thread. An earlier iteration of the **DB** class used the singleton pattern for the database, but because of the fact that a new thread is created for

each request, a single database connection between threads was deemed not to be thread safe (deadlocks, race conditions, and data integrity issues could occur). Therefore, each thread has its own connection to the database.

For high-traffic applications, creating a new database connection on each request could introduce performance issues, but this is a non-issue until specific evidence points to the contrary. It could also be argued that creating a new thread for each request, like how Python Flask is architectured would make it hard to scale for high-traffic applications, but for our use case it has been deemed perfectly fine.

13 In-progress database work

JCDH | -

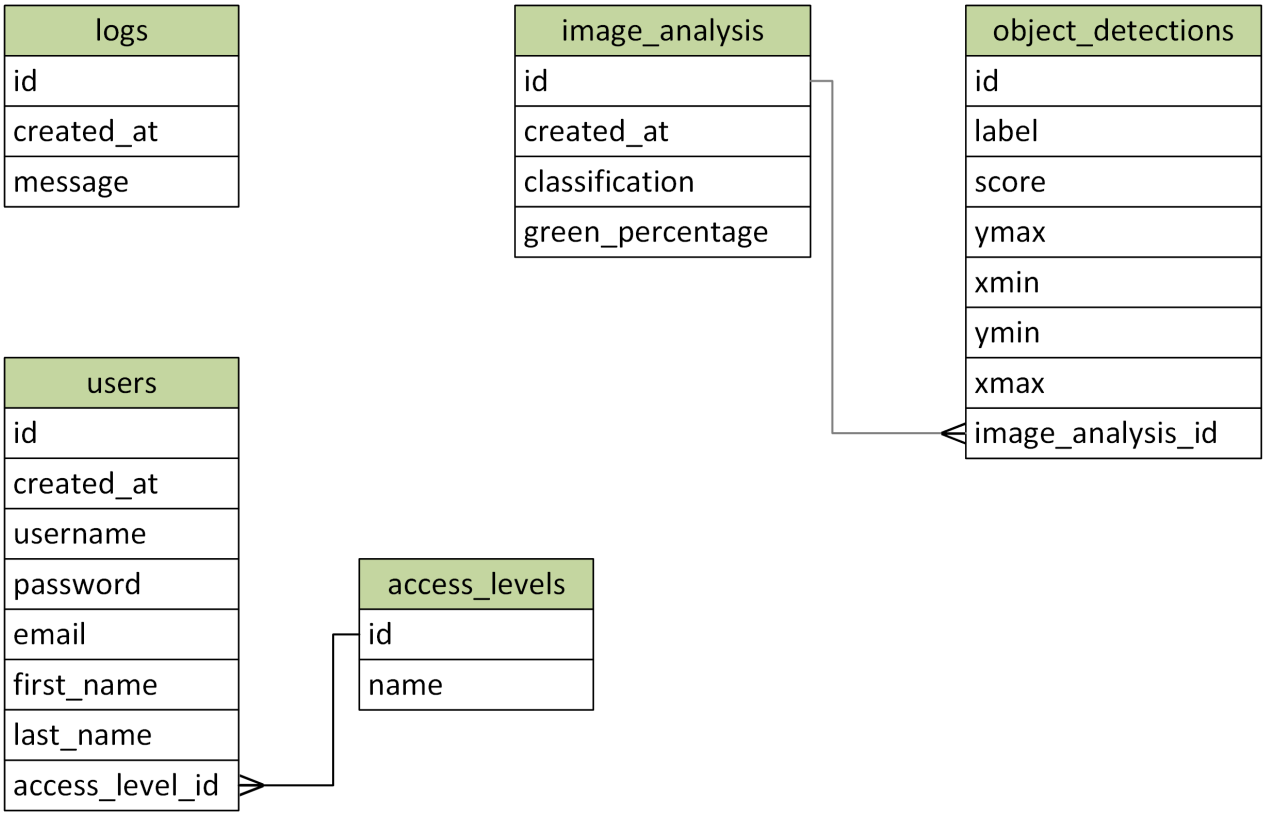


Figure G.47: Database overview diagram

Note: Arrows indicate foreign key relationships.

13.1 Logs table (logs)

JCDH | -

The **logs** table is responsible for storing system logs and messages. Devices like the Arduino and ESP32-CAM report on their current status which is then sent to and stored in the logs table.

It contains information like the timestamp of the message and the message itself.

Name	Description	Datatype	Metadata
id	The unique id of the specific log entry	INTEGER	AUTO INCREMENT, PRIMARY KEY
created_at	The time at which the log entry was created.	TEXT	CURRENT_TIMESTAMP
message	The log message	TEXT	NOT NULL

Table G.1: Database: Logs table structure

id	created_at	message
1	1743357828	picture_taken

Table G.2: Database: Logs table example

13.2 Image analysis table (image_analysis)

JCDH | -

Name	Description	Datatype	Metadata
id	unique id	INTEGER	*AUTO INCREMENT, PRIMARY KEY*
created_at	The timestamp for when this was stored	TEXT	*CURRENT_TIMESTAMP*
classification	The image classification	TEXT	NOT NULL
green_percentage	Amount of green hue in the image	REAL	NOT NULL
label	The label from the object detection	TEXT	NOT NULL
score	Confidence score for the object detection	REAL	NOT NULL

Table G.3: Database: Image analysis table structure

id	timestamp	classification	green_percentage	label	score
1	1743357828	lettuce	47.2	broccoli	0.83

Table G.4: Database: Image analysis table example

13.3 Bounding boxes table (bounding_boxes)

JCDH | -

Name	Description	Datatype	Metadata
id	unique id	INTEGER	*AUTO INCREMENT, PRIMARY KEY*
xmin	Boundingbox xmin	INTEGER	NOT NULL
ymin	Boundingbox ymin	INTEGER	NOT NULL
xmax	Boundingbox xmax	INTEGER	NOT NULL
ymax	Boundingbox ymax	INTEGER	NOT NULL
image_analysis_id	connects the bounding box to the specific image_analysis row	INTEGER	*FOREIGN KEY to image_analysis.id*, NOT NULL

Table G.5: Database: Bounding boxes table structure

id	image_analysis_id	xmin	ymin	xmax	ymax
1	1	10	8	150	100

Table G.6: Database: Bounding boxes table example

14 API JSON schema

JCDH | -

The Leafy Automation Central API communicates using the JSON format for request and response payloads. This format was chosen because of its easy of use, group experience, simplicity, compatibility and availability of coding libraries.

```
{
    "img-capture-time": 22,
    "img-capture-req-time": 243,
    "image-classification": "green lettuce",
    "green-percentage": 0.43,
    "log": ["System connected to WiFi", "Picture taken"]
}
```

Listing G.5: JSON response example

14.1 API status codes

JCDH | -

The Leafy Automation Central API makes use of HTTP status code. HTTP status codes are a collection of codes which tells you something about the success / failure of a response. As an example response code *200* indicates success, while other codes like *404* or *500* indicates that something was not found, or an internal server error occurred, respectively.

14.2 API routes

JCDH | -

API routes defines the routes used for the web server, which the Leafy Automation Core communicates with. This API architecture is written in a REST API style.

Route	Method	Description	Arguments	Returns
/api/v1	GET	Index route, used for pinging the system to check connection status	{}	{}
/api/v1/status	GET	Returns current status of the system	{}	{"img-capture-time": ..., "img-capture-req-time": ..., "image-classification": ..., "green-percentage": ..., "log": ...}
/api/v1/log-stats	GET	Log information related to Core latency	**image-capture-time** (Time it took to capture an image), **image-capture-req-time** (Time it took to send the image to the Central)	{"msg": "Data captured"}
/api/v1/log	GET	Stores the supplied message in a log datastructure	**msg** (The message to log)	{}
/api/v1/camera-feed	GET	Gets the latest image in the buffer as a jpg	{}	*[binary-data]*
/api/v1/capture-image	POST	Stores the supplied image in the buffer as jpg	*[binary-data]*	{"msg": "Image saved"} | {"msg": "No image data provided"}, 400
/api/v1/classify-image	POST	Classifies an image using AI model	*[binary-data]*	{"class": ...}

Table G.7: Leafy Automation Central - API routes

15 Specialized Computer Vision with OpenCV and PlantCV

JCDH | -

15.1 Understanding OpenCV and PlantCV

JCDH | -

OpenCV and PlantCV are widely used Computer Vision (CV) libraries for analyzing images. They have good documentation, and interface with most programming languages [85]. PlantCV is built on top of Computer Vision (CV), but contains a more specialized set of functions for plant specific tasks [86].

15.2 Color segmentation

JCDH | -

Another interesting technology, which has been explored (partly because of its simplicity) is **color segmentation**. This allows us for our use case to figure out the percentage of green color in an image. Our theory is that this information can be used as a data point which tells us how close the camera (if placed on a robot arm) is to the plant.

Application in robotic proximity detection

By analyzing the proportion of green pixels detected in an image, we hypothesize that the relative distance between the camera and the plant can be inferred. As the robot arm moves closer to the plant, the percentage of green pixels should increase (green percentage). This provides us with a simple, yet effective data point.

Conversion from RGB to HSV color space

Images are normally coded in the Red, Green, Blue (RGB) color space. Because Hue, Saturation, Value (HSV) provides a better format for color extraction, and natural light conditions, it is converted [87].

Color thresholding

Color thresholding is used to only extract the colors within a specified range (in this case green colors), so that anything in the image not of interest is ignored.

Normalization and scaling

Finally, in order to calculate the percentage of "green" pixels in relation to other colors, the formula defined in Equation G.1 is used.

$$\begin{aligned} P_{green\_percentage} &= \frac{N_{green}}{N_{total}} \times 100 \\ N_{total} &= H \times W \end{aligned} \tag{G.1}$$

where H is the height and W is the width of the image, both in pixels.

15.3 Mask Generation

JCDH | -

Figure G.48, G.49 and G.51 shows the steps involved in generating clean masks without any noise.

Figure G.48: Lettuce top-down image

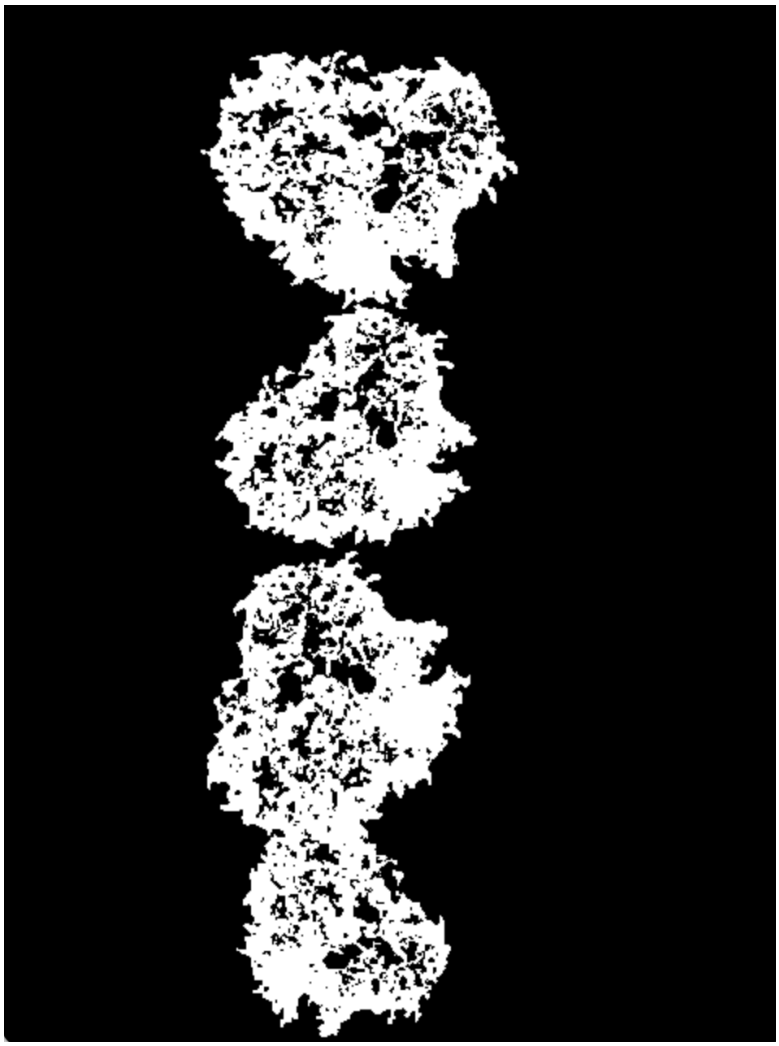


Figure G.49: Lettuce top-down image mask

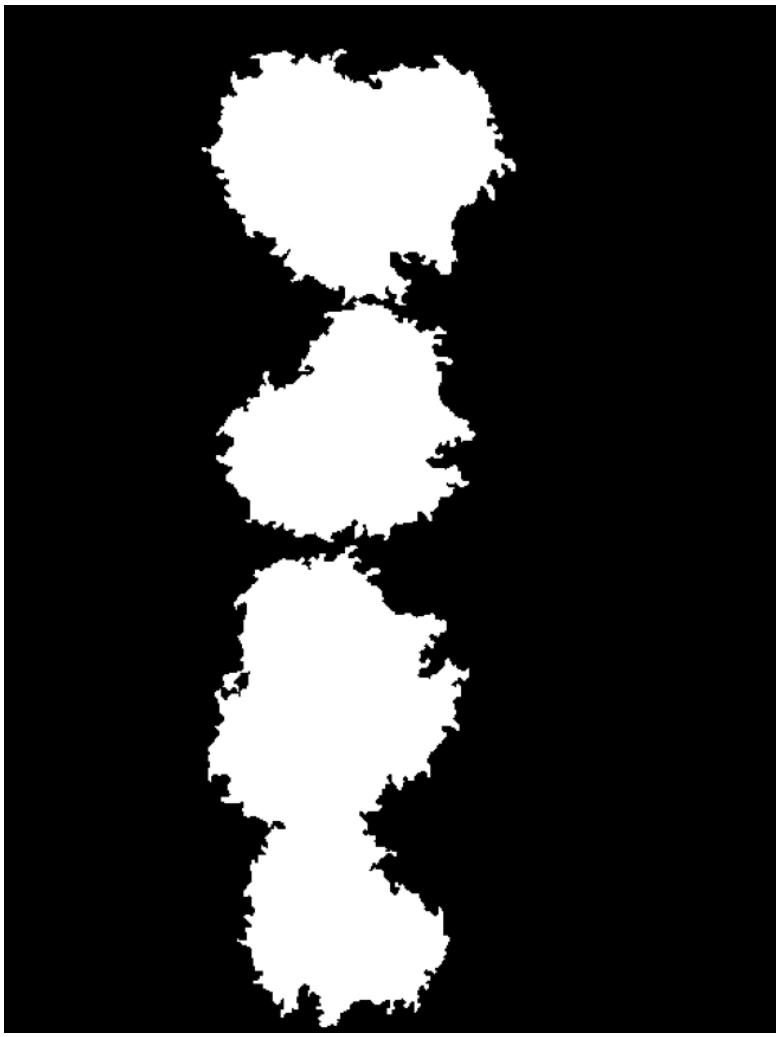


Figure G.50: Lettuce top-down image mask (fill holes and specs of noise)

15.4 Handling overlapping lettuce

JCDH | -

Once masks have been generated, you may encounter overlap, as seen in Figure G.51. [12] This is solved using watershed segmentation [88]. Watershed segmentation treats the image like a landscape with peaks and valleys, where low intensity are valleys, and high intensity parts of the image are peaks. This information can then be used to detect edges (where two peaks meet) [89]. This information gives us a close approximation to the correct boundaries for each lettuce. Figure G.52 shows the drawn on contours based on the results from the watershed algorithm, and looking at the intersecting point between the two lettuce plants, it is clear that an approximation is being done.

[12] An important area of research for Hydroplant Technologies was how to handle overlapping lettuce.

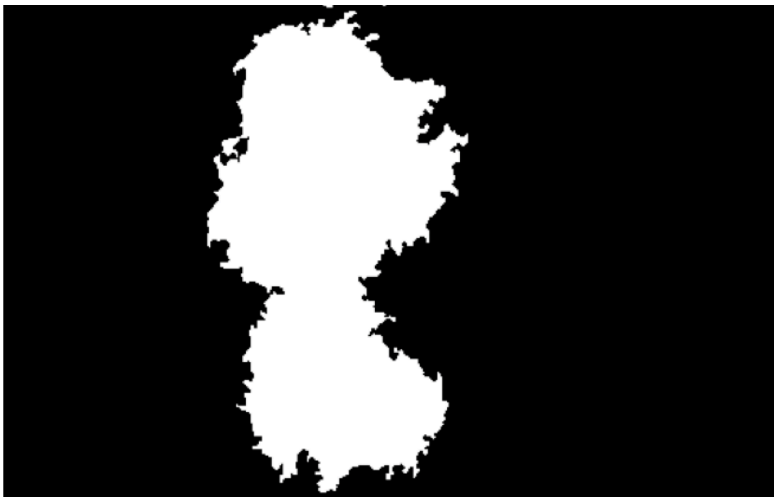


Figure G.51: Lettuce top-down image mask (overlap)

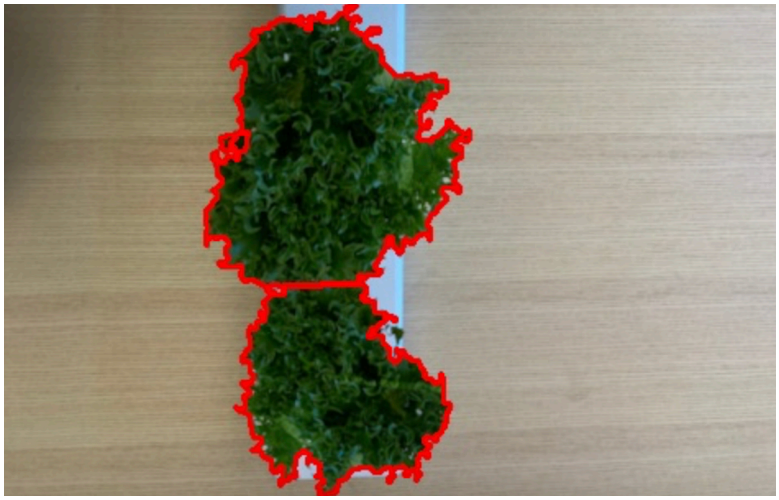


Figure G.52: Lettuce top-down (watershed)

15.5 Chessboard pattern for camera calibration

JCDH | -

A chessboard pattern allows the system to deduce the camera rotation and location in space, camera parameters and the chessboard location in the scene. This information could be useful to make more accurate approximations of the lettuce location. To generate a chessboard pattern, the script supplied by OpenCV for this purpose was used [90].

15.6 Generating a chessboard pattern

JCDH | -

Listing G.6 shows the command that was used to generate a chessboard pattern. This "gen_pattern.py"

script receives parameters like number of rows and columns in the chessboard, and the square size in millimeters [13].

```
poetry run python scripts/opencv/gen_pattern.py -o chessboard.svg --rows 9
    --columns 6 --type checkerboard --square_size 20
```

Listing G.6: OpenCV checkerboard / chessboard pattern generation command

The generated chessboard pattern is sized in the dimensions of an A4 paper as seen in Figure G.53. Measurements indicated that each square was *19 mm* in size, which is a small difference from the prompted size of *20 mm*. This difference indicates a small error in the print configuration or printer hardware, but as long as one is aware of this deviation (and enter the deviated values into the algorithms), it does not pose an issue.

[13] Note that the word "chessboard" and "checkerboard" is being used interchangeably here, which is something the official OpenCV documentation also does.

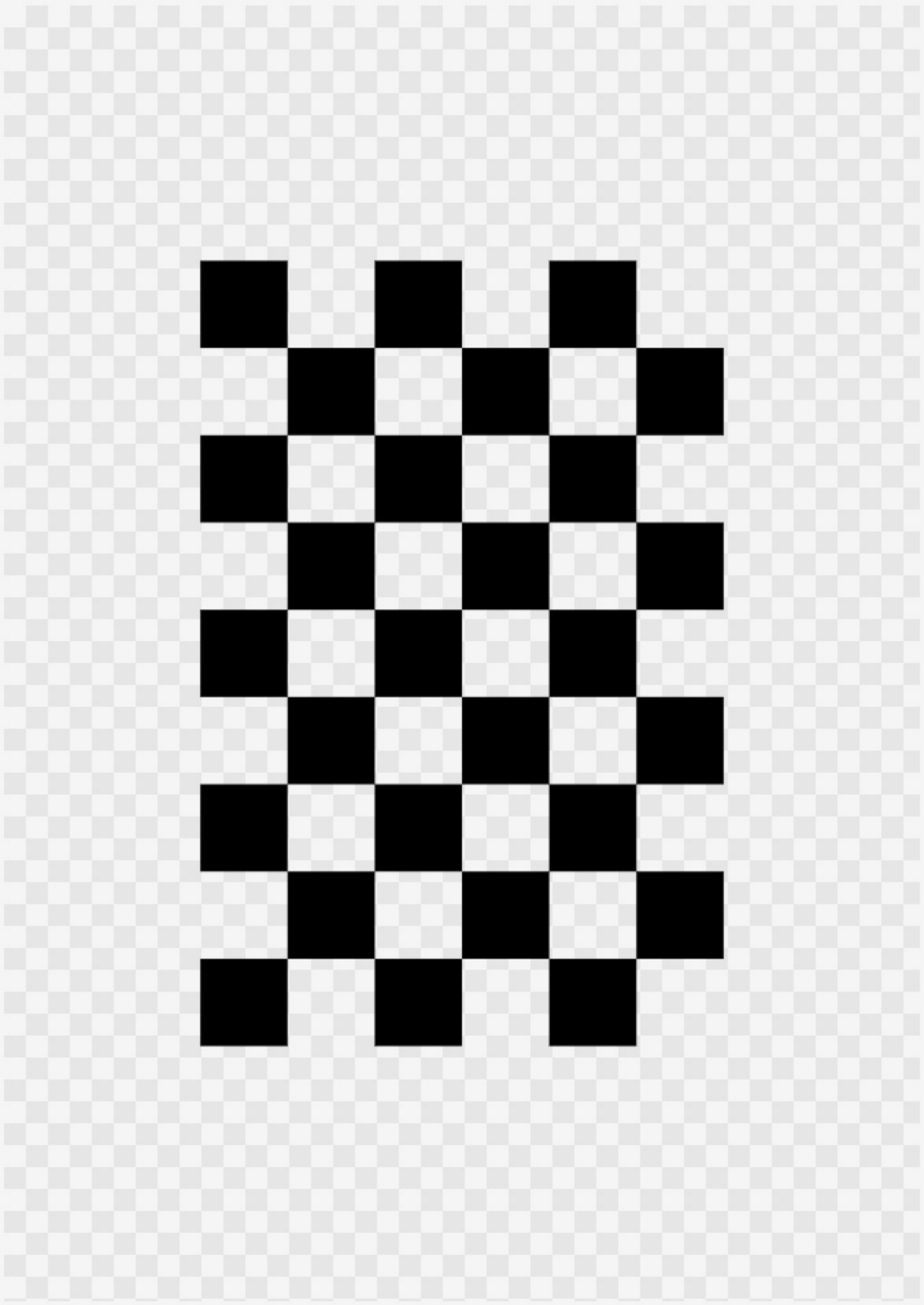


Figure G.53: Chessboard pattern

15.7 Using the chessboard pattern in practice

JCDH | -

Figure G.54 shows the OpenCV "findChessboardCorners" algorithm, and "drawChessboard-Corners" in action [91]. The algorithm defines the points in the supplied image corresponding to the corners of the chessboard squares, which is useful for deducing real-world scale of objects

and camera orientation.

Figure G.54: Chessboard detection

According to the OpenCV documentation, at least a collection of 10 images of the chessboard from different angles are required to get accurate calibration data [55].

15.8 Creating a 3D representation of the scene

JCDH | -

A scene is made up of the following objects:

- Lettuce
- Camera
- Robot arm

The aim of the scene is to represent the position of the lettuce in three dimensions, so that the robotics can pick up the lettuce.

The pinhole camera model is used to estimate real-world lettuce coordinates from a two-dimensional image, and the depth estimation Z coordinate [91]. This coordinate is from the perspective of the camera, so the distance vector between the camera center and robot arm

center must then be subtracted to give correct gripping points from the perspective of the robot arm.

16 Earlier system architecture work

JCDH | -

16.1 High-level architectural relational overview

JCDH | -

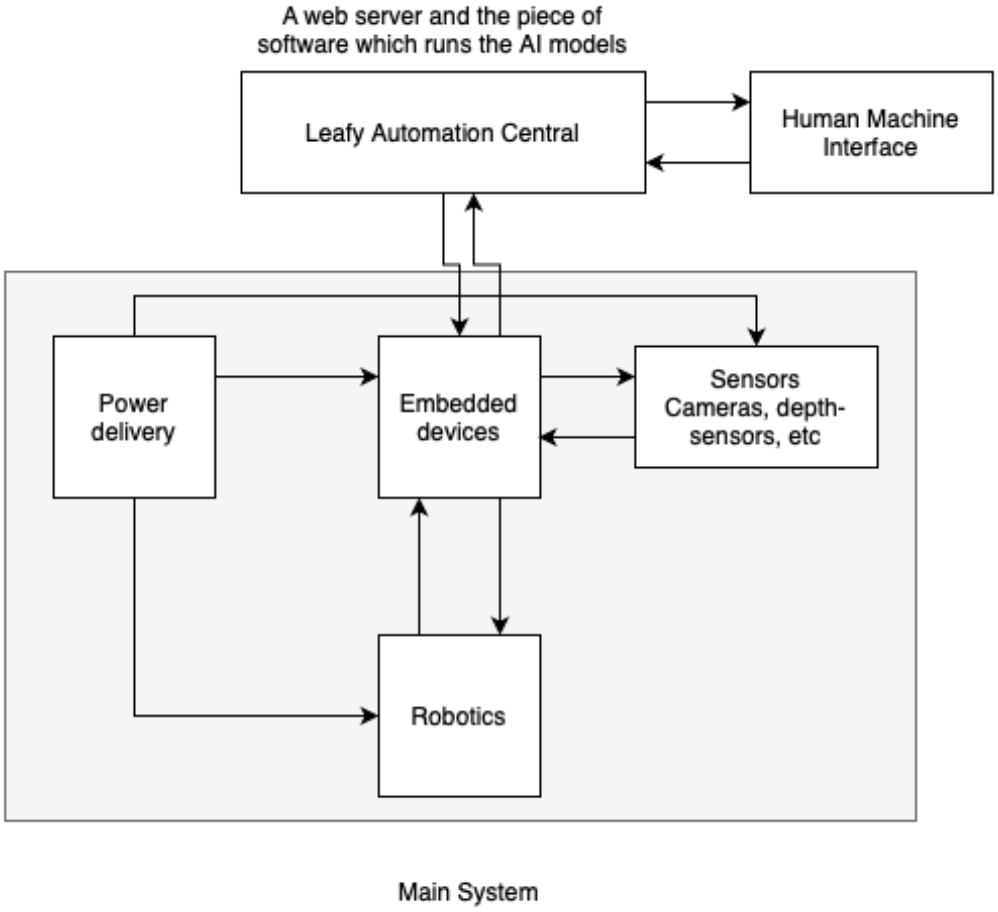


Figure G.55: High-level architectural overview

16.2 Main System relational overview

JCDH | -

A simple summary of the Main System would be the structural components, robotics and micro controllers.

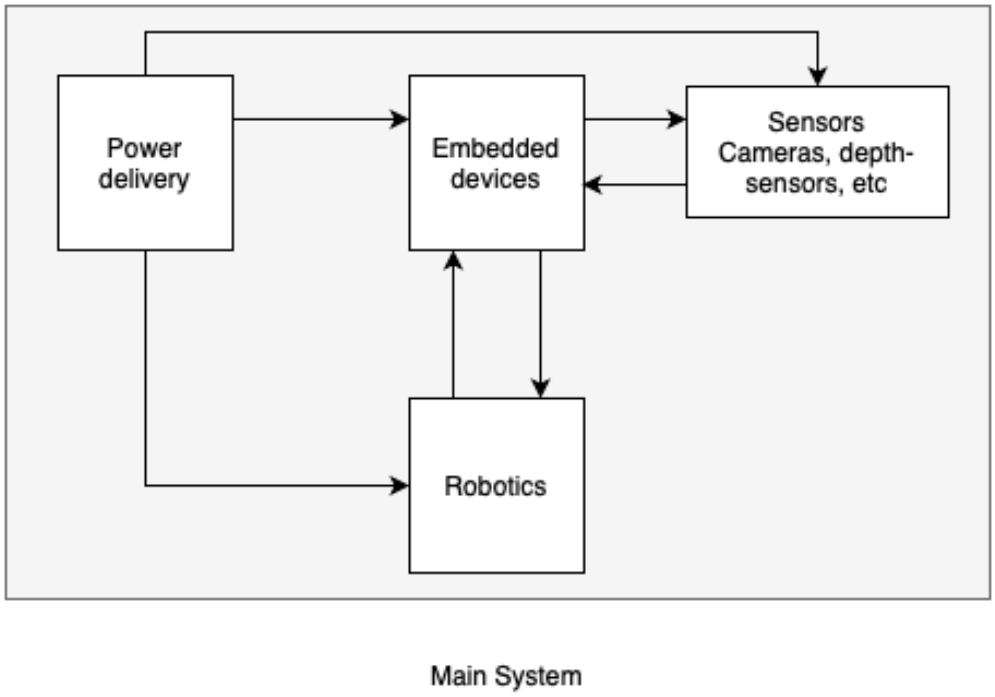


Figure G.56: Main System architectural diagram

16.3 Communications protocols / pipeline

JCDH | -

The following diagram tells us something interesting. It shows in simple terms how communication throughout our system works. The Arduino serves as an important intermediary binding both Networking and Signals together.

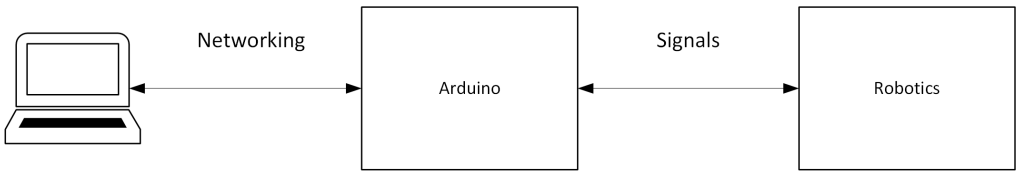


Figure G.57: Communications pipeline

16.4 AI stack

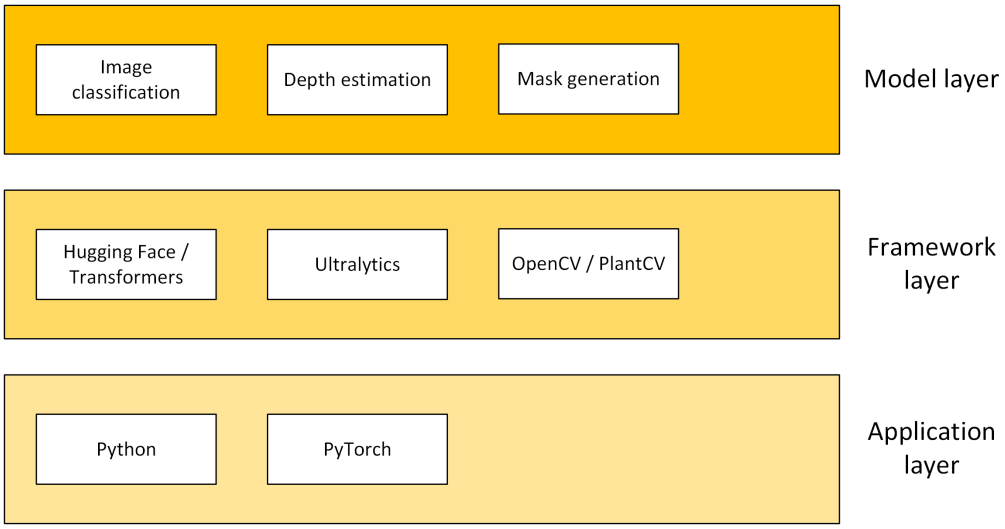


Figure G.58: AI stack

17 Considering development boards

JCDH | -

The following microcontrollers are of interest:

- ESP32
- STM32

Considering development boards:

- Arduino
- Raspberry Pi
- BBC Micro:bit
- Any ESP32 board

17.1 Development boards supplied by Hydroplant

JCDH | -

- Arduino UNO R4 Wifi
- Raspberry Pi 5

18 Explaining scrypt

JCDH | -

Scrypt is a password-based key derivation function. Although, many such password-based key derivation functions exist (like bcrypt), scrypt aims to make the algorithm more secure against a common weakness in most alternatives; the ability to use custom-designed circuits to brute force the password-based key derivation functions [92].

Figure G.59 shows an example of a scrypt password hash. It's made up of the following components [92]:

- [**scrypt**] - Password hashing algorithm name.
- [**32768**] - Amount of memory and computing power available.
- [**8**] - Block size.
- [**1**] - The amount of parallelization.
- [**$Ym0gtO...**] - The hash of the password itself.

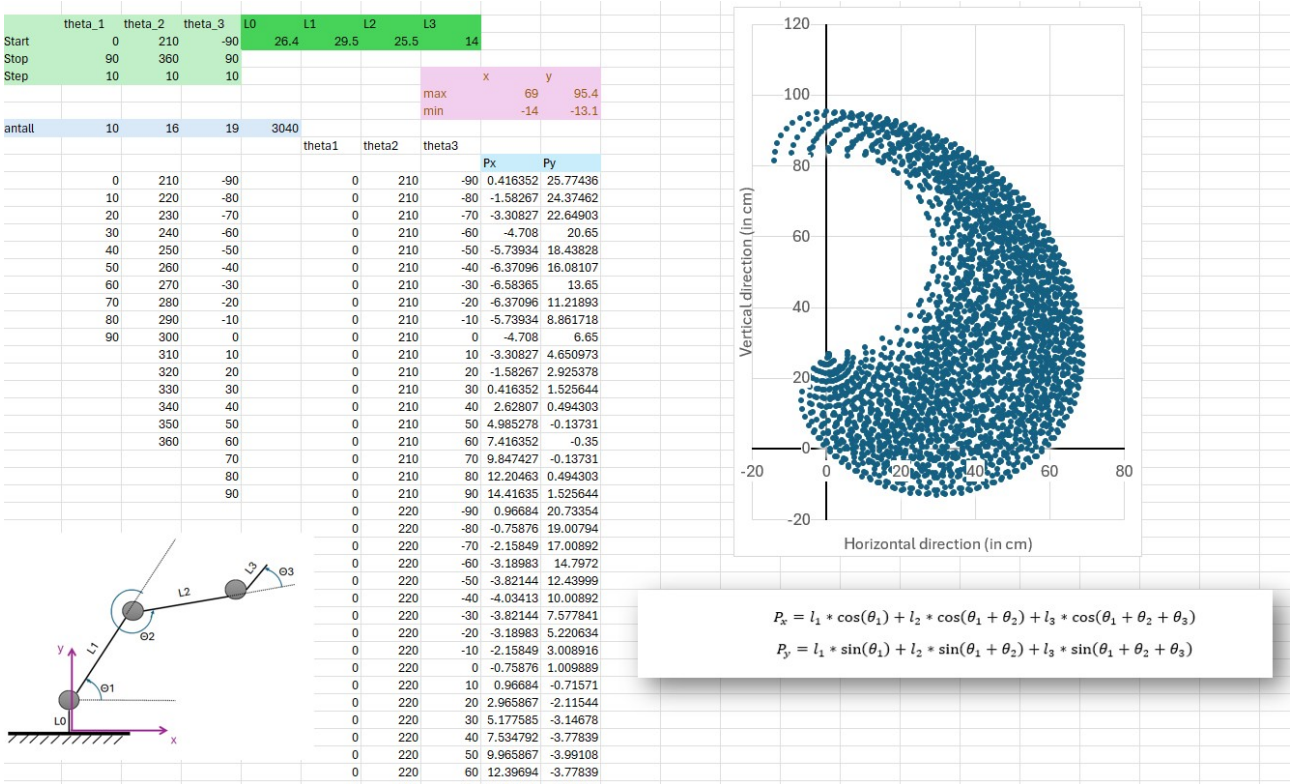
```
scrypt:32768:8:
1$Ym0gtOFvl0wxf0GE$413087246aabeeef0b89e316f3e5c87d92fdb9c7bbe0dd21b56327f478a
7f72bbb23a4e40a57f0bd0d84e6fdf728f4ab9f39ab5d8542f5fc11f10008e8215140
```

Figure G.59: scrypt password hash used for system authentication

Appendix H

Calculations

1 Configuration Space Excel sheet



	theta_1	theta_2	theta_3	L0	L1	L2	L3
Start	0	210	-90	26.4	29.5	25.5	14
Stop	90	360	90				
Step	10	10	10				

	x	y
max	69	95.4
min	-14	-13.1

antall							
10	16	19	3040				

theta1	theta2	theta3	Px	Py
0	210	-90	0.416352	25.77436
0	210	-80	-1.58267	24.37462
0	210	-70	-3.30827	22.64903
0	210	-60	-4.708	20.65
0	210	-50	-5.73934	18.43828
0	210	-40	-6.37096	16.08107
0	210	-30	-6.58365	13.65
0	210	-20	-6.37096	11.21893
0	210	-10	-5.73934	8.861718
0	210	0	-4.708	6.65
0	210	10	-3.30827	4.650973
0	210	20	-1.58267	2.925378
0	210	30	0.416352	1.525644
0	210	40	2.62807	0.494303
0	210	50	4.985278	-0.13731
0	210	60	7.416352	-0.35
0	210	70	9.847427	-0.13731
0	210	80	12.20463	0.494303
0	210	90	14.41635	1.525644
0	220	-90	0.96684	20.73354
0	220	-80	-0.75876	19.00794
0	220	-70	-2.15849	17.00892
0	220	-60	-3.18983	14.7972
0	220	-50	-3.82144	12.43999
0	220	-40	-4.03413	10.00892
0	220	-30	-3.82144	7.577841
0	220	-20	-3.18983	5.220634
0	220	-10	-2.15849	3.008916
0	220	0	-0.75876	1.009889
0	220	10	0.96684	-0.71571
0	220	20	2.965867	-2.11544
0	220	30	5.177585	-3.14678
0	220	40	7.534792	-3.77839
0	220	50	9.965867	-3.99108
0	220	60	12.39694	-3.77839

$$P_x = l_1 * \cos(\theta_1) + l_2 * \cos(\theta_1 + \theta_2) + l_3 * \cos(\theta_1 + \theta_2 + \theta_3)$$

$$P_y = l_1 * \sin(\theta_1) + l_2 * \sin(\theta_1 + \theta_2) + l_3 * \sin(\theta_1 + \theta_2 + \theta_3)$$

Figure H.1: Excel sheet with configuration space calculation

	theta_1	theta_2	theta_3	L0	L1	L2	L3		
Start	0	210	-90	26.4	29.5	25.5	14		
Stop	90	360	90						
Step	10	10	10					x	y
							max	69	95.4
							min	-14	-13.1
antall	10	16	19	3040					
					theta1	theta2	theta3		
								Px	Py
	0	210	-90		0	210	-90	0.416352	25.77436
	10	220	-80		0	210	-80	-1.58267	24.37462
	20	230	-70		0	210	-70	-3.30827	22.64903
	30	240	-60		0	210	-60	-4.708	20.65
	40	250	-50		0	210	-50	-5.73934	18.43828
	50	260	-40		0	210	-40	-6.37096	16.08107
	60	270	-30		0	210	-30	-6.58365	13.65
	70	280	-20		0	210	-20	-6.37096	11.21893
	80	290	-10		0	210	-10	-5.73934	8.861718
	90	300	0		0	210	0	-4.708	6.65
		310	10		0	210	10	-3.30827	4.650973
		320	20		0	210	20	-1.58267	2.925378
		330	30		0	210	30	0.416352	1.525644
		340	40		0	210	40	2.62807	0.494303
		350	50		0	210	50	4.985278	-0.13731
		360	60		0	210	60	7.416352	-0.35
			70		0	210	70	9.847427	-0.13731
			80		0	210	80	12.20463	0.494303
			90		0	210	90	14.41635	1.525644

Figure H.2: Excel sheet with configuration space (portion)

2 Moment calculations

BMR | -

The 2 first excel sheets show the moment calculations for direct drive, the last two show the moment calculations with belt drive.

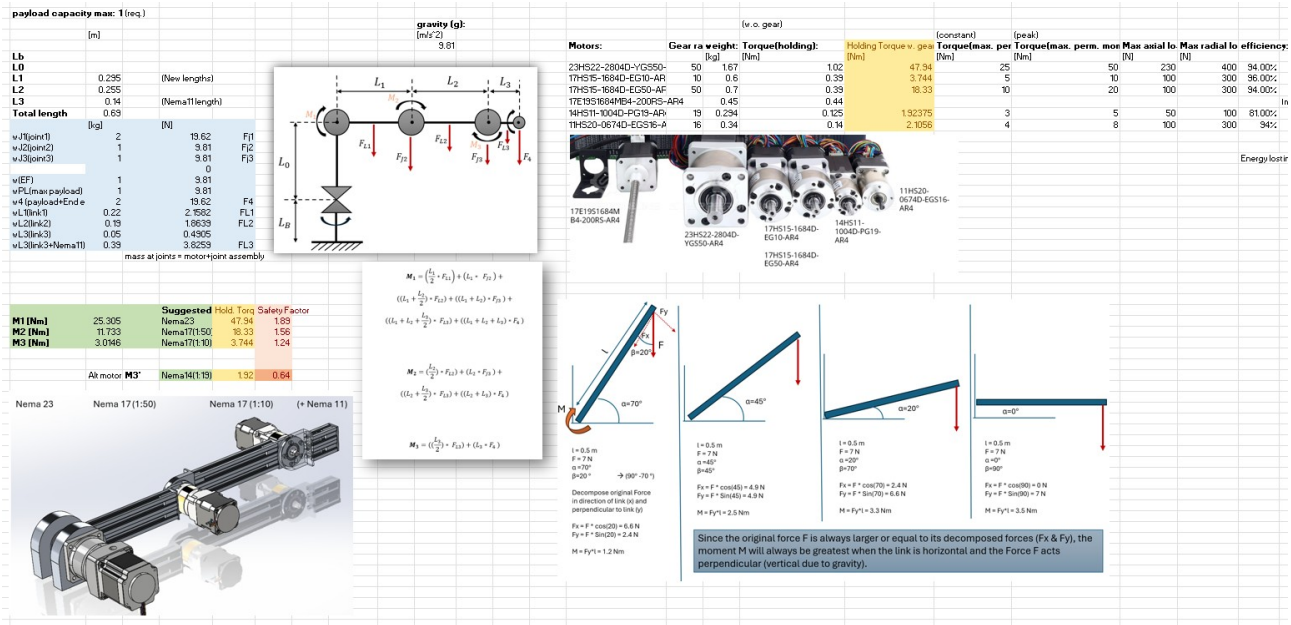


Figure H.3: Moment calculations screenshot V1 (picture and CAD of motors from [3])

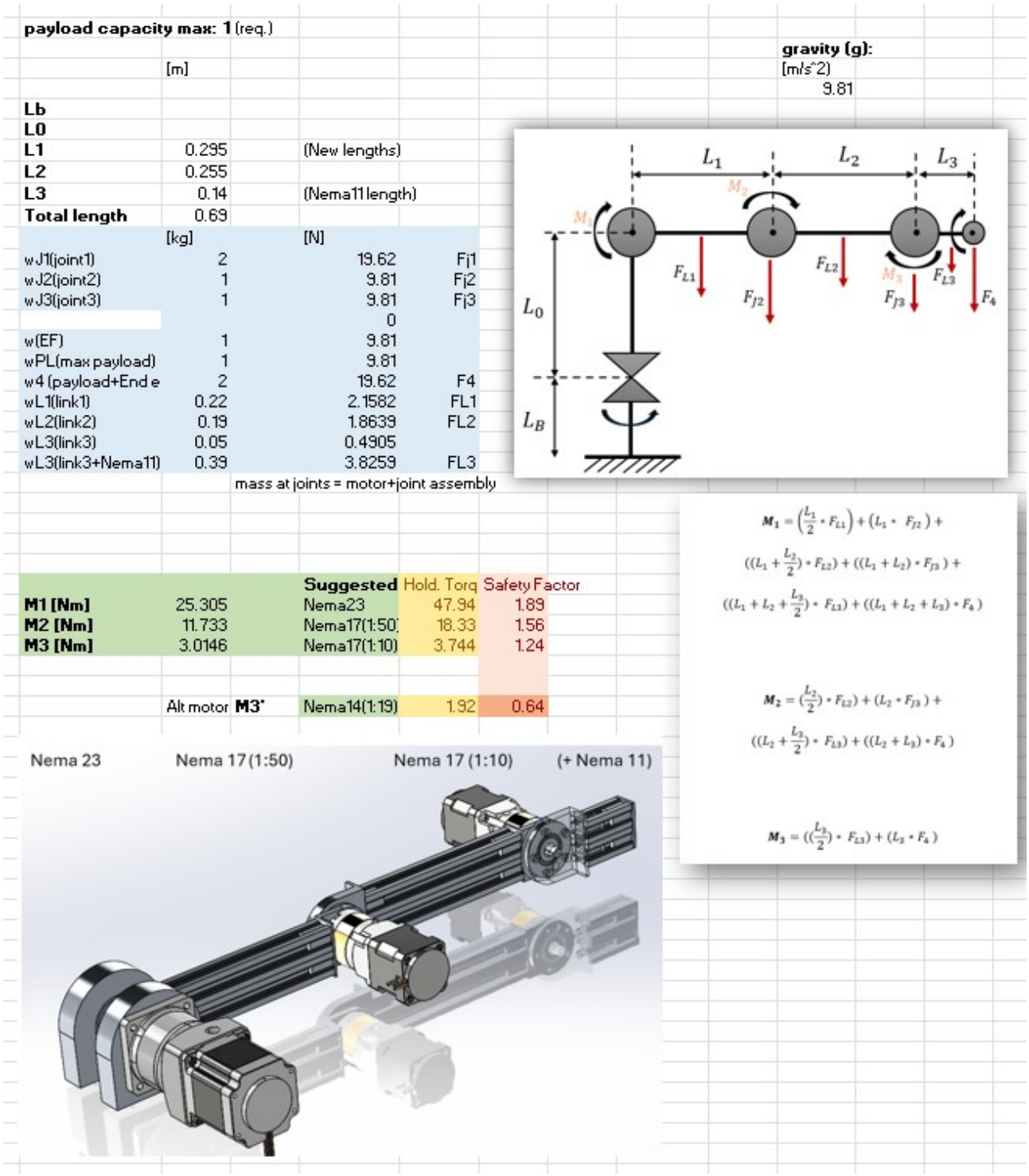


Figure H.4: Moment calculations screenshot (portion) - (CAD motors from [3])

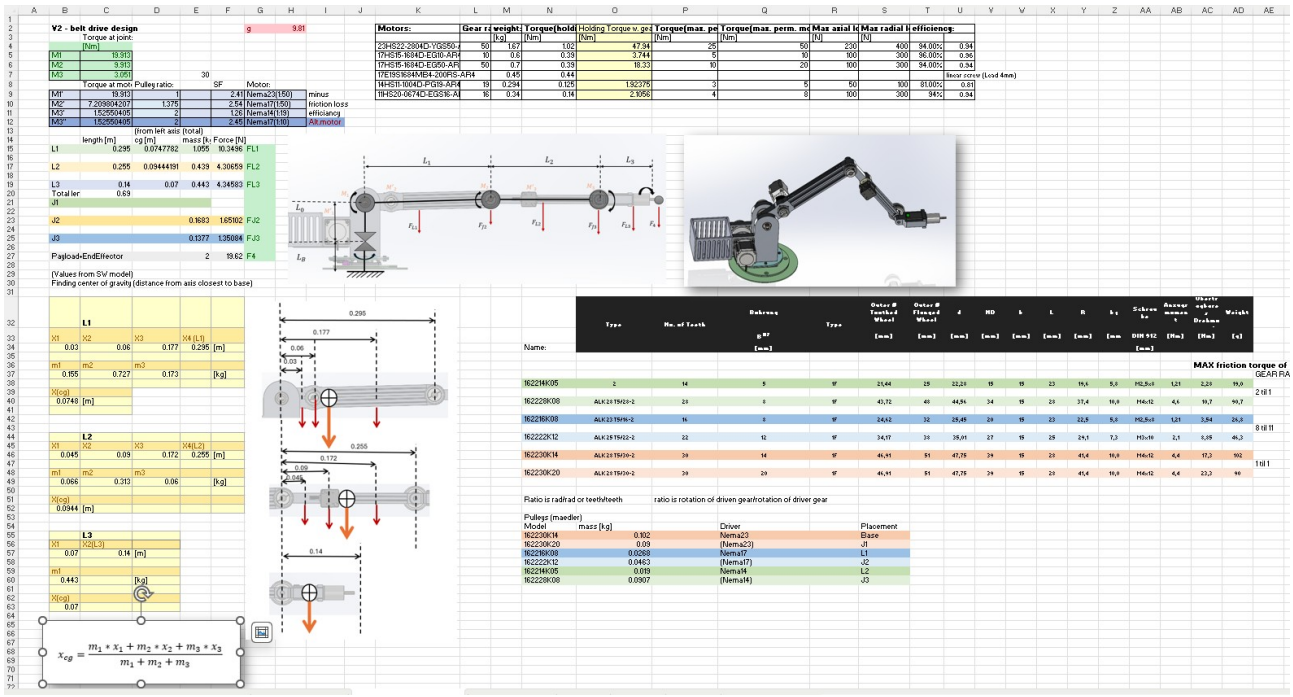


Figure H.5: Moment calculations screenshot V2

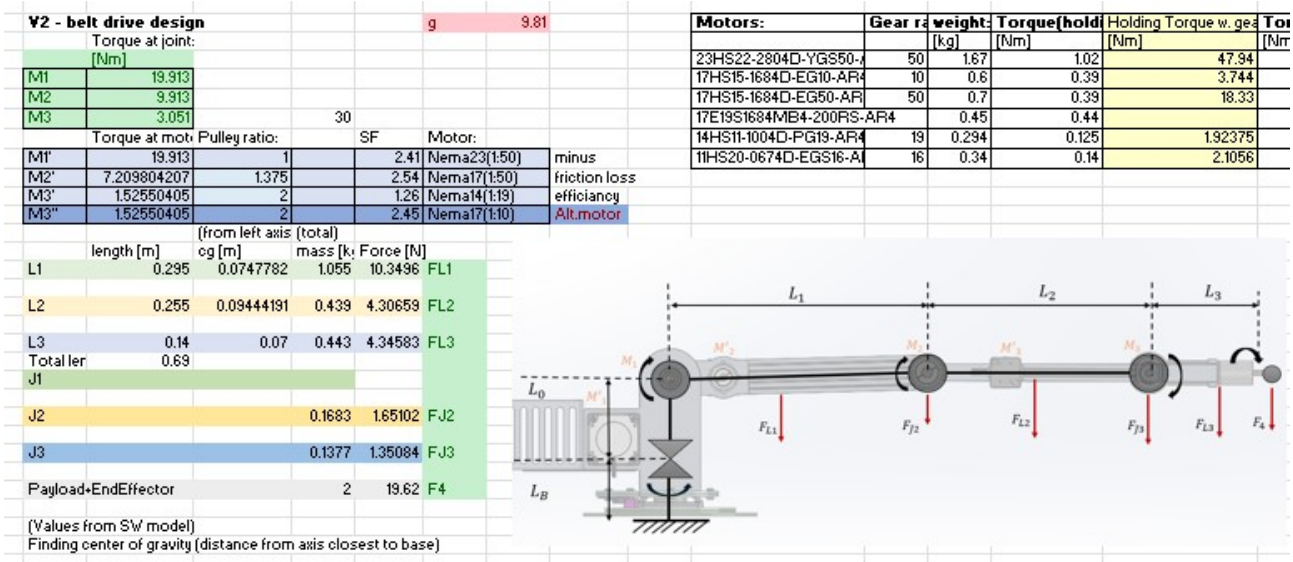


Figure H.6: Moment calculations screenshot V2 (portion)

3 Varied payload

BMR | -

payload capacity max: 1 kg		(req.)											
								gravity (g):					
	[m]							[m/s^2]					
								9.81		**Motors:**	**Gear ratio**	**weight:**	**T**
Lb												[kg]	[
L0										23HS22-2804D-YGS50-AR4	50	1.67	
L1	0.295		(New lengths)							17HS15-1684D-EG10-AR4	10	0.6	
L2	0.255									17HS15-1684D-EG50-AR4	50	0.7	
L3	0.14		(Nema11 length)							17E19S1684MB4-200RS-AR4		0.45	
Total length	0.69				Payload	M1	M2	M3					
	[kg]		[N]		0.3	20	9	2					
wJ1(joint1)	2		19.62	Fj1	0.5	22	10	2					
wJ2(joint2)	1		9.81	Fj2	0.8	24	11	3					
wJ3(joint3)	1		9.81	Fj3	1.0	25	12	3					
			0		1.3	27	13	3					
w(EF)	1		9.81		1.5	29	14	4					
wPL(max payload)	1		9.81		1.8	30	15	4					
w4 (payload+End ef)	2		19.62	F4	2.0	32	16	4					
wL1(link1)	0.22		2.1582	FL1	2.3	34	17	5					
wL2(link2)	0.19		1.8639	FL2	2.5	35	18	5					
wL3(link3)	0.05		0.4905		2.8	37	19	5					
wL3(link3+Nema11)	0.39		3.8259	FL3	3.0	39	19	6					
		mass at joints = motor+joint assembly			3.3	41	20	6					
					3.5	42	21	6					
					3.8	44	22	7					
					4.0	46	23	7					

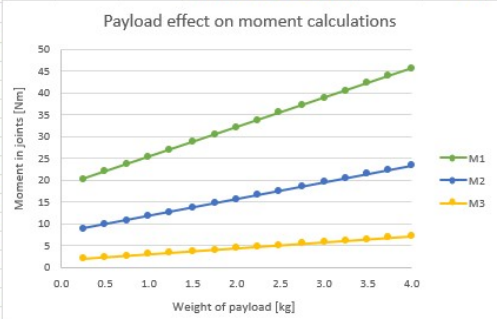


Figure H.7: Excel sheet with moment calculations for varied payload

Appendix I

Project expenses

1 Project expenses

BMR | —

This is an overview of all bought and given components for the bachelor project "Leafy Automation", see fig. I.1

Leafy Automation bachelor project expenses				
	Purchases (NOK):			**Sum (NOK):**
	Items:		With MVA (VAT):	
Software:			0, -	0, -
Mechanical Materials:	Bearings x 4 shipping Bearings x 2 shipping Pulleys and belts		568, - 70, - 328, - 70, - 1324, -	2360, -
Electronics:	Stepper motor kit Shipping Circuit board Components		9206, - 941, - 1974, - 2449, -	14 570, -
Common:	Nametags for Expo		920, -	920, -
			Partial sum:	17 850, -
	Given resources from HPT:	**Estimated cost:**		
	Rasberry Pi 5	1000,-		
	Arduino UNO R4 Wi-Fi	500, -		
	Aluminum profiles (20x20)	110, -		
	Servo motor for gripper	250,-		
	Camera (ESP-32-CAM)	100, -		
			Partial sum:	1950, -
	Resources from USN and group members:	**Estimated cost:**		
	Filament (3D print)	500, -		
	MDF (laser cut)	100, -		
	Limit switches, wires, etc.	300, -		
	Leafy greens	200, -		
			Partial sum:	1100, -
			Total sum:	20 900, -

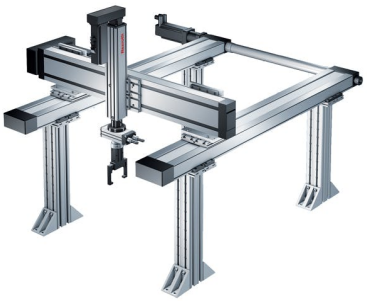
Figure I.1: Project expenses overview

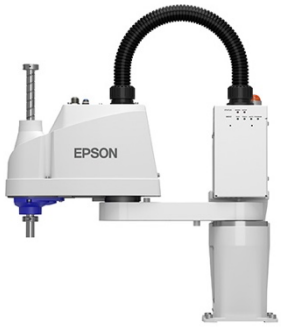
Appendix J

Robot Concepts

Robot concepts

For our robot arm, we researched different pick and place types to ensure that we chose one that would fit with our project. The table below describes the five types we looked into.
Some are more like each other than others, and some types we could rule out from an early stage. We needed to focus on our requirements to help choosing the most suitable robot type. Factors like work area, type/ weight of load, precision, speed requirements and others, needed to be put in consideration when choosing a concept to move forward with.

Robot type	Description	
Articulated robot arm	- This robot type resembles the human hand and allows mechanical movement and configurations. - One of the most common types of robotic arms for industrial automation. - Single arm attached to a base with a twisting joint.	
Cartesian robot arm	- These are linear/ gantry robots. They work on three linear axis (up and down, in and out, side to side). - Popular in the industry and for manufactures who want high flexibility in their configurations.	

Cylindrical robot arm	- This type of robotic arm is designed around a single arm that is capable of moving up and down vertically. - They have a rotary joint at the base and prismatic joint to connect the links. - Very compact and cover small and simple tasks	
Delta robot arm	- This robot type is also referred as parallel robot arms, because they facilitate three arms connected to a single base mounted over a workspace. - They have high speed options and are therefore used for automation.	
SCARA robot arm	- Selective compliance assembly robot arm - These types are designed with a horizontal arm that moves in two directions - Known for high speed and precision - Limited range of motion in the vertical place, designed to move primary in the horizontal plane.	

From valuation, research and the Pugh matrix, we decided to use an articulated robot arm. The robots we did not choose was because of several different reasons. Many of the industrial robot types are built on large frames in which the end defector (gripper) moves along the different axis (Gantry-and cartesian robots). These could be suitable for the purpose of this project and are quite simple in their overall construction but also have some drawbacks.

While being very scalable they take up a lot of space since their frame needs to cover the whole working area. For our project we want to put most weight on the flexibility and design of the end effector, therefore we wanted to make an articulated robot arm. The other robot arms have more limitations in the end effector and makes it more difficult to pick plants from different angles.
The delta robot arm is also used in pick and place applications and has good applications, precision and speed. The reason we discharged this type is because it is not able to lift a product from the side, which can be needed in our project.
For the SCARA and cylindrical robot arm, they are types of robots that is fast and have high precision, while taking up little space. The downside is that it has limited flexibility that again makes it hard to pick plants from different angles.

Sources robot pictures:

- Articulated robot arm
- Cartesian robot arm
- Cylindrical robot arm
- Delta robot arm
- SCARA robot arm