Bachelor's thesis

Spring of 2024

KONGSBERG

University of
South-Eastern Norway

Course: TS3000 Bacheloroppgave
Date: May 21, 2024
Title: Delirium

This report forms part of the basis for assessment in the subject.

Project group: 6
Group members: Stian Nordholm,
Helge K. Kopland,
Andreas B. Sørensen,
Anders M. Minde

Internal Supervisor: Joakim Bjørk
External supervisors: Sirajuddin Asjad,
Jan Dyre Bjerknes

Project partners: Kongsberg Defence & Aerospace

Acknowledgements

We would like to express our sincere gratitude to Sirajuddin Asjad and Jan Dyre Bjerknes, our external supervisors, for the task we were given and the advice we received from them during this project. Their support combined has been crucial for us, in terms of getting the equipment we needed in place and providing us with tips, tricks, and tools to perform our best.

Our internal supervisor, Joakim Bjørk, was instrumental in keeping us on the right path, provided guidance, and gave us calming advice throughout the project. It would not have been possible for us to keep an even keel and deliver a report or product as required without his insight.

Delirium further would like to show our greatest appreciation to Line Minde, both the creator of our logo and our design overseer. Her expertise in graphical design has been important for the look and feel of our produced artifacts.

We would like to thank Tina Tolleskoven for her time, as she willingly and happily endured our presentation rehearsals and provided us with instrumental feedback.

For tips along the way, excellent insights, and help with formatting our report, we would like to thank Ruben Sørensen.

Abstract

This report details the development of the bachelor's project Delirium, which was carried out at USN Kongsberg in the spring of 2024. Delirium is a portable system designed to attack unmanned aerial vehicles with radio frequency jamming and spoofing of the civil Global Positioning System (GPS) signal. It is both a general jammer capable of denying Global Navigation Satellite System (GNSS) reception with several strategies, and a GPS "spoofer" capable of generating its own faux GPS baseband data and transmitting this data to an arbitrary target device. These capabilities can optionally be performed at the same time. The novelty of this product lies in the synthesis of existing stand-alone programs and modern software-defined radio technology to create a completely mobile penetration testing -and attacking device to be used against devices utilizing GNSS signals.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 31, 70, 72

BPSK Binary Phase Shift Keying. 26, 28

CPLD Complex Programmable Logic Device. 51, 52

dB Decibel. 59, 60

DFRM Digital Frequency Radio Memory. 24, 169

FPGA Field-Programmable Gate Arrays. 52

GHz Giga Hertz. 52, 67

GNSS Global Navigation Satellite System. 14, 15, 20, 22, 84

GUI Graphical User Interface. 9, 49, 70–72, 82, 84

HAT Hardware Attached on Top. 64

KDA Kongsberg Defence & Aerospace. 17, 37, 39, 40, 42, 43, 47, 48, 50, 52, 57, 84, 99

Kg Kilogram(s). 54

LNA Low Noise Amplifier. 65, 66

mA Milli Ampere(s). 56, 57

mAh Milli Ampere Hour(s). 57, 58

MHz Mega Hertz. 28, 67, 75, 76

NASA National Aeronautics and Space Administration. 53

NKOM Norsk kommunikasjonsmyndighet (Norwegian Communication Authority). 33, 34, 59

NOK Norske kroner. 54

PPM Parts Per Million. 52

PSK Phase Shift Keying. 28

QPSK Quadrature Phase Shift Keying. 28

RAM Random Access Memory. 55, 169

SAW Surface Acoustic Waves. 65–67

SBAS Satellite Based Augmentation System. 65

SDR Software Defined Radio. 19, 30, 48, 51–53, 70, 71, 82

SMA SubMiniature version A. 63, 126

UAS Unmanned Aerial System. 17

UI User Interface. 47

UML Unified Modeling Language. 71

USB Universal Serial Bus. 55, 56

USB-PD USB Power Delivery. 56

USN University of South-Eastern Norway. 35, 70

W Watt(s). 56–58

Glossary

almanac Within GNSS systems, the almanac is a dataset that contains coarse orbit and status information for the satellites in the respective GNSS's constellation as well as data related to error correction.. 21, 68

Atlassian An Australian-American corporation that makes software solutions for software development teams and project managers. 39

AWGN Additive White Gaussian Noise. A random disturbance (noise) signal, following a bell-shaped curve, evenly spread across all frequencies [14]. 25, 26

BeiDou The Chinese GNSS. 20

boilerplate code "In computer programming, boilerplate code, or simply boilerplate, are sections of code that are repeated in multiple places with little to no variation." [15]. 32

Chip A chip is a string of pseudo-random bits transmitted with a pre-determined length and rate. Chip is often used to avoid confusion with bits containing useful data.. 29

ephemeris In astronomy and celestial navigation, an ephemeris is a dataset with tables that gives the trajectory of naturally occurring astronomical objects as well as artificial satellites in the sky, i.e., the position (and possibly velocity) over time.[16]. 21, 28, 29, 53, 68, 79

flight controller A circuit board or box, that with the help of multiple sensors controls everything a drone does, i.e. the drone's brain. 64

flowgraph The concept of a flowgraph in GNU Radio is an acyclic directional graph with one or more source blocks (to insert samples), one or more sink blocks (to terminate or export samples) and any processing blocks in between.. 70

Galileio The European GNSS operated by the European Union Agency for the Space Programme (EUSPA). 20

GIL Python's Global Interpreter Lock. 31, 72

GLONASS Global'naya Navigatsionnaya Sputnikovaya Sistema, the Russian GNSS. 20

GPS Global Positioning System, the American GNSS. 20, 22, 72, 82, 84

high-level language High-level programming languages have formats close to the English language. They are very abstracted from the machine code which it represents, and express instructions in a more human-readable form than low-level languages [RefH2].. 31

L band The L band is the IEEE designation for the range of frequencies in the radio spectrum from 1 to 2 gigahertz (GHz). This is at the top end of the ultra high frequency (UHF) band, at the lower end of the microwave range.[17]. 20

low-level language A low-level programming language has a small or nonexistent amount of abstraction between the language and machine language [RefH2].. 31

LTspice ”LTspice is a powerful, fast, and free SPICE simulator software, schematic capture, and waveform viewer with enhancements and models for improving the simulation of analog circuits.” [18]. 60

MVP A Minimum Viable Product is an early version of a product that satisfies the stakeholder's requirements in the simplest ways possible so that any discontent can be weeded out early on [19][20]. 42

Piezoelectric "The ability of certain materials to generate an electric charge in response to applied mechanical stress." [21]. 66

shared object shared object files (.so) are - like Windows' dynamic-link library (.dll) files - shared library files which allow multiple executing programs access to the same source code libraries.. 32, 33, 72

SV Space Vehicle, a common way to shorten the word "satellite" in the GNSS field. 22, 23

UBX-CFG-GNSS Configuration field in ucenter, used to select one or more frequency bands for a particular constellation. E.g. GPS: L1C/A /L2C/L5 and/or GLONASS: L1 C/A. 65

Chapter 1

Introduction & Scope

This chapter serves as an introduction, both to this report itself and the task which defined this project.

1 Introduction

HK | *SN*

Global satellite navigation was originally a military invention. Developed in the 1960s it began as a two-dimensional positioning service. After the invention of solid-state microprocessors, bandwidth utilization techniques, and computers, its accuracy improved and led to the invention of the American system Navstar, later renamed to Global Positioning Service (GPS) [RefH3]. In 1983 President Ronald Reagan authorized the use of GPS by civilian airlines, this was the first time the system was to be used in a non-military application. By the end of the 1980s, there were handsets available for consumers to buy, these were however expensive at around 3000$

With the continued development in computer technology, and widening usage of the GPS system, receivers now cost around 1.5$ instead and are everywhere. Cars, watches, cellphones, computers, and even some dog collars have them. We have come to rely on navigation services for many things in our daily lives. Therefore we believe it is important to study the effect it would have on us if this technology could be stopped in some way.

In our bachelor thesis, we have studied the process of developing a system that can block or disrupt the GPS signal, and thus either obstruct the navigation or give the user a false location. We have called the project "Delirium" as the state of delirium is a mental disorder manifesting as a state of confusion, encapsulating our project goal.

The introduction section of this report is about the group, the problem statement, and the previous works on this subject. The "background" section is technical background information about the systems we interact and interface with. The "method" section explains our approach, the tools we used, and how we implemented them into our system. Finally, in the "epilogue" section; we show our results, discuss our findings, and suggest how to expand on our work.

1.1 Group Members

SN | *ABS*

The Delirium project consists of four members: Anders Minde, Stian Nordholm, Helge Kopland, and Andreas Bondal Sørensen. Each group member's portrait, full name, initials, and engineering discipline can be found in Table 1.1.

1.2 Initials

SN | *ABS*

The initials of the author and proofreader for each section are indicated in the headings throughout this document. This approach clarifies the contributions of each group member, as the project is graded individually. Clearly displaying each member's contributions is essential for individual assessment. Each section has been proofread by a different group member to ensure the document's quality. The format for crediting is as follows: the author's initials are listed first, followed by a vertical line |, and then the proofreader's initials.

2 Problem Statement

HK | *ABS*

We were tasked with creating a 'red-side' (attacking) system for confusing the positioning service of an Unmanned Aerial System (UAS) using GPS jamming. The system was to physically appear as a small container with an antenna, and be capable of neutralizing a drone through GPS jamming.

If this was achieved we were then supposed to iterate on the system and add spoofing capabilities, with the aim of depriving the receiver of authentic signals (jamming) and sending our own GPS signals that tell the receiver it is in another location.

KDA wants to use the system we develop internally for penetration testing of drones and other systems that use GPS services, for example, their missiles. The system described by the customer can be visualized as seen in figure 1.1.

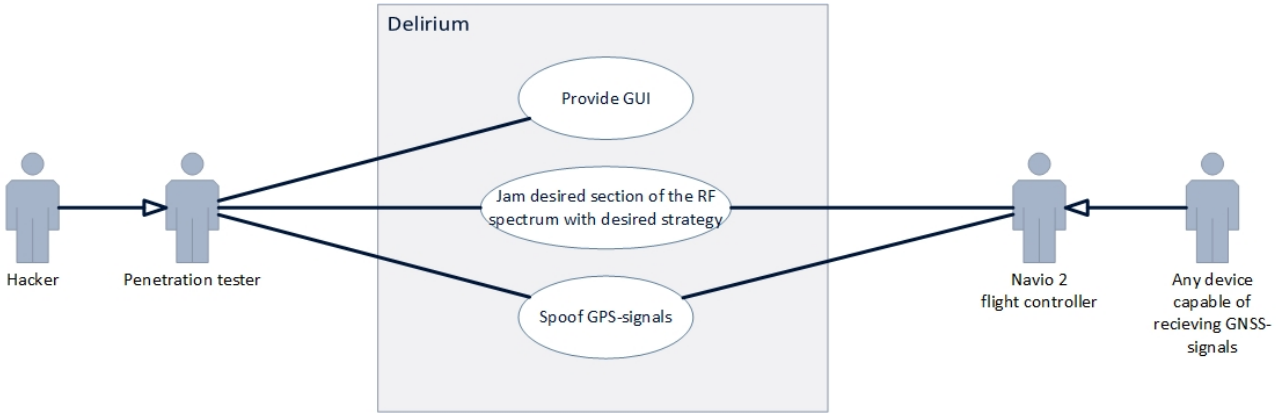


Figure 1.1: Main use cases for Delirium, created by AM. Legend can be seen in figure N.1.

3 Previous Work

HK | *ABS*

In recent years the wide usage of consumer-grade drone systems has generated issues for several concerned parties and the number of drones sold to consumers is expected to double by 2030 [22]. In this context, experts are concerned about the increase in the malicious use of these drones,

	***Name***	Anders Minde
	Initials	AM
	Discipline	Computer engineer - Cyber physical systems
	***Name***	Stian Nordholm
	Initials	SN
	Discipline	Computer engineer - Cyber physical systems
	***Name***	Helge Kopland
	Initials	HK
	Discipline	Electrical engineer - cybernetics
	***Name***	Andreas Bondal Sørensen
	Initials	ABS
	Discipline	Electrical engineer - cybernetics

Table 1.1: Group members

in particular the security, safety, and privacy within society[RefH4]. Therefore the operation of hindering drone operations in certain areas is becoming more important in the coming years. Recent studies have reviewed the possibility of manipulating a drone's positioning services in order to gain control of the perceived location of the drone [RefH5][RefH6][RefH7].

The consensus among these studies is that recent advancements in Software-Defined Radio (SDR) peripherals and open-source software are making the concept of interfering with drone positioning services more accessible to the layman. Previously in order to transmit the signals needed to block GPS reception, one would need a signal generator, but the prices of these systems make this approach rather infeasible: "Despite the ease of mounting a spoofing attack with a signal simulator, there are some drawbacks. One is cost: the price of modern simulators can reach $400k" [RefH5]. Another reason this method was seen as unlikely by Humphreys et al., was the size of the signal generator system. A signal simulator is an advanced signal generator, which generates simple sine waves with a set amplitude and frequency. The signal simulator can in addition to this simulate specific signals, such as the pseudo-random ranging codes found in the GPS L1-signal.

Lately, the development of SDR has made those points mute. In [RefH7] Renato Ferreira et al. proved that with an SDR-peripheral costing in the order of a few hundred dollars, they were able to jam the signals of a consumer-grade drone. Iterating on their work, we aim to develop a system consisting of similarly priced components that can both jam and spoof the signals of a consumer-grade drone.

Chapter 2

Background Theory

This chapter includes all the necessary background theories one would need to understand the topics discussed later in this document. Subjects like jamming and spoofing, modulation techniques, software languages, modern software development, as well as general information on GNSS are all explained in detail. This is an important read to fully understand the extent of our project and our task.

4 Global Navigation Satellite Systems

AM | *HK*

There exist 4 Global Navigation Satellite Systems (GNSSs) which orbit the earth, here enlisted from first to last:

- Global Positioning System (GPS), made by the United States, first launch: 1978
- Global'naya Navigatsionnaya Sputnikovaya Sistema (GLONASS), made by Soviet/Russia, first launch: 1982
- BeiDou, made by China, first launch: 2000
- Galileio, made by the European Union, first launch: 2011

Each of these systems manages about 30 Space Vehicles (SVs, meaning satellites in this case) in orbit at any time, depending on health, age, system modernization plans, and so on. The set of SVs belonging to a certain system is not uniform, it might contain older and newer SVs with different capabilities depending on emerging technologies and demand. Although each system is operated by individual entities representing different powers around the world, their method of operation is quite similar. Each GNSS orbits at a specific radius relative to the earth, roughly as illustrated in figure 2.1.

4.1 Generally About GNSS Signals

AM | *HK*

Each GNSS broadcasts both open signals for civilian use and encrypted signals for authorized personnel. Our project scope covers the signals available for civilian use. The satellites continuously transmit navigation signals in two or more frequencies in the *L band* (from 1 to 2

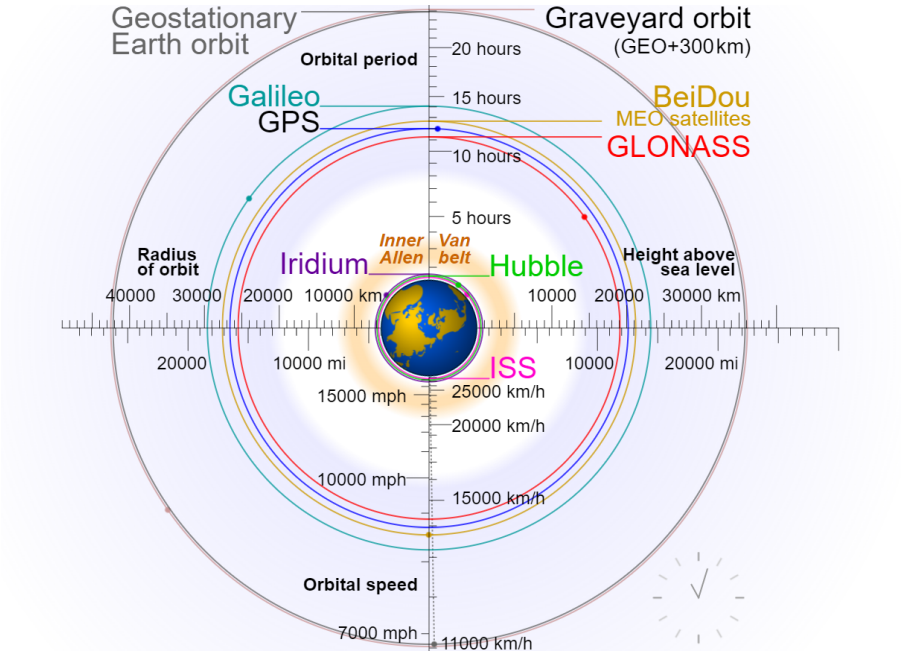


Figure 2.1: Illustration of several satellite navigation system orbits[1]. The illustration is animated and can be seen and interacted with by following the reference

gigahertz). The signals' spread along the band are shown in figure 2.2. The GNSS signals generally contain the following main components[23]:

- **Carrier wave:** Radio frequency sinusoidal signal at a given frequency.
- **Ranging code:** Sequences of 0s and 1s (zeroes and ones), which allow the receiver to determine the travel time of a radio signal from a satellite to the receiver. They are called pseudorandom noise (PRN) sequences or PRN codes.
- **Navigation data:** A binary-coded message providing information on the satellite *ephemeris* (satellite position and velocity) which enables the computation of satellite coordinates at any point in time, clock bias parameters, *almanac* (with a reduced accuracy ephemeris data set), satellite health status, and other complementary information.

4.2 Determining the Position

AM | *HK*

A device receiving GNSS signals can by algorithmic calculation determine its position in all 3 dimensions, its velocity and the time. For this to happen, the receiving device needs to acquire the ranging code and navigation data, as described in the section above. The actual calculation of the position from this data is quite exhaustive, several examples of which can be found on GNSS-SDR's web page [24]. The key is that each SV broadcasts omnidirectional signals which form a sphere surrounding them. When the signal from one SV reaches the receiving device with its ranging code and navigation data, the distance between them can be calculated. The position on Earth cannot be determined, however, until signals from several SVs are received.

This is a consequence of the fact that the position of the receiver is calculated as being in

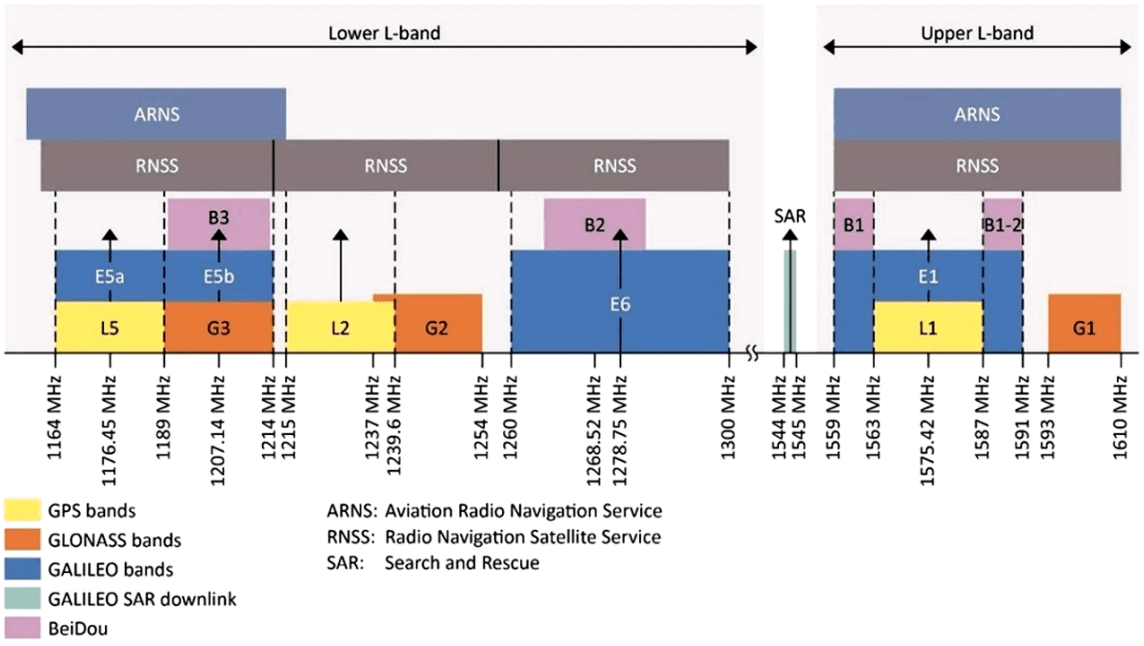


Figure 2.2: GNSS signal frequencies[2]

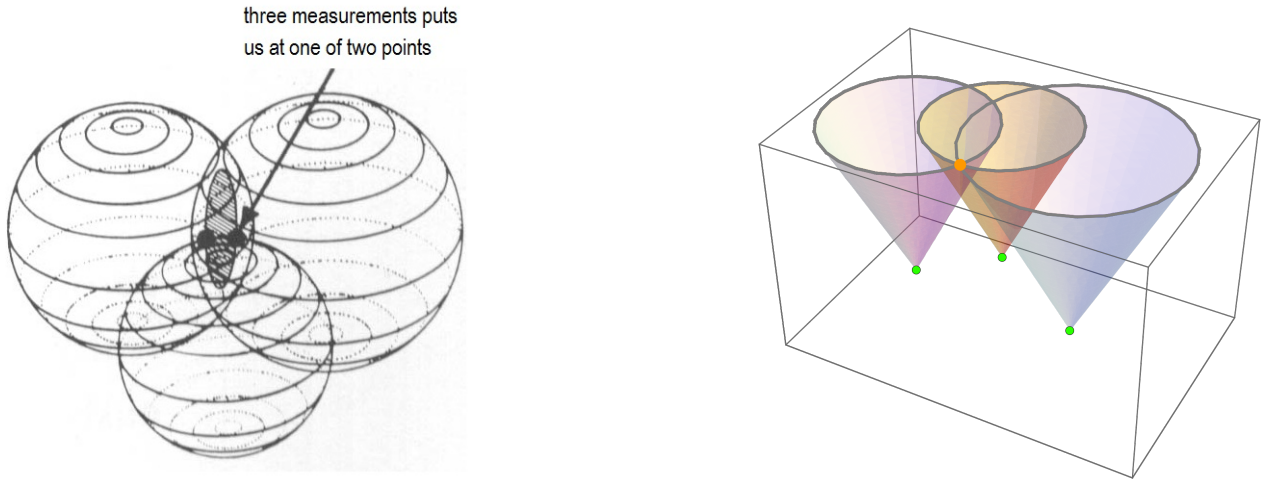
the intersection of the "signal spheres" arriving on it (see figure 2.3). As such, we could say that the signal from one SV only provides enough information to assume a distance from the SV itself as the position of the receiver. The signals from two SVs will place the receiver on Earth, but in the largest possible area, etc. In fact, it takes signals from a minimum of four SVs in view to get an actual position, where the 4th SV's signal is used to correct for timing inaccuracies. Receiving more than 4 signals further makes the position calculation more accurate [25] [26] [27].

4.3 The GPS Example

AM | *HK*

The by far most used GNSS in Western countries is the GPS, it therefore made sense for us to focus on this system. The main GPS carrier signal, L1, at 1575.42MHz, consists of two carrier components which are in phase quadrature with each other, both of which are referred to as the "legacy signals" by the US Space Force. Each carrier component is a bi-phase shift key (BPSK) modulated by a separate sequence of bits (more on that in section 6). One such sequence is the modulo-2 sum of the P(Y)-code (precision/secure code) and legacy navigation (LNAV) data, which is reserved by cryptographic techniques for military and authorized civilian users. The other sequence is the modulo-2 sum of the C/A code (coarse/acquisition code) and the LNAV data, which is freely available for public use.

All satellites in the GPS constellation currently broadcast the signal mentioned above with its two carrier components, and therefore L1 C/A is the go-to civilian navigation signal within the GPS. Newer satellites additionally broadcast more accurate signals on other frequencies. Code-division-multiple-access (CDMA) techniques allow differentiating between the SVs even though they may transmit at the same frequencies [RefH1] [28] [29].



(a) Omnidirectional illustration [25]

(b) Light cones [26]

Figure 2.3: Position calculation

4.3.1 LNAV Data

AM | *HK*

The LNAV data includes SV (Space Vehicle) ephemerides, system time, SV clock behavior data, status messages, and C/A to P (or Y) code handover information, etc. The 50 bps (bits per second) data is modulo-2 added to the P(Y)- and C/A- codes; the resultant bit-sequences are used to modulate the L1 carrier. The actual broadcasted data is a 1500-bit long "frame" made up of five subframes, where each subframe is 300 bits long (see the example in fig 2.4). Subframes 4 and 5 are sub-communicated 25 times each, meaning that a complete data message requires the transmission of 25 full frames [RefH1].

5 Jamming

HK | *AM*

Drone Jamming can be accomplished by several methods. There are two main vectors of attack for drone jamming. The communication link between the pilot and the drone, and the location signals between the drone and the satellites.

In order to have a counter-drone system, there are several prerequisites that need to be met. First, the drone must be detected, then it needs to be localized and finally, it needs to be identified before beginning the drone jamming. There are numerous ways to perform the detection, localization, and identification. One of these is visually detecting, locating, and identifying the drone. Given the scope of our project; neutralization, we assume that the prerequisites are met.

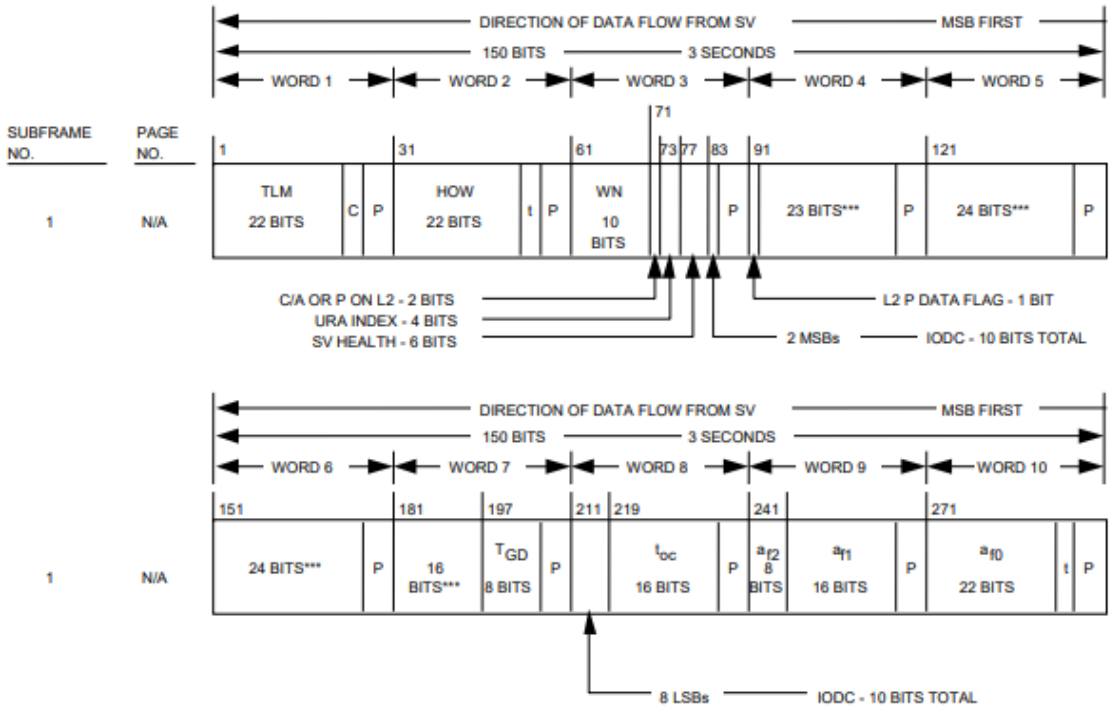


Figure 2.4: The first subframe of the LNAV data [RefH1].

Figure 2.5: Steps in a drone jamming operation. Made by HK.

Neutralizing the drone can be split into two categories; hard-kill and soft-kill. Drone jamming falls into the latter category which means there are no physical components launched or hurled at the drone. Instead, it is attacked wirelessly through radio frequencies.

5.1 Jamming Methods

ABS | *HK*

There are two main jamming techniques, noise-jamming (spot, sweep, barrage) and repeater-jamming (DFRM). Noise-jamming works by sending noise in the form of additive white Gaus-

sian noise (AWGN) on either a specific frequency or multiple frequencies to make the target unable to communicate on said frequencies. The repeater technique is based on receiving a signal from a radar, modulate either the frequency, phase, or amplitude, and then sending the signal back, creating false "hits" on the target radar. This method can be read more about in appendix Q. Figure 2.6 shows the different forms of noise-jamming.

Jamming is highly dependent on power density and the jamming-to-signal ratio (J/S), where "J" is the jammer's transmitting power and "S" is the target signal strength, or the received GPS-signal strength [3]. In a jammer, we want "J" to be as big as possible, but as described in the next sections, this varies with which method being used.

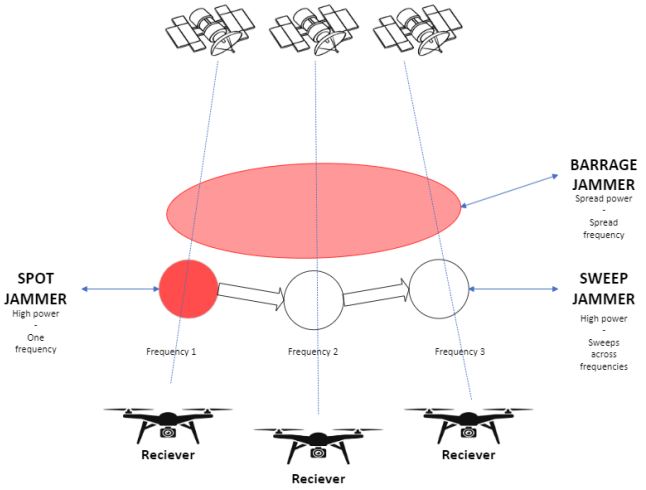


Figure 2.6: Noise jamming methods. (Created by ABS.)

5.1.1 Barrage Jamming

ABS | *HK*

As mentioned in the intro above, power density is very important in jamming. However, with barrage-jamming (fig. 2.7), power density is sacrificed for covering greater bandwidth [3]. This means that the jammer will be less powerful, but will work better against frequency agile receivers and several receivers even. By spreading the power there is some level of jamming no matter what frequency the receiver operates at. The main advantage of barrage jamming is its simplicity and that it can cover much of the electromagnetic spectrum [3]. The main disadvantage is the low power, given that modern receivers often demand high power to be jammed.

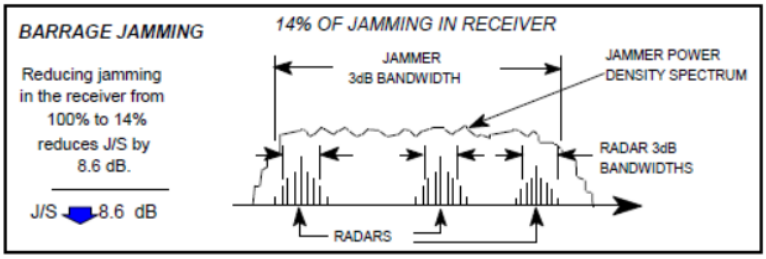


Figure 2.7: Barrage jamming. [3]

5.1.2 Spot Jamming

ABS | *HK*

A way to fully raise the signal power density of a jammer is to use a spot jammer (fig. 2.8). The spot jammer works by being tuned to the anticipated centre frequency of the victim receiver and focusing all its energy on that specific frequency. This increases the jamming-to-signal ratio to a maximum. In contrast to the barrage jammer, the spot jammer cannot jam several frequencies at once. If the need of jamming several frequencies is present, multiple jammers are needed. Even though spot jammers are powerful, they are also quite vulnerable to frequency agile receivers, or receivers that can change their frequency, which means that the operator, or jammer, have to tune the the jamming signal to meet the receiver's frequency [3].

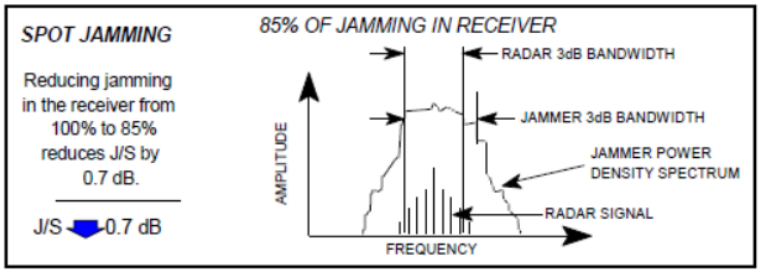


Figure 2.8: Spot jamming. [3]

5.1.3 Sweep Jamming (Swept Spot Jamming)

ABS | *HK*

Sweep jamming, or swept spot jamming, works as a middle point between spot jamming and barrage jamming. As with spot jamming, sweep jamming can only cover one frequency at once. However, as the name implies, the sweep jammer will sweep through several frequencies after one another. This allows the jammer to cover a broader spectrum, but still focus all its energy on one "point". The efficiency of a sweep jammer depends both on transmitting power and sweeping speed (how fast it changes frequency).

Another ability of the sweep jammer is causing vibrations in the receiver made from power bursts from the jammer. If the sweeping speed is fast enough, the vibrations can last from the first power burst until the next power burst arrives [3]. This phenomenon is known as "ringing", and increases the effect of the sweep jammer with a great amount.

5.1.4 Protocol Aware Jamming

ABS | *HK*

In difference to the other jamming methods discussed in this section, protocol-aware jamming does not involve transmitting jamming signals in the form of AWGN. Instead, it transmits a random stream of bits that are modulated through BPSK modulation. The bit-stream, or jamming signal, is specified towards the receiving units, taking center frequency, bandwidth, and data rate into consideration. The constructed signal will then be very similar to the original GPS signal (section 5). This gives it a lower detection probability than the other jamming methods and will cause less interference with other communication systems [RefH8].

6 Modulation Techniques

HK | *ABS*

Radio signals are around us all the time, our phones are transmitting and receiving them constantly. The signals are analog in nature, but through modulation, digital data are implemented into the carrier signal. This modulation happens through different methods and can be explained through how they work [RefH9]:

- ASK Carrier Amplitude modulation.
- FSK Carrier Frequency modulation.
- PSK Carrier Phase modulation.
- QAM Carrier amplitude and phase modulation.

In amplitude modulation, the carrier amplitude is increased or decreased in accordance with the signal properties. In fig. 2.9 we can see that the output has a varying amplitude based on the input above. In the middle, when the signal is zero, the output is also zero.

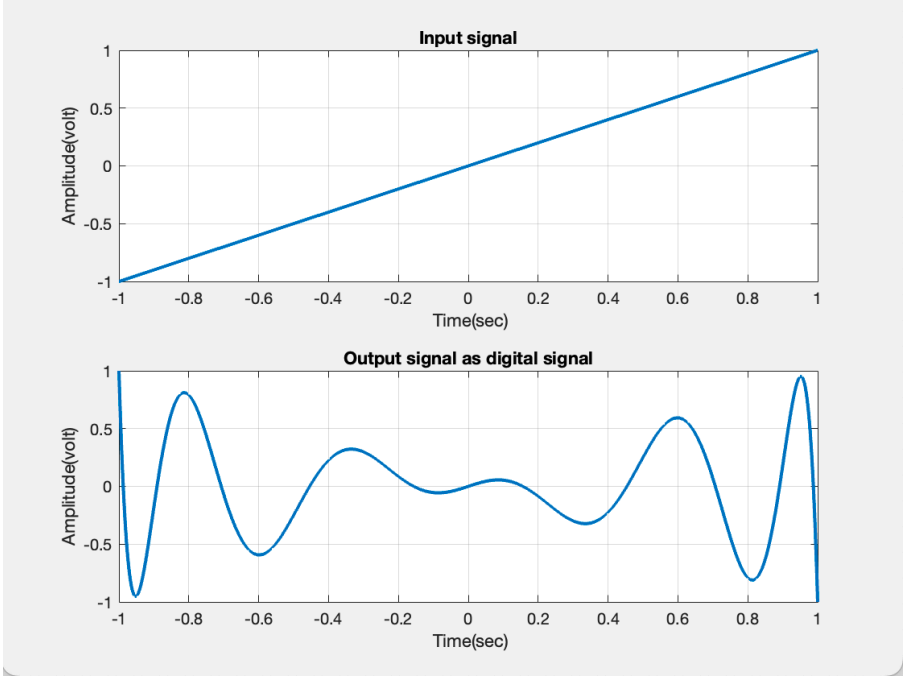


Figure 2.9: Amplitude modulation viewed in time-domain

In frequency modulation, the carrier frequency is adjusted according to the signal properties. For an example, we simulated a two-component sine wave with a 30 Hz and 60 Hz frequency. Then we modulated that signal onto a carrier frequency of 200 Hz with a set frequency deviation of 100 Hz. The resulting modulated signal can be seen in fig. 2.10 together with the original sine wave.

In phase modulation, the phase of the carrier frequency is adjusted according to the signal properties. See fig. 2.11 for an example. In the figure, the top line is the digital signal that will be added to the carrier. The middle is the carrier with the signal modulated onto it (the phase shifts every time the signal goes from one state to the other).

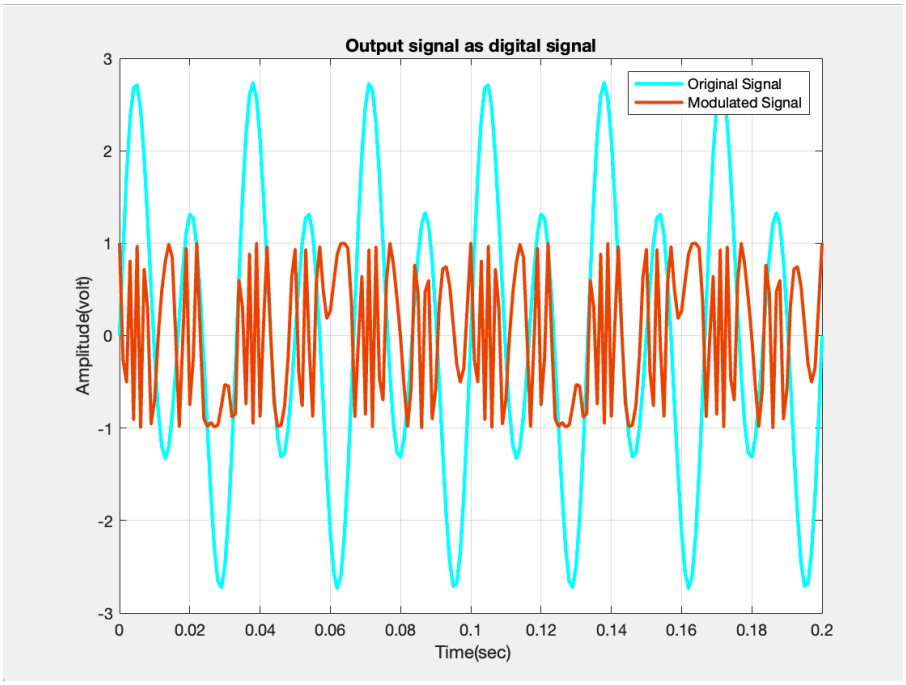


Figure 2.10: Frequency modulation viewed in time-domain

And in quadrature, or amplitude and phase modulation, both the amplitude and phase of the carrier are adjusted according to the signal properties.

GPS signals, specifically the L1-band are modulated using BPSK-modulation where the C/A-code and P(Y)-code are transmitted using two separate bit streams. The C/A-code is transmitted as a 1.023MHz signal and the P(Y)-code is transmitted as a 10.23MHz signal.

These two signals are out of phase by 90° to each other and are added together before being modulated into the carrier. This is a form of PSK called Quadrature Phase Shift Keying or QPSK. See fig. 2.13 for a representation of the final signal.

From fig. 2.12 we can see that the codes are modulated onto the carrier with a central frequency of 1575.42 MHz. The bandwidth of the channel is 15.345MHz.

7 Spoofing

HK | *AM*

In section 5 we introduced the concept of jamming a signal, which is the act of denying access to a radio frequency by sending noise at an amplitude exceeding that of the target signal. Jamming is a relatively simple approach to the denial of service for the GNSS receiver, requiring no real signal analysis or signal processing (except for protocol-aware jamming).

Spoofing on the other hand is the method of generating simulated signals using real ephemeris data from NASA, then transmitting these at the correct frequency in the same manner as the real satellites do. This gives the appearance of a genuine satellite connection for the receiver, and it can lock on to these fake signals and calculate its own position based on the data emitted from the spoofing device. This means that the spoofing device can send out positioning data such that the receiver believes it is in a completely different place than it is. It can also

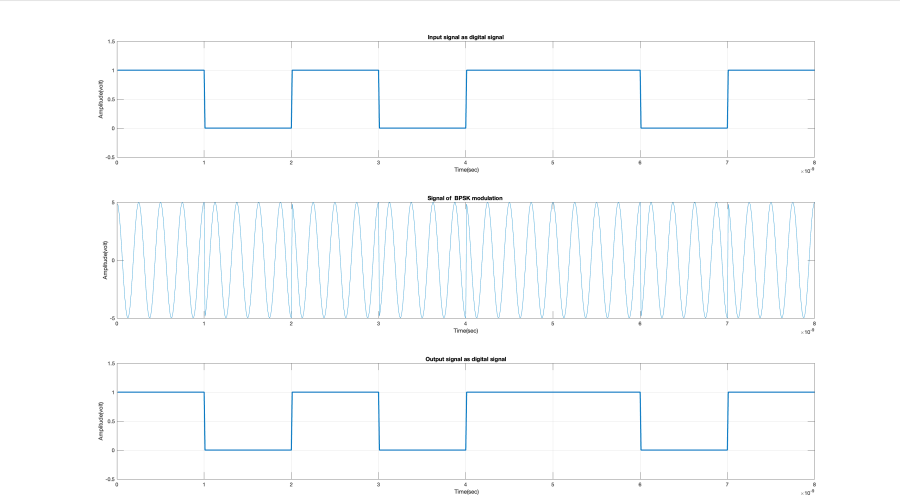


Figure 2.11: BPSK modulation viewed in time-domain

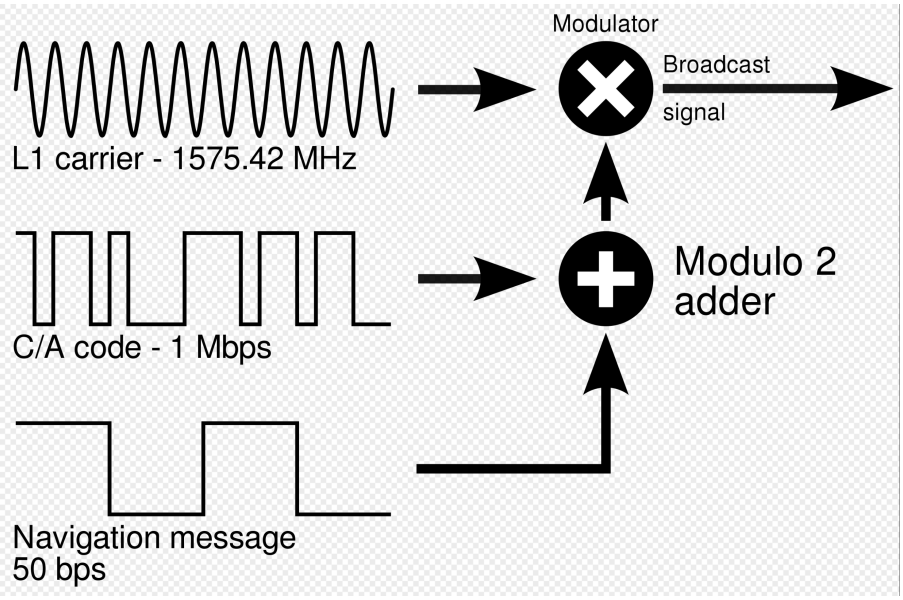


Figure 2.12: C/A-code and NAV message modulated into the carrier signal

change the perceived velocity and the time of the receiver. The goal of spoofing may be denial of service, removing the navigation reliability of the receiver, or to manipulate the perceived location of the receiver to gain control of the system the receiver is connected to.

7.0.1 Asynchronous versus Synchronous Spoofing

HK | *AM*

There are two main approaches to spoofing; asynchronous and synchronous. In asynchronous spoofing, the spoofer needs to transmit data the same way the satellites do, with the same transmission rate, using legitimate ephemeris data. Synchronous on the other hand demands more accuracy. It also requires that the spoofing signal is accurate to within microseconds of the original signal, as well as the Chip delay must be zero. The GPS L1 signal has two separate bit streams (C/A- and P-code). The C/A PRN codes are 1023 chips in length and are transmitted at 1.023 Mchips/s.

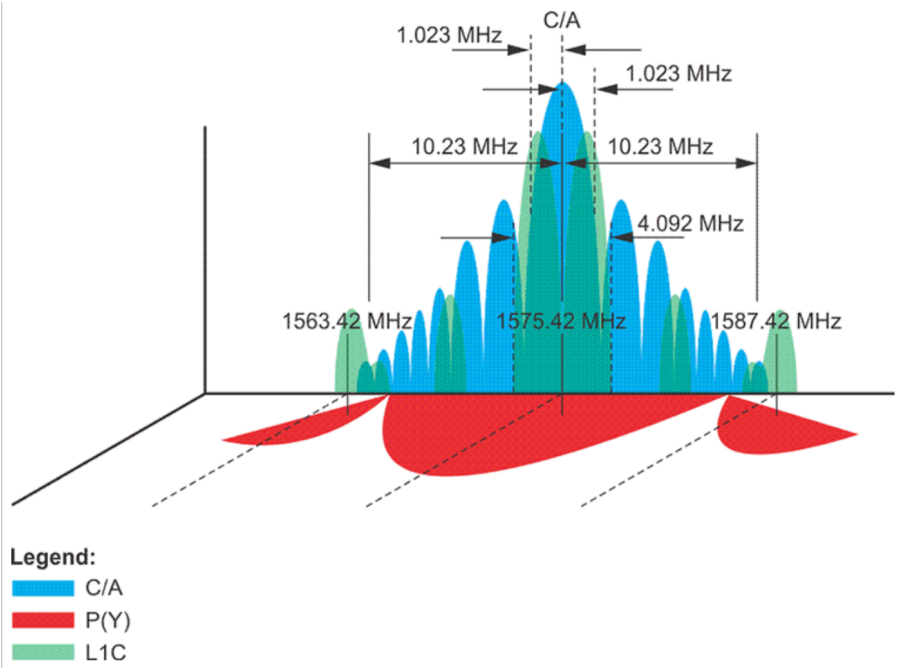


Figure 2.13: Figure showing the final complex modulated signal [4].

This means the code repeats every millisecond. The P codes however are much longer, and even though they are transmitted at ten times the rate (10.23 Mchips/s) they are only repeated once every week [30]. The reason they are longer is to improve the accuracy of the GNSS system. To do this, most solutions for synchronous spoofing involve a separate GPS receiver that is connected to the SDR-peripheral in order to gather information about the ranging codes [RefH10] [RefH5]. The spoofing system must also have knowledge about the location of the receiver to transmit signals accurately to the receiver's antenna [RefH5].

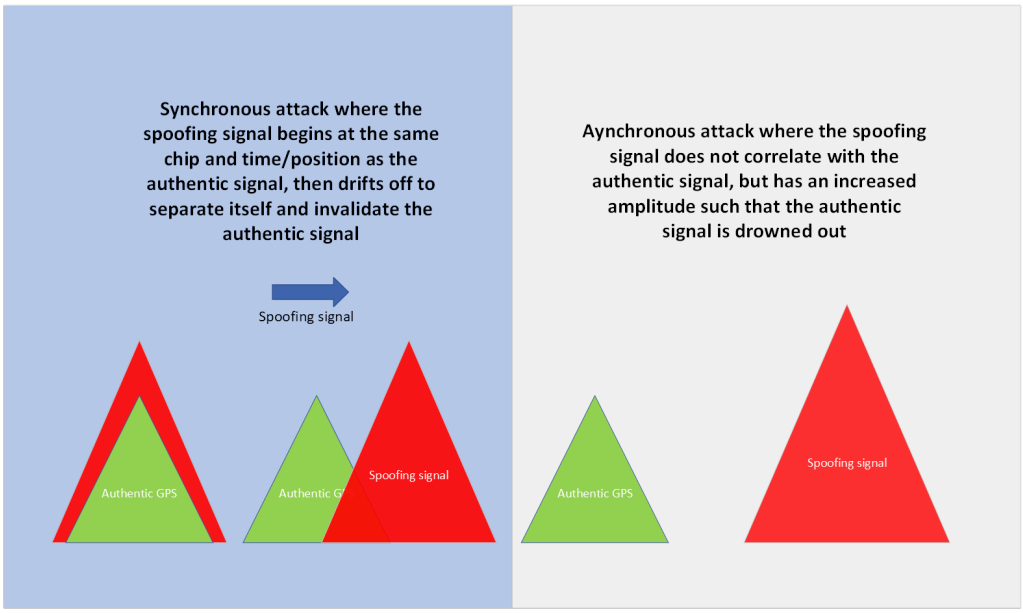


Figure 2.14: How asynchronous spoofing works. Made by HK.

In figure 2.14 we can see the difference between synchronous and asynchronous spoofing from the receiver's point of view. The green triangle is the authentic signal and the red triangle is the signal from the spoofing device. The distance between the peaks is the chip delay. Notice how, in the asynchronous spoofing scenario, the spoofing signal does not overlap the authentic signal.

8 The Python Language

AM | *SN*

Python is a very high-level (see glossary for "high-level language" and "low-level language") general-purpose programming language focusing on object-oriented programming with high readability and is one of the most if not the most popular programming language today [31] [32]. The Python programming language has several different implementations, the reference of which is CPython, which is what Delirium was implemented in [33]. CPython consists of source code written in both Python and C and provides an API for using C/C++ with it (see 10). Despite the advantages CPython provides in terms of readability and abstraction, it does come with some clear disadvantages.

CPython's GIL (Global Interpreter Lock) does not support threading in the sense that one would expect coming from a language such as C++. The interpreter is locked to the main thread, and no threads are allowed interpretation in actual parallel, only one at a time, which prevents multi-threaded programs from taking full advantage of multiprocessor systems [34]. This meant that we had to take special care to not implement any features which could result in a blocking state in the main process.

9 The C & C++ languages

AM | *SN*

The C language is a relatively old one, created in 1970 at the AT&T Bell Laboratories to write the Unix operating system (OS). It has had a huge impact, being used to create OS' and generally applications that require high efficiency and speed [RefH11]. The C language can be characterized as being higher-leveled than assembly languages, but lower-leveled than e.g. Python, making it a middle-level language [RefH2]. The C++ language was also created at AT&T Bell Laboratories but at a later time. The purpose of C++ was to expand the exclusively procedural C language with object-oriented programming by adding functionality such as classes. C++ has since then become a mature language of its own, but is still largely compatible with its origin C in terms of compilation [RefH11].

The C/C++ source code files need to be compiled in order to produce executable programs for a computer. This is a more time-consuming and complicated procedure for the developer than e.g., interpreting Python source code, which is as simple for the developer as starting a program. There do, however, exist excellent tools to make the compilation process for the C languages less time-consuming and even automated. Such a tool is the well-known CMake.

CMake is an open-source, cross-platform family of tools designed to build, test, and package software. CMake provides control of the software compilation process using simple independent configuration files, labeled as "CMakeLists.txt" [35]. CMake is widely used within the development of C/C++ projects, and its popularity has led to it being included as an extension in the Visual Studio Code editor, which provides a simple user interface for performing builds and debugging, with the premise that a compiler is installed on the system.

10 Combining Python with C

AM | *SN*

Python can be extended with C and C++ source code to provide greater speed and to take advantage of C/C++ standard libraries and functionality [31]. There are several ways of doing this in CPython, both with built-in Python packages and with external solutions. The built-in way is described in Python's own documentation [36]. The documentation shows that to integrate C/C++ source code with a Python program, one must include the "Python.h" header file in the desired C/C++ source. This is a C-language header file that comes with CPython by default.

The header file exposes the "Python API", which defines a set of functions, macros, and variables that provide access to most aspects of the Python run-time system [36]. This API is then used to bind the C functions/types to callable Python objects. Next, a setup file must be made that uses the standard Python module "distutils". This file ensures the compilation of the C/C++ extension module and makes a ".so"-file (see glossary for "shared object") which provides access to the module [37]. The built-in extension solution is available on practically every system that has CPython installed and is documented from the official source. It does, however, cause significant amounts of boilerplate code.

```
#include <pybind11/pybind11.h>
namespace py = pybind11;

PYBIND11_MODULE(hackrfstat, m) {
    m.doc() = "Delirium C++ extension to collect status from individual HackRF's"; // optional module docstring

    m.def("getStatus", &getStatus, "Get status of individual HackRF"); // Declaring the function for callability in python
}
```

Figure 2.15: Binding a C++ function to CPython via pybind11. Screenshot from the Delirium source code repository.

Another way of combining C/C++ code with Python is with the external pybind11 library. The pybind11 library is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goal is to minimize boilerplate code compared to the built-in way and other external solutions [38]. The binding of C++ code to Python with pybind11 is done by including its header file in the desired C++ source code, and then using the functions defined in this header to declare the C++ functions/types in Python format, as can be seen in figure 2.15.

The pybind11 library provides a CMake function called "pybind11_add_module" which adds a library target to be built from the listed source files [39]. When this function is used in a CMakeLists file, a *shared object* file is produced which exposes the functions/types in the C++ source file as a Python module, making it available for import statements in CPython source files.

11 Continuous integration & version control

AM | *SN*

Continuous integration (CI) is a popular software development practice that involves forcing developers to upload their work to a central repository and integrate their changes to the code base within it often, perhaps several times a day. This practice has become an industry standard within software development as it dramatically reduces the time spent on making different people's code work together (integration), compared to earlier practices where people would work individually over long periods and then integrate, which can cause enormous compatibility issues which can take easily as much time to fix as it took to make the software itself. CI is usually combined with - or built into - a version control system (VCS) [RefH12][40].

A VCS in the software development context is a software system that manages changes to software projects and enables inspection of who made changes, when they were made, and what was changed. A VCS further allows reverting a project to a previous state, among other optional functionalities [41]. The most popular VCS at the moment is Git, which is a distributed VCS (DVCS). The feature that makes Git distributed is the fact that once a repository has been cloned down from e.g. GitHub to a computer, it exists in its entirety on the local machine and the repository has been completely mirrored [42].

12 Legal restrictions

HK | *ABS*

Jamming, the act of deliberate disruption of communication signals, is strictly regulated and prohibited by law. According to Paragraph 6-2 of "EKOM-loven" (The Electronic Communications Act), any transmission that utilizes the electromagnetic frequency spectrum requires explicit approval from the relevant authorities. This regulation is in place to ensure that the electromagnetic spectrum is without interference, protecting critical communications. Norwegian Communications Authority (NKOM) highlights aviation, shipping, military and police as examples of critical use cases of the electromagnetic spectrum. The strict regulation shows the importance of maintaining reliable and uninterrupted communications in various sectors.

12.1 NKOM Application

ABS | *SN*

For us to be able to test our system/jammer in free-air, we need approval from NKOM. This involves sending an application at least 2 months prior to testing, including a description of our signal characteristics, the transmitting power, and the geographical extent of the transmission

(all of NKOM's requirements can be found in appendix O). This is because NKOM needs time to notify all affected parties, especially emergency authorities, as tampering with navigational systems can cause dangerous situations. We would also have to notify local emergency services, as well as send a "notice to airmen (NOTAM)" if deemed necessary.

As mentioned, the application involves considerations regarding transmitting power. As we only have been able to test with wired connections, we had to support our conclusions based on our testing and J/S considerations. The J/S calculations are based on the transmitting power of our strongest radio (measured while cabled), in addition to information on the signal from the GPS satellites (i.e. transmitting power, gain, distance). These calculations also include transmission loss from the satellites down to specified heights (app. O, sec. 4, pt. 5).

In addition to signal strength, NKOM wants a geographical representation of our noise transmission at our test site. This is again for them to get a better understanding of our test. The test site of choice is at our team member Helge's farm, and its belonging gravel pit. The destination is chosen because the tall walls of the gravel pit will block the transmitted signals in most directions except upwards, and affect the surroundings as little as possible. With the help of Matlab, we could make a script that simulates our radio taking terrain into consideration. However, we could not find the right tool to show this from different heights, from 5 feet to 30000 feet, as NKOM demands.

(The application to NKOM can be found in appendix P.)

Chapter 3

Method

In this section, we will provide detailed descriptions of how we have handled the management of our project. This is an important part of our assignment and a requirement from USN as a part of our bachelor's thesis. We will explain how we have approached the assignment and how it has developed along the way, as well as the foundations of our requirements, design choices, and our milestones, all put together in our "red thread". Further, all the equipment we have used will be presented, both hardware and associated software and firmware. In the end, our development approach will be laid out in detail.

13 Project Management

AM | *ABS*

> "Project management is the application of processes, methods, skills, knowledge, and experience to achieve specific project objectives according to the project acceptance criteria within agreed parameters" [43].

Project management is something any group of people with a goal of creating something useful must consider. For us, it meant that we had to establish a work pattern, make use of a known project model, make use of existing tools, and generally agree upon how we were going to work to achieve our goals.

13.1 General Work Pattern

AM | *ABS*

It was a requirement and a necessity that we establish a work pattern within our group. In this way, like in most businesses, we would know what to expect from each other each working day, which in turn lowers the likelihood of relational friction between group members. We decided in plenary to work with core working hours so that in the time frame 09:30 to 15:00 on working days all team members had to be at the office, with 30 minutes granted for lunch. It was also imperative that we divided the work pattern into two parts; before Easter and after. This had to be done because we all had an extra discipline-specific course to attend to before Easter. Our work pattern became:

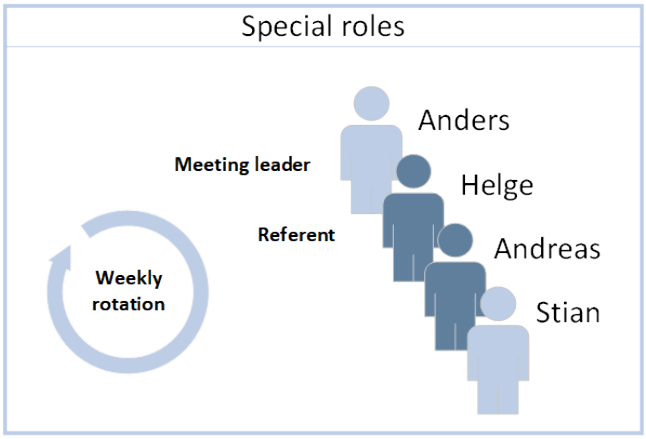


Figure 3.1: Illustration of how we rotated the special roles weekly, created by AM.

- **Before Easter:** Wednesday to Friday each week, with Scrum sprints lasting 2 regular weeks granting 6 workdays. Mondays were reserved for the extra course, and Tuesdays were "joker-days" free to use as one pleased.
- **After Easter:** Monday to Friday each week, with Scrum sprints lasting 1 regular week.

It was also a requirement that we at any time had appointed the special roles of meeting leader and a referent. The meeting leader was responsible for inviting supervisors to meetings and leading the meetings themselves. This was later extended to include all meetings internally. The referent was responsible for taking notes during the meetings and then sending them out afterward. These special roles were rotated between group members weekly, as seen in figure 3.1.

13.2 Project Model

AM | *ABS*

A requirement from the faculty was that no matter which project model we chose, we had to make it "agile". The term agile within this context was made famous by "The Agile Manifesto" which is a document created by several high-standing software developers in 2001 [44]. Our group needed either to find a project model that enabled us to work like this or make one fit into the description. We chose a well-known model that intrinsically supports agile development; Scrum.

Scrum is a framework for a team's work pattern which is intentionally simple and lightweight. It focuses on *transparency* - everyone in the team can see everything that is being done and therefore is equipped to take part in meaningful discussions, *inspection* (enabled by transparency) - everyone is responsible for holding each other responsible by paying attention to the work that is being done, and *adaptation* (enabled by inspection) - the ability to adapt to sudden changes in requirements or other factors [45]. What follows is an explanation of the artifacts and events that comprise Scrum and how we used these artifacts in our project:

- **Sprints** - Some number of periodical and fixed workdays in which the team is expected to complete the sprint goal. We chose 1 regular working week as the sprint's length to

compromise between ensuring enough agility and getting some actual work done.

- **The Product backlog** - A comprehensive to-do list containing all efforts of work that must be undertaken to complete the finished product. In our case, this was renamed to Project backlog because we wanted non-product-related tasks required from the faculty such as presentations, documentation, and so on to be reflected as work done. Both the project backlog and the sprint backlog were kept and maintained in the Jira Software application.
- **The Sprint backlog** - A smaller to-do list containing the sprint goal, which is the expected increment of the product during the sprint it is attached to. See the relation between the backlogs in figure 3.2.
- **Daily scrums** - Daily meetings set at the beginning of each workday used to provide oversight and give all developers the chance to provide or get input from other developers. We had such meetings every workday, lasting a minimum of 10 minutes (due to realizations made), and ensured that they lasted no more than 15 minutes.
- **Sprint planning** - A meeting held at the beginning of each sprint where the team decides which tasks should be included in the sprint backlog. We held such meetings each sprint and ensured that they never lasted more than 2 hours.
- **Sprint review** - A meeting held near the end of a sprint with the intention of inspecting its outcome and adapting future work to any changes that might have come up. Such meetings were held by our team lasting a maximum of one hour with the inclusion of our primary stakeholder KDA.
- **Sprint retrospective** - A meeting held at the very end of a sprint to determine what went well, what didn't go so well, and what can be done to make things better. Such meetings were held internally by our team at the end of every sprint during the entire project.

The plan for how we implemented Scrum is illustrated in figure 3.3. Although we followed Scrum quite closely to its definition, some adaptations were made to better suit our needs. The two special roles within Scrum, namely the Scrum master and the product owner were left out. This was done because we deemed them unnecessary in our case. The tasks associated with these roles fell under the responsibilities of each group member. In addition to the backlogs and task overview which were kept in the Jira Software application, we relied on a separate timeline for visualizing important delivery dates and events. The timeline can be seen in appendix A.

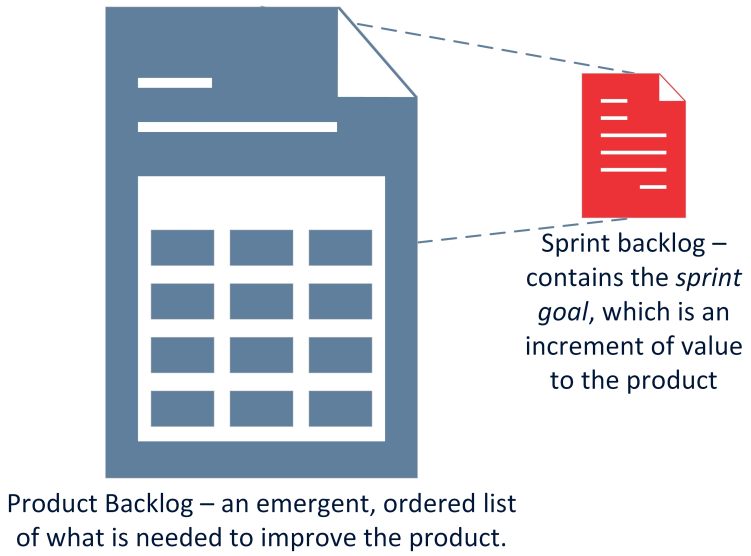


Figure 3.2: Illustration of the backlogs and their relation. Diagram created by AM.

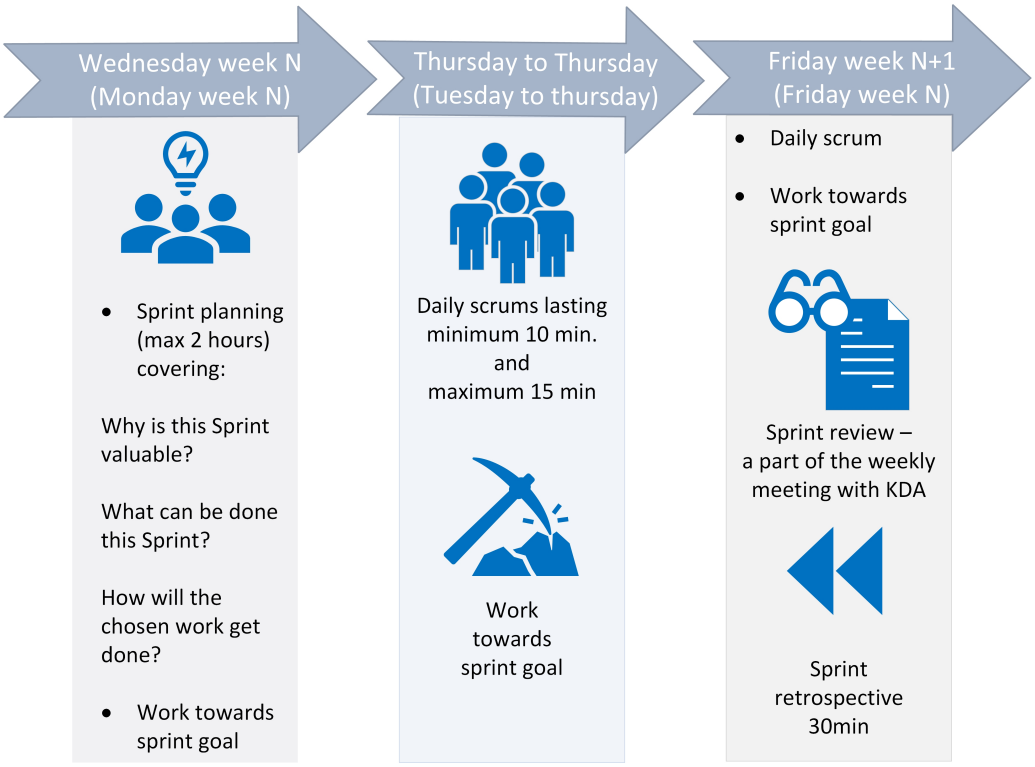


Figure 3.3: Illustration of our project model and work pattern. The days written outside parentheses are before Easter, and inside parentheses are after Easter. Diagram created by AM.

13.3 Project Management Tools

AM | *ABS*

Several tools were used to keep track of progress, working hours, storage, and so on during the lifetime of the Delirium project. In this section, we will cover the most important tools used by us to satisfy our own requirements in regard to oversight, and the faculty's requirements in regard to documentation.

13.3.1 Jira

ABS | *AM*

To help organize our project, we applied a tool called "Jira Software". Jira Software is a software application developed to help projects organize, improve workflow and for general tracking of activities. Atlassian (the creators of Jira) originally created Jira for software development teams, but have over the years been adapted for other types of businesses [46]. KDA use Jira extensively in their day-to-day affairs and was highly recommended by our external supervisor. This allowed us to work similar to the way KDA works and was a great way for our external supervisor to follow our progress.

Within Jira, there are several modules to help manage your project. You can create timelines, backlogs, and Kanban boards and then use milestones and issues/tasks to connect them all together. Every group member can make tasks and enter them into our project backlog, and in our sprint planning, we will agree on which tasks are most important for the milestone we are currently working on reaching. In figure 3.4, one can see our timeline with milestones spread across the calendar as it is displayed in Jira. As our project timeline was based on our milestones, Jira fit perfectly into our plan. Using the timeline feature we could connect tasks directly to each milestone and fill them in from our project backlog.

To keep track of tasks we used Kanban-boards for each sprint. We picked tasks from the backlog and put them into the current sprint. This helped us keep track of what had to be done and gave us an overview of what each group member was working on. As soon as a sprint was finished, we could retrieve a report with an overview of which tasks were included, finished, or not completed. The remaining tasks were then automatically moved to the next sprint, or we could move them back to the project backlog if that suited our timeline better.

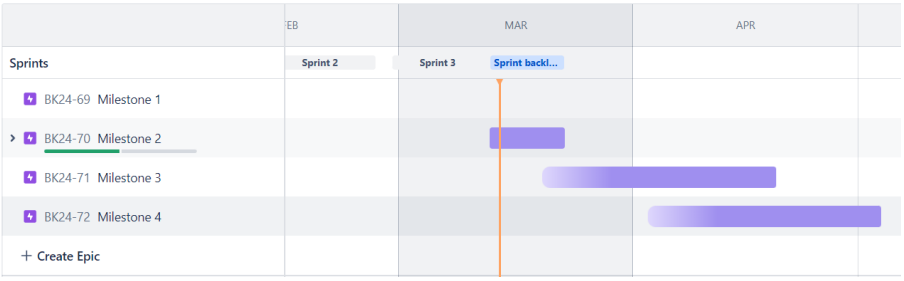


Figure 3.4: Screenshot of our timeline within Jira.

13.3.2 Time-tracking

AM | *ABS*

We needed some mechanism for tracking hours spent working on the project as this was a requirement from our faculty. Based on anecdotal recommendations, we chose to use Clockify. Clockify is a time-tracking application made for teams. It has very good availability: It can be used in a web browser, desktop app, or via their smartphone app. It is free of charge, although one can subscribe to gain more features [5]. We chose to use Clockify as it remained free of charge for our uses, its availability was more than adequate, time-tracking with it was very simple (see figure 3.5), but most importantly because of the reports that can be produced with it in both weekly format and total sum, which made our supervisors very happy.

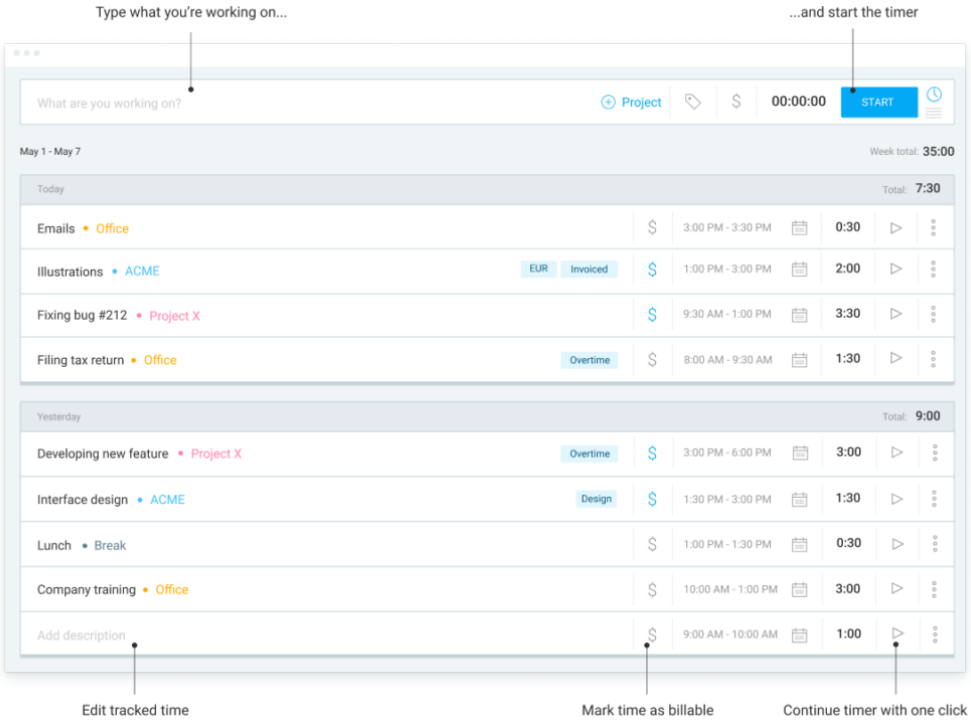


Figure 3.5: Time-tracking in Clockify [5]

13.3.3 Storage, Communication & Report

AM | *ABS*

Right from the start we agreed on using Microsoft's OneDrive cloud solution for storing general files such as diagrams, pictures, documents, etc. All group members had access to this platform through their school account and were familiar with it, so this was an easy choice. The source code for our project was kept on GitHub (see section 16.3 for more on this). Even though we spent every workday in our office at "Krona" (the USN Kongsberg school building) together, we needed platforms to communicate with each other in case of unforeseen events such as sickness or important notifications. Facebook's Messenger application was chosen for keeping an informal chat and Microsoft Teams was used in cases where some group members or supervisors had to work from home. Our customer, KDA, requested a Signal chat to be set up for informal communication with them (Signal is an encrypted messaging service for instant

messaging, voice calls, and video calls [47]).

As sprouting engineers and academics, we needed a platform for writing our report in a "proper" way and to be able to write simultaneously. Based on the experiences we all shared from previous projects and recommendations from our supervisor and earlier students, Overleaf was chosen as our tool for this. Overleaf is an online LaTeX writing and publishing tool that provides an environment for compiling LaTeX code to PDF, simultaneous writing and spell checking, etc.

13.4 Lessons Learned

AM | *ABS*

During the lifetime of our project, we made several changes to improve efficiency, communication, etc. based on emerging thoughts expressed in our retrospective meetings. What follows are the main lessons learned about project management during the project life cycle.

The special role of meeting leader was initially only intended to lead meetings with supervisors. Our group leader took the lead on all internal meetings such as scrums, retrospectives, and so on. This caused notable extra overhead for the leader and made all internal meetings possibly biased. Therefore it was decided in plenary that all meetings and their respective preparations should be led by the current week's meeting leader.

Our Scrums were, from the start, conducted before each working day. However, they were done sitting down and with no minimum time duration. This caused them to become banal, short-lived, and uninformative very quickly, plus it allowed distracting screen time. This issue was taken up during a retrospective meeting and mended by introducing a stand-up policy and a minimum duration of 10 minutes. After this subtle change, our Scrums became very useful, we talked to each other's faces and shared information and plans in a way everyone agreed was helpful.

We started writing the report early on in the project life cycle. It did not take much time for "yellow errors" to start popping up in the Logs section in the Overleaf user interface (see figure 3.6). If these errors were not mended immediately, we experienced that the number of errors soon grew almost exponentially, which caused very time-consuming fixing sessions after some time had passed. We agreed in plenary that we must compile the report often when writing, and correct all yellow and red errors *immediately*.

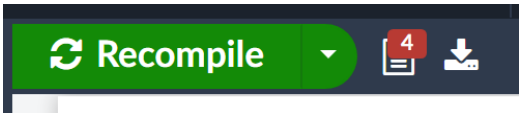


Figure 3.6: Errors in the Overleaf project

14 The Red Thread

AM | *ABS*

In any system engineering project, it is important to retain traceability from what the customer says it wants to what is contained in the end product. This traceability is what we refer to as "the red thread". The red thread in our project is the path that is taken from a stakeholder requirement, through its implementation in the product and with its end in verification. Our red thread can be seen visualized in appendix B, we recommend the reader to consult with this diagram as it shows the full picture. The following sections detail the steps of our red thread, and show how we utilized common systems engineering concepts to achieve traceability and to ensure customer satisfaction.

14.1 Milestones & MVP

AM | *ABS*

The task we were given was not well defined, it was more of a research task rather than a product specification. We were advised from early on that we should somehow define certain milestones so that we had specific goals to reach regarding the end product. It was also important for both us and the customer that we communicated the iterations of the product frequently. This led us to define 4 milestones to be placed on our timeline (see appendix A) and to adopt the concept of the Minimum Viable Product (MVP). By utilizing the concept of MVP, we ensured that no matter how much research/testing we did, we always had a working product to deliver, into which discoveries made were iteratively merged. Our MVP can be seen in figure 3.7. The

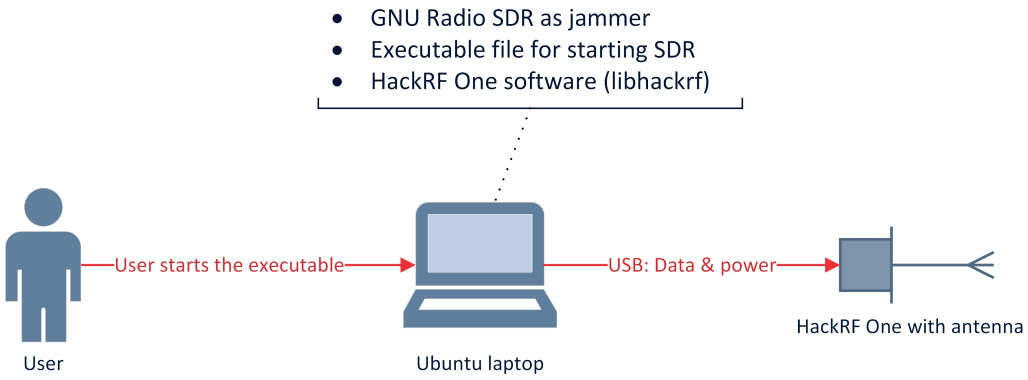


Figure 3.7: Our MVP No. 1, called "basic jammer", is the simplest product that satisfies the most important requirements of both faculty and customer. Diagram made by AM.

milestones were made to spread out the system requirements such that we could iteratively make a working product with increasing complexity. Our milestones can be seen as a diagram with short explanations in appendix C. The diagram presents our milestones as steps in a stair, to illustrate how we iterated on Delirium towards our goal; Milestone 4. The diagram further shows "Low priority/ideal features", which stems from conversations with our customer KDA. These were not requirements at the beginning of the project, but they became requirements as we passed Milestone 4.

14.2 User Stories

ABS | *SN*

User stories are great aids to any project. They were mostly used in agile software development, but have in later years become a natural part of any product development. The user stories are created from the user's perspective, describing functions and goals in simple everyday language without any technical details [48]. This provides context to the development team and their contributions to the end product, as well as creating a framework for the team's day-to-day work [48].

14.2.1 How They Work

ABS | *SN*

The foundation stones of a user story are "who", "what" and "why". These are essential when creating a user story. First of all, one has to know who the user story is for, what the product shall do, and why the stakeholders want the product. In our case, "who" is a penetration tester at KDA's cybersecurity department. A penetration tester wants to test methods of breaking into their systems to test their security. This can be a firewall, missile system, or in our case a flight controller. KDA wants our system to test the flight controller and its ability to withstand attacks through jamming and spoofing, to fix security holes and hinder outsiders from exploiting them.

When the fundamentals ("who", "what", "why") are in place, we can be more certain that we have the customer in focus. All features are added because the customer wants it, not because it is "cool to have", or because features are added just because they can.
Another feature of user stories is the ability to divide the project into interim goals. As the main goal of our project is quite big, we can use the user stories to dissect the project into smaller goals. As we generate requirements from the user stories, we can connect them to the interim goals to give our project stable progress.

To us, the user stories will work as an aid throughout our project. We can with greater certainty know that we are headed the right way and pick out the correct requirements for our system.

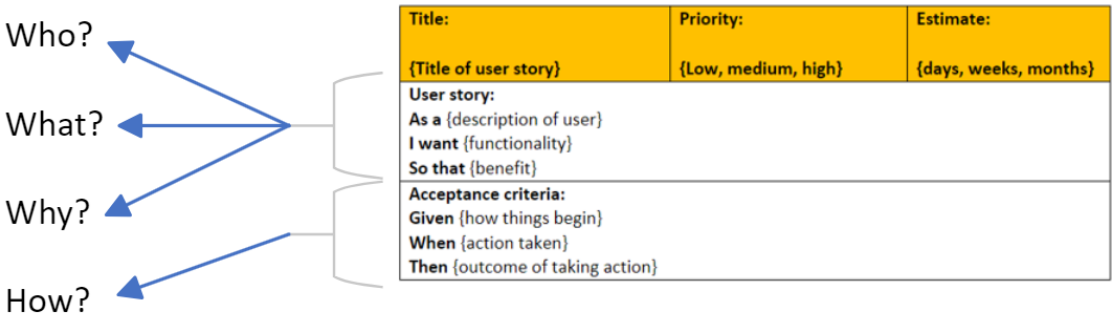


Figure 3.8: User story template.

14.2.2 Our User Stories

ABS | *SN*

In the figure above (3.8), the template for our user stories is shown. Each user story is given

a title, a priority, and a time estimate. These are all directed toward the project and will be used to establish requirements. The user story is then divided into two sections:

- User story: In the user story field, the who, what, and why are established. Description of user, functionality, and benefit are described in short sentences, keeping it simple and easy for everyone to understand, including those without technical knowledge.
- Acceptance criteria: In the acceptance criteria field, the "how" is established. This is where the action is described and what the desired outcome of said action shall be.

As we intend to work agile during our project, none of the user stories are set in stone. Neither is there a finite number of user stories. As the project moves forward, new requirements will present themselves, both from our stakeholders and around the technicalities of our product. We are then able to create new user stories in cooperation with our stakeholders, to make sure the framework we have established is preserved. All our user stories can be found in appendix D.

14.3 System Requirements

HK | *ABS*

The system requirements for Delirium were initially derived through collaborative efforts in stakeholder meetings, where the user stories played a pivotal role in capturing essential functionalities and ensuring the system aligned with the customer's specific needs and objectives. The system requirements for Delirium can be found in their entirety in appendix E along with their respective derived requirements, our design choices, and the respective technical requirements for each of these. This overview serves as the basis for the formal description of Delirium from the customer and other stakeholder's statements.

14.3.1 System Requirements Structure

AM | *ABS*

The fields that each requirement consists of can be seen in table 3.1. For some of the items in the overview (see appendix E), all fields were not deemed necessary. This decision was made because the design choices and technical requirements serve to realize either a top-level or a derived requirement, and so they simply inherit the e.g. test plan for that requirement. The following is an explanation of each requirement's fields as seen in the overview:

ID:	**Source:**	**Attached milestone:**
Verification method:	**Compliance status:**	
"Requirement description"		

Table 3.1: System requirement fields, creator: AM

- **ID:** Identification. Is of one of the following types:
    1. "TLRx" = Top-Level Requirement no. x. A direct or translated requirement from a stakeholder.

2. "DRx.y" = Derived Requirement from TLRx number y.
3. "DCx.y" = Design Choice from TLRx number y.
4. "TRx.y" = Technical Requirement from TLRx number y.

- **Source:** Where the requirement comes from. Can be from a User Story (US) or another requirement.
- **Attached milestone:** The milestone for which this requirement is relevant, as seen on the milestones diagram in appendix C.
- **Verification method:** How this requirement will be verified. Refers to a test plan, identified as "Tx" = Test number x.
- **Compliance status:** Whether or not this requirement is satisfied. "Complies in ..." has been used in this field. The meaning of this is that the requirement has been satisfied and verified in a certain passed milestone.
- **Type:** The classification of the requirement in regards to the Systems Engineering Body of Knowledge (SEBok) wiki page about system requirements (Table 2) [RefH13]. This classification was used by us to ensure a "full picture" understanding of the scope of our system. The type also, for technical requirements, specify the discipline for which it is relevant (Electronics or Software).
- **Requirement description:** Textual description of the requirement.

To ensure the top-level and derived requirements' quality during their formulation we made sure that each of them fulfilled the "system requirement characteristics"-checklist defined on the SEBoK wiki page regarding stakeholder requirements (at [RefH13] in table 3).

14.3.2 Testing the Requirements

HK | *ABS*

After we had our requirements, we had to create a framework for testing so that we could verify our system met the requirements. To give us a high confidence level that our testing was sufficient, it was decided to have some preset fields that could be filled out before the test so each member of the group could understand the testing methodology. From fig. 3.9 we can see that this test was done to verify requirement DR1.1. The description is "Using the equipment without power infrastructure". We then specify the steps to execute, the expected result, who has conducted the test, and who it has been peer-reviewed by. Finally, there are fields for actual results, changelog, and comments.

The Changelog field is for specifying changes made to the connected requirements such that the testing can be updated to follow the new requirements. The actual result is where we filled in what our observations were, and then we could later compare that to the expected result to see if there were any discrepancies.

Scenario Test ID	DR1.1 (derived requirement)	**Status**	Completed and verified
Scenario Description	Using the equipment without power infrastructure	**Priority**	High
Test Case ID	T1.1.1		
Pre-Condition	System must be connected to a power bank or laptop	**Steps to Execute**	1: Make sure the Micro-USB interface is fully seated, 2: The laptop has battery and is powered up 3: The program is loaded and ready to run
Exp. Result	System operates as normal and sends out signals on the frequency determined by the software	**Act. Result**	Using the Ubuntu laptop and the signal analyzer we tested the system wired and found the HackRF sent out a 20MHz wide signal at 1.57548GHz
Work done by/date	Created: HK/15.02.2024 Executed: HK/01.03.2024 Peer-reviewed: AM/01.03.2024	**Changelog**	
Comments	The delta central frequency between the signal we sent out, and the central frequency reported by the signal analyzer was 60kHz, this was unexpected, but could be down to measurement inaccuracies, or explained by the lackluster oscillator in the HackRF One which is 20ppm, meaning that at 1.57542 GHz, it could deviate by as much as 31.5kHz.		

Figure 3.9: Our fields for ensuring correct testing

If the expected and actual results matched, we could mark the status field as "complete", and after peer review "completed and verified".

This testing documentation can be found in full in appendix G. Towards the midpoint of the project we were at the stage of importing the testing documentation into the report. We then decided to write subreports into LaTeX, and all testing from that point onwards was only written into the report, therefore the Excel sheet is not complete. For the updated testing documentation, review appendix 8.3.

14.4 Risk Assessment

HK | *ABS*

Behind every project lay latent risks that could disrupt or ruin the project should they come to fruition. Therefore it is crucial to identify and concatenate these risks in a fashion that all team members can see. Being open to the fact that these risks are present, how likely they are, and how big of an impact they can have is important to a successful project. After they are identified, the team can create mitigation strategies and document what actions they take to avoid being vulnerable to the identified risks.

A risk can be identified using some common identifiers:

- Uncertainty: A common point of risk is uncertainties in the external or internal environment.
- Dependencies: Projects are often dependent on other factors, like external entities and components
- Change: Changing the requirements, technology, and regulations among others could impact the budget, timeline, and overall success of the project
- Human factors: Team dynamics, skill gaps, turnover or inadequate communication are also potential sources of risk.

ID	RISK DESCRIPTION	Effect on project			Mitigation Strategy
101	The team work falls apart and our project progress stalls	Our product does not meet its requirements			Using SCRUM project method with short deadlines should visualize the progress in a better way, so we can identify if and where progress has halted in time to rectify the problem and devote resources accordingly.
		Chance	Impact	Risk	
		2	5	**10**	

Figure 3.10: Risk Assessment example

- Financial factors: Budget constraints, fluctuations in financial markets and unexpected costs can all lead to risks within the project.

Our list of risks can be viewed in appendix F. In the initial iteration of our risk matrix, we had several additional fields, and our risk factor was divided into effects on health & safety and the effect it would have on the project. Each of these factors was then scored individually before being added together to become the score for that risk. We found out that this way of dividing the factors up made it more time-consuming to evaluate risks, and eventually, it became ambiguous if the effects could be distinguished from each other. There was a high level of correlation between the effect on the project and the "HSE"-score in most cases, therefore we decided to merge the two without losing any significant data.

A crucial field for us was the mitigation strategy (see fig. 3.10), which forced us to look at possible solutions to these hypothetical situations. Often this made us more aware of the risks and how to avoid them and forced us to acknowledge them.

14.5 Design Choices

ABS | *SN*

Not all features and functions of a system can be tied to a requirement. There can be several reasons for this. In our case, the task given from KDA was somewhat open to interpretation and we could define it the way we understood it. This gave us some challenges regarding defining what a requirement was and how to distinguish requirements from design. To separate requirements from design, we had to establish what the difference between the two was. The difference can be explained as: "a requirement represents a need, the design represents a solution" [49]. As mentioned in section 14.2, there has to be a clear separation between what the customer asks for and extra features. However, given the freedom of the task, we could still choose some features or solutions we wanted to use. This is what we call ”design choices”. The design choices enable us to organize the process the way we see best. This doesn't mean that we can disregard the requirements, but we can choose which path we use to get there and in a way work in parallel with the requirements. This also involves deciding what components to use, what kind of UI to create, and other aspects of the process and finished product.

Although the design choices are not directly connected to requirements through the needs of the stakeholder, they still share roots with the requirements in the sense that they offer a solution to a problem. As the work progresses towards our milestones, many problems appear along the way. The design choices we make help solve these problems, and ensure stable progression toward our end goal.

The extent of the design choices became bigger towards the last couple of months of the

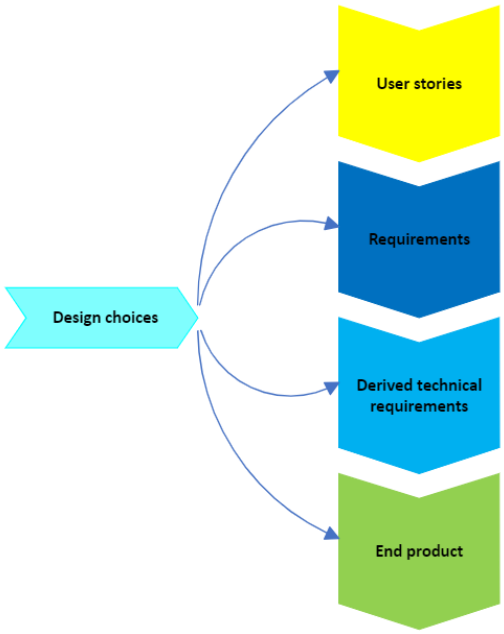


Figure 3.11: Design choices and the "red thread" (Created by ABS).

project. This is due to us realizing that we would reach our milestones and give us more freedom to explore options we had earlier set aside, as well as develop the end product itself. It was important for us that all design choices stayed relevant to the task, were well thought out, and that they all had a significant contribution to the product. This was to make sure that the stakeholders were satisfied with what we delivered, as well as giving us more ownership of the project itself.

The list below presents some of the design choices we have made during the project:

- **UI:** The requirement from KDA is that the system must have a user interface. The requirement states no other details, so we decided to go for an easy-to-use graphical user interface (GUI) to answer their request (section 16.6).
- **HackRF One:** In conjunction with jamming, we needed a transmitter to send out our jam signals. The decision fell on the HackRF One. By using two of these, we can both jam and spoof the Navio 2 (15.2).
- **SDR:** This choice is connected to the requirement to jam the Navio 2's GPS signals. We went for software-defined radio (SDR) to create and configure our radios (jammers) (section 15.1).
- **GPS-SDR-SIM:** The requirement says to spoof the Navio 2 flight controller. We utilized GPS-SDR-SIM to create false GPS signals and send them to the flight controller (section 15.3).
- **Portable system:** We initially thought of a laptop with a radio transmitter as our system. However, we went for a system including two HackRFs controlled by a mini-PC (Raspberry Pi 5) and touchscreen, all packed into a briefcase (section 15.4).

(All design choices are explained in detail in their respective sections.)

15 Equipment

AM | *ABS*

This section covers the equipment we used throughout the project, what they are, their capabilities, and why we chose them. Some of the equipment discussed here was not part of Delirium directly, as we scoped the project down to attacking a specific flight controller (The Navio 2, starting at section 15.9), which we had to set up and interact with in parallel with Delirium.

15.1 GNU Radio

AM | *ABS*

GNU Radio is a free and open-source software development toolkit that provides signal processing "blocks" to implement software-defined radios (SDR). It provides a graphical interface called "GNU Radio Companion" which reduces the complexity of implementing an SDR down to placing "blocks" around, much like a flowchart. In GNU Radio jargon, "blocks" are a general term referring to filters, demodulators, decoders, sources, sinks, etc. which traditionally are found in radio systems. The "flowgraph" is the basic data structure in GNU Radio, which represents the connections of the blocks through which a continuous stream of samples flows. The concept of a flowgraph is an acyclic directional graph with one or more source blocks (to insert samples), one or more sink blocks (to terminate or export samples), and any processing blocks in between. A simple example flowgraph can be seen in figure 3.12.

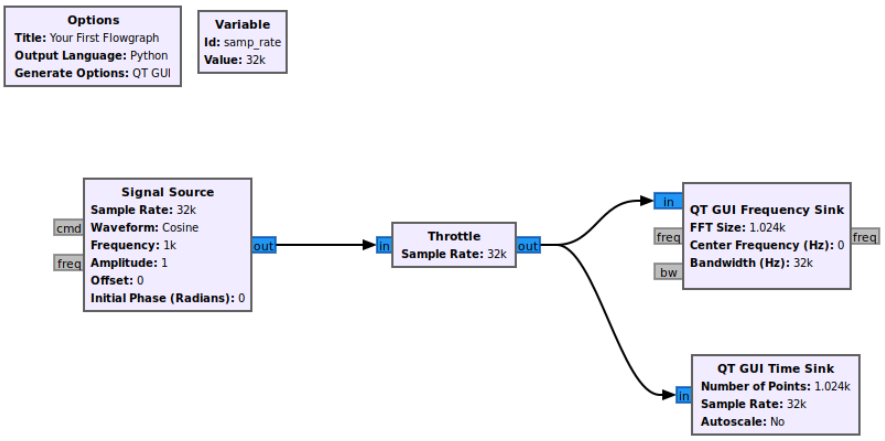


Figure 3.12: An example GNU Radio flowgraph within the GNU Radio Companion GUI [6].

GNU Radio is made to support readily available low-cost external RF hardware, often called SDR peripherals, or to work without hardware in a simulation environment. It is a popular way to implement SDRs within academia, among hobbyists, and also commercially [50][51]. GNU Radio is licensed under the GNU General Public License version 3 (GPL v3), which gives us permission to copy, distribute and/or modify it under its terms. Integration of HackRF One into GNU Radio is provided either via the integrated gr-soapy, which is a GNU Radio wrapper for the Soapy SDR library or via gr-osmosdr, which is a generic GNU Radio SDR I/O block

that interfaces with libhackrf [52][53][54].

When a flowgraph has been made in GNU Radio Companion, the flowgraph can both be run and "generated". When a flowgraph is generated, a runnable Python (or C++, depending on the block's compatibility) source file is produced which contains a class definition of the whole flowgraph with get-and-set-functions for its variables. The flowgraph can then be imported just like any other Python module and be manipulated at runtime.

15.1.1 Creating Custom Blocks

AM | *ABS*

GNU Radio supports modification to a long extent. It enables the creation of one's own custom blocks to be used just like any native blocks within the flowgraph. There are 2 main types of such custom blocks: embedded blocks and Out Of Tree (OOT) modules. Embedded blocks are tools to quickly prototype blocks within a flowgraph. They can only be created with Python and can only be used within the flowgraph it was created. An OOT module can be thought of as a collection of custom GNU Radio blocks. It is the more permanent alternative to the embedded blocks and can be created in either Python or C++. The GNU Radio wiki is packed with tutorials for these subjects, and it suggests the following general steps to create an OOT module: [55][56][57]

- Create an out-of-tree module using gr_modtool
- Create a new Python/C++ block using gr_modtool
- Modify the Python code or C++ .h and .cc code in a text editor so the block will function
- Modify the YAML file so it can be displayed in GNU Radio Companion (GRC)
- Install and run the block in a flowgraph

15.1.2 Why We Chose GNU Radio

AM | *ABS*

Right from the introductory conversations with our customer, KDA, we were advised to take a look at Software Defined Radios (SDR) and to assess whether this concept could help us in the project. After some research, several members of our group simultaneously had found GNU Radio and its respective GitHub repository and wiki-page. It became clear that GNU Radio was a commonly used, seemingly beginner-friendly (as much as it could be) way of implementing SDRs. After some experimenting, it also became clear that the radios produced with it could be controlled by custom scripts and that the libraries supplied by GNU Radio could be interfaced with, thanks to them being open-sourced. When we in addition to the facts quoted here also found an SDR peripheral device that was compatible with GNU Radio (HackRF One, see section 15.2), we settled with this choice.

15.2 HackRF One

AM | *ABS*

Software Defined Radio (SDR) is - perhaps unsurprisingly - mainly software-based. It is,

however, essential for radio communication that there exists a mechanical component capable of receiving/transmitting the signals, as the signals themselves are physical entities moving through space. In our case, this component was chosen to be the HackRF One by Great Scott Gadgets. HackRF One is an SDR peripheral designed to enable test and development of radio technologies. It can be used as a USB peripheral or programmed for stand-alone operation, is a continuously developed open-source platform, and is capable of transmission or reception of radio signals from 1 MHz to 6 GHz [7]. A HackRF One device can be seen in figure 3.13.

Figure 3.13: The HackRF One with its antenna output visible on the left, and clock synchronization inputs/outputs on the right [7].

15.2.1 Software

AM | *ABS*

With the HackRF comes the opportunity and necessity to take advantage of its tailor-made open-source software. The software made for the device includes command-line utilities, called tools, which allow interaction with the device. Among the tools lies the hackrf_spiflash program which is used to update the device's firmware [58]. The software also includes the low-level library "libhackrf" which enables a host computer to interface with the device through its USB ports [59]. Libhackrf is a C-language library consisting of many functions related to polling connection status, transferring data over USB and settings modification. The software can be found in its entirety at GitHub [60], all of which is licensed with the GPL 2.0 (GNU General Public License), which grants us permission to both distribute and modify the software under its terms [61].

15.2.2 Hardware

HK | *ABS*

The HackRF One is run by an embedded microcontroller that is responsible for managing various functions of the device, such as USB communication, controlling the transceiver, and executing user commands. The microcontroller works in conjunction with a Complex Programmable Logic Device (CPLD) which interfaces with the microcontroller and translates the

user commands into functions. It performs tasks such as signal mixing, filtering, and controlling the frequency synthesizer. It is a core reason for making the HackRF a suitable peripheral for software-defined radio usage. Many similar systems use Field-Programmable Gate Arrays (FPGA) to accomplish the same tasks. FPGA's are more configurable and usually deliver lower latency in real-time signal processing. In our usage of the HackRF this does not affect us, and the CPLD is more than sufficient in this regard. Another advantage of a CPLD is the low-power usage and cost efficiency of the chip, reducing the overall cost of the HackRF [62].

The HackRF also has an internal clock system, utilizing a 20 PPM oscillator to generate and manipulate signals. 20 PPM means that for a signal centered around the GPS-L1 center frequency, we could drift off center by as much as 31.5 kHz. This was not crucial for our MVP-1 but would make spoofing the signal near impossible. Therefore we ordered an oscillator that has a 0.1 PPM accuracy and is temperature compensated in order to mitigate this risk. The HackRF has an internal header that the ordered part interfaces with, so no soldering is required to perform the upgrade.

The radio frequency transceiver on the HackRF One is a half-duplex transceiver, meaning that it can both receive and send radio frequency signals. Half-duplex means that it is not capable of doing this simultaneously. For our use case, this is not necessary. Should it become necessary there is also a remedy to implement two HackRF systems together, one for receiving and one for sending signals.

For the transmission method of the signals, we had the option of using any passive antenna with an SMA-M connector. Our HackRF came with an ANT500-antenna, our antenna of choice. Given the fact that it is highly illegal to send GNSS signals over the air [63], we also acquired some cabling, which enabled us to test our system outdoors without legal ramifications. This included a splitter and SMA-MCX couplings to interface with the antenna port on the Navio2 chip.

15.2.3 Why We Chose HackRF One

HK | *ABS*

When we began the project in January, we saw that there had to be a physical radio transmitting the jamming and spoofing signals. Therefore we began investigating options and saw that there were several options. In parallel, we increased our understanding of the signals we needed to emit in order for our system to function properly. There had been mentions of SDR by KDA before, and we saw that this could be a solution. There was some initial uncertainty as to which signals we were supposed to interfere with, this being the controller signals or the GPS signals. Therefore we focused on getting hardware that could accomplish both. This means it needed to be able to transmit on a frequency in excess of 2.4 GHz. We found several solutions that could encompass both solutions. The next important attribute of our solution was cost and availability. This was a main concern, given the constraint of our budget and the short timeline.

When considering this, we compared the different options in appendix I. Here we found that with the exception of RTL-SDR, all of the solutions could work. The availability of the AntSDR

at the time was not good, and shipping time was high. The Ettus is not fully open-source, and PlutoSDR was on backorder. We were therefore left with the option of the LimeSDR and HackRF One. The LimeSDR is a crowd-funded SDR peripheral and quite new. The HackRF One on the other hand is a well-established solution with good documentation and a GitHub page where users can post questions about issues and get help [60]. Given that the HackRF One was cheaper and had a shorter shipping time as well, we decided to use that peripheral.

15.3 GPS-SDR-SIM

AM | *ABS*

GPS-SDR-SIM (Global Positioning System - Software Defined Radio - Simulator) is a C-language program freely available on GitHub that generates GPS baseband signal data streams based on a given ephemeris data set. The ephemeris data set must be downloaded from NASA's website [64]. The generated data stream can then be converted to radio frequency waves and broadcast from common SDR peripheral devices such as HackRF One. After the program has been locally built, the user can run the program "gps-sdr-sim" followed by options to either specify a user-defined trajectory contained in a local file or specify a static location with coordinates. The position(s) given is then used by the program, together with the GPS ephemeris data set, to generate the simulated ranges and LNAV data (see section 4.3.1) for the GPS satellites in view. This range data is then used to generate the digital I/Q (In-phase and quadrature component) samples for the GPS signal [65].

15.3.1 Why We Used GPS-SDR-SIM

AM | *ABS*

If Delirium was to be capable of spoofing GPS signals, we needed a way to generate false signals with good quality. We chose to use GPS-SDR-SIM because in our thorough preliminary research, we saw that others had used it for similar purposes with success [RefH14]. Furthermore, it is open-source and licensed with the MIT license which grants us permission to use and modify it under its terms. The program is also compatible with HackRF One and provides explicit instructions to work with it.

15.4 Briefcase

ABS | *AM*

Since our design is based around a briefcase to fit our components inside, we had to find a fitting case. Our choice is based on three main factors: size, price, and quality. The briefcase must be of good quality, as we want to use our system in remote locations, as well as delivering a good product to our stakeholders. Another aspect of portability is the size and weight of the briefcase. The components we use are all quite lightweight, so the main weight will be the briefcase itself. That makes it important that the case is as light as possible, but at the same time meets our other requirements. Another aspect of choosing a briefcase is the price of it. Since we are working on a limited budget, this is quite heavily weighted in the process of choosing a briefcase.

After researching and finding several options, we initially wanted a type called Peli Aircase (generally known as Pelicase). Pelicase is well-known in the industry, and especially known for

Figure 3.14: Jula Protection Case M [8].

being robust and of good quality. Their cases come in a variety of sizes, and we were able to find several that suit our needs. However, a glance at our budget quickly excluded the Pelicase from the list, and we had to search for other options. Since our first choice was out of reach, we further searched for options within our price range. Even though we had to lower the price range, we still didn't want to lower our requirements regarding quality. We searched on for other vendors and brands and found several options with varying quality. Some of the cases could be excluded merely based on size and others based on quality.

We quickly concluded that our budget simply wouldn't allow a case with quality like the Pelicase. This ended with choosing a case from Jula. Jula's "Protective case M" suited our needs of both price and size, however at a lower quality than what Pelicase offers. Still, the looks and quality of the case, as well as the reviews of it made us more comfortable with choosing this case, and it even seemed to outrank some of the pricier options from Opticase/Nuprol. The case weighs 3,5 Kg and has inside measurements of 425x155x284 mm, which is plenty of space for our components. The price of the case is 549,- NOK, which suits our budget well. The inside of the briefcase is filled with three layers of foam that can easily be cut out/removed (squared pattern) to fit our components. This gives the components extra protection and gives us the opportunity to arrange the case the way we want. The case is shown in figure 3.14.

In table 3.2, one can see the briefcases we considered using, with price, size, and quality review. The table is divided by colors, where red is the options that was disregarded, either because of prize or size. Yellow are options that we could have used but didn't choose, and green are the best options. Quality is ranked by medium and high, based on customer reviews and previous experience with the brands and stores in question.

15.5 Raspberry Pi 5

ABS | *SN*

Name:	Vendor:	Measurements:	Weight:	Quality:	Price:
Beskyttelseskoffert M	Jula	425x155x284 mm	3,5 Kg	Med	549,-
Beskyttelseskoffert L	Jula	505x350x140 mm	4,5 Kg	Med	899,-
Koffert m/hjul, IP65	Clas Ohlson	510x285x195 mm	6,2 Kg	Med	998,-
Opticase	Fosen Tools	442x355x170 mm	-	Med	1323,-
Nuprol - M - Utstyrskoffert	Game On	463x372x182 mm	-	Med	1199,-
Peli 1485 Air Case	Rufo	451x259x156 mm	2,4 Kg	High	4320,-
Peli 1555 Air Case	Rufo	584x324x191 mm	4,1 Kg	High	5760,-
Peli protector Case 1600	Elfa Distrelec	493x616x220 mm	5,9 Kg	High	4992,-
Robust transportkoffert	Max Sievert	430x300x170 mm	2,9 Kg	Med	1519,-

Table 3.2: Briefcase options.

Figure 3.15: The Raspberry Pi 5 (screenshot from kjell.com).

Since our system is built as a stand-alone product, we needed a computer to control all its functions. We decided on the Raspberry Pi 5. This mini-PC comes with all the ports we need, as well as providing enough processing power to run all our peripherals. The most process-demanding peripheral is the HackRF One. However, with 8 GB of RAM and high-speed USB 15.5, this will be no problem for the Raspberry Pi 5. The Raspberry Pi also comes with a designated connector for an external display which is specially made for the Raspberry Pi Touchscreen, and frees the USB ports for other peripherals.

There are other options for PCs to run our system, but the price, form factor, amount of ports, and the ability to power the Pi through a powerbank make the Pi 5 the best option for us. Figure 3.15 shows the Raspberry Pi 5 before it is installed on the touchscreen.

15.6 Touchscreen

ABS | *SN*

Since our system includes a user interface, we needed a peripheral to enable the user to interact with it. With the Raspberry Pi 5 comes a few alternatives: either a keyboard and mouse, a console controller (or similar), or a touchscreen. Since we would like to keep our system as compact and portable as possible, we quickly decided to go for a touchscreen.

After deciding what solution to go for, we started researching possible options for touch-screens. As with the other components like the briefcase, the price was weighted quite heavily in the process of finding an appropriate screen. With that in mind, we decided to go for the

Figure 3.16: a) Raspberry Pi Touchscreen. b) Backside of the touchscreen (screenshot from elfadistrelec.no).

screen made by the same company as the Raspberry Pi, the "Raspberry Pi Touchscreen". This is a cheaper, yet optimal screen for our use. The screen is easy to set up with the Raspberry Pi and the Raspberry Pi can be mounted to the back of the screen for a snug fit.

15.7 Power Management

ABS | *SN*

Since our system must be portable and used in remote locations, we had to find a way to power all the components in the briefcase. There are a few ways to do this, but the best solution involves either a battery pack or a powerbank. To decide which solution is best, we had to take a look at what components we were using. The main component to power is the Raspberry Pi 5 (section 15.5), the system's "brain". The other components: HackRF One and the touchscreen, will be powered via the Pi. All the components will have their requirements regarding power and power consumption, but the main focus is on the Raspberry Pi. This PC requires 25 W power input to function the way we want it to. The Pi's power input can be reduced to 15 W, but that will limit the amount of peripherals it can power to 600 mA. As one can see in table 3.3, we will exceed this limit with double the amount of the limited power, so we had to find a solution that could deliver at least 25 W [66]. The power adapter that comes with the Raspberry pi 5, is a 27 W USB-PD adapter. So, the desired option was to match this.

The power requirement from the HackRf One and the touchscreen are rather moderate [67][68]. The touchscreen draws about 250 mA of power (full brightness), and the HackRF one draws around 500 mA. Our system uses two Hack RF Ones, so the accurate amount is 1000 mA. As mentioned earlier, this exceeds the peripheral power draw on a limited input of 15 W, which again means that we had to aim for a power supply that could provide at least 25 W.

The Raspberry Pi 5 has several USB ports to use, as well as supporting power input via USB-C. This facilitates the possibility of using a powerbank to supply the components with power, as USB is standard on powerbanks. We also found that many other Raspberry Pi projects used a powerbank as their power supply, so the solution has been proven to work.

To calculate the maximum running time of our system, we first had to find out how much power we consumed. This is shown in equations 3.1, 3.2 and 3.3. First, we add together the numbers from table 3.3, which can be used to find the running time in hours in equation 3.3.

Component:	Consumption:	Stanby:
Raspberry Pi 5	2400 mA	1290 mA
Raspberry Pi Touchscreen 7”	200 mA	250 mA
HackRF One	500 mA	-

Table 3.3: Power consumption.

Equation 3.1 also takes into account that we use two HakRF Ones. The total consumption comes to be 3600 mAh. We can now use this number together with the capacity of the desired powerbank (20000 mAh, 15.7.1)

$$Total\ Consumption = 2400\ mAh + 200\ mAh + (2 \times 500)\ mAh = 3600\ mAh, \tag{3.1}$$

Battery life can then be found by equation 3.2:

$$Battery\ Life\ (in\ h) = \frac{battery\ capacity\ (mAh)}{load\ current\ (mA)}, \tag{3.2}$$

$$Battery\ Life\ (in\ h) = \frac{24000\ mAh}{3600\ mA} = 6.66\ h^*, \tag{3.3}$$

**The calculation in equation 3.3 is based on completely draining the powerbank. The Anker 737 can be drained to 0 % battery, but it is not recommended as it reduces the life cycle of the powerbank. The battery should be kept in the 20-80 % range for the best life expectancy.*

As we can see from equation 3.3, we can run our system for 6.66 hours. This is with the Raspberry Pi 5 and the HackRFs running "under load", and a powerbank with 20000 mAh capacity. In standby, the running time becomes significantly longer with over 15 hours (15.58 hours).

There are no requirements regarding running time for our system, neither from our stakeholder, KDA. However, we decided to go for a powerbank with 24000 mAh to get as much running time as possible. This makes the user able to use the system for quite some time before needing to recharge.

15.7.1 Powerbank

ABS | *SN*

To power our system we decided to go with a powerbank. The powerbank will be connected to the Raspberry Pi, and then distributed through the Pi to the other components. Since the other components will be connected through the Pi, the main requirements for the powerbank will come from the Pi. The Raspberry Pi 5 requires a 5V/5A input. This translates to an output of at least 25 W from the powerbank. However, the recommended power delivery is 27 W. This limits our choices to only a few, since most powerbanks either only offer 5V/3A (15 W) outputs, or don’t feature enough USB ports. We could still find a few options that deliver

Figure 3.17: Our Powerbank of choice (screenshot from Elkjop's online store).

Name:	Capacity:	Power Delivery:	Weight:	Price:	No. of Ports:
Clas Ohlson PB	20000 mAh	27 W	390 g	699,-	1xUSB-A, 1xUSB-C
Samsung EB-P3400	10000 mAh	25 W	210 g	449,-	2xUSB-C
Baseus Bipow	20000 mAh	25 W	425 g	890,-	2xUSB-A, 1xUSB-C
Anker 737 GaN Prime	24000 mAh	up to 140 W	630 g	1790,-	1xUSB-A, 2xUSB-C

Table 3.4: Powerbank options.

enough power, which can be seen in table 3.4.

The choice of powerbank fell on the "Anker 737 GaN Prime". The reasoning behind this is the capacity, power delivery, and availability. The powerbank can deliver up to 140 W of power, well above the desired 27 W, and has a capacity of 24000 mAh. This powerbank weighs 3 times as much as the Samsung EB-P3400, but we chose power delivery over weight in this case. As mentioned in section 15.7, we have no requirements regarding the running time of our system, but we still wanted to go for at least a 20000 mAh powerbank to ensure enough capacity to run our system for a good amount of time. The Anker 737 features the ability to both charge and deliver power to other units via its two USB-C ports. This gives it an advantage over for example the Clas Ohlson powerbank, which only has one USB-C port and is unable to deliver enough power via the USB-A port (restricted to 15 W).

Other features of the Anker 737 GaN Prime:

- "Smart temperature monitoring system".
- Screen with relevant information, such as battery percentage and power draw.

15.8 Attenuator

ABS | *AM*

As an addition to our system, we have looked into attenuators. An attenuator is used to decrease the transmitting power of RF transmitters. As mentioned in section 5.1, we want the J/S-ratio to be as high as possible. However, in some cases we want to decrease our jammer's

signal strength (J). In our case, we will use the attenuator in regards with testing outside. Since the rules on jamming and spoofing are quite strict, and the calculations on our transmitting power (appendix P) suggest that we should lower the power when testing in an open environment, we will use the attenuator to reduce the amount of disturbance on other systems reliant on GPS. There is also a requirement from NKOM to keep the transmit power to an absolute minimum (appendix O), with special precautions concerning local emergency services.

As mentioned earlier, the attenuator will be used to decrease the transmitting power of our jammer/spoofer. The attenuator should be able to lower the transmit power by approximately 20-30 dB. There are several options open to the purchase of attenuators in this range. However, these are often a "single-stage" attenuator, and the 30dB ones are quite expensive. There is also an opportunity to buy a cheaper variant, but these are often lower quality and have a long delivery time. Therefore, we decided to look into making our own attenuator. We could then make a low-cost variant with "multi-stage" attenuation. Our research on the topic suggested that making a passive attenuator was a possibility. We first had to decide what kind of attenuation we wanted, then calculate resistor values and put them in the correct configuration. The process of doing this is shown in sections 15.8.1 and 15.8.2.

15.8.1 T-pad Attenuator

ABS | *AM*

Figure 3.18 shows the configuration we will use to make the attenuator. This is made out of

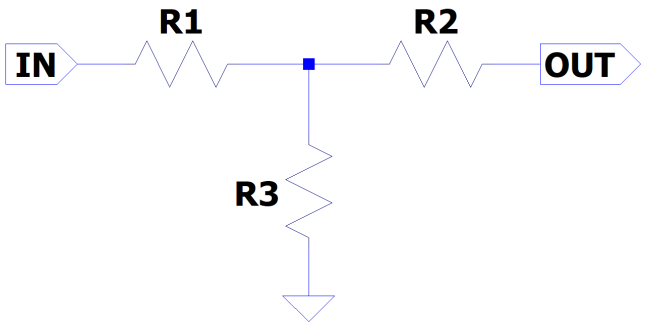


Figure 3.18: T-pad configuration.

three resistors in the the shape of a T, hence the name "T-pad". There are other types of configurations that makes an attenuator, such as the PI-pad and the L-pad. However, the T-pad is a good choice when working with equal impedance on input and output, and is often used with high frequency.

The T-pad attenuator consists of three resistors, R_1, R_2, and R_3. R_1 and R_2 are usually of equal values and form a voltage divider that has the function of ensuring the right amount of attenuation. R_3 is there to help terminate the output signal and minimize signal reflections, as well as ensure the right attenuation. In the next section, we will take a look at calculating resistor values for two specific attenuation levels, namely 10 dB and 20 dB.

15.8.2 Calculations

ABS | *AM*

To begin the calculations of the resistor values, one first have to calculate the N-factor (also referred to as K-factor [69]). "The K-factor or value is the ratio of the voltage, current or power corresponding to a given value of attenuation." [69]. This means that the N-factor will vary depending on what level of attenuation that is desired. Onwards, calculating R_1, R_2 and R_3 is then dependent on this factor, as shown in equations 3.4, 3.5 and 3.6. To ensure accurate resistor values for our system, the characteristic impedance, Z_0, also has to be accounted for. In our case, Z_0 equals 50 Ω, as the rest of the components in our system have a specified impedance of 50 Ω.

Calculations of N-factor for X dB attenuation:

$$N = 10^{\frac{XdB}{20}}, \tag{3.4}$$

Calculations on resistor-values for X dB attenuation, with respect to Z_0 and the N-factor:

$$R_1 = R_2 = Z_0 \times \left(\frac{N-1}{N+1}\right) \ [\Omega], \tag{3.5}$$

$$R_3 = 2 \times Z_0 \times \left(\frac{N}{N^2-1}\right) \ [\Omega], \tag{3.6}$$

Using equations 3.5 and 3.6 for a 10 dB and 20 dB yields the results found in table 3.5. With the values in the table put in the T-pad configuration, the attenuation will be at the intended levels.

XdB	N	R_1	R_2	R_3
10 dB	3.16	25.96 Ω	25.96 Ω	35.16 Ω
20 dB	10	40.90 Ω	40.90 Ω	10.10 Ω

Table 3.5: N-factor and resistor values for 10dB and 20dB attenuation.

15.8.3 Simulations

ABS | *AM*

In addition to calculations, we can use a simulation tool called LTspice. In LTspice we are able to create analog circuits and simulate their behaviour. We did the same for the attenuator, where we set up the simulation with our calculated values, and ran the simulation with an appropriate input source.

In figure 3.19, one can see how the configuration of the T-pad attenuator is set up. In this schematic, there has been added a 50 Ω resistor on both the input and the output. This is to get the correct input- and output impedance for the simulation. Figure 3.20 shows a simulation of two variants of the attenuator with different values on the resistors in the T-pad as in figure 3.19. The blue line is with the resistor values calculated for a 10 dB attenuation, and the red line shows the simulation where the resistor values have been fine-tuned to get the correct level

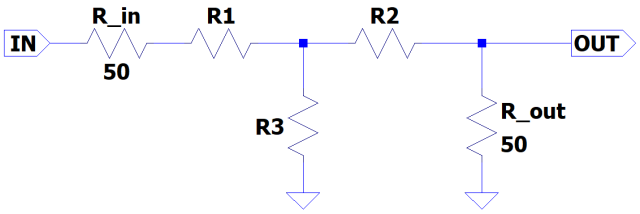


Figure 3.19: T-pad configuration with impedance match.

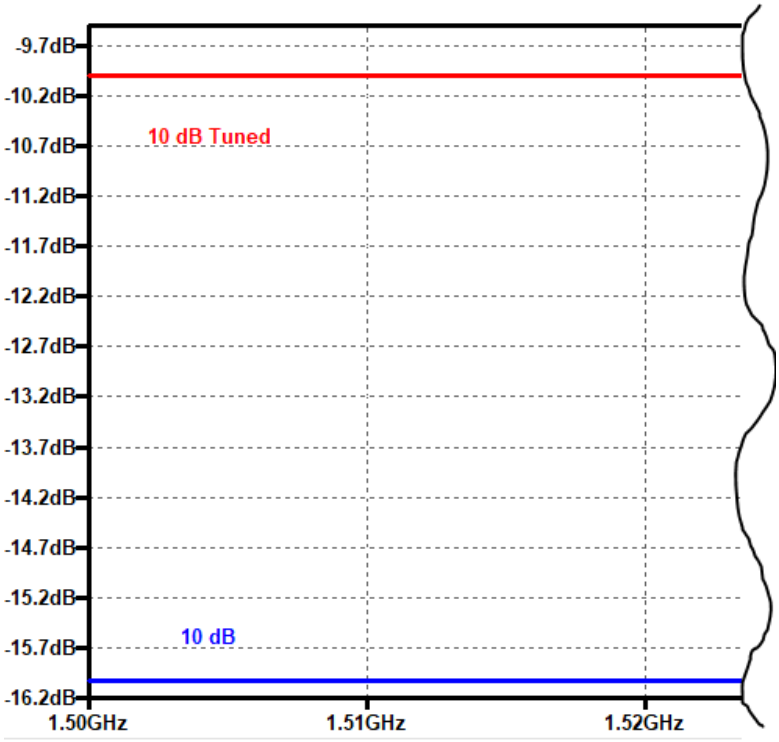


Figure 3.20: Tuned vs. untuned resistor values.

of attenuation. The values used in the tuned version is $R_1 = R_2 = 11.24\ \Omega$ and $R_3 = 105.59\ \Omega$. After fine-tuning, the attenuation comes out to be -9.99 dB (red) instead of -16.03 dB (blue). In our case, it is not highly important to get exactly 10- and 20 dB. However, we would like to get as close to the standard level as possible, as attenuators usually come in levels of 3 dB, 6 dB, 10 dB, 20 dB, and so on.

The resistor values from both table 3.5 and the tuned ones are not standard values for resistors. Therefore one has to find standard values and place them in parallel with each other to get the correct values. Using the following formula (3.7), one can find standard resistors to put in the T-pad [70]:

Find closest standard resistor:

$$R_{parallel} = \left(\frac{1}{R_{P1}} + \frac{1}{R_{P2}}\right)^{-1}\ [\Omega], \tag{3.7}$$

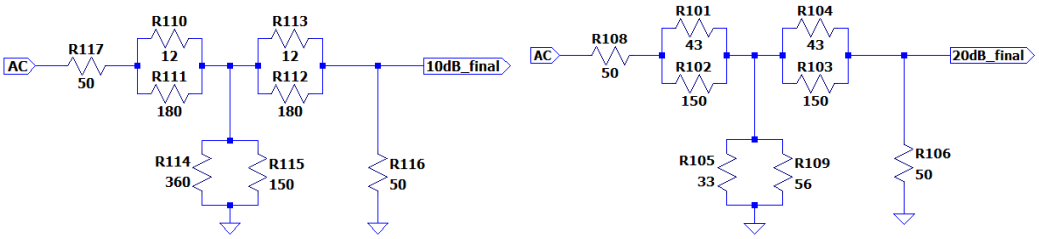


Figure 3.21: a) 10 dB attenuator. b) 20 dB attenuator.

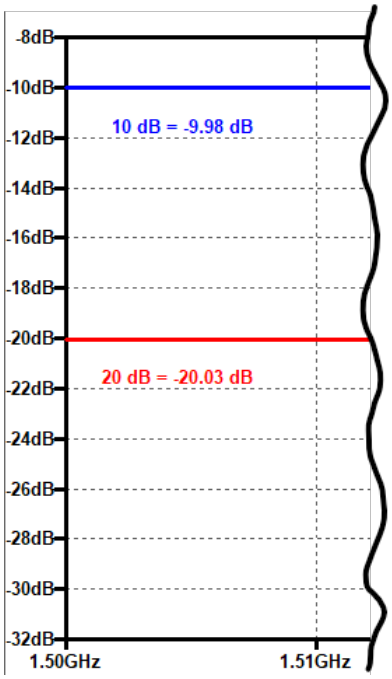


Figure 3.22: Simulation results with parallel connections.

Example from the 10 dB-stage, where the targeted values are 11.24 Ω and 105.59 Ω:

$$R_{parallel} = \left(\frac{1}{12} + \frac{1}{180}\right)^{-1} = 11.25\ \Omega, \tag{3.8}$$

$$R_{parallel} = \left(\frac{1}{360} + \frac{1}{150}\right)^{-1} = 105.88\ \Omega, \tag{3.9}$$

From equations 3.8 and 3.9, we find that the resistor values to get 11.24 Ωs can be 12 and 180 Ω i parallel with each other, and 360 and 150 Ω in parallel to get 105 Ωs. We can then put these values into the T-pad configuration as shown in figure 3.21. The simulation results from this configuration can be seen in figure 3.22. Here the red line represents the 10 dB stage and the blue line represents the 20 dB stage. Their attenuation levels come out to -9.98 dB and -20.034 dB respectively.

15.8.4 Multi-stage Attenuator

ABS | *AM*

As mentioned earlier in this section, we wanted to make a multi-stage attenuator. Until now, we have looked at 10- and 20 dB attenuators. With these two we can also make a third stage.

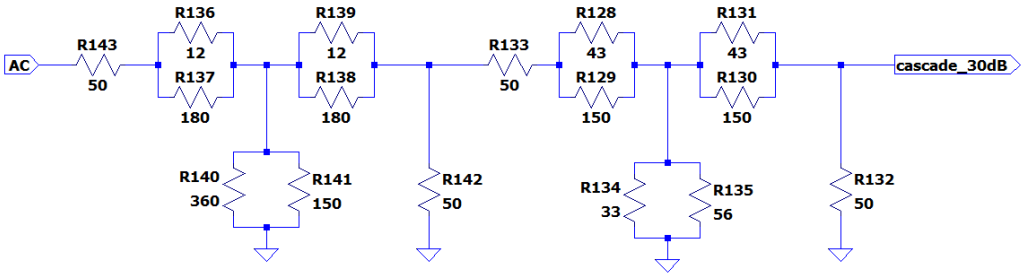


Figure 3.23: Cascade connection.

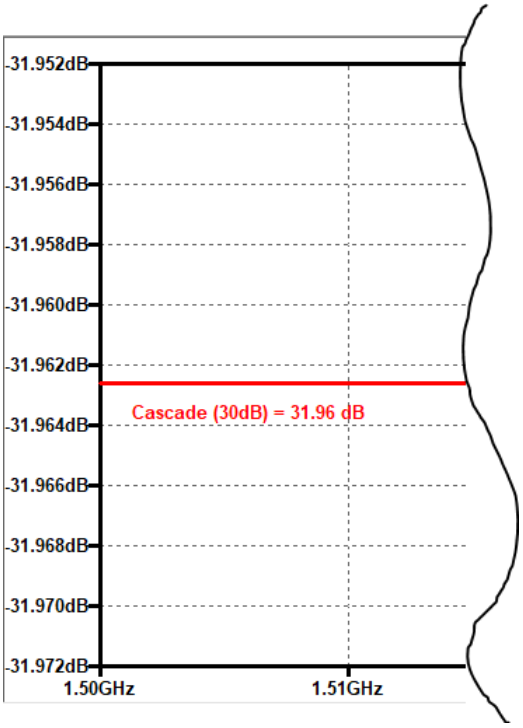


Figure 3.24: Cascade simulation result.

By simply connecting these two together we get a 30 dB attenuator. In figures 3.23 and 3.24, we can see the simulation setup and results for the two attenuators in a cascade connection. As one can see, the attenuation is now 30 dB (31.96 dB). Another reason for cascading the two attenuator stages instead of making a 30 dB attenuator, is that attenuators of higher stages (30 and up) tend to become unstable at high frequencies when configured in the T-pad-, L-pad- and PI-pad forms.

In reality, the connection will look a little different as all the 50 Ω resistors will be replaced by an SMA-connector. A more realistic configuration is shown in figure 3.25. With this configuration, we can reach both 10, 20, and 30 dB of attenuation.

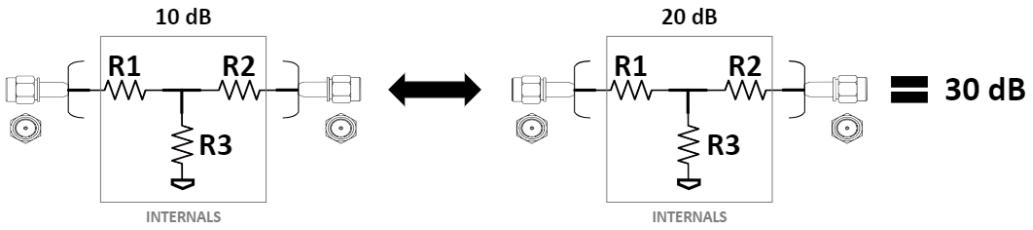


Figure 3.25: Visual representation of cascading 10 dB and 20 dB attenuators (requires cables) (Created by ABS).

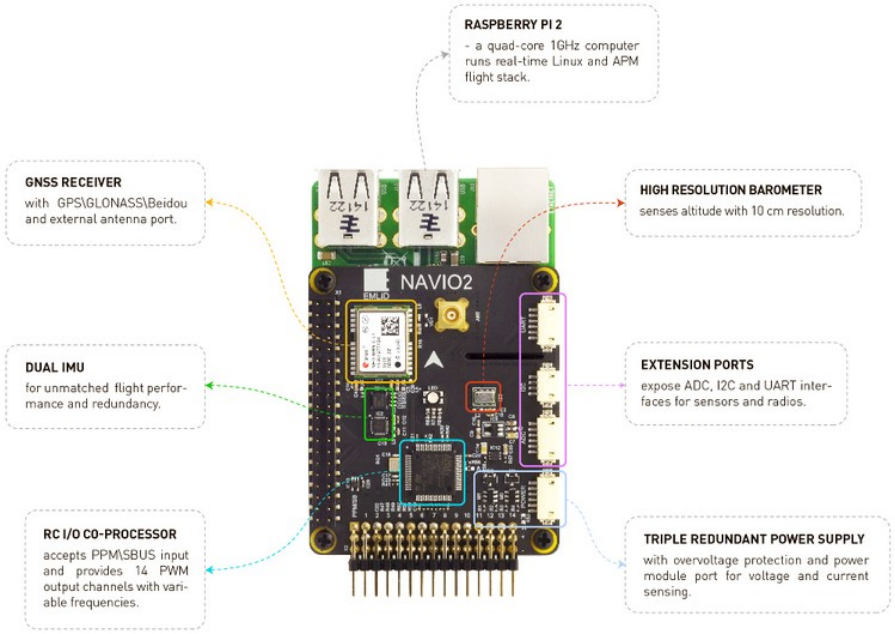


Figure 3.26: Navio 2 features.

15.9 Navio 2

ABS | *AM*

The Navio 2 controller is an autopilot HAT that together with a Raspberry Pi turns into a flight controller. Navio 2 uses several sensors and controllers on board to support every need a flight controller has. The sensor includes accelerometers, gyroscopes, and magnetometers. The autopilot HAT is also equipped with the NEO-M8N GNSS-chip which can be read about in section 15.10. Navio 2 is mainly designed to be used in drones, but can easily be used in other radio-controlled and autonomous vehicles like planes, cars, and motorbikes. In figure 3.26 some of Navio 2's features are shown.

15.10 NEO M8N

ABS | *AM*

As mentioned in section 15.9, the Navio 2 is equipped with a NEO-M8N GNSS chip from ublox. The NEO-M8N chip utilizes concurrent reception of up to three GNSS systems (GPS and Galileo together with BeiDou or GLONASS). The following is an overview of the NEO-M8's capabilities regarding each GNSS [71]:

- Global Positioning System (GPS, the American GNSS) - receive and track the L1 C/A signals broadcast at 1575.42MHz

- GLONASS (the Russian GNSS) - receive and track the L1OF signals GLONASS provides at 1602 MHz + k*562.5 kHz, where k is the satellites frequency channel number (k = -7,..., 5, 6).
- BeiDou (the Chinese GNSS) - receive and track the B1I signals broadcast at 1561.098MHz
- Galileio (the European GNSS) - receive and track the E1-B/C signals centered on the GPS L1 frequency band. Galileo reception is by default disabled, but can be enabled by sending a configuration message (UBX-CFG-GNSS) to the receiver. Galileo has been implemented according to ICD release 1.2 (November 2015) and verified with live signals from the Galileo in-orbit validation campaign.

In addition to these, NEO-M8N also supports SBAS satellites, which send correctional signals from geostationary satellites to improve accuracy and quality of the received GNSS signals. This provides greater positioning accuracy in scenarios like urban environments, where the signals can be weakened by surrounding buildings and clusters of trees.

Figure 3.27: The NEO M8N GNSS-chip [9].

15.10.1 Anti-jamming and Anti-spoofing Detection

ABS | *AM*

The NEO M8N has integrated jamming and spoofing detection. However, other than the integrated SAW-filter, there is no other means of stopping jamming or spoofing. The spoofing detection feature, a spoofing flag, is only usable for notifying the pilot of the drone that there is an attack currently happening. The chip will check for irregular patterns indicating that the unit is being spoofed and let the operator know by combining a number of checks on the received signal looking for inconsistencies [71].

15.10.2 SAW Filter

ABS | *AM*

The NEO M8N GNSS chip does not only feature jamming and spoofing detection. It also offers a way of trying to stop it. Incorporated onto the GNSS chip, there is a filter called a SAW-filter. SAW stands for Surface Acoustic Waves and is used together with a low noise amplifier (LNA) to enhance the signal received from the external antenna. As well as enhancing the

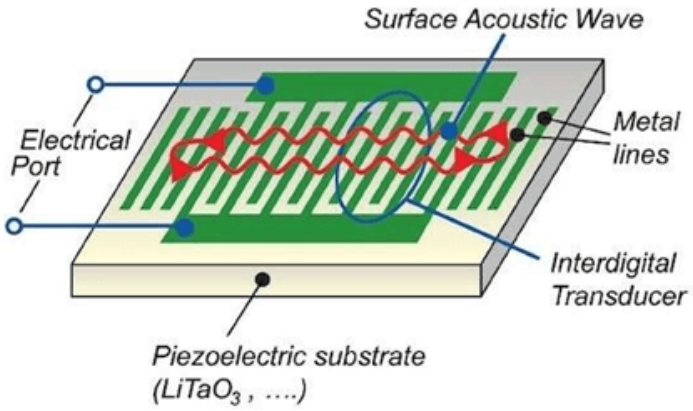


Figure 3.28: Building blocks of a SAW-filter [10].

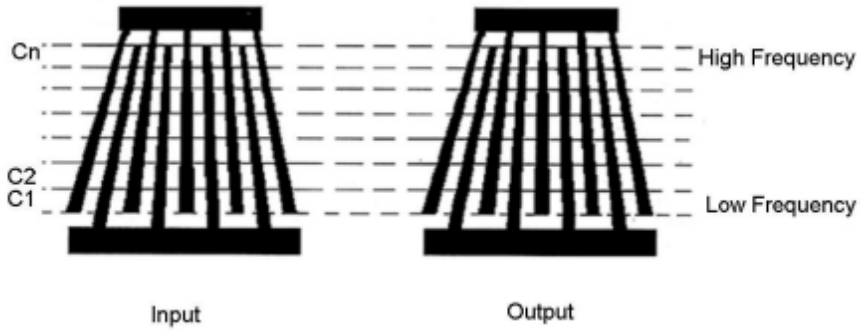


Figure 3.29: SAW-filter bus bars [11].

signal received from the antenna, the SAW-filter also works as an anti-jamming component. Functioning like a bandpass filter, it lets through signals of the desired frequency as well as attenuating unwanted frequencies. SAW-filters are used in most electrical devices, such as mobile phones, computers, etc.

A SAW-filter works by turning electrical signals into acoustic waves, sending the waves over the surface of a Piezoelectric material, then turning the acoustic waves back into an electrical signal [11]. This is done with the use of an input transducer and an output transducer. The transducers are composed of an inter-digital electrode connected to a bus bar as shown in figure 3.28. Each electrode will function either as an acoustic source or a detector. The transducers will exchange energy while the piezoelectric material will absorb radio frequency energy and function as a transportation medium, turning the acoustic waves back into electric energy.

The operating frequency of the SAW device will be determined by the wavelength of the electrode and its neighboring spaces as shown in figure 3.29. The amplitude and phase are decided by the electrodes length and position respectively [11]. After this process, the signal will then be sent to an LNA, and further processing will be done in the RF block of the GPS module.

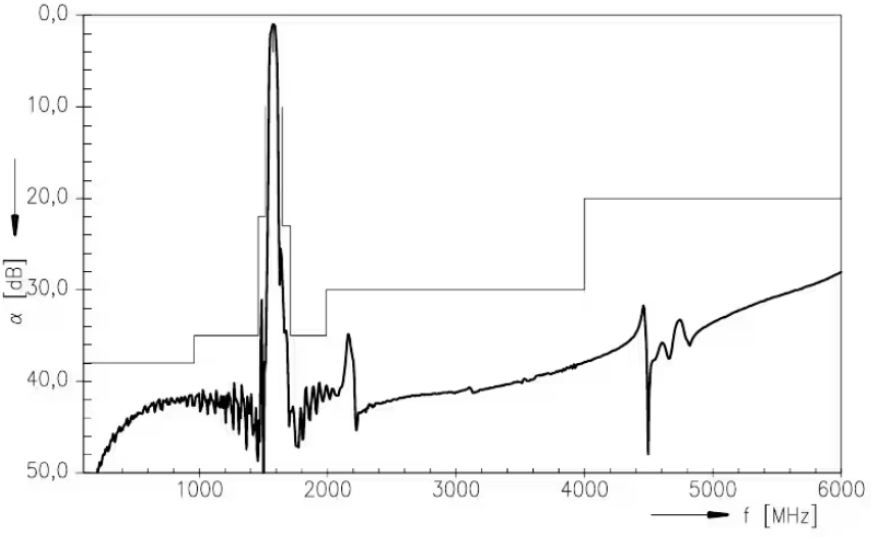


Figure 3.30: Frequency response (wide-band) of the EPCOS B7839 SAW-filter [12].

15.10.3 SAW Filter Disadvantages

ABS | *SN*

As NEO-M8N contains a SAW-filter, the jamming will be a little harder to accomplish. However, SAW-filters have a few disadvantages:

- **Temperature sensitivity:** Although SAW filters have countermeasures to battle the effects of temperature variations, their characteristics will still be affected.
- **Frequency drift:** As the chip ages, its performance will start to decay.
- **Tunability:** The chip has a static range, so there is no way to tune the frequency range.
- **Narrow passband:** In coherence with tunability. The passband of the chip is both static and quite narrow. This can be exploited in the form of being able to concentrate much energy in the frequency range of the passband.

  Typical bandwidth: 2 to 10 MHz for narrow-band and up to several hundred MHz for wide-band.
- **Working area:** The filter's working area is usually from 0 GHz to 3 GHz. However, at around 1.7-1.8 GHz (depending on the filter), and even more distinctly around 2.2 GHz, the filter's frequency response will start to decay, as shown in figure 3.30. This is due to the wavelength of the acoustic waves will get impractically short at such frequencies, meaning that the filter will struggle to handle them.

The above-mentioned disadvantages can all be exploited to some degree, depending on what jamming technique is being used. Both spot- and barrage-jamming can be suitable methods for exploiting the chips static and narrow passband. With spot-jamming one can concentrate all energy on a specific frequency, and even if the frequency is around the filters center frequency where it has the best working conditions, it might not be able to separate the counterfeit signal from the real one. The same applies to barrage-jamming where the energy is spread across

several frequencies. If the jammer can spread its energy across the frequencies in the passband of the filter, the filter might not be able to distinguish the counterfeit signals from the real ones. This means that the jamming signals will not be as powerful as with spot-jamming but can still be a successful method.

15.11 Emlid OS

SN | *ABS*

Emlid is both the name of the creator of the Navio 2 flight controller used in our project and the name of the operating system designed to run on this flight controller. Although our project does not require a functional drone, using this OS combined with Emlid's utilities, available on their GitHub repository https://github.com/emlid/Navio2, has been extremely useful.
For all testing of Delirium, we use the "U-blox SPI to TCP bridge utility" from this repository. This utility allows our Raspberry Pi to open a port and share data with a u-center application on another PC. However, this setup requires both the Raspberry Pi and the u-center PC to be on the same WiFi network.

15.12 u-center

SN | *ABS*

To be able to see the effect of our work, we have to be able to see what the drones see, and we can do that through the use of u-center. u-center is a program developed by u-blox, the same one that developed the NEO M8N GNSS chip. By connecting the flight controller and a Windows computer to the same wireless 1network u-center can connect to the flight controller and display all we require.
Through the use of u-center, our group can view positional data, each connected satellite and its signal strength, the drone's positional accuracy, and much more (see fig 3.31).
While being able to view data in u-center, we also have some options to control the GNSS chip of the flight controller, this could help us in our testing of our systems. For example, being able to cold-start the GNSS chip, i.e. deleting stored information about the estimate of its position, its speed, nearby satellites, almanac data, and ephemeris data. From u-center, we can choose from which GNSS-systems we want to receive data, see fig 3.32.

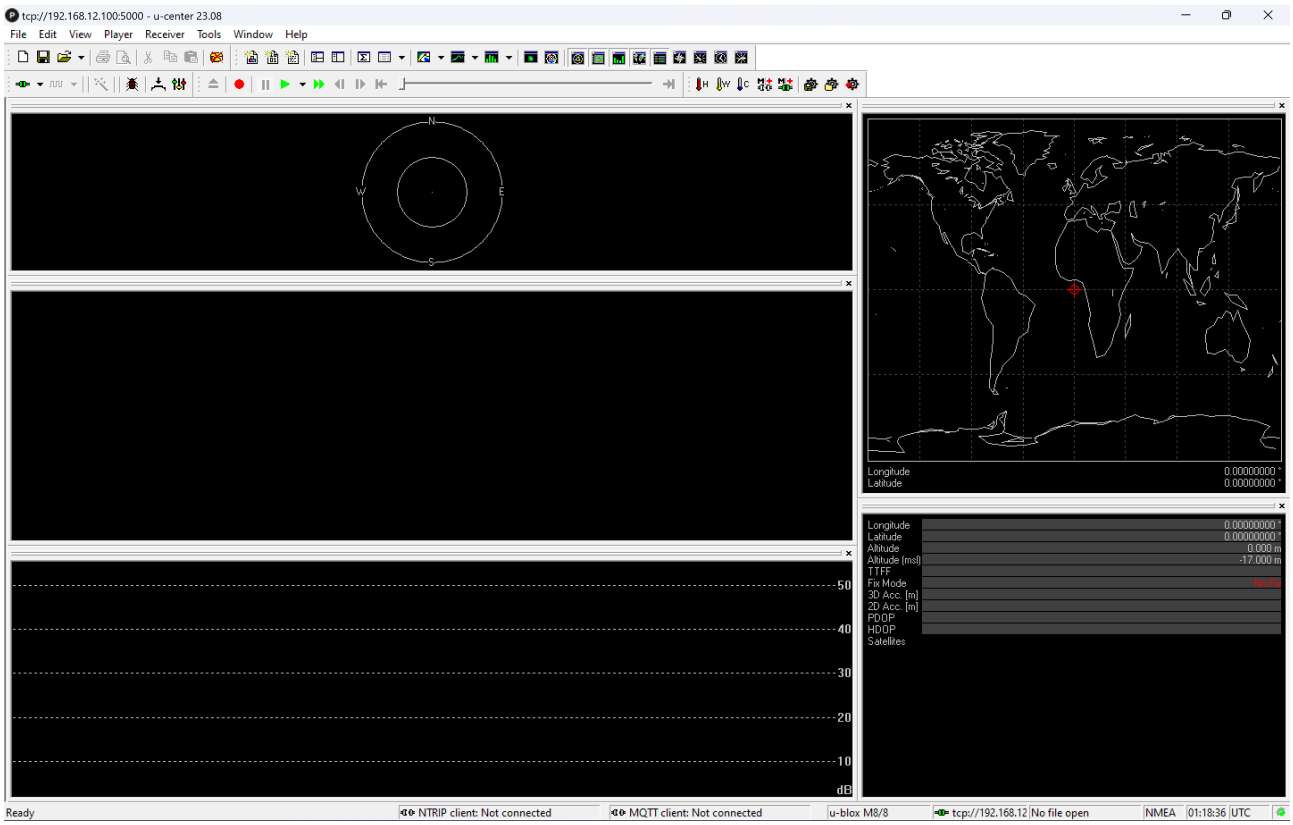


Figure 3.31: Screenshot from u-center.

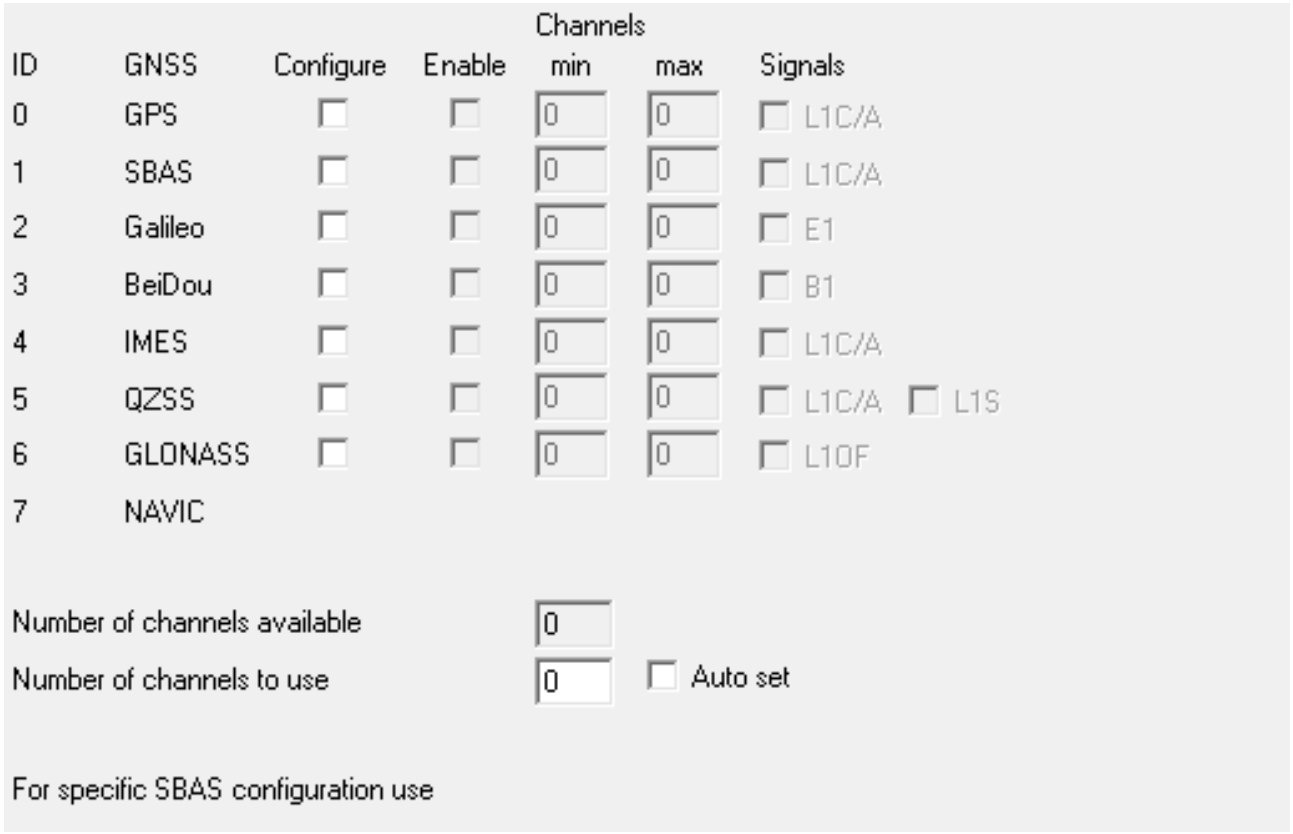


Figure 3.32: Screenshot from u-center. Choosing GNSS systems in the settings menu.

16 Software Development

AM | *SN*

This section covers how the software for Delirium was developed and which components and tools were used. The high-level software architecture for Delirium can be seen in appendix M, and we recommend the reader to inspect this diagram if doubts occur.

16.1 Modular Approach

AM | *SN*

We chose a modular approach for developing the software of Delirium. Modular software development is a software design technique that involves partitioning a program into separate modules which in turn contain specific functionality of the system [72]. The purpose of using the modular approach was to achieve a clear division of labor and avoid cumbersome version control collisions. The modular approach further fulfills a request from our customer regarding the possibility of expanding Delirium in the future. Such an expansion is much simpler if functionalities are relatively isolated, and thus can be extracted from the source code without changing much to achieve compatibility.

We established 3 main bodies of work that needed to be done in software; the development of the SDR's in GNU radio Companion (which generates Python source code files), a user-friendly GUI and an API (Application Programming Interface) connecting the two in a seamless way. The API solution was chosen because not only did we need to divide the labor intensity between members, but we also needed to divide the research amount fairly, as neither of us had any experience with the equipment we had chosen.

16.2 Programming Languages Used

AM | *SN*

Delirium was developed mainly in Python 3.10, which was released on October 4, 2021 [73]. An extension module for the Delirium API was written in C++. As we already knew it well, both Python and C++ development were performed in Microsoft's Visual Studio Code (VS Code). It was a requirement for us to use Python as we needed to properly interface with the generated GNU Radio flowgraphs (see section 15.1). Both GNU Radio and the HackRF software recommend using Ubuntu, which comes with a default installation of Python 3.10, and as no other dependencies dictated otherwise, we chose to stick with this [59] [74].

The need to use C++ came from the libhackrf (see section 15.2.1) library, which is implemented in the C-language, which is largely compatible with C++ (see section 9 for more on this). Therefore, C++ was chosen over C, as we have been taught C++ in several courses at USN.

16.3 Implementation of CI & VCS

AM | *SN*

Based on past experience, availability of help, cost, and known industry norms/anecdotes, we chose to use the popular DVCS Git and the web-based GitHub to keep our code repository. The

repository was kept in private mode due to the project's possible non-legal nature (depending on how it's used). To enable CI, we used Git in the way described by Martin Fowler in his article on the topic (see full explanation in appendix: K). This meant that we had to establish a service that builds the project (including all dependencies), performs critical unit tests, and alerts us if something went wrong in this process on the main repository on GitHub.

GitHub Actions was chosen to implement this feature. A workflow based on the "Python Application"-template was made, which consists of a .yml file containing instructions to build the entire project on a GitHub-hosted Ubuntu Virtual Machine (VM). The .yml-file ensures that the VM builds the project and all dependencies, checks the Python code for obvious syntactical errors and runs the specified unit tests with the Python module "pytest". After all, builds and tests are done, GitHub sends an email to the commit-er if at any stage the build/test failed.

16.4 Development of the Delirium API

AM | *SN*

The purpose of the Delirium API (Application Programming Interface) is to serve as an intermediary between the Delirium GUI and the Gnu Radio SDR's plus gps-sdr-sim (see section 15.3), as can be seen in figure 3.33. It does this by providing control classes and functions which, when instantiated/called, enable the starting/stopping/manipulating of the underlying SDR's. The development of any functionality in the API had its roots in the system requirements. After formulating technical requirements from the top-level requirements and design choices, tentative diagrams were made both for our own sake in terms of keeping oversight, but also for documentation purposes. A Unified Modelling Language (UML) class diagram of the Delirium API can be seen in appendix 10.

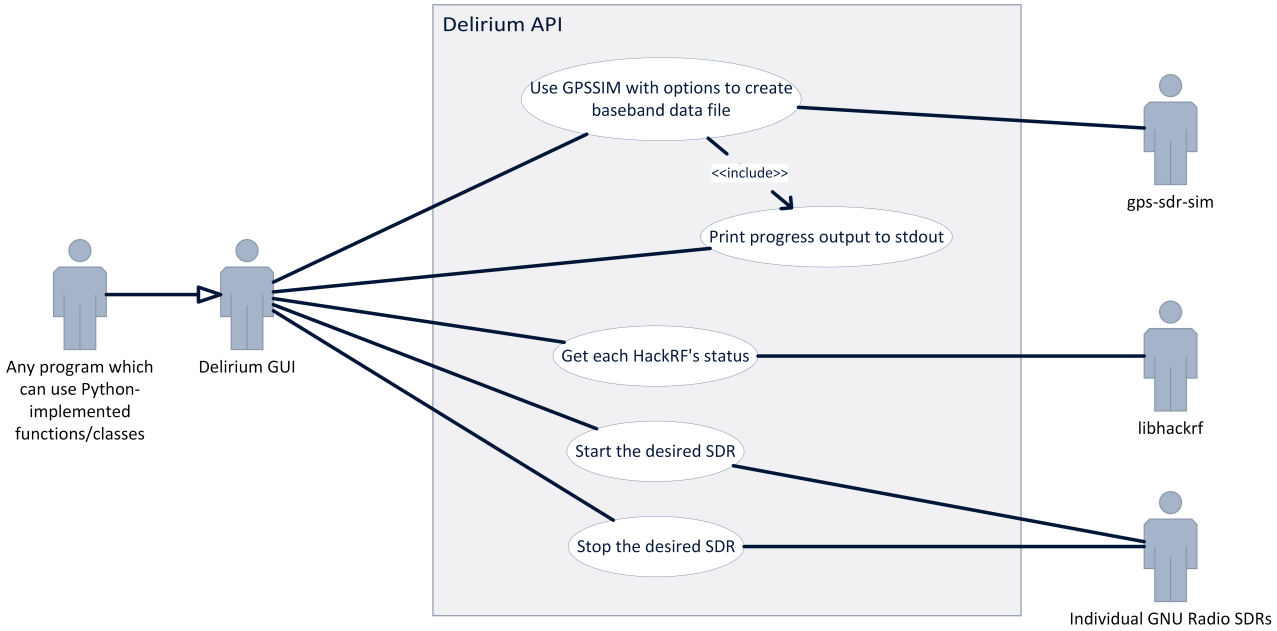


Figure 3.33: Use cases for the Delirium API, created by AM. Legend can be seen in figure N.1.

In the early planning phase, only one class was proposed, a controlling class for each GNU

Radio SDR. After some testing, this proved to be difficult, as the spoofing radio needed special treatment for both the collection of the user's options and for the generation of the GPS baseband data. There also emerged an issue with our implementation of sweep-jamming (see section 5.1.3) where the actual sweeping did not work in spite of it being proven to work in an isolated test case. After an exhaustive test (see appendix 7), the cause was speculated to be a result of Python's Global Interpreter Lock (GIL, see section 16.2) not allowing the actual parallel execution of threads. As a remedy for these issues, functions were made which could then be called using the "multiprocessing.Process" Python package. The multiprocessing package supports the spawning of processes, with a similar API to the threading library found in e.g. C++. It effectively side-steps the GIL by creating subprocesses instead of threads. The multiprocessing module, in this way, allows the programmer to fully leverage multiple processors on any machine, as these subprocesses can be scheduled in parallel with the main process [75].

16.4.1 Extending the API with C++

AM | *SN*

After milestone 4 was deemed a success (see appendix C), the decision was made to further expand Delirium with the ability to - at runtime - display the status of each HackRF device. In this way, the user would be able to see both if any of the devices were running or not, and whether an error existed at runtime. Functions for collecting such information already existed within the libhackrf software (see section 15.2.1), so for us to utilize these functions, a C/C++ implementation had to be made which could communicate with the C language implemented libhackrf and then pass the collected information through the Delirium API to the Delirium GUI. For the Delirium-side implementation, the C++ language was chosen to implement this, due to experiences the team members had gained with this language in previous courses.

A C++ file with a respective header file ("module.cpp" and "module.h") was made, which called upon the necessary libhackrf functions to determine the HackRF's status. The pybind11 library was also called upon within this file to bind the C++ function to CPython. The source code file structure is illustrated in figure 3.34. Linking the libraries and compilation was then made possible using a CMakeLists file, which linked the libhackrf's own CMakeLists file and the pybind11 library to the resultant target. The pybind11 library's CMake function "pybind11_add_module()" was then used in this top-level CMakeLists file to create the target. The resultant CMake build produced a shared object file, which made the extension callable from the Delirium API. The CMake build process is illustrated in figure 3.35. The behavior of the solution is illustrated in the UML sequence diagram in appendix 12.4.

16.4.2 Unit Testing

AM | *SN*

Unit testing was used extensively during the development of the Delirium API. Unit testing involves testing software code at its smallest functional entities, which is typically a single class, or perhaps a single function. By testing every such unit individually, most of the errors that

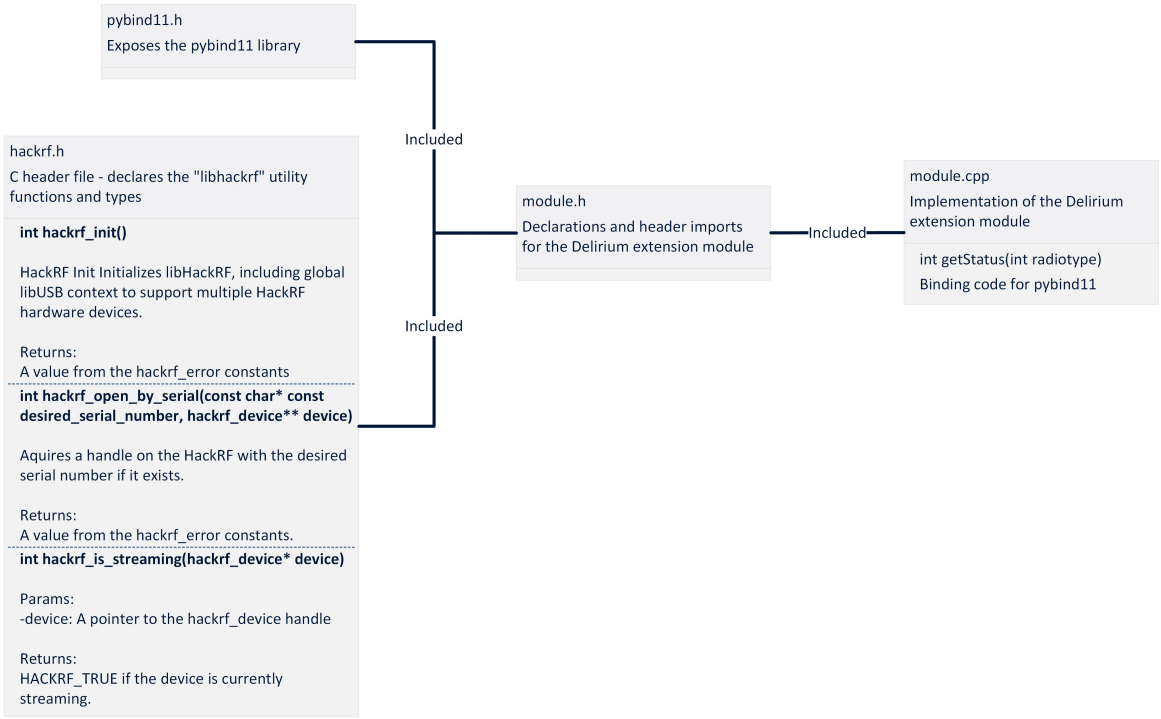


Figure 3.34: An overview of the file structure needed to implement the Delirium C++ extension, created by AM.

Figure 3.35: An overview of the CMake build process for the Delirium C++ extension module, created by AM.

might be introduced into the code throughout a project can be detected early and prevented from propagating to other parts of the project [RefH15]. The Delirium API unit tests consisted of isolated functions (contained in their own file) which instantiated the API's classes and called its functions to enable verification of the system requirements on an oscilloscope. An oscilloscope had to be used during these tests, as it was the only way to observe the radio frequency output of Delirium.

16.5 Creating the Flowgraphs in GNU Radio

HK | *ABS*

When designing the flow graphs for the jamming part of our system we had to ensure that GNU Radio Companion could communicate with our peripheral. There are two opportunities to make this happen, the osmocom-block, and the Soapy-sink. Sinks in this context is a virtual sink to pour the signal into, and then the peripheral transmits it using the settings in the sink-block. A graphical view of the blocks can be seen in fig.3.36.

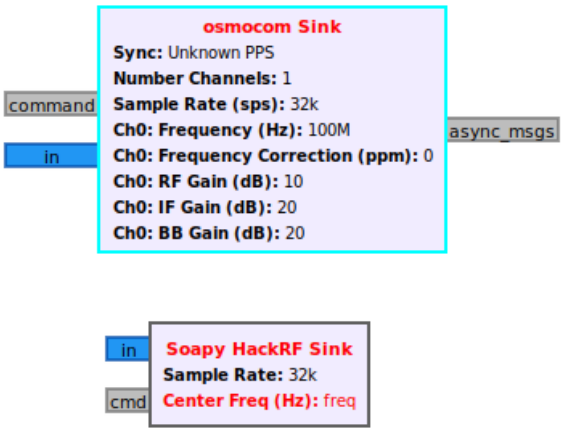


Figure 3.36: Osmocom and Soapy blocks.

Then the next task was to generate the signals to send into the sink. This process is different for each jamming script, but there are several variables that are similar or equal for each script, therefore we used variable blocks to simplify the building process for each script. The variable blocks we used are represented in fig. 3.37.

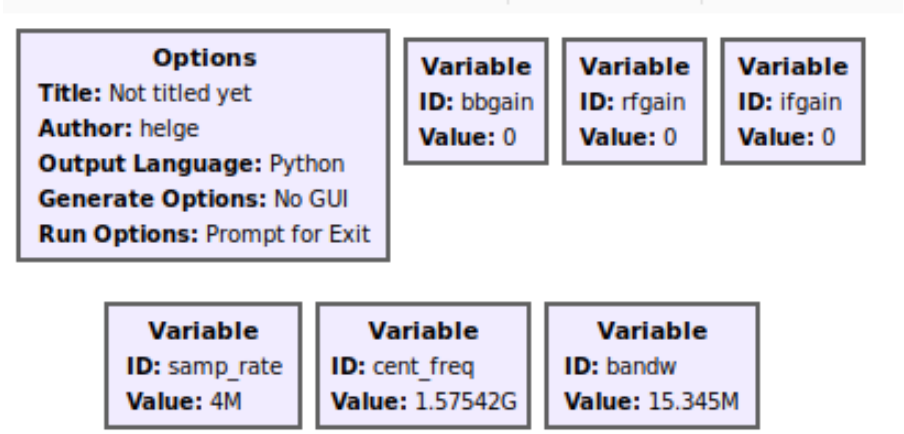


Figure 3.37: Variables we used for protocol-aware jamming.

In these variable blocks, there are two options; "ID" and "Value". The "ID" is the name of the variable and can be called from other blocks such as the sink-blocks. For example, the "cent_freq"-block is defined as being the same frequency as the carrier frequency of the GPS L1 signal. This block has a value corresponding to that frequency. We can then retrieve this information in other blocks that need this setting. When we then need to change this setting, we only need to change the variable, instead of changing the settings in every block that calls upon it.

16.5.1 Barrage Jamming

HK | *ABS*

Our first radio was the barrage jammer. This jammer uses a simple flowgraph (as shown in fig. 3.38)

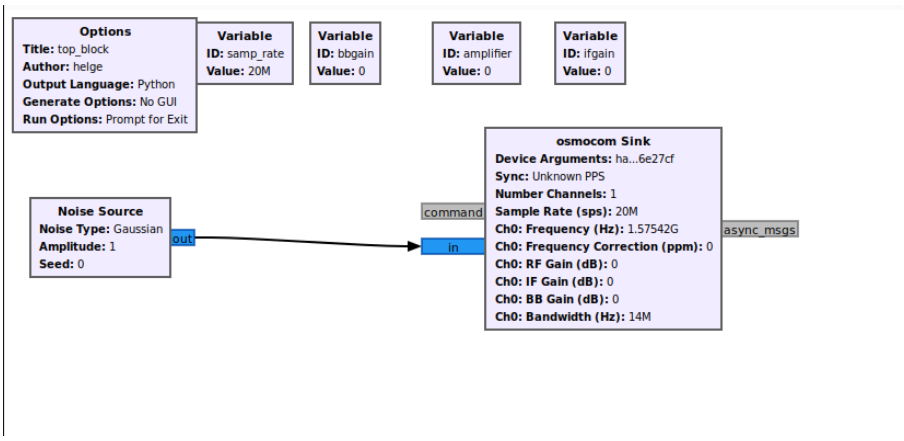


Figure 3.38: Barrage jamming flowgraph

The goal of the barrage jammer was to engulf the entire bandwidth of the GPS signal, therefore the bandwidth is set to 15.3 MHz [76]. This is set using the variable "samp_rate", and the value is set to the same as bandwidth.

16.5.2 Spot Jamming

HK | *ABS*

The next flow graph we designed were the spot jamming. The idea of spot jamming is explained in 5.1.2. One limitation we met with this approach is the bandwidth of the HackRF One, which only goes down to 1 MHz.

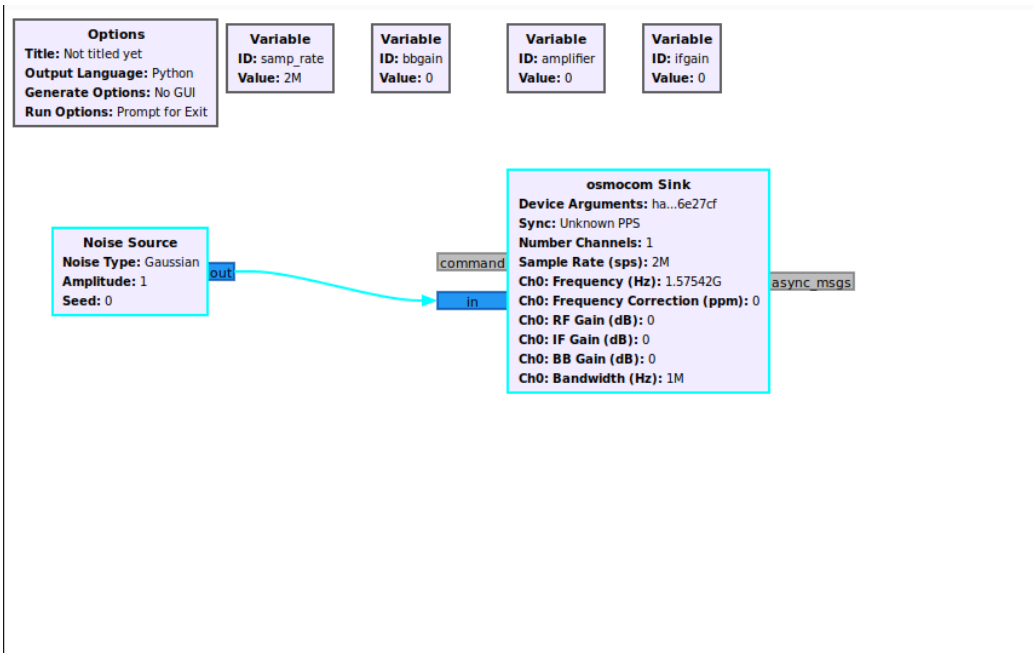


Figure 3.39: Spot jamming flow graph

16.5.3 Sweep Jamming

HK | *ABS*

Designing the sweep jammer we had three different approaches. The first approach involved the saw tooth signal source, but this did not have acceptable results when testing on the signal analyzer. The range of the sweeper was only within the bandwidth of the HackRF One, which is smaller than the range we need in order to sweep the other GNSS signals.

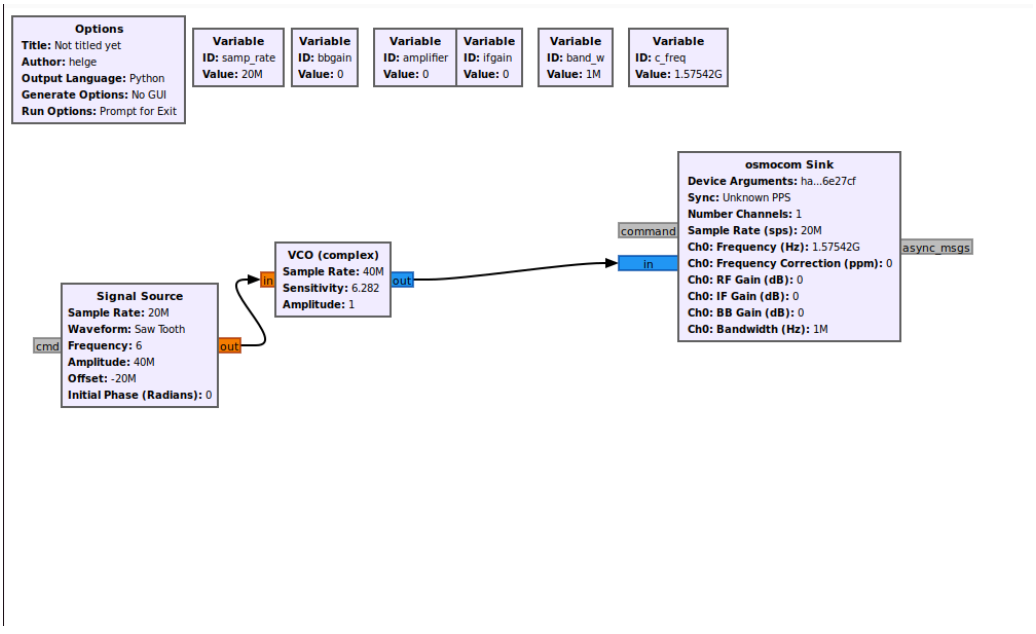


Figure 3.40: Sweep jamming flowgraph

16.5.4 Protocol Aware Jamming

HK | *ABS*

Protocol-aware jamming was the most sophisticated jamming solution we implemented in the final system. It takes into account the digital modulation of the GPS signal and simulates a similar modulation on an array of random binary bits. The first iteration is also the one that got implemented (see fig. 3.41). Here the "Random Source"-block generates a random set of zeroes and ones (binary digits), these are in char-format, so they need to be transformed into floats in order to be processed further. This happens in the aptly named "UChar To Float"-block. From here the bits are multiplied by two (zero remains zero and ones become twos) before they are subtracted one. This makes every zero a negative one, and every two a one. Doing this gives us a set of negative and positive ones, exactly what we need to have a complex signal. The set is still in float format, however, and we need them to be in a complex format to feed them into our SDR peripheral. This is done with the "Float To Complex"-block. Here we input the set of floats and inject a constant source of zeroes to the imaginary part of the complex number. Then we had the correct format and the correct set to transmit to the HackRF One.

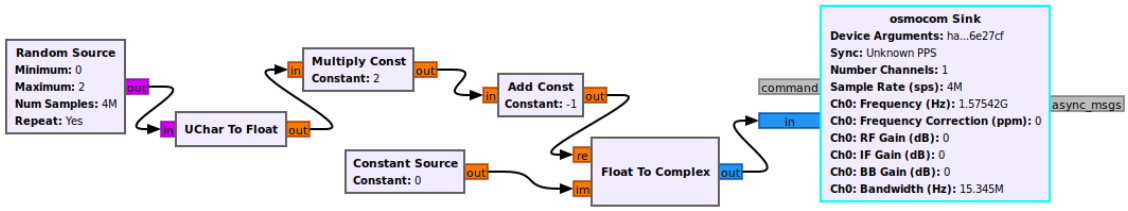


Figure 3.41: Protocol aware jamming flowgraph

To get the full QPSK signal, we iterated on this to create a more accurate modulation, but through testing, this proved to have the same efficiency (emitted power measured on the signal analyzer) and demanded more storage. See fig. 3.42. The modulation of the signal had to be implemented in such a fashion that the phase-shifted component had to have a sample rate of 1/10th the non-phase-shifted signal. We accomplished this by reducing the sample size of the source feeding into the adder.

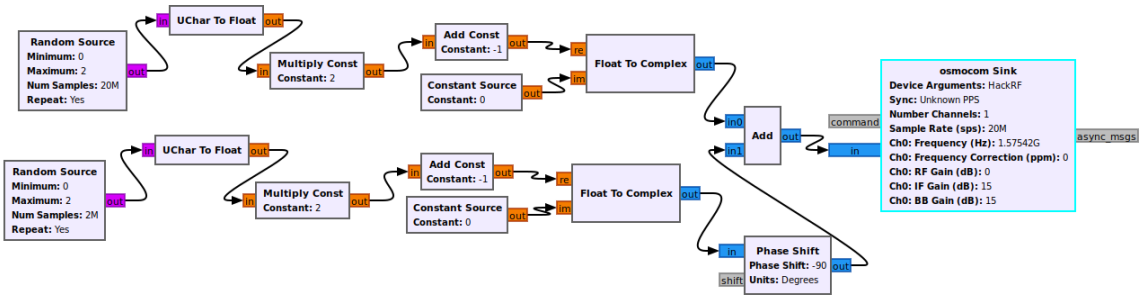


Figure 3.42: QPSK jamming flowgraph

Our final iteration on this jammer was to use an embedded Python block to make the SDR peripheral sweep the other GNSS bands with this jammer. This proved to be ineffective as the embedded Python block only was able to adjust the signal within the bandwidth of the HackRF One, which is not wide enough. See fig. 3.43 for flowgraph. The signal going into the osmocom sink is the same as for the QPSK-modulated signal, but it first passes through a frequency translating finite impulse response filter, where the center frequency (within the bandwidth) gets determined by an embedded python block, which has a code that takes an array with the center frequency of the Galileo center frequency and the GLONASS center frequency and switches between these two frequencies with a polling rate of 1000Hz, the maximum transmission rate of the UART-connection between the PC and the HackRF One.

To remedy this we replaced the embedded Python block with an option in our API that enables the same functionality but acts on the variable instead.

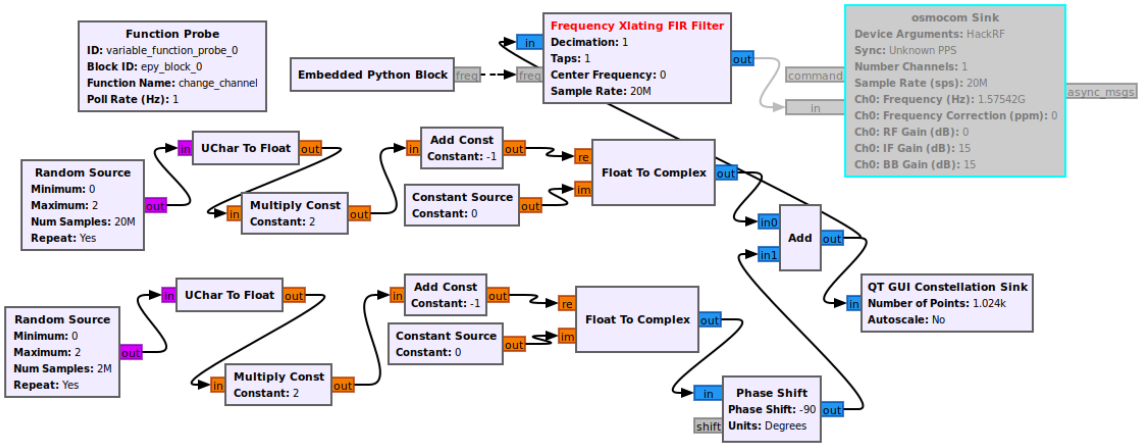


Figure 3.43: Protocol jamming with sweep functionality

16.6 Development of Delirium GUI

SN | *AM*

The Delirium Graphical User Interface (GUI) is a pivotal part of our project, created to ease the use of the underlying system as much as possible without limiting the control the user has. Developed using Python's tkinter library, the Delirium GUI provides a way to use our prebuilt jamming solutions and an integrated spoofing utility using osqzss's Github repo, gps-sdr-sim, to generate the radiowaves to be transmitted by the hackrf. The GUI can be seen in fig 3.44.

The GUI in its entirety is contained in the delirium.py file, within this file, there is also all logic used to start, stop, control, and keep track of the systems SDRs using the Delirium API. Delirium.py is built in the following structure. Firstly, importing all necessary libraries, tkinter, ttkthemes, and themedtk to build the GUI and its looks, os, sys and multiprocess, and deliriumAPI. With deliriumAPI being our library built in-house.

Due to the continuous integration approach adopted in our software development process, the functions and methods have evolved throughout the project. As a result, some functions and methods have continued to expand in size. At this time, a few days before this document is finalized, the main function to control starting and stopping jamming is around 70 lines long. For future work, it is recommended to refactor this function to enhance readability and maintainability. Future work for the GUI will be discussed more in-depth later in this document, see section 19.2.

16.6.1 Using tkinter Library to Develop the GUI

SN | *AM*

The GUI was built using tkinter, a popular standard Python library for creating graphical user interfaces. Tkinter was chosen for its simplicity and effectiveness in rapidly developing functional interfaces.

16.6.2 Structure of the GUI

SN | *AM*

The GUI is split into different sections using tkinter frames:

- The jamming frame is placed in the top left. It includes a drop-down menu to select the type of jamming, a start/stop button, and a gain button used to toggle an increase in signal strength.
- In the top right is the spoofing frame, which only consists of two buttons, a button called Generate and a button called Control. These buttons can dynamically place another frame inside the spoofing frame to display the respective usage of spoofing.
- The generate frame, see fig 3.45, gives the user the ability to create the spoofing Bin file with options the user chooses. These options are choosing an ephemeris file, downloaded either to the raspberry itself or a USB plugged-in with the file, choosing a location to be spoofed, the time that is transmitted in the spoofed signal, should be left empty to let the API choose the current time as this usually gives the best results, and the name for the bin file.
- The control frame (see fig 3.46) gives the user the ability to start and stop the spoofing. In this frame, the user is presented with a button to choose the file to be used for spoofing, and a button to start and stop.

On the bottom part of the GUI is an empty textbox. A feature of Delirium is that stdout, the standard output of the program is directed to this textbox, this means that all print statements will appear in this textbox. This is better than the alternative, which is the feedback given in the terminal used starting Delirium, making the user have to switch between windows or create a lot more code to place all information manually with Tkinter. This gives us the ability to give the user feedback on the program, either logging of jamming and spoofing turning on and off, and errors in edge cases. This works for all print statements even if they are in the API code or in the flowcharts for the jamming methods.

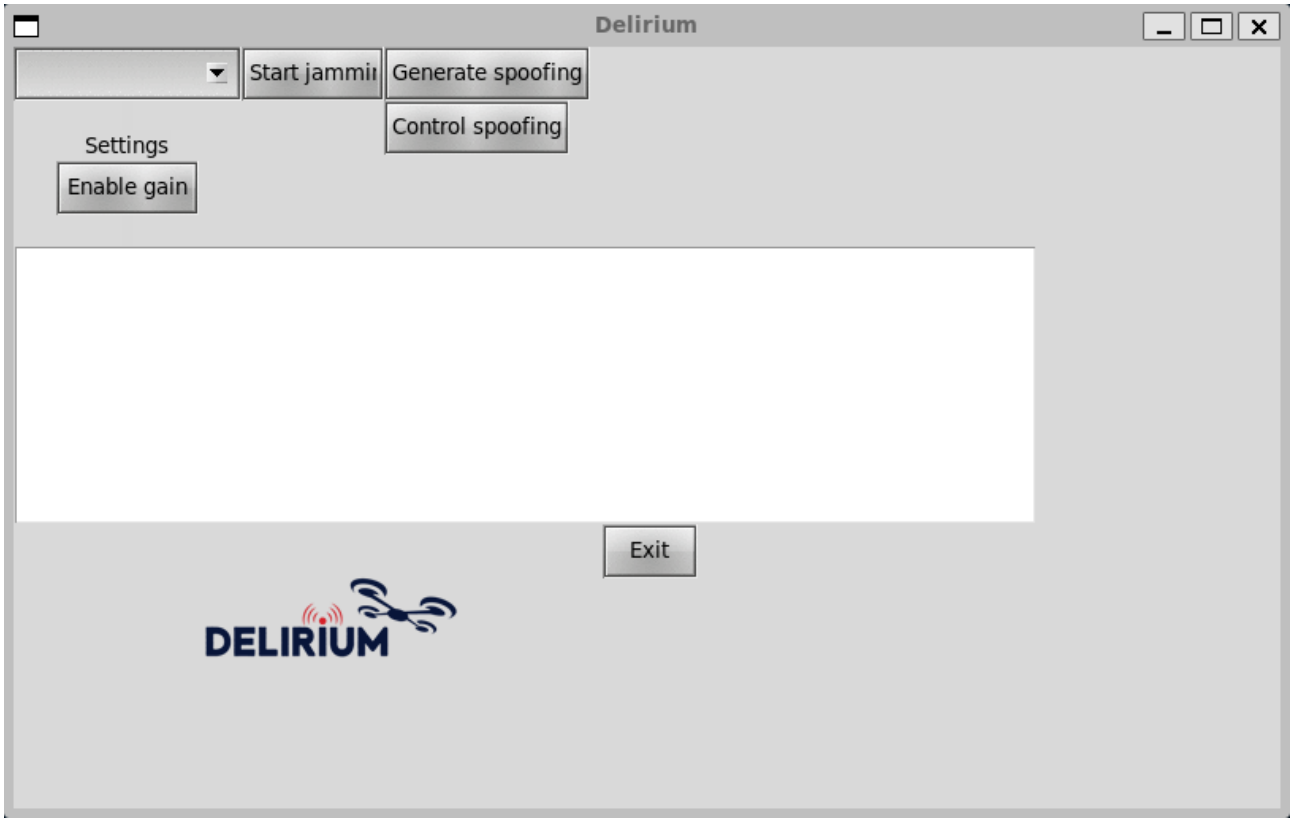


Figure 3.44: Screenshot of Delirium GUI.

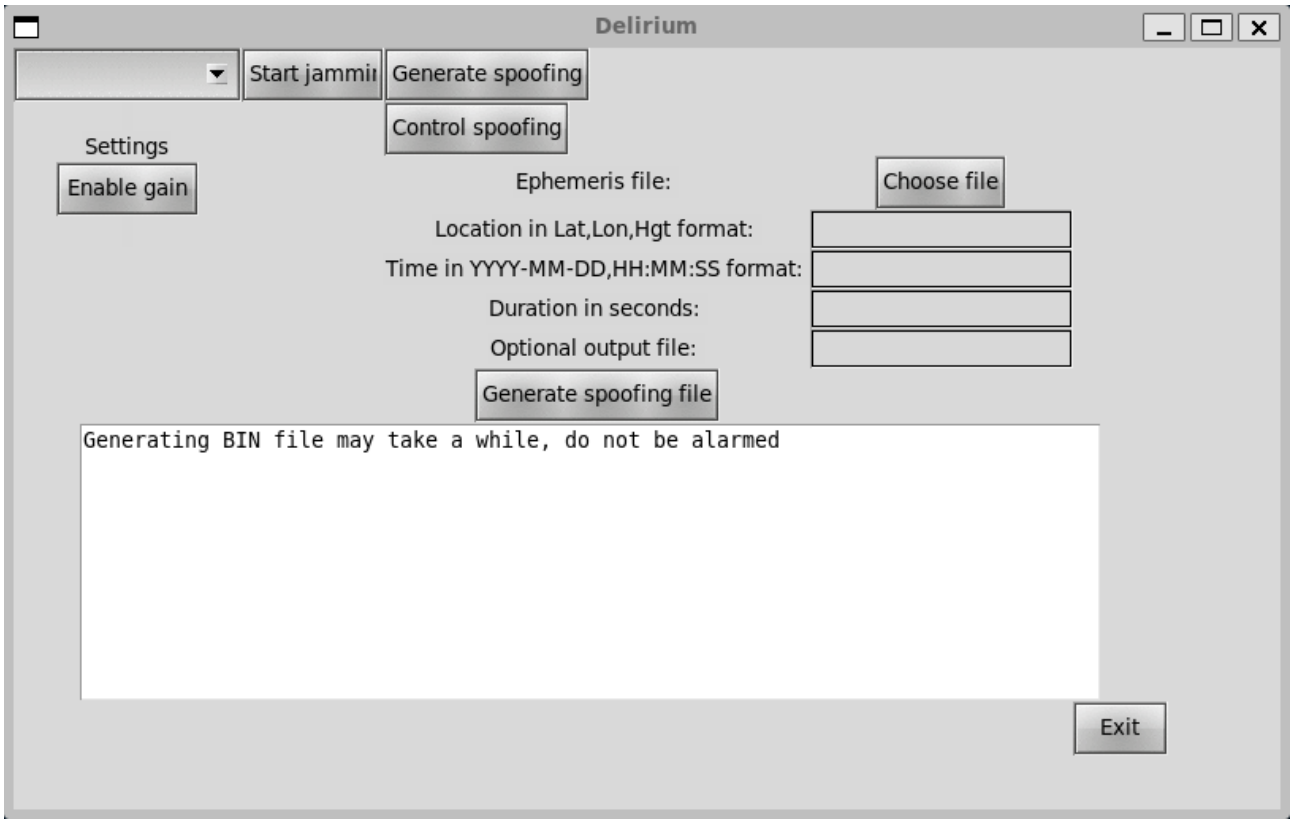


Figure 3.45: Screenshot of generate frame in GUI.

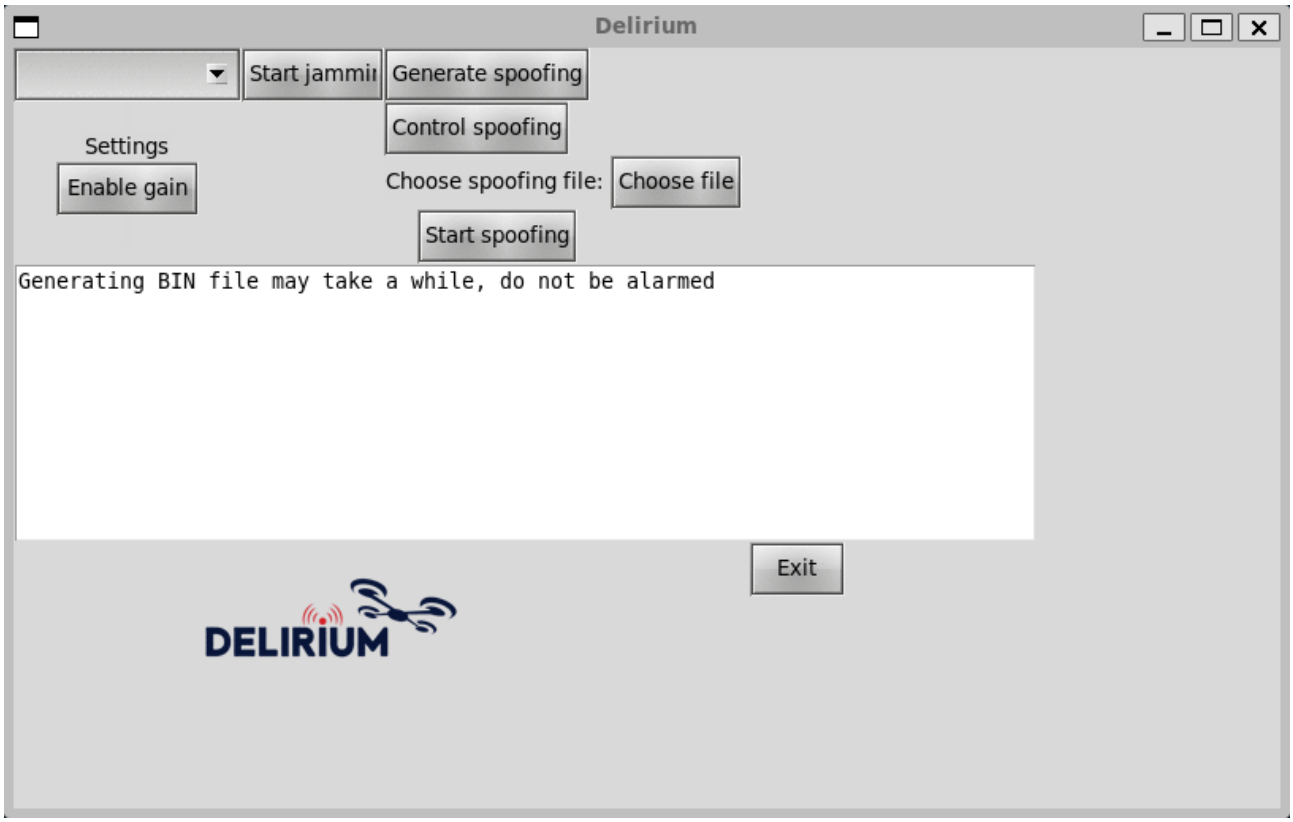


Figure 3.46: Screenshot of control frame in GUI.

Chapter 4

Epilogue

This chapter covers the conclusion of project Delirium. Here we will present our results, the conclusions that can be drawn from our results, and propose some recommendations for further expansion of Delirium.

17 Results

AM | *ABS*

This project's resultant system - Delirium - is a highly capable jammer that provides several strategies to interfere with the reception of radio frequency signals. Furthermore, it is capable of simultaneously spoofing GPS signals, providing the user with a simple interface to type in coordinates, create faux GPS baseband data, and broadcast this data as credible signals to any GPS receiver. A diagram showing the architecture of Delirium in both software and hardware can be seen in appendix L, this reference diagram is an overview of the system in terms of the connections between all the components within it, and schematically shows the system in its final state. Delirium is also a very portable system capable of deployment in any undeveloped area. Pictures taken of the actual system in its final physical state can be seen in the figures starting at 4.1a.

The Delirium GUI is a simple and user-friendly interface which allows the user access to many of Deliriums functions. It allows the starting/stopping and gain-tuning of our GNU Radio implemented SDR jammers, which are:

- Barrage jammer
- Spot jammer
- Protocol aware jammer
- Protocol aware jammer with sweeping

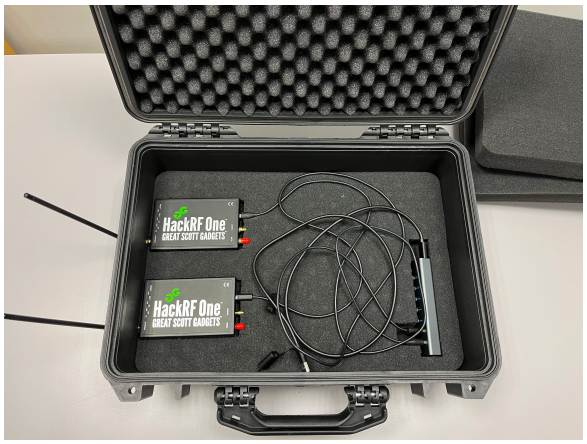
The GUI further allows the generation of faux GPS baseband data via the gps-sdr-sim application (for more information, see section 15.3), and then the starting/stopping of our "spoofer", which is it's own GNU Radio implemented SDR. Previously generated baseband data can also be loaded in, making Delirium a reliable testing device. Delirium allows the running of both a

(a) Delirium with its test-cables connected.

(b) With its antennas connected, Delirium can interact with radio frequency receivers through the air.

(c) The front side, showing a simplified 3D-printed version of our logo.

(d) Delirium from the user's perspective with the laptop and powerbank visible. The antennas can be conveniently stored in front of the laptop.



(e) In the layers below the laptop the HackRF One's can be found, along with the USB-hub needed to connect them to the laptop.

spoofer and a jammer simultaneously, enabling an effective attack against GNSS receivers by e.g., jamming other GNSS's while spoofing GPS signals. A screenshot of our GUI can be seen in figure 4.2, and screenshots of it in different states can be found in section 16.6.

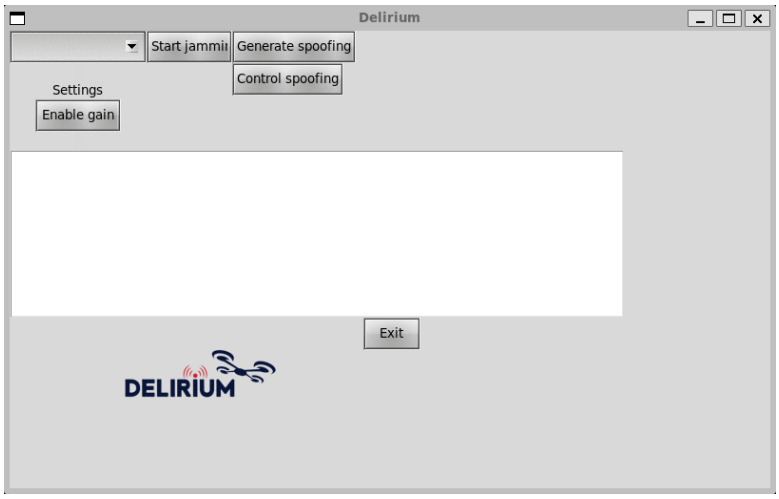


Figure 4.2: Screenshot of the Delirium GUI.

Despite many hours of effort, and considerable amounts of research to ensure compatibility (see sections 15.5 and 15.6), we did not succeed with the intended integration of either the Raspberry Pi 5 nor the 7" touch screen. We knew that Ubuntu was the recommended operating system (OS) for the software dependencies Delirium had, so efforts were made to install it on the Raspberry. When doing this, however, we did not succeed in enabling the touch functionality of the screen. The Raspberry Pi official OS was then installed. This enabled the touch functionality but caused a compatibility error with GNU Radio, which proved itself unsolvable despite much effort made. Our initial concept of a laptop running the Ubuntu OS was chosen instead, on which we earlier had verified the requirements of Delirium.

18 Conclusions

HK | *SN*

This was the first attempt at a multi-disciplinary engineering project for the participants. We realized we had to create a solid development plan and structure our work so that we could avoid miscommunication and cooperate effectively. We studied different project management techniques and landed on SCRUM, as discussed in section 13.2. This technique has proved suitable for us and enabled us to work on our separate prioritized fields, and then integrate our work.

Then we had to identify our stakeholders' needs. This resulted in several meetings with KDA and the user stories in section 14.2. From these, we made a red thread through requirements (see appendix E) and discussed and noted the risk factors around our project. We feel that although the task was not precisely defined by our stakeholders, we managed to derive requirements that matched our stakeholdersť wishes well, and they are satisfied with our current solution.

Given the large scope of our initial project, we decided to divide our project into several

milestones, as explained in section 14.1. This enabled us to work on a smaller project and focus on reaching the goal of our current milestone. This also gave us several alternate goals, should our progress not enable us to reach the full scope of the project. We could then integrate these milestones into our timeline which helped us visualize the deadlines we were working with. We ended up completing milestone 4, which was our end goal.

A central part of our system is the SDR and we did thorough research into which system would fit our needs best (ref. section 15.2). During this research, we had to take into account the time of arrival of the components we needed, because of the relatively short timeline (one semester) of our project. Doing this ensured that we got the HackRF One in good time. We have been satisfied with how the SDR peripheral have operated, and besides a planned upgrade to the spoofing HackRF, with an external oscillator, we had no issues with them.

We envisioned early on to test our system outside wirelessly but knew this was illegal and required explicit consent from NKOM (ref. section 12). Therefore we planned our approach around wired testing and testing in the basement Faraday cage. We began working on the application to NKOM in March, see appendix P, but mainly focused on the main project. This application proved to be more technical and time-consuming than first anticipated, therefore we did not apply in time, and we have thus not been able to legally test wireless transmission outside on real-world signals. However, through extrapolation of our wireless testing in the Faraday cage and our wired testing on the Navio chip and on the signal analyzer, we are confident that our system would have been successful in the real-world.

To conclude this project we can confidently say that our solution fulfills the task that was presented to us. Our package is a small container with the capability of jamming and spoofing GPS signals and is portable and not dependent on any infrastructure. It can be used by our customer to penetration test their internal systems that are using GNSS receivers and locate any weaknesses that may be present.

There are some functionalities we did not have time to integrate into our solution, and we elaborate on these in section 19.

19 Recommendations

SN | *ABS*

As we conclude the development of the Delirium project, several avenues for future enhancement and refinement have been identified. While Delirium successfully integrates radio frequency jamming and GPS spoofing into a portable system, there are specific areas that could benefit from further development to maximize the system's potential and usability. This section outlines key recommendations for future work, focusing on improving the GUI, incorporating additional features, and performing code cleanup. These enhancements aim to not only streamline the user experience but also expand the functional capabilities and maintainability of the system, ensuring Delirium remains a robust and versatile tool for penetration testing and GNSS signal interference.

19.1 Spoofing a drifting GPS-signal

SN | *AM*

A significant feature for future development of Delirium is the ability to spoof a drifting GPS signal. This functionality would enable the user to cause the target of Delirium to "drift" in a specified direction. Implementing this feature in the future should not present significant difficulties. Currently, we generate the faux GPS signal data through gps-sdr-sim by passing the "-l" option with a static location in latitude/longitude/height (LLH) format, such as "30.286502,120.032669,100". Instead, the "-u" option could be used and with a provided CSV (comma-separated values) file. This CSV file must then contain dynamic location data at a 10Hz frequency, representing ten location entries per second. More information about this feature can be found on gps-sdr-sim's GitHub page [65].

This feature could also be implemented as a "CSV generator" in Delirium. The user would e.g., input three variables: latitude, longitude, and height. They would also specify the drift's velocity and the file's duration in seconds. A Python function could then generate a CSV file where the location data changes according to the user's settings.

19.2 Future Work for the Delirium GUI

SN | *ABS*

Several features and potential developments for the Delirium GUI were not completed due to time constraints.

19.2.1 Cleanup of Codebase

SN | *ABS*

As discussed in section 16.6, a cleanup of the delirium.py code is recommended. Some functions have become quite lengthy, and code repetition is evident. For example, in the toggle jamming function, six lines of code are repeated four times to update the GUI when starting or stopping the jammer. These lines could be refactored into a separate function to improve readability and reduce technical debt.
Another example of the need for cleanup is the code for creating the GUI elements. In tkinter, each element requires at least two lines of code: one for creating the element and one for placing it on the screen. As we added more functionality to the Delirium GUI, the codebase grew increasingly unorganized. This growth has made it more difficult to keep track of elements and understand the overall structure intuitively. The lack of organization has resulted in a codebase that is harder to maintain and extend.

References

[1] Comparison of several satellite navigation system orbits. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/b/b4/Comparison_satellite_navigation_orbits.svg

[2] L1, l2, l5, l3, and simply l frequency bands. [Online]. Available: https://gnss.store/blog/post/l1-l2-l5-l3-and-simply-l-frequency-bands.html

[3] A. M. Palash Choudhari, Varun Karthikeyan. (2018) Electronic warfare - radar noise jamming. [Online]. Available: http://fullafterburner.weebly.com/next-gen-weapons/electronic-warfare-radar-noise-jamming

[4] Spread spectrum and code modulation of l1 gps carrier. [Online]. Available: https://www.e-education.psu.edu/geog862/node/1753

[5] Introduction to clockify. [Online]. Available: https://clockify.me/help/getting-started/introduction-to-clockify

[6] Your first flowgraph. [Online]. Available: https://wiki.gnuradio.org/index.php/Your_First_Flowgraph

[7] Hackrf one. [Online]. Available: https://greatscottgadgets.com/hackrf/one/

[8] Beskyttelseskoffert m. [Online]. Available: https://www.jula.no/catalog/bygg-og-maling/oppbevaring/oppbevaring-av-verktoysmasaker/verktoykasser/beskyttelseskoffert-001594/

[9] Neo-m8n-0. [Online]. Available: https://no.mouser.com/ProductDetail/u-blox/NEO-M8N-0?qs=zW32dvEIR3unZhZI0KRbew%3D%3D

[10] What is a saw filter. [Online]. Available: https://www.everythingrf.com/community/what-is-a-saw-filter

[11] Apitech, "Introduction to saw filter theory and design techniques," pp. 1–16, 2018. [Online]. Available: https://www.spectrumcontrol.com/globalassets/documents/rf2m-us/white-paper---saw-filter-2018.pdf

[12] Saw components b7839. [Online]. Available: https://www.mouser.com/datasheet/2/136/B7839-77972.pdf

[13] A. Zaccaron. Digital radio frequency memory drfm for ecm applications. [Online]. Available: https://www.emsopedia.org/entries/digital-radio-frequency-memory-drfm-for-ecm-applications/

[14] Additive white gaussian noise. [Online]. Available: https://en.wikipedia.org/wiki/Additive_white_Gaussian_noise

[15] Boilerplate code. [Online]. Available: https://en.wikipedia.org/wiki/Boilerplate_code

[16] Ephemeris. [Online]. Available: https://en.wikipedia.org/wiki/Ephemeris

[17] L band. [Online]. Available: https://en.wikipedia.org/wiki/L_band

[18] Ltspice. [Online]. Available: https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html

[19] E. Ries. Minimum viable product: a guide. [Online]. Available: https://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html

[20] M. Rouse. Minimum viable product. [Online]. Available: https://www.techopedia.com/definition/27809/minimum-viable-product-mvp

[21] Nanomotion. What is the piezoelectric effect? [Online]. Available: https://www.nanomotion.com/nanomotion-technology/the-piezoelectric-effect/

[22] Consumer drone unit shipments worldwide from 2020 to 2030. [Online]. Available: https://www.statista.com/statistics/1234658/worldwide-consumer-drone-unit-shipments/#:~:text=The%20total%20number%20of%20consumer,unit%20shipments%20globally%20by%202030.

[23] European Space Agency. Gnss signal. [Online]. Available: https://gssc.esa.int/navipedia/index.php/GNSS_signal

[24] Pvt. [Online]. Available: https://gnss-sdr.org/docs/sp-blocks/pvt/#precise-point-positioning

[25] Positioning computation. [Online]. Available: https://galileognss.eu/positioning-computation/

[26] Light cones. [Online]. Available: https://en.wikipedia.org/wiki/Satellite_navigation_solution

[27] The global positioning system. [Online]. Available: https://oceanservice.noaa.gov/education/tutorial_geodesy/geo09_gps.html

[28] European Space Agency. Gps general introduction. [Online]. Available: https://gssc.esa.int/navipedia/index.php/GPS_General_Introduction#:~:text=GPS%20Signal%20Structure,-GPS%20satellite%20program&text=The%20main%20GPS%20carrier%20signal,military%20and%20authorized%20civilian%20users.

[29] U.S. Space Force. Space segment. [Online]. Available: https://www.gps.gov/systems/gps/space/

[30] Gps wiki. [Online]. Available: https://en.wikipedia.org/wiki/GPS_signals

[31] D. Kuhlman, "A python book: Beginning python, advanced python, and python exercises." [Online]. Available: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html#introduction-python-101-beginning-python

[32] S. Cass, "The top programming languages 2023," aug 2023. [Online]. Available: https://spectrum.ieee.org/the-top-programming-languages-2023

[33] Python implementations. [Online]. Available: https://wiki.python.org/moin/PythonImplementations

[34] Globalinterpreterlock. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock

[35] About cmake. [Online]. Available: https://cmake.org/about/

[36] Extending python with c or c++. [Online]. Available: https://docs.python.org/3.10/extending/extending.html

[37] Building c and c++ extensions. [Online]. Available: https://docs.python.org/3.10/extending/building.html#building

[38] pybind11 seamless operability between c++11 and python. [Online]. Available: https://github.com/pybind/pybind11

[39] Build systems. [Online]. Available: https://pybind11.readthedocs.io/en/stable/compiling.html#pybind11-add-module

[40] Continuous integration. [Online]. Available: https://en.wikipedia.org/wiki/Continuous_integration

[41] Version control. [Online]. Available: https://en.wikipedia.org/wiki/Version_control

[42] Getting started - about version control. [Online]. Available: https://web.archive.org/web/20151225053703/http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[43] What is project management? [Online]. Available: https://www.apm.org.uk/resources/what-is-project-management/#:~:text=Project%20management%20is%20the%20application,a%20finite%20timescale%20and%20budget.

[44] "Manifesto for agile software development." [Online]. Available: https://agilemanifesto.org/

[45] K. Schwaber and J. Sutherland, "The scrum guide," 2020. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

[46] ProductPlan. What is jira? [Online]. Available: https://www.productplan.com/glossary/jira/

[47] Signal (messaging app). [Online]. Available: https://en.wikipedia.org/wiki/Signal_(messaging_app)

[48] M. Rehkopf. User stories with examples and a template. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories

[49] Is there a difference between requirements and design? [Online]. Available: https://baknowledgeshare.com/is-there-a-difference-between-requirements-and-design/

[50] What is gnu radio? [Online]. Available: https://wiki.gnuradio.org/index.php?title=What_is_GNU_Radio%3F

[51] Handling flowgraphs. [Online]. Available: https://wiki.gnuradio.org/index.php/Handling_Flowgraphs

[52] Gnu radio wiki: Hardware. [Online]. Available: https://wiki.gnuradio.org/index.php?title=Hardware#Great_Scott_Gadgets_HackRF

[53] Github: gr osmos. [Online]. Available: https://github.com/osmocom/gr-osmosdr

[54] Github: gr soapy. [Online]. Available: https://github.com/dicta/gr-soapy

[55] Gnu radio wiki: Creating your first block. [Online]. Available: https://wiki.gnuradio.org/index.php?title=Creating_Your_First_Block

[56] Gnu radio wiki: Creating c++ oot with gr-modtool. [Online]. Available: https://wiki.gnuradio.org/index.php?title=Creating_C%2B%2B_OOT_with_gr-modtool

[57] Gnu radio wiki: Creating python oot with gr-modtool. [Online]. Available: https://wiki.gnuradio.org/index.php?title=Creating_Python_OOT_with_gr-modtool#Creating_an_OOT_Module

[58] Updating firmware. [Online]. Available: https://hackrf.readthedocs.io/en/latest/updating_firmware.html

[59] Installing hackrf software. [Online]. Available: https://hackrf.readthedocs.io/en/latest/installing_hackrf_software.html

[60] hackrf github repo. [Online]. Available: https://github.com/greatscottgadgets/hackrf

[61] Gnu general public license, version 2. [Online]. Available: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html

[62] Only-VLSI. Complex programmable logic device. [Online]. Available: https://only-vlsi.blogspot.com/2008/05/complex-programmable-logic-device.html

[63] N. Kommunikasjonsmyndighet. Forbud mot jammere. [Online]. Available: https://nkom.no/frekvenser-og-elektronisk-utstyr/jammere-og-repeatere/jammere

[64] Earth data. [Online]. Available: https://urs.earthdata.nasa.gov/

[65] gps-sdr-sim. [Online]. Available: https://github.com/osqzss/gps-sdr-sim

[66] R. P. Foundation/Broadcom. (2023) Raspberry pi 5. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[67] GreatScottGadgets. Minimum host system requirements for hackrf. [Online]. Available: https://hackrf.readthedocs.io/en/latest/hackrf_minimum_requirements.html

[68] R. P. Foundation/Broadcom. Raspberry pi touch display. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/display.html

[69] E. Tutorials. T-pad attenuator. [Online]. Available: https://www.electronics-tutorials.ws/attenuators/t-pad-attenuator.html

[70] Colour coding and standard resistor values. [Online]. Available: https://www.open.edu/openlearn/science-maths-technology/an-introduction-electronics/content-section-2.4

[71] *NEO-M8 u-blox M8 concurrent GNSS modules Data sheet*, u-blox, 12 2022, rev. 12. [Online]. Available: https://content.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_UBX-15031086.pdf

[72] Modular programming. [Online]. Available: https://en.wikipedia.org/wiki/Modular_programming

[73] Whats new in python 3.10. [Online]. Available: https://docs.python.org/3/whatsnew/3.10.html

[74] Binary package python3 in ubuntu jammy. [Online]. Available: https://launchpad.net/ubuntu/jammy/+package/python3

[75] multiprocessing process-based parallelism. [Online]. Available: https://docs.python.org/3/library/multiprocessing.html

[76] gps-modulation. [Online]. Available: https://www.e-education.psu.edu/geog862/book/export/html/1407#:~:text=For%20example%2C%20if%20a%20satellite,the%20P(Y)%20code.

References of high regard

[RefH1] *NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*, Space Systems Command (SSC), 2022, rev. N. [Online]. Available: https://www.gps.gov/technical/icwg/IS-GPS-200N.pdf

[RefH2] B. Kahanwal, "Abstraction level taxonomy of programming language frameworks," 11 2013.

[RefH3] M. Vogel, *Handbook of Space Engineering, Archaeology, and Heritage, edited by A.G. Darrin and B.L. O'Leary*. Contemporary Physics, 2009.

[RefH4] G. Lykou, D. Moustakas, and D. Gritzalis, "Defending airports from uas: A survey on cyber-attacks and counter-drone sensing technologies." *Sensors*, 2020.

[RefH5] T. E. Humphreys, B. M. Ledvina, M. L. Psiaki, B. W. Q'Hanlon, and P. M. Kintner, "Assessing the spoofing threat: Development of a portable gps civilian spoofer," *Proceedings of the 21st International Technical Meeting of the Satellite Division of the Institute of Navigation*, 2008.

[RefH6] R. V. Karpe and S. Kulkarni, "Software defined radio based global positioning system jamming and spoofing for vulnerability analysis," *Institute of Electrical and Electronics Engineers*, 2020.

[RefH7] R. Ferreria, J. Gaspar, P. Sebastiao, and N. Souto, "Effective gps jamming techniques for uavs using low-cost sdr platforms," *Wireless Personal Communications*, 2020.

[RefH8] A. Hussain, N. A. Saqib, U. Qamar, M. Zia, and H. Mahmood, "Protocol-aware radio frequency jamming in wi-fi and commercial wireless networks," *JOURNAL OF COMMUNICATIONS AND NETWORKS VOL. 16 NO. 4*, pp. 397–406, 2014.

[RefH9] W. Stallings, *Data and Computer Communications*. Pearson Education Limited, 2014.

[RefH10] J. R. van der Merwe, X. Zubizarreta, and A. Rügamer, "Classification of spoofing attack types," *Institute of Electrical and Electronics Engineers*, 2018.

[RefH11] B. I. Tuleuov and A. B. Ospanova, *Beginning C++ Compilers : An Introductory Guide to Microsoft C/C++ and MinGW Compilers*. Imprint: Apress, 2024.

[RefH12] M. Fowler. (2024) Continuous integration. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration

[RefH13] A. Faisandier and G. Roedler, "Stakeholder requirements definition," the relevant content are tables 2 and 3; respectively requirements classification and characteristics conformity. [Online]. Available: https://sebokwiki.org/wiki/Stakeholder_Requirements_Definition

[RefH14] X.-C. Zheng and H.-M. Sun, "Hijacking unmanned aerial vehicle by exploiting civil gps vulnerabilities using software-defined radio," *Sensors and Materials*, vol. 32, p. 2729, 08 2020.

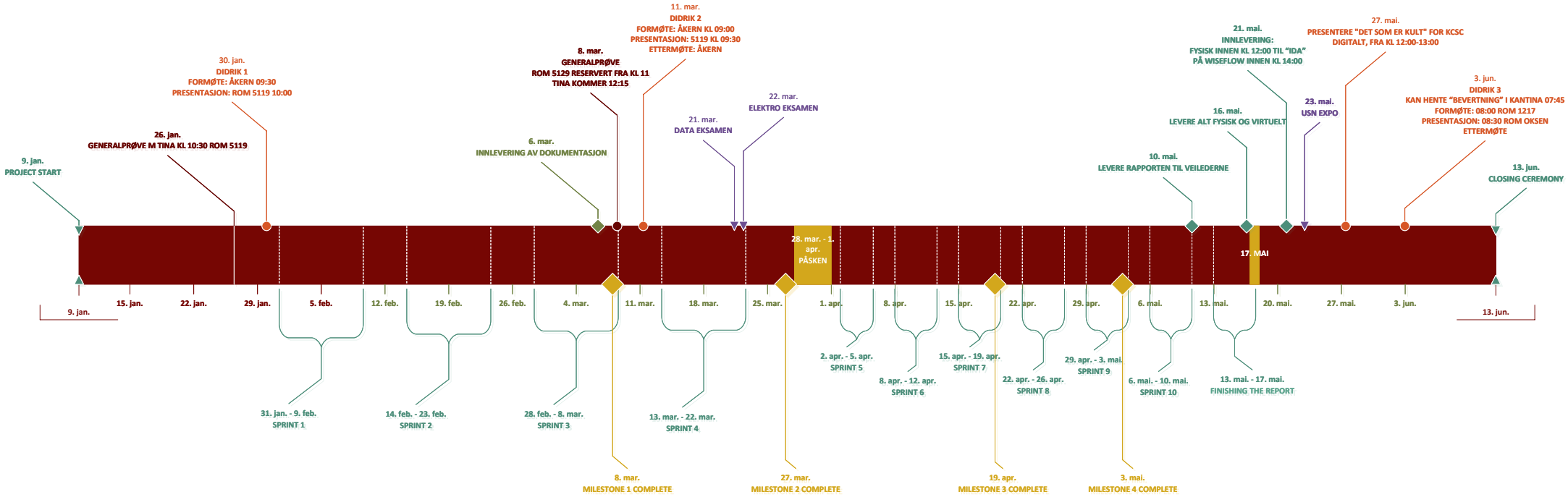
[RefH15] D. Huizinga and A. Kolawa, *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Press, 2007.

Appendices

Appendix A

Project Timeline

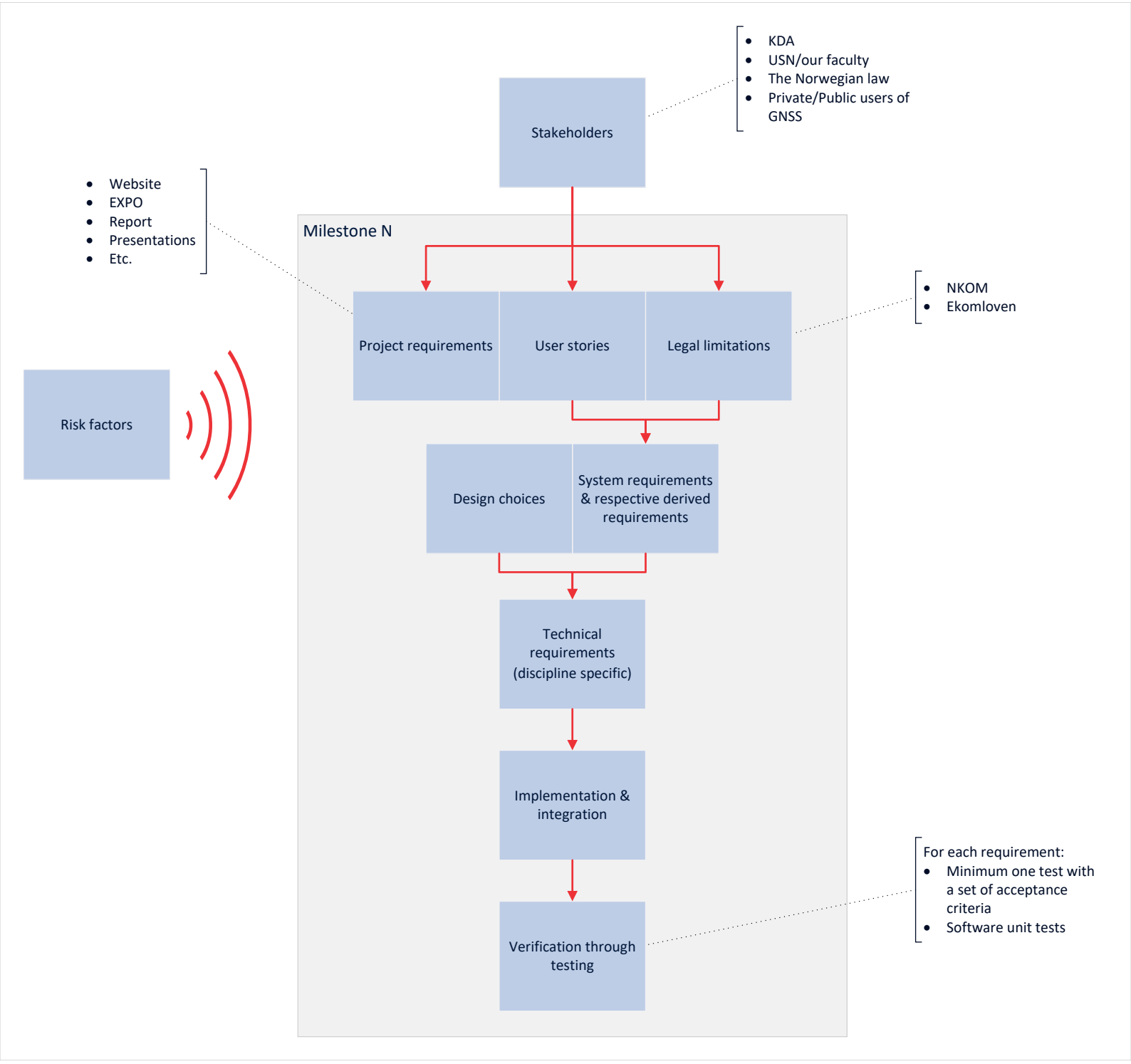
The following page shows the timeline for project Delirium. It was tentatively updated, and was used as an overview for planning our efforts.


9. jan.
PROJECT START
26. jan.
GENERALPRØVE M TINA KL 10:30 ROM 5119
30. jan.
DIDRIK 1
FORMØTE: ÅKERN 09:30
PRESENTASJON: ROM 5119 10:00
6. mar.
INNLEVERING AV DOKUMENTASJON
8. mar.
GENERALPRØVE
ROM 5129 RESERVERT FRA KL 11
TINA KOMMER 12:15
11. mar.
DIDRIK 2
FORMØTE: ÅKERN KL 09:00
PRESENTASJON: 5119 KL 09:30
ETTERMØTE: ÅKERN
21. mar.
DATA EKSAMEN
22. mar.
ELEKTRO EKSAMEN
28. mar. - 1. apr.
PÅSKEN
10. mai.
LEVERE RAPPORTEN TIL VEILEDERNE
16. mai.
LEVERE ALT FYSISK OG VIRTUELT
21. mai.
INNLEVERING:
FYSISK INNEN KL 12:00 TIL "IDA"
PÅ WISEFLOW INNEN KL 14:00
23. mai.
USN EXPO
17. MAI
27. mai.
PRESENTERE "DET SOM ER KULT" FOR KCSC
DIGITALT, FRA KL 12:00-13:00
3. jun.
DIDRIK 3
KAN HENTE "BEVERTNING" I KANTINA 07:45
FORMØTE: 08:00 ROM 1217
PRESENTASJON: 08:30 ROM OKSEN
ETTERMØTE
13. jun.
CLOSING CEREMONY
9. jan.
15. jan.
22. jan.
29. jan.
5. feb.
12. feb.
19. feb.
26. feb.
4. mar.
11. mar.
18. mar.
25. mar.
1. apr.
8. apr.
15. apr.
22. apr.
29. apr.
6. mai.
13. mai.
20. mai.
27. mai.
3. jun.
13. jun.
31. jan. - 9. feb.
SPRINT 1
14. feb. - 23. feb.
SPRINT 2
28. feb. - 8. mar.
SPRINT 3
13. mar. - 22. mar.
SPRINT 4
2. apr. - 5. apr.
SPRINT 5
8. apr. - 12. apr.
SPRINT 6
15. apr. - 19. apr.
SPRINT 7
22. apr. - 26. apr.
SPRINT 8
29. apr. - 3. mai.
SPRINT 9
6. mai. - 10. mai.
SPRINT 10
13. mai. - 17. mai.
FINISHING THE REPORT
8. mar.
MILESTONE 1 COMPLETE
27. mar.
MILESTONE 2 COMPLETE
19. apr.
MILESTONE 3 COMPLETE
3. mai.
MILESTONE 4 COMPLETE

Appendix B

The Red Thread

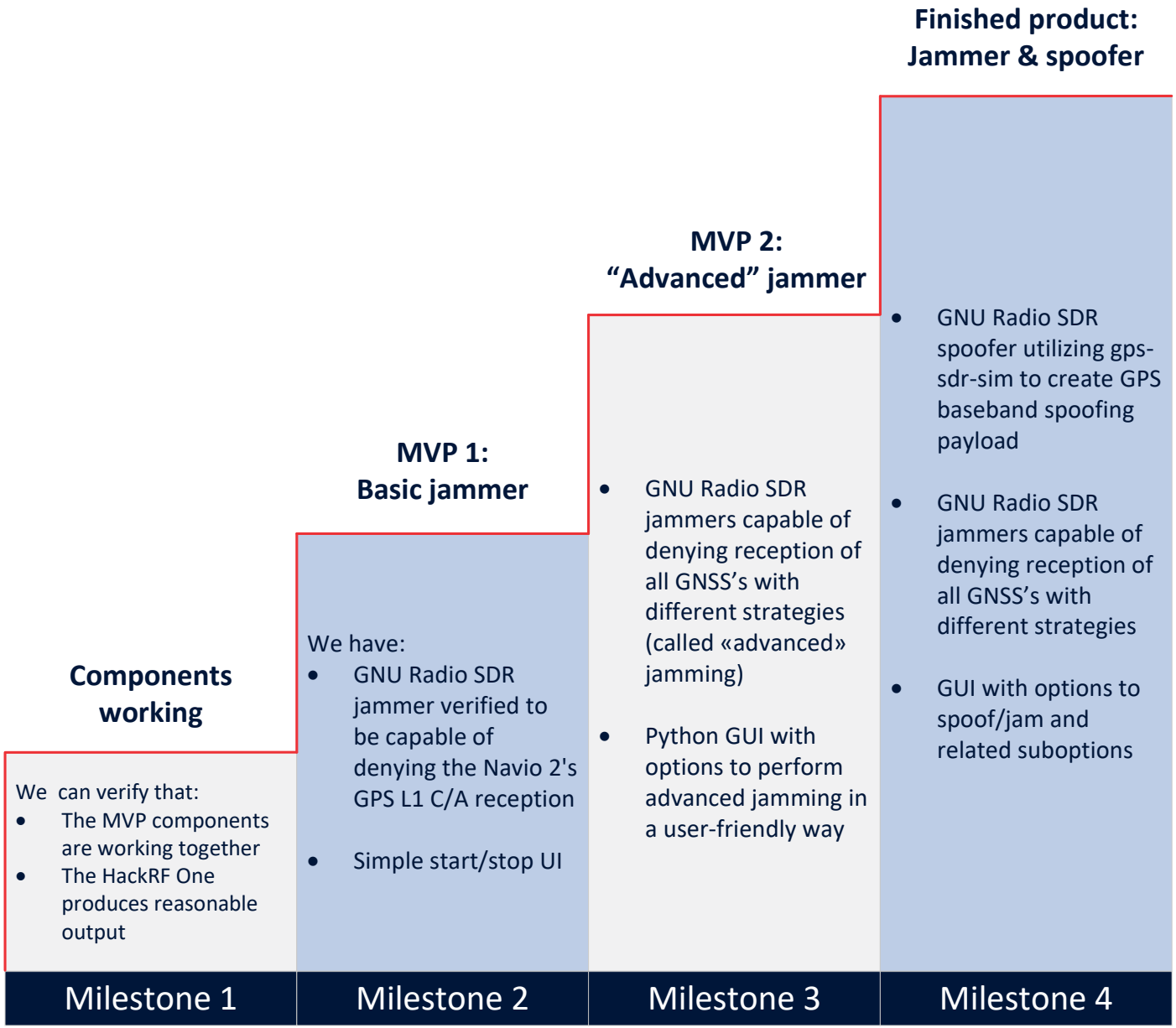
The following diagram shows our red thread - the path which is taken from a stakeholder requirement, through its implementation in the product and with its end in verification (see section 14 for more information). This diagram was used to illustrate both internally and to our stakeholders how we should relate the systems engineering artifacts we produced to each other.


Stakeholders
KDA
USN/our faculty
The Norwegian law
Private/Public users of GNSS
Website
EXPO
Report
Presentations
Etc.
Milestone N
Project requirements
User stories
Legal limitations
NKOM
Ekomloven
Risk factors
Design choices
System requirements & respective derived requirements
Technical requirements (discipline specific)
Implementation & integration
Verification through testing
For each requirement:
Minimum one test with a set of acceptance criteria
Software unit tests

Appendix C

Milestones

The following page shows our milestones diagram containing our iterative goals for the project in ascending order. It was widely used as a reference for the team in regards to planning and progress tracking, and was approved by our customer, KDA, and our internal supervisor.



Low-priority/ideal features

Embed the system within a "tactical" suitcase with:

- Powerbank
- Small computer (e.g., Rasp.Pi 5)
- Touch screen To display the GUI
- 2 HackRF's; one for spoofing, one for jamming (can be used simultaneously)

Appendix D

User Stories

Title: **Jamming GPS signals**	**Priority:** **High**	**Estimate:** **3 months**
User story: **As a** penetration tester **I want** to jam the GPS-signals received by the drone **So that** the drone no longer has communication with GPS-satellites		
Acceptance criteria: **Given** that the drone is within visible reach, and the jammer is set to function on the correct frequency **When** the jammer starts **Then** the drone shall be unable to communicate with GPS-satellites		

Title: **Position spoofing**	**Priority:** **High**	**Estimate:** **5 months**
User story: **As a** penetration tester **I want** to spoof the drone's position **So that** the drone thinks it is somewhere else than its actual position		
Acceptance criteria: **Given** that I am able to send credible fake packets of information to the drone **When** a command is prompted from my system to spoof the drone **Then** the drone will receive the false signals, and perceive them as real		

Title: **Location-specific spoofing**	**Priority:** **Medium -> High**	**Estimate:** **5 months**
User story: **As a** penetration tester **I want** to spoof the GNSS-receiver by tricking it into thinking it is at a specific location decided by me (an airport, a different country, etc.) **So that** I can give my system the ultimate test against spoofing		
Acceptance criteria: **Given** that the jamming of the drone's GPS signals is successful, and I am able to send credible fake packets of information to the GNSS-receiver **When** the system starts to jam GPS-signals, and send credible fake packets of information to the GNSS-receiver **Then** the GNSS-receiver shall receive the fake packets of information, perceive them as real, and "believe" it is at the location decided by my system		

Title: **Portability**	**Priority:** **High**	**Estimate:** **3 months**
User story: **As a** penetration tester **I want** a portable system **So that** I can easily bring it to remote locations		
Acceptance criteria: **Given** high focus on a compact and portable design **When** developing our system **Then** the system will be portable and easy to handle		

Title: **Navio 2/NEO-M8N**	**Priority:** **High**	**Estimate:** **6 months (end goal)**
User story: **As a** penetration tester **I want** to test the flight controller, Navio 2 and its GPS-chip NEO-M8N's ability to withstand attacks through jamming and spoofing **So that** I can use this information to prevent similar attacks from intruders/outsiders		
Acceptance criteria: **Given** the outcome of testing the Navio 2 flight controller **When** testing its ability to prevent attacks **Then** the results from the test will decide the future use of said component		

Title: **User interface**	**Priority:** **Medium -> High**	**Estimate:** **5-6 months**
User story: **As a** penetration tester **I want** a simple user interface **So that** I can easily control my system		
Acceptance criteria: **Given** that a UI is available **When** operating the system **Then** I have a simple user interface		

Appendix E

System Requirements

The following overview show the system requirements for Delirium, made by AM. Coloring has been used to make it easier to separate the types:

- Dark blue - Top-level system requirements
- Light blue - Derived requirements
- Red - Legal requirements
- Orange - Design choices
- Gray - Technical, discipline-specific requirements

The Derived requirements and so on relating to a single top-level requirement are shown on the same page. For more information about their contents and coherence, see section 14.3.1.

ID: TLR1	Source: US "Portability"	Attached milestone: M2, M3, M4
Verification: T2.1.1	Type: Top-Level Physical	Compliance status: Complies in M2, M3, M4
The system must have the ability to be deployed in an undeveloped area. Definition of undeveloped area: "An undeveloped place or piece of land has not been built on or used for farming". - https://dictionary.cambridge.org/dictionary/english/undeveloped		

ID: DR1.1	Source: TLR1	Attached milestone: M2, M3, M4
Verification: T2.1.1	Type: Derived Functional	Compliance status: Complies in M2, M3, M4
To enable the required system portability, the system must be self-sufficient in regards to power.		

ID: TR1.1	Source: TLR1	Attached milestone: M2, M3, M4
Verification: -	Type: Technical Electronics	Compliance status: Complies in M2, M3, M4
The system must consist of a laptop containing a battery which is capable of running the system by itself.		

ID: DC1.1	Source: TLR1	Attached milestone: Beyond M4
Verification: -	Type: Design choice	Compliance status: Integration failed
We will expand the system with a powerbank capable of powering the entire new configuration consisting of a Raspberry Pi 5 with a touch screen instead of the laptop.		

ID: TR1.2	Source: DC1.1	Attached milestone: Beyond M4
Verification: -	Type: Technical Electronics	Compliance status: Integration failed
There must be aquired and integrated a powebank which fulfills the new configuration's needs in regards to power.		

ID: TLR2	Source: US "Portability"	Attached milestone: M2, M3, M4
Verification: T2.1.1	Type: Top-Level Physical	Compliance status: Complies in M2, M3, M4
The system must have the ability to be carried in an undeveloped area. Definition: undeveloped areas are not used for farming or industry, or do not have any buildings on them - https://dictionary.cambridge.org/dictionary/english/undeveloped		

ID: DR2.1	Source: TLR2	Attached milestone: M2, M3, M4
Verification: T2.1.1	Type: Derived Physical	Compliance status: Complies in M2, M3, M4
To enable the required portability, the system must fit entirely into an average student's backpack, along with an average-sized laptop and its charger		

ID: DC2.1	Source: TLR2	Attached milestone: Beyond M4
Verification: -	Type: Design choice	Compliance status: -
We will expand the system with a "tactical suitcase" which will fit the entire system within it.		

ID: TR2.1	Source: DC1.1, DC2.1	Attached milestone: Beyond M4
Verification: -	Type: Technical Electronics	Compliance status: Integration failed
The "tactical suitcase" must fit the entire "Beyond M4"-system within it including: The powerbank Raspberry Pi 5 with touch screen 2x HackRF One 2x RF antennas		

ID: TR2.2	Source: DC2.1	Attached milestone: Beyond M4
Verification: -	Type: Technical Electronics	Compliance status: Complies beyond M4
The "tactical suitcase" must fit the entire system within it including: The powerbank Laptop 2x HackRF One 2x RF antennas		

ID: TLR3	Source: US "Jamming GPS signals "	Attached milestone: M2, M3, M4
Verification: T3.1	Type: Top-Level Functional	Compliance status: Complies in M2, M3, M4
The system must be able to jam the Navio 2 flight controller's reception of GPS-signals. As long as the system is jamming, the Navio 2 cannot determine its location using the GPS. Jamming is interpreted as "the deliberate blocking of or interference with wireless communications" -https://en.wikipedia.org/wiki/Radio_jamming		

ID: DR3.1	Source: TLR3, TLR4	Attached milestone: M2, M3, M4
Verification: T1.1.1	Type: Derived Functional	Compliance status: Complies in M2, M3, M4
To be able to jam the Navio 2 flight controller's GPS-signal reception and spoof its location, the system must be able to transmit radio frequency waves at 1575.42MHz, which is the GPS L1 C/A singal's carrier frequency.		

ID: DR3.2	Source: TLR3, TLR4	Attached milestone: M2, M3, M4
Verification: T1.1.1	Type: Derived Functional	Compliance status: Complies in M2, M3, M4
The system must emit a signal centred around the GPS-L1 frequency with a bandwidth of atleast 15,3 MHz.		

ID: DR3.3	Source: TLR3, TLR4	Attached milestone: M2, M3, M4
Verification: T3.2.1	Type: Derived Functional	Compliance status: Complies in M2, M3, M4
The signal strength must be adjustable and atleast 30 dBm over background noise.		

ID: DC3.1	Source: TLR3, DR3.*, DR4.2	Attached milestone: M2, M3, M4
Verification: -	Type: Design choice	Compliance status: -
We will use use GNU Radio as the means to create several Software Defined Radios (SDR) which satisfy the source requirements.		

ID: TR3.1	Source: DC3.1	Attached milestone: M2, M3, M4
Verification: -	Type: Technical Electronics	Compliance status: Complies in M2, M3, M4
At the very least, there must be implemented one SDR which is capable of jamming GPS-signals and one which is capable of transmitting GPS-signals.		

ID: TLR4	Source: US "Position spoofing"	Attached milestone: M4
Verification: T4.1	Type: Top-Level Functional	Compliance status: Complies in M4
The system must be able to tell the Navio 2 flight controller where it is by location-specific GPS spoofing. Definition of spoofing: "A global positioning system (GPS) spoofing attack attempts to deceive a GPS receiver by broadcasting fake GPS signals, structured to resemble a set of normal signals" -paraphrased from		

ID: DR4.1	Source: TLR4	Attached milestone: M4
Verification: T4.1	Type: Derived Functional	Compliance status: Complies in M4
The system must be able to produce a credible GPS-baseband data stream		

ID: DR4.2	Source: TLR4	Attached milestone: M4
Verification: T4.1	Type: Derived Functional	Compliance status: Complies in M4
The system must be able to broadcast a stream of GPS baseband data		

ID: DC4.1	Source: DR4.1	Attached milestone: M4
Verification: -	Type: Design choice	Compliance status: -
We will use use the gps-sdr-sim program to produce credible GPS baseband data		

ID: TR4.1	Source: TLR4, DR4.1, DC4.1	Attached milestone: M4
Verification: -	Type: Technical Data	Compliance status: Complies in M4
The system must implement the gps-sdr-sim program such that the user can specify an exact position in lat/long/height-coordinates, so that gps-sdr-sim produces the baseband data accordingly.		

ID: TR4.2	Source: DR4.2	Attached milestone: M4
Verification: -	Type: Technical Data	Compliance status: Complies in M4
To be able to broadcast GPS data, the system must include a GNU Radio SDR specific for this functionality.		

ID: TLR5	Source: US "User Interface"	Attached milestone: M2, M3, M4
Verification: User feedback	Type: Top-Level Useability	Compliance status: Complies in M2, M3, M4
The system must provide a User Interface (UI) which is simple to use. "Simple to use" is interpreted as being possible for a non-expert to use without confusion with the premise that they understand what they are hoping to achieve with the system.		

ID: TR5.1	Source: TLR5, TLR6	Attached milestone: M2
Verification: -	Type: Technical Data	Compliance status: Complies in M2
A simple start/stop program capable of starting the desired SDR must be made.		

ID: DC5.1	Source: TLR5, TLR6	Attached milestone: M3, M4
Verification: -	Type: Design choice	Compliance status: -
We will make a Python Graphical User Interface (GUI) with the tkinter Python package. The intention of this is to make usage of the system as easy as possible for the user.		

ID: TR5.2	Source: DC5.1, TLR5, TLR6	Attached milestone: M3, M4
Verification: -	Type: Technical Data	Compliance status: Complies in M3, M4
A GUI must be made using the Python language and the tkinter package.		

ID: TLR6	Source: US "User Interface"	Attached milestone: M2, M3, M4
Verification method: T6.1	Type: Top-Level Functional	Compliance status: Complies in M2, M3, M4
The system must provide a User Interface (UI) which itself provides - to the user - the ability to utlize the required functionality of the system at any point in time.		

ID: DR6.1	Source: TLR6, DC5.1	Attached milestone: M2, M3, M4
Verification: T6.1	Type: Derived Functional	Compliance status: Complies in M2, M3, M4
Through the UI the user must have the ability to both start and stop an SDR (The user must easily be able to start an SDR, as this is the whole concept of our product and will grant the user its value. The SDR must quit when told to do so, as NKOM stricly forbids jamming of any kind)		

ID: TR6.1	Source: DR6.1, DR3.3, DC4.1	Attached milestone: M3, M4
Verification: -	Type: Technical Data	Compliance status: Complies in M3, M4
The GUI must provide the following abilities to the user: -Start/stop any implemented SDR -Control the gain on each SDR -In M4; pass user options to gps-sdr-sim		

ID: DC6.1	Source: DR6.1	Attached milestone: Beyond M4
Verification: -	Type: Design choice	Compliance status: -
We will implement the ability for the user to verify the state of each peripheral device without the need of an external oscilloscope or the indicators on each HackRF. Possible states include "idle", "N/A" and "transmitting".		

ID: TR6.2	Source: DC6.1	Attached milestone: Beyond M4
Verification: -	Type: Technical Data	Compliance status: Integration with GUI failed; ran out of time
There must be implemented a program which uses the C-implemented libhackrf API to poll each SDR peripheral about their status		

ID: TLR7	Source: NKOM (EKOM-loven §6-2)	Attached milestone: All
Verification: -	Type: Legal Requirement	Compliance status: -
By Norwegian law it is strictly forbidden to perform any radio freqency jamming in the public or private space.		

ID: TLR8	Source: NKOM (EKOM-loven §6-2)	Attached milestone: All
Verification: -	Type: Legal Requirement	Compliance status: -
By Norwegian law it is strictly forbidden to perform spoofing of GNSS signals, this is regarded as malicious behavior as it could interfere with critical systems.		

ID: TR7.1	Source: TLR7, TLR8	Attached milestone: All
Verification: -	Type: Technical Electronics	Compliance status: Aquired
The project group must aquire and use cables (SMA/MCX) for all testing regarding spoofing or jamming in public or private space, the only exception being when the tests are performed within the Faraday cage at USN Kongsberg.		

Appendix F

Risk Assessment

This appendix shows the main functionalities of the SDRs we considered, except for price, as they vary based on currency exchange rates, and availability and shipping time. The figure was gathered from: from https://www.crowdsupply.com/microphase-technology/antsdr-e200.

PROJECT
Risk Matrix

Drone Jamming Threat

For our bachelor project, we have identified risks and challenges we deem plausible and rated them in likelyhood, impact and a risk score by multiplying the likelyhood and impact together.

Kilde for template: https://templatelab.com/risk-matrix/

ID	RISK DESCRIPTION	Effect on project	Chance	Impact	Risk	Mitigation Strategy
101	The team work falls apart and our project progress stalls	Our product does not meet its requirements	2	5	**10**	Using SCRUM project method with short deadlines should visualize the progress in a better way, so we can identify if and where progress has halted in time to rectify the problem and devote resources accordingly.
102	Our understanding of the signal processing and emitting does not allow us to spoof the drone	Our deliverables does not meet the expectations	3	5	**15**	Divide the project into partial goals, where the first goal will be easier to accomplish, if we do not meet the end result, we can scale back our project and set a new end goal.
103	Challenging reserach draws out too much time, and our parts ordered does not arrive in time	Poor planning is revealed	3	5	**15**	Verify our component needs with our external supervisor early in the project timeline. Thus making sure we order the necessary components in time.
104	Our solution faulters in the design and does not encapsulate what the customer envisioned	Not delivering what we should shows poor scope	1	1	**1**	Confere with our stakeholders in iterative steps and have a two way communication with our derived requirements, sending them to our supervisor regularly.
105	Lacking communication between project members and stakeholders	Our solution is not what stakeholders ordered	2	3	**6**	Dedicate a contact person between the bachelor group and the stake holder for each sprint, and create a checklist for subjects we need to confere with our stakeholders.
106	Project cost nearing limit of funding/exceeding limit	Cost is one of the main focus of any functional company	2	5	**10**	We will create a budget with the components we need and send this to our stakeholder so that we can be sure our needs are not above the limits of the project.
107	Research taking too much time, individual expertise is not high enough	Calling for increased help from supervisor	2	3	**6**	
108	Disagreement internally in the group causes friction and fallout between members	Unprofessional behaviour is not good for grading	2	3	**6**	We have implemented several mechanisms for disagreements in order to have a way of resolving them should they appear. Since we have an even number of members, we have stated a dice toss game for resolving ties i.e.
109	Data loss in case of cloud storage bankruptcy or computer crash	Problems with delivering documents	2	2	**4**	All members of the group download and backup the files from the OneDrive cloud storage every friday end-of-work.
110	Components we are implementing does not meet their production standards, and are not suitable because of this.	Our faultering equipment is not able to jam the drone	1	4	**4**	Verify components when they arrive and file a verification protocol ensuring that our components are within specification.

Figure F.1: Our list of risks that were determined to be likely or have a high impact.

Appendix G

Testing Excel

This appendix shows the testing documentation we filled out in excel, before we decided to directly fill it out in LaTeX to save some overhead on later implementing it into the report.

Scenario Test ID	DR1.1 (derived requirement)	Status	Completed and verified	
Scenario Description	Using the equipment without power infrastructure	Priority	High	
Test Case ID	T1.1.1			
Pre-Condition	System must be connected to a power bank or laptop	Steps to Execute	1: Make sure the Micro-USB interface is fully seated, 2: The laptop has battery and is powered up 3: The program is loaded and ready to run	
Exp. Result	System operates as normal and sends out signals on the frequency determined by the software	Act. Result	Using the Ubuntu laptop and the signal analyzer we tested the system wired and found the HackRF sent out a 20MHz wide signal at 1.57548GHz	
Work done by/date	Created: HK/15.02.2024 Executed: HK/01.03.2024 Peer-reviewed: AM/01.03.2024	Changelog		
Comments	The delta central frequency between the signal we sent out, and the central frequency reported by the signal analyzer was 60kHz, this was unexpected, but could be down to measurement inaccuracies, or explained by the lackluster oscillator in the HackRF One which is 20ppm, meaning that at 1.57542 GHz, it could deviate by as much as 31.5kHz.			

Scenario Test ID	DR2.1 (derived requirement)	Status	Completed and verified	
Scenario Description	Testing portability of the system	Priority	High	
Test Case ID	T.2.1.1			
Pre-Condition	The system must be contained in its entirety in the backpack of one of the students in the bachelor group	Steps to Execute	1: Backpack must be present 2: The laptop and HackRF One must be placed inside 3: The weight of the backpack must be	
Exp. Result	We expected this to be feasible	Act. Result	It was feasible	
Work done by/date	Created: HK/02.03.2024 Executed: HK/10.03.2024 Peer-reviewed: ABS/10.03.2024	Changelog	Write potential changes to connected DR/TLR/US that necessitates changing test scenario	
Comments	The weight of the backpack was approximately 5 kilograms, and the system fit inside easily in its entirety.			

Scenario Test ID	TLR4	Status	Not completed	
Scenario Description	Spoofing	Priority		
Test Case ID	T4.1			
Pre-Condition		Steps to Execute		
Exp. Result		Act. Result		
Work done by/date	Created: INIT/dd.mm.yyyy Executed: INIT/dd.mm.yyyy Peer-reviewed: INIT/ss.mm.yyyy	Changelog	Write potential changes to connected DR/TLR/US that necessitates changing test scenario	
Comments	Write comments about execution, unexpected results, testing environment, constraints hindering test success etc.			

Scenario Test ID	DR3.2 (derived requirement)	Status	Completed and verified	
Scenario Description	Signal strength	Priority		
Test Case ID	T3.2.1			
Pre-Condition	System must be powered up and connected to measuring device	Steps to Execute	1: System must be powered up and ready to transmit signals to the measuring device 2: Start the transmission and read out the signal strength from the measuring device 3: Compare the strength of the signal to the strength of the GPS L1-signal	
Exp. Result	Expect to have a signal in the order of 50-80 dB higher than the original signal	Act. Result	Signal with amplifier enabled is -39 dBm, compared to the -140 dBm of the GPS-L1 signal	
Work done by/date	Created: HK/05.03.2024 Executed: HK, ABS/20.03.2024	Changelog	Write potential changes to connected DR/TLR/US that necessitates changing test scenario	
Comments	The signal strength is 100 dB stronger, meaning we do not need the amplification in order to reach our estimated range, we rather would need an attenuator to not exceed it by a factor of 10.			

Scenario Test ID	TLR3	Status	Not completed	
Scenario Description	Jamming	Priority		
Test Case ID	T3.1			
Pre-Condition		Steps to Execute		
Exp. Result		Act. Result		
Work done by/date	Created: INIT/dd.mm.yyyy Executed: INIT/dd.mm.yyyy Peer-reviewed: INIT/ss.mm.yyyy	Changelog	Write potential changes to connected DR/TLR/US that necessitates changing test scenario	
Comments	Write comments about execution, unexpected results, testing environment, constraints hindering test success etc.			

Scenario Test ID	TLR6	Status	Not completed	
Scenario Description	User Interface	Priority		
Test Case ID	T.6.1			
Pre-Condition		Steps to Execute		
Exp. Result		Act. Result		
Work done by/date	Created: INIT/dd.mm.yyyy Executed: INIT/dd.mm.yyyy Peer-reviewed: INIT/ss.mm.yyyy	Changelog	Write potential changes to connected DR/TLR/US that necessitates changing test scenario	
Comments	Write comments about execution, unexpected results, testing environment, constraints hindering test success etc.			

Figure G.1: Testing documentation in Excel

Appendix H

Test reports

The following pages show the test reports made for the tests performed to verify the Delirium system against its requirements.

1 Test Report: T1.1.1

Report conducted by:HK, ABS

Report written by:HK

Date:10.04.2024

1.1 Pre-condition:

System must be connected to a powerbank or laptop

1.2 Method:

- Make sure the USB-interface is fully seated
- The laptop/powerbank is charged and is powered up
- The program is loaded and ready to run

1.3 Hypothesis:

System operates as normal and sends out signals on the frequency determined by the software

1.4 Equipment used:

- Ubuntu Laptop with GNU Radio
- HackRF One
- USB cable
- Keysight CXA Signal Analyser

1.5 Results:

Using the Ubuntu laptop and the signal analyzer, we tested the system wired and found the HackRF sent out a 20MHz wide signal at 1.57548GHz. See fig.H.1.

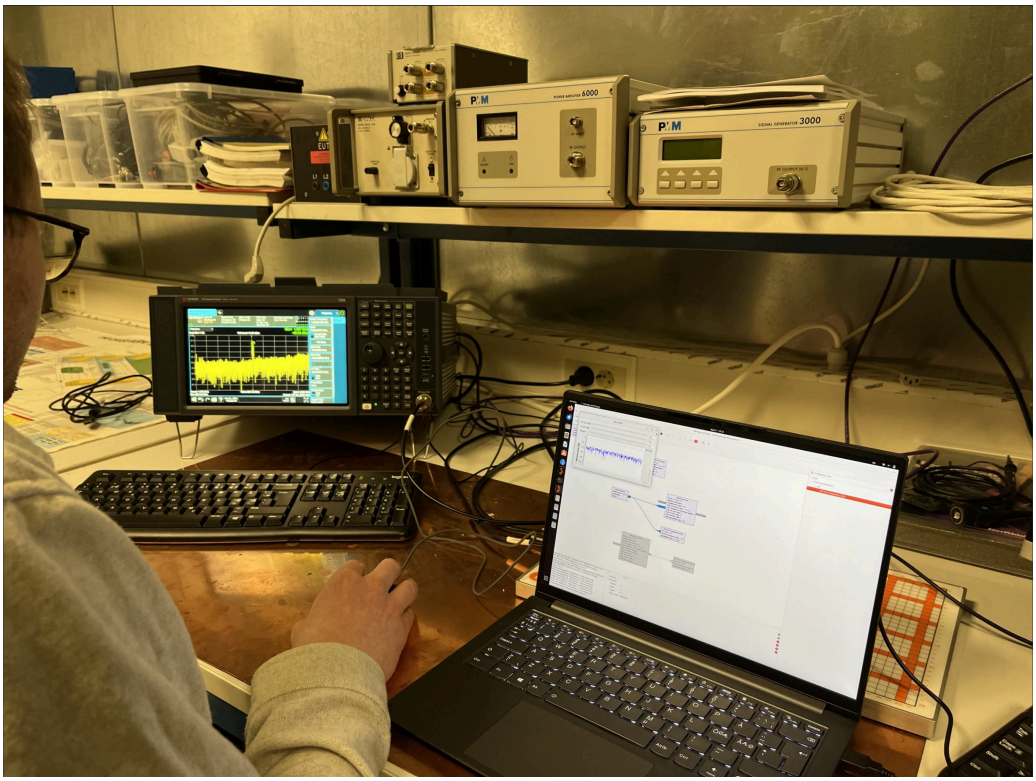


Figure H.1: CXA Keysight signal analyzer showing signal peak in middle of spectrum.

1.6 Conclusions:

The delta central frequency between the signal we sent out, and the central frequency reported by the signal analyzer was 60KHz, this was unexpected, but could be down to measurement inaccuracies. The other reason could be the lackluster oscillator in the HackRF One which is 20ppm, meaning that at 1.57542GHz it could deviate by as much as 31.5KHz.

2 Test Report: T2.1.1

Report conducted by:HK

Report written by:HK

Date:10.03.2024

2.1 Pre-condition:

System must be contained in its entirety in the backpack of one of the students in the bachelor group

2.2 Method:

- Backpack must be present
- The laptop and HackRF One must be placed inside
- The weight of the backpack must be less than 20 kg.

2.3 Hypothesis:

System operates as normal and sends out signals on the frequency determined by the software

2.4 Equipment used:

- Ubuntu Laptop with GNU Radio
- HackRF One
- USB cable

2.5 Results:

The system fit inside the backpack easily and the weight was approximately 5 kg.

2.6 Conclusions:

The portability of the system has been proven.

3 Test Report: T3.1

Report conducted by:HK, SN, AM, ABS

Report written by:HK

Date:01.03.2024

3.1 Pre-condition:

System must be connected to a powerbank or laptop.

3.2 Method:

- Make sure the USB-interface is fully seated
- The laptop/powerbank is charged and is powered up
- The program is loaded and ready to run

3.3 Hypothesis:

When we activate the system the GPS-reception in the receiver is disrupted and the GPS lock is disabled.

3.4 Equipment used:

- Ubuntu Laptop with GNU Radio
- HackRF One
- USB cable
- Navio2 Flight Controller with ucenter open and monitoringing enabled

3.5 Results:

Using the Ubuntu laptop and the HackRF One without the external oscillator. We tested outside with the HackRF One being connected to the receiver with the GNSS-antenna via an SMA T-connector. Then we waited until the receiver had good GPS-lock and activated our protocol aware jammer. Within half a second the GPS reception was totally blocked, and the receiver lost its position bearing.

3.6 Conclusions:

We saw that our solution succsessfully blocks the GPS-signals from reaching the receiver when we activate our protocol aware jammer.

4 Test Report: T3.2.1

Report conducted by:HK

Report written by:HK

Date:10.03.2024

4.1 Pre-condition:

System must be powered up and connected to measuring device.

4.2 Method:

- System must be powered up and ready to transmit signals to the measuring device
- Start the transmission and read out the signal strength from the measuring device
- Compare the strength of the signal to the strength of the GPS L2-signal

4.3 Hypothesis:

Expected signal strength is 50-60 dB higher than the original signal.

4.4 Equipment used:

- Ubuntu Laptop with GNU Radio
- HackRF One
- USB cable
- CXA Signal Analyser

4.5 Results:

The signal strength with the amplifier enabled is -39 dBm, compared to the -140 dBm of the GPS-L1 signal.

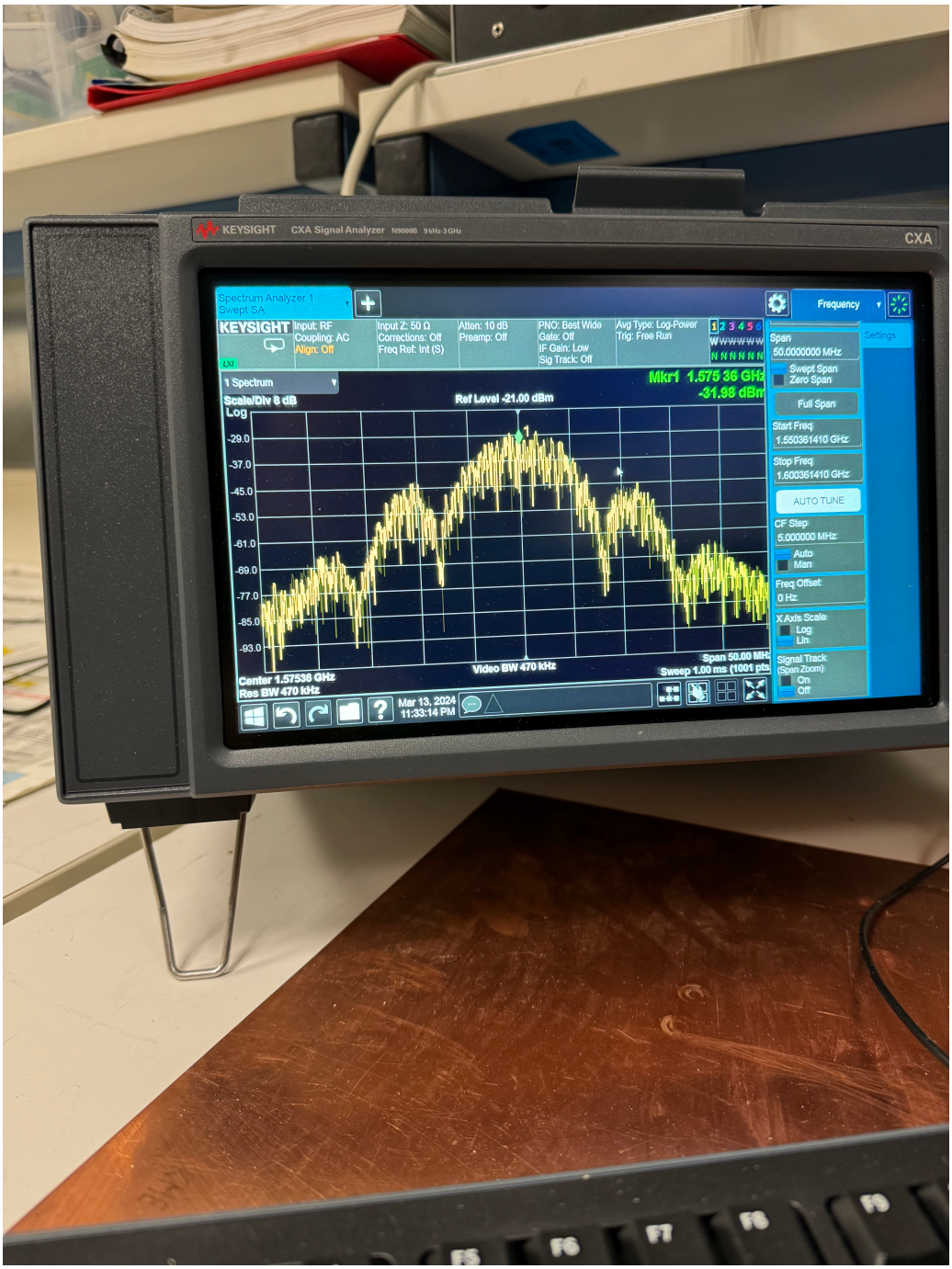


Figure H.2: CXA Keysight signal analyzer showing signal peak in middle of spectrum.

4.6 Conclusions:

The signal strength is 100 dB stronger, meaning we do not need the amplification in order to reach our estimated range, we rather would need an attenuator to not exceed it by a factor of 10.

5 Test Report: T4.1

Report conducted by: HK, AM, SN, ABS

Report written by:HK

Date:01.05.2024

5.1 Pre-condition:

System must be connected to a powerbank or laptop, test must be conducted either cabled or inside a Faraday cage.

5.2 Method:

- Make sure the USB-interface is fully seated
- The laptop/powerbank is charged and is powered up
- The program is loaded and ready to run
- Only start the spoofer once the door to the Faraday cage is fully sealed

5.3 Hypothesis:

When we activate the system the receiver starts locating our faux GPS satellites and eventually locks on to them.

5.4 Equipment used:

- Ubuntu Laptop with GNU Radio
- HackRF One
- USB cable
- Faraday cage
- Navio2 Flight Controller with ucenter open and monitoringing enabled

5.5 Results:

Using the Ubuntu laptop and the HackRF One with the external oscillator. We tested with the HackRF One transmitting to the receiver with the GNSS-antenna via the ANT500-antenna. Then we saw the satellites pop up in ucenter, but were not accepted due to the mismatch in time/date and location of the cached data within the receiver. After two minutes the our signal

was accepted as genuine and we saw the receiver change its time and date in ucenter. It then displayed the time and date parameters we designated in the spoofing .bin-file.

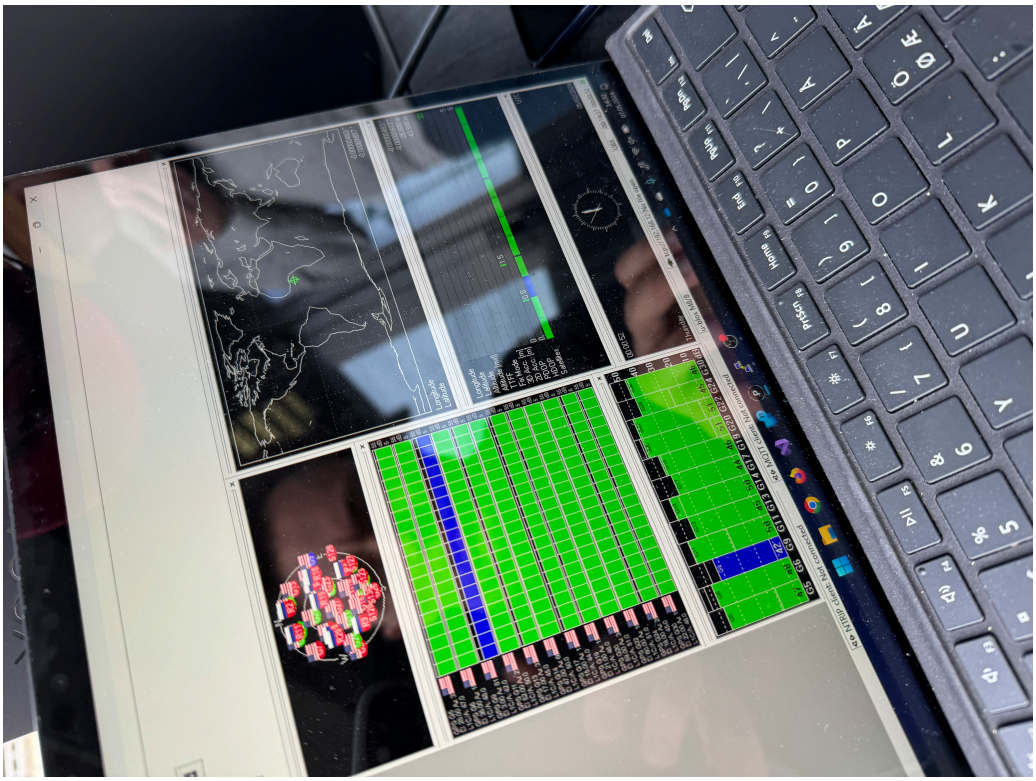


Figure H.3: Picture showing ucenter where the receiver believes it is in the middle of the ocean.

5.6 Conclusions:

When the receiver was hot started (meaning it retains the cached data about its location and time from the previous GNSS-lock) the time it took for our signal to be perceived as authentic was longer than when we did a cold start of the receiver. This backs up the claim that a spoofer with accurate time and location of the receiver will be locked onto faster, and can then diverge from the real signal slowly, to invalidate the real signal.

6 Test Report: T6.1 - GUI/API Integration test

Conducted by: HK, AM, ABS, SN

Report written by: AM

Date: 15.04.2024

6.1 Hypothesis:

Delirium complies with the milestone 3 description (see appendix: C).

6.2 Equipment used:

- One of USN's Oscilloscope's (KEYSIGHT CXA Signal Analyzer)
- Our test-computer (Lenovo laptop running Ubuntu)
- Two HackRF One's and their respective USB cables
- An SMA male-to-male cable

6.3 Method:

We connected one HackRF One to the laptop using USB (for power and data transfer), then connected the SMA cable to the HackRF and the oscilloscope. The GUI was then opened and each SDR started then stopped in turn. All radios were tested, including the spoofing-sdr. We have bound each radio to one of the HackRF's serial numbers, the jamming radios are run on one, the spoofer is run on the other. This is why we needed both devices.

6.4 Results:

Each SDR started when told to, and quit running when told to, we observed this clearly on the oscilloscope. We observed that the oscilloscope produced the expected results for each SDR in terms of bandwidth, centre frequency and gain.

6.5 Conclusions:

The test proves that we have accomplished our goals for milestone 3, meaning a successful integration of advanced jamming features and that the GUI-API relationship is working.

7 Test Report: Multiple Radios in Parallel

Report conducted by: AM

Report written by: AM

Date: 24.04.2024

7.1 Hypothesis:

It is possible to run both our spoofing radio and any of our jamming radios in parallel with Delirium

8 Equipment used:

- One of USN's Oscilloscope's (KEYSIGHT CXA Signal Analyzer)
- Our test-computer (Lenovo laptop running Ubuntu)
- Two HackRF One's and their respective USB cables
- USB hub and two USB cables
- SMA T-connector and cables connecting both HackRF One's to the oscilloscope
- Python test-scripts

8.1 Method:

Both HackRF's were connected to the oscilloscope via SMA and to the laptop via the USB hub. The test-scripts were then run in sequence. The test-scripts tested the running of our sweep-jamming radio and spoofing radio in parallel, spoofing radio on its own, sweep-jamming on its own, and finally barrage-jamming and spoofing in parallel.

8.2 Results:

Sweep-jamming by itself: When start was called, the radio started jamming at the right centre frequency. The sweeping did not happen, and no errors occurred, so it seemed as if the sweeping-method was not run at all. The assumption was made that the GIL (Global Interpreter Lock) present in the Python language was keeping the method from executing. A hot-fix was attempted using the "threading.Thread" and "threading.timer" modules without success.

Spoofing by itself: Works exactly as intended. Output from the generation of the .bin-file is shown in the terminal and we could verify on the oscilloscope that the underlying radio is

transmitting at the right centre frequency when started. It also stops when told to, as could be observed on the oscilloscope

Spoofing and barrage jamming at the same time: Works exactly as intended. Both radios were started then stopped individually before they were run at the same time. When run at the same time it is clear that the barrage-jamming is interfering with the spoofing, meaning that both radios must be active.

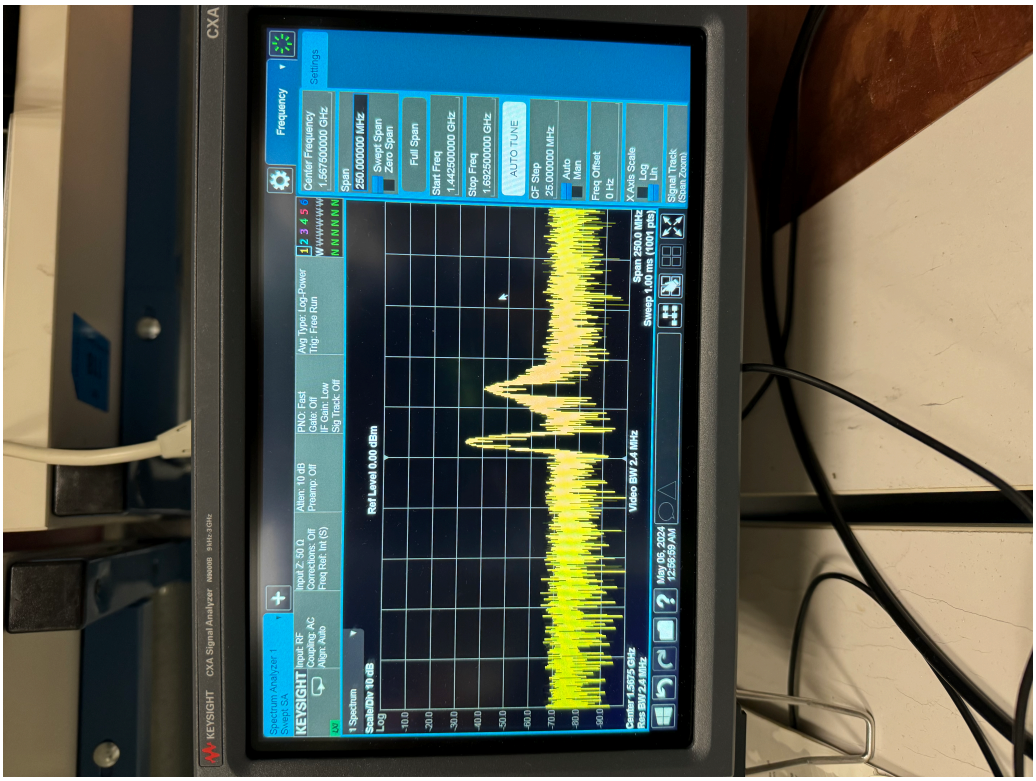


Figure H.4: CXA Keysight signal analyzer showing signal peak of both sweep jamming and spoofing signals.

8.3 Conclusions:

This test was crucial for understanding Delirium's needs regarding scheduling. GNU Radio is implemented in C++ with a Python-wrapper on top for interfacing with the underlying radios. The radios are created in C++ and threaded individually, so running two radios at the same time should not be - and has been proven here to not be - an issue. The issues faced here are probably the results of Pythons GIL which does not allow for actual C-like threading to occur. The sweep-functionality is implemented in Python, and thus will not be run.

Appendix I

SDR Components

This appendix shows the main functionalities of the SDRs we considered, except for price, as they vary based on currency exchange rates, and availability and shipping time. The figure was gathered from: from https://www.crowdsupply.com/microphase-technology/antsdr-e200.

Comparisons

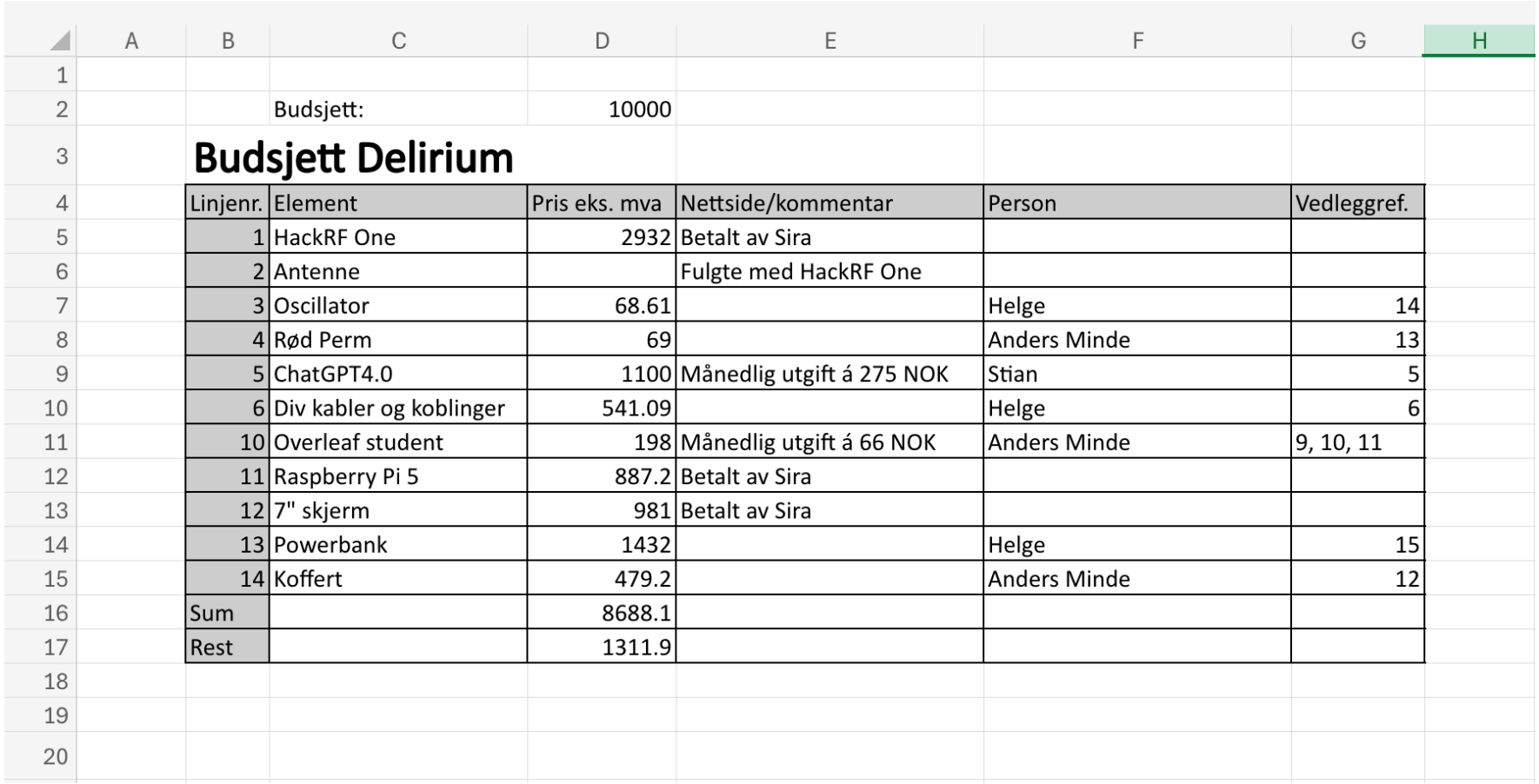
	AntSDR E200 AD9363	AntSDR E200 AD9361	RTL-SDR	PlutoSDR	Ettus B205mini	LimeSDR mini 2.0	HackRF One
Chipset	AD9363	AD9361	RTL2832U	AD9363	AD9364	LMS7002M	MAX5864, MAX2837, RFFC5072
Frequency Range	325 MHz - 3.8 GHz	70 MHz - 6 GHz	500 KHz - 1766 MHz	325 MHz - 3.8 GHz	70 MHz - 6 GHz	10 MHz - 3.5 GHz	1 MHz - 6 GHz
Interface	Gigabit Ethernet	Gigabit Ethernet	USB 2.0	USB 2.0	USB 3.0	USB 3.0	USB 2.0
Embedded	Yes	Yes	No	Yes	No	No	Yes
RF Bandwidth	20 MHz	56 MHz	3.2 MHz	20 MHz	61.44 MHz	40 MHz	20 MHz
Sample Depth	12 bits	12 bits	8 bits	12 bits	12 bits	12 bits	8 bits
Sample Rate	61.44 MSPS	61.44 MSPS	3.2 MSPS	61.44 MSPS	61.44 MSPS	30.72 MSPS	20 MSPS
Transmitter Channels	2	2	0	2	1	1	1
Receivers	2	2	1	2	1	1	1
Duplex	Full	Full	No	Full	Full	Full	Half
Programmable Logic Gates	85k	85k	N/A	28k	150k	44k	64 macrocell CPLD
Open Source	Schematic & firmware	Schematic & firmware	No	Full	Schematic & firmware	Full	Full
Oscillator Precision	+/-2 ppm	+/-2 ppm	+/-1 ppm	+/-20 ppm	+/-2 ppm	+/-1 ppm initial, +/-4 ppm stable	+/-20 ppm
Transmit Power	Up to 10 dBm (depending on frequency)	Up to 10 dBm (depending on frequency)	N/A	Up to 6 dBm (depending on frequency)	10 dBm+	Up to 10 dBm (depending on frequency)	-10 dBm+ (15 dBm @ 2.4 GHz)

Figure I.1: Comparison of different hardware solutions

Appendix J

Delirium Budget

This appendix shows the budget of Delirium, and the costs associated with acquiring the tools needed to build Delirium. Costs were calculated without VAT. The budget was made using Microsoft Excel. Made by HK.



Budsjett: 10000

Budsjett Delirium

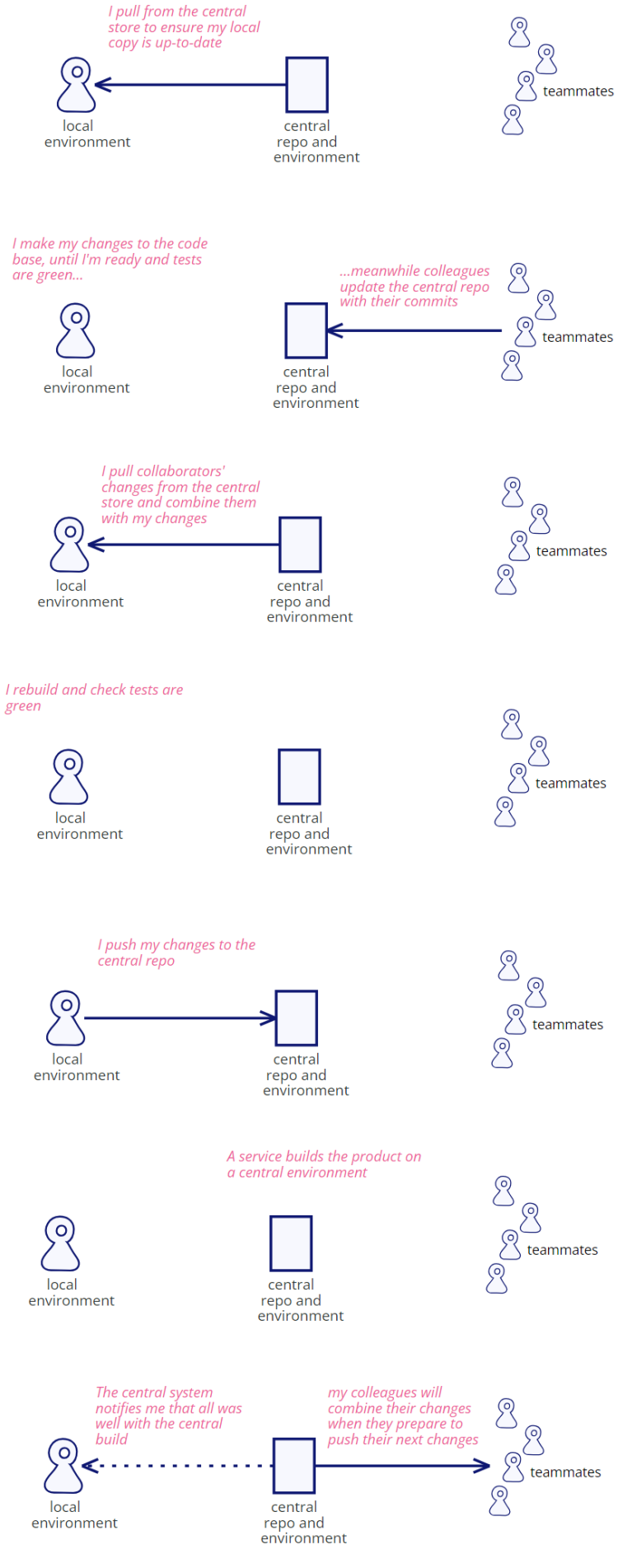
Linjenr.	Element	Pris eks. mva	Nettside/kommentar	Person	Vedleggref.
1	HackRF One	2932	Betalt av Sira		
2	Antenne		Fulgte med HackRF One		
3	Oscillator	68.61		Helge	14
4	Rød Perm	69		Anders Minde	13
5	ChatGPT4.0	1100	Månedlig utgift á 275 NOK	Stian	5
6	Div kabler og koblinger	541.09		Helge	6
10	Overleaf student	198	Månedlig utgift á 66 NOK	Anders Minde	9, 10, 11
11	Raspberry Pi 5	887.2	Betalt av Sira		
12	7" skjerm	981	Betalt av Sira		
13	Powerbank	1432		Helge	15
14	Koffert	479.2		Anders Minde	12
Sum		8688.1			
Rest		1311.9			

Figure J.1: Delirium budget

Appendix K

Martin Fowler's CI Model

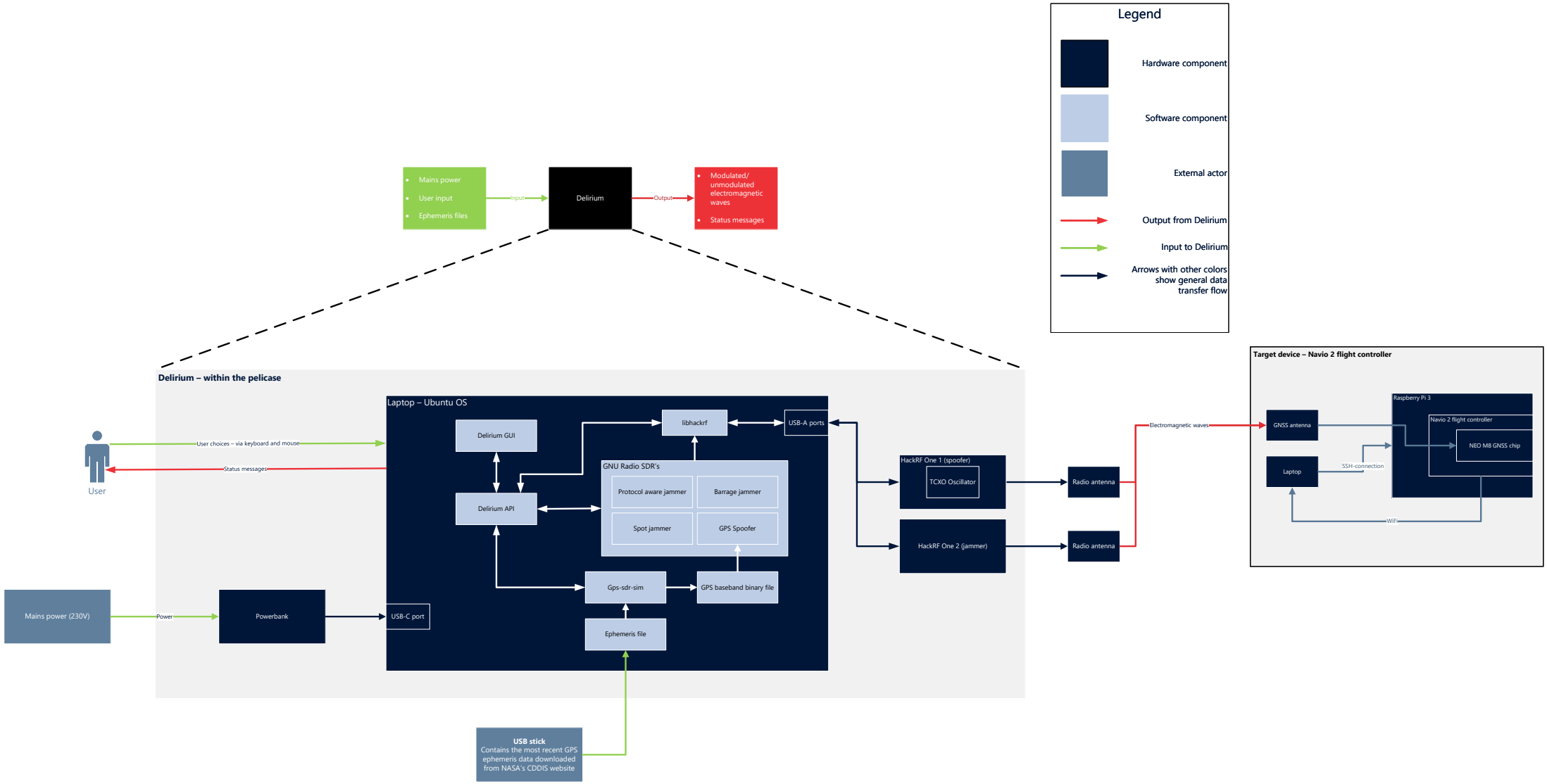
The following is a collage made by AM containing figures used by Martin Fowler on his web-page to describe a simple Continuous Integration (CI) model [RefH12].


I pull from the central store to ensure my local copy is up-to-date
local environment
central repo and environment
teammates
I make my changes to the code base, until I'm ready and tests are green...
...meanwhile colleagues update the central repo with their commits
local environment
central repo and environment
teammates
I pull collaborators' changes from the central store and combine them with my changes
local environment
central repo and environment
teammates
I rebuild and check tests are green
local environment
central repo and environment
teammates
I push my changes to the central repo
local environment
central repo and environment
teammates
A service builds the product on a central environment
local environment
central repo and environment
teammates
The central system notifies me that all was well with the central build
my colleagues will combine their changes when they prepare to push their next changes
local environment
central repo and environment
teammates

Appendix L

Delirium Diagram; Hardware & Software

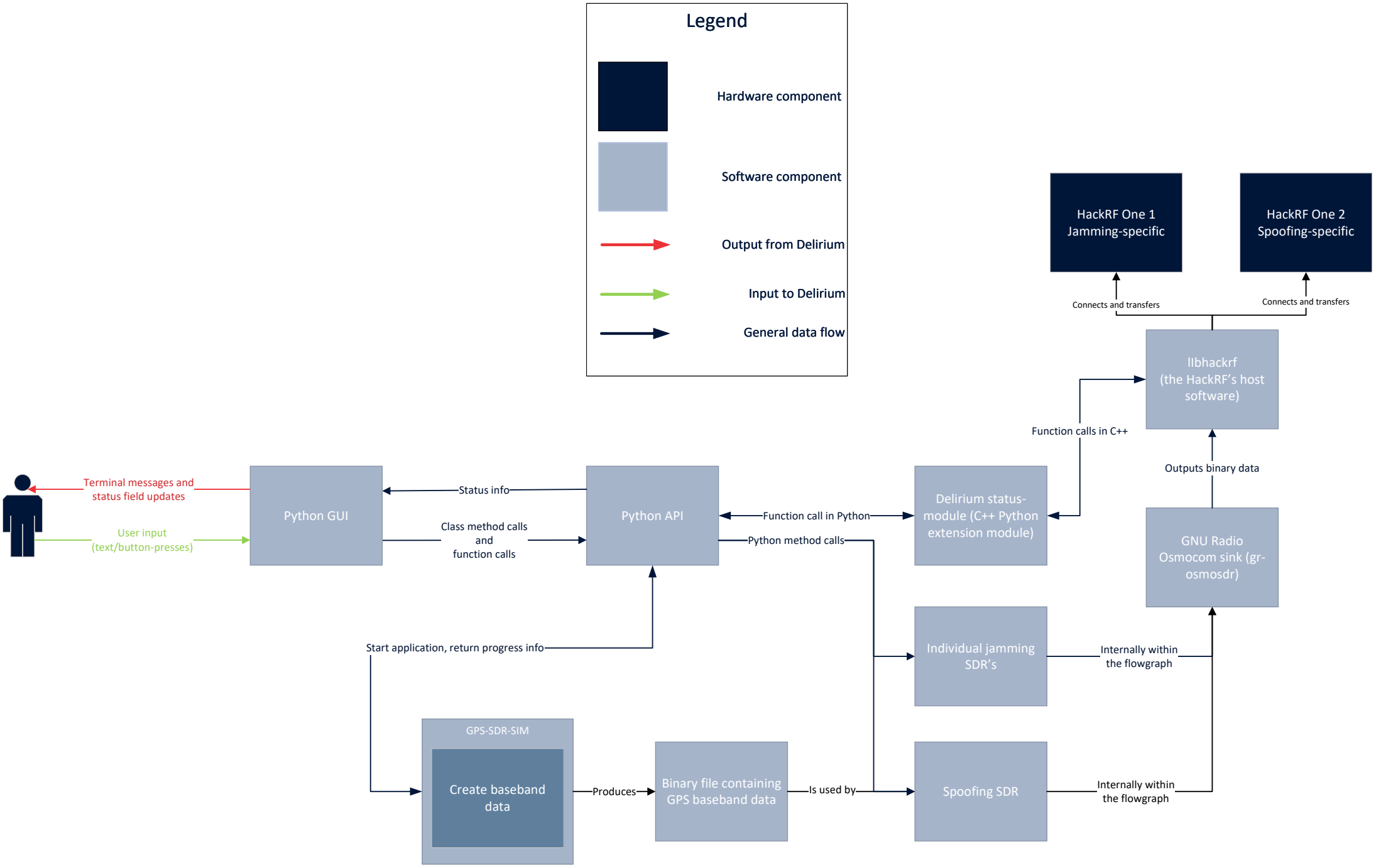
The following diagram is the result of our tentative system decomposition. It was used as a reference for keeping track of all system components, and can be used by the reader for the same purpose. The diagram shows the system schematically in its final state. Created by AM.


Legend
Hardware component
Software component
External actor
Output from Delirium
Input to Delirium
Arrows with other colors show general data transfer flow
Mains power
User input
Ephemeris files
Input
Delirium
Output
Modulated/ unmodulated electromagnetic waves
Status messages
Delirium – within the pelicase
Laptop – Ubuntu OS
Delirium GUI
libhackrf
USB-A ports
User choices – via keyboard and mouse
Status messages
User
GNU Radio SDR's
Protocol aware jammer
Barrage jammer
Spot jammer
GPS Spoofer
Delirium API
Gps-sdr-sim
GPS baseband binary file
Ephemeris file
Mains power (230V)
Power
Powerbank
USB-C port
USB stick
Contains the most recent GPS ephemeris data downloaded from NASA's CDDIS website
HackRF One 1 (spoofer)
TCXO Oscillator
HackRF One 2 (jammer)
Radio antenna
Radio antenna
Electromagnetic waves
Target device – Navio 2 flight controller
GNSS antenna
Raspberry Pi 3
Navio 2 flight controller
NEO M8 GNSS chip
Laptop
SSH-connection
WiFi

Appendix M

Software Architecture

This diagram shows Deliriums software architecture on a high level of abstraction. Its purpose is to serve as a reference for the the logical coherency between the software components in the system. The diagram was developed by AM.


Legend
Hardware component
Software component
Output from Delirium
Input to Delirium
General data flow
HackRF One 1
Jamming-specific
HackRF One 2
Spoofing-specific
Connects and transfers
Connects and transfers
libhackrf
(the HackRF's host
software)
Function calls in C++
Outputs binary data
Terminal messages and
status field updates
User input
(text/button-presses)
Python GUI
Status info
Class method calls
and
function calls
Python API
Function call in Python
Python method calls
Delirium status-
module (C++ Python
extension module)
GNU Radio
Osmocom sink (gr-
osmosdr)
Start application, return progress info
Individual jamming
SDR's
Internally within
the flowgraph
GPS-SDR-SIM
Create baseband
data
Produces
Binary file containing
GPS baseband data
Is used by
Spoofing SDR
Internally within
the flowgraph

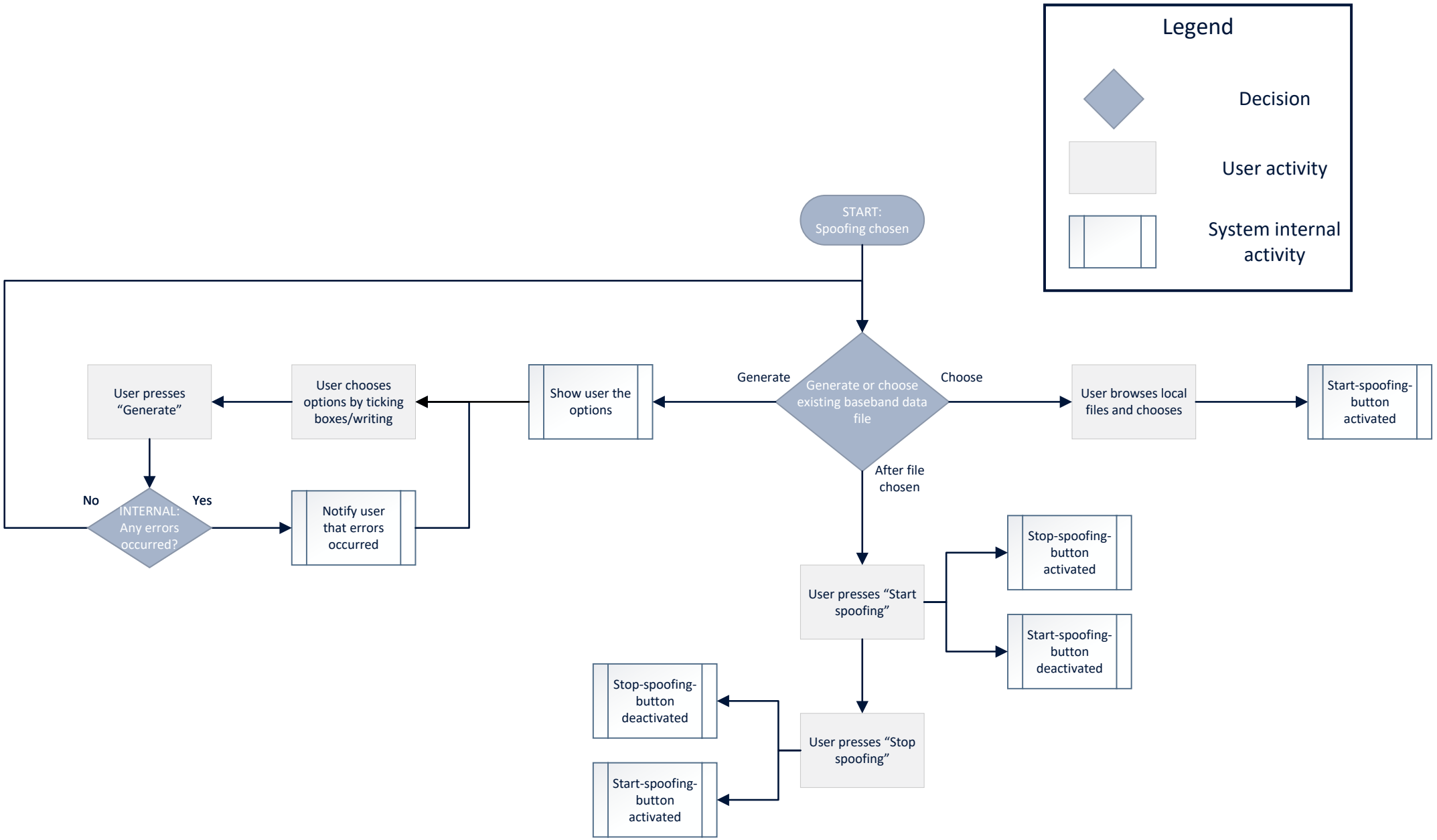
Appendix N

Delirium UML Diagrams

The following diagrams show different aspects of the software architecture of Delirium from different perspectives. Their intention is to provide context for the implementation, both as a guideline for the developers and for documentation purposes.

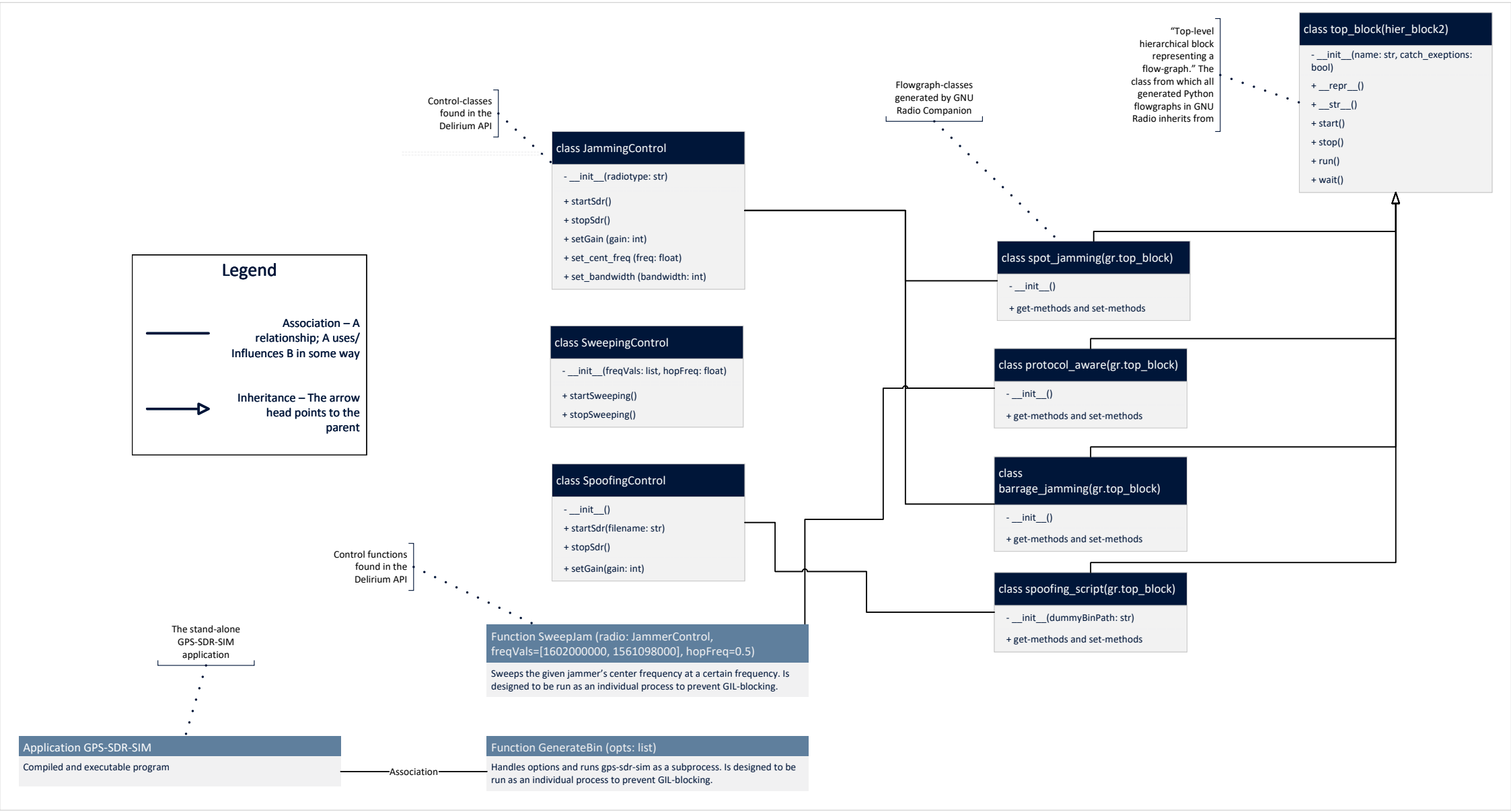
9 Activity Diagram - GUI

The following UML activity diagram shows the behavior of the Delirium GUI after the user has chosen spoofing. It was made during the planning phase before the implementation took place. Made by AM.


Legend
Decision
User activity
System internal activity
START: Spoofing chosen
Generate or choose existing baseband data file
Generate
Choose
Show user the options
User chooses options by ticking boxes/writing
User presses "Generate"
INTERNAL: Any errors occurred?
No
Yes
Notify user that errors occurred
User browses local files and chooses
Start-spoofing-button activated
After file chosen
User presses "Start spoofing"
Stop-spoofing-button activated
Start-spoofing-button deactivated
User presses "Stop spoofing"
Stop-spoofing-button deactivated
Start-spoofing-button activated

10 Class Diagram - API

The following is a UML class diagram which shows the functions/types provided by the Delirium API and how it associates with other software components within the system.


Control-classes found in the Delirium API
class JammingControl
- __init__(radiotype: str)
+ startSdr()
+ stopSdr()
+ setGain (gain: int)
+ set_cent_freq (freq: float)
+ set_bandwidth (bandwidth: int)
class SweepingControl
- __init__(freqVals: list, hopFreq: float)
+ startSweeping()
+ stopSweeping()
class SpoofingControl
- __init__()
+ startSdr(filename: str)
+ stopSdr()
+ setGain(gain: int)
Control functions found in the Delirium API
Flowgraph-classes generated by GNU Radio Companion
"Top-level hierarchical block representing a flow-graph." The class from which all generated Python flowgraphs in GNU Radio inherits from
class top_block(hier_block2)
- __init__(name: str, catch_exeptions: bool)
+ __repr__()
+ __str__()
+ start()
+ stop()
+ run()
+ wait()
class spot_jamming(gr.top_block)
- __init__()
+ get-methods and set-methods
class protocol_aware(gr.top_block)
- __init__()
+ get-methods and set-methods
class barrage_jamming(gr.top_block)
- __init__()
+ get-methods and set-methods
class spoofing_script(gr.top_block)
- __init__(dummyBinPath: str)
+ get-methods and set-methods
Legend
Association – A relationship; A uses/ Influences B in some way
Inheritance – The arrow head points to the parent
Function SweepJam (radio: JammerControl, freqVals=[1602000000, 1561098000], hopFreq=0.5)
Sweeps the given jammer's center frequency at a certain frequency. Is designed to be run as an individual process to prevent GIL-blocking.
The stand-alone GPS-SDR-SIM application
Application GPS-SDR-SIM
Compiled and executable program
Association
Function GenerateBin (opts: list)
Handles options and runs gps-sdr-sim as a subprocess. Is designed to be run as an individual process to prevent GIL-blocking.

11 Use Case Diagrams

The following are UML use case diagrams which were developed to supplement our user stories and to gain insight before implementation began.

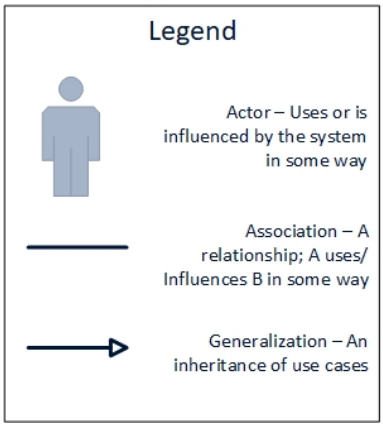


Figure N.1: Legend explaining the Delirium use case notation

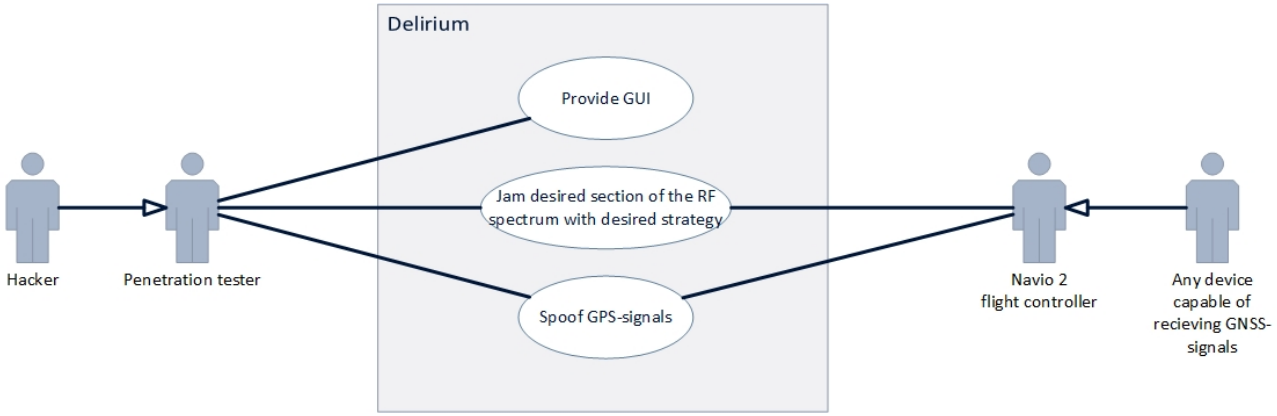


Figure N.2: Use case diagram for Delirium as a whole. Created by AM

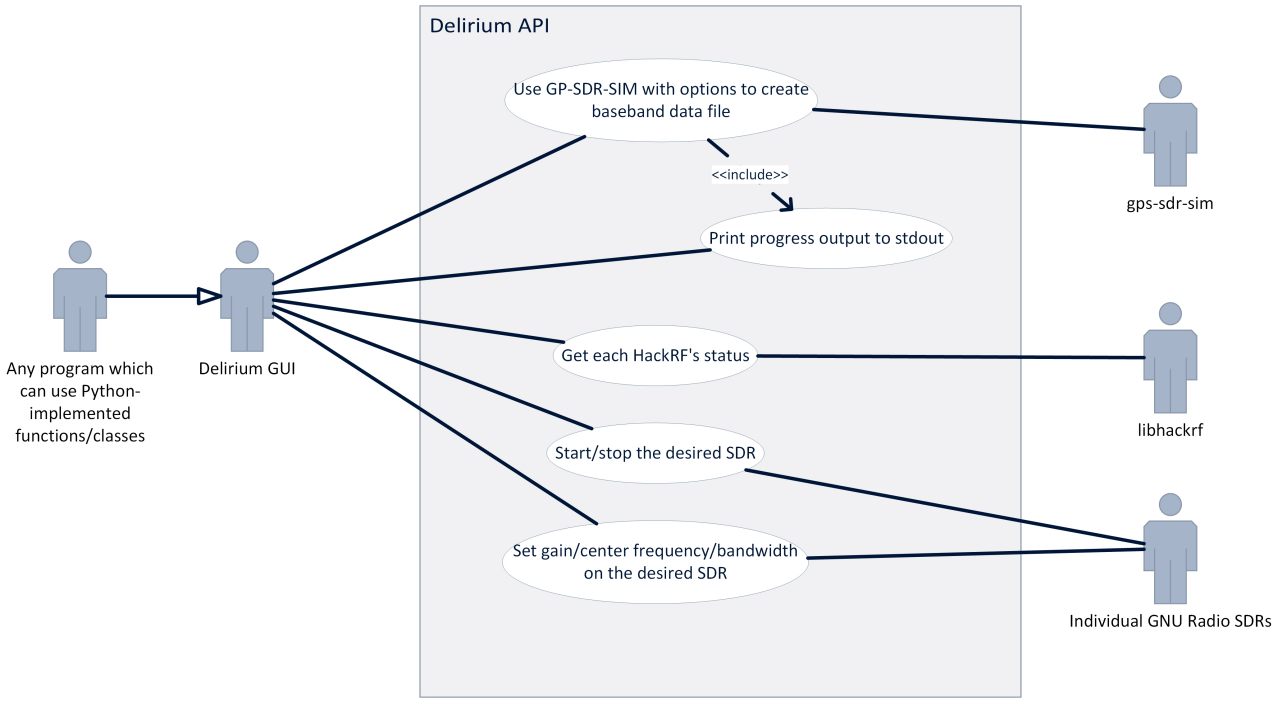


Figure N.3: Use case diagram for the Delirium API. Created by AM

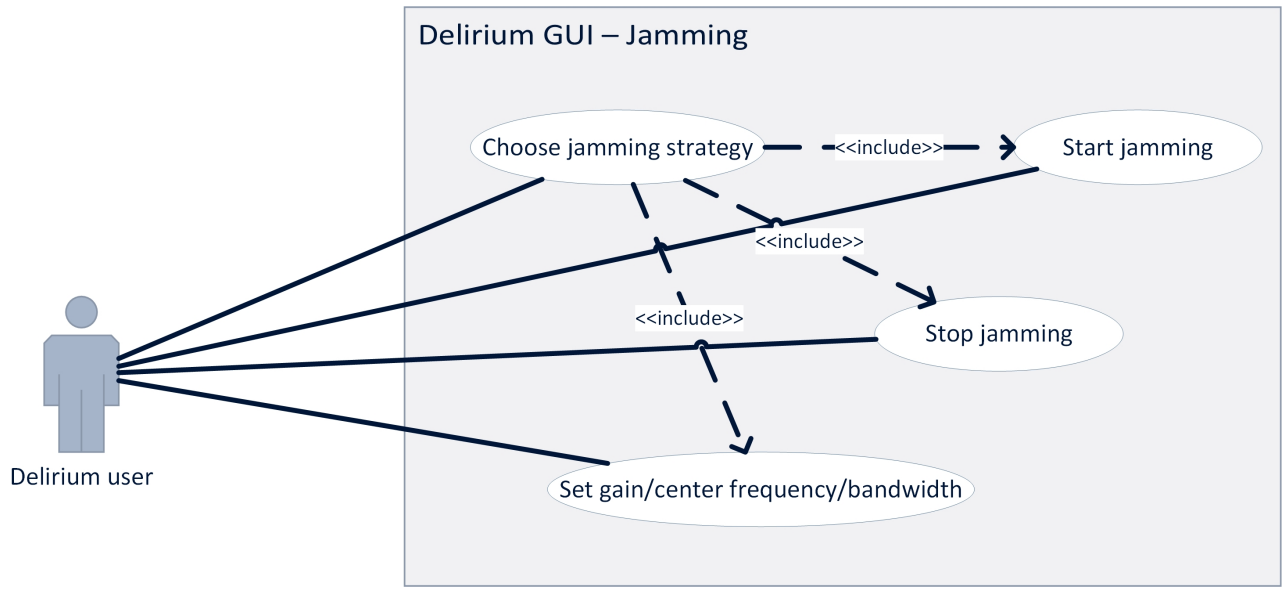
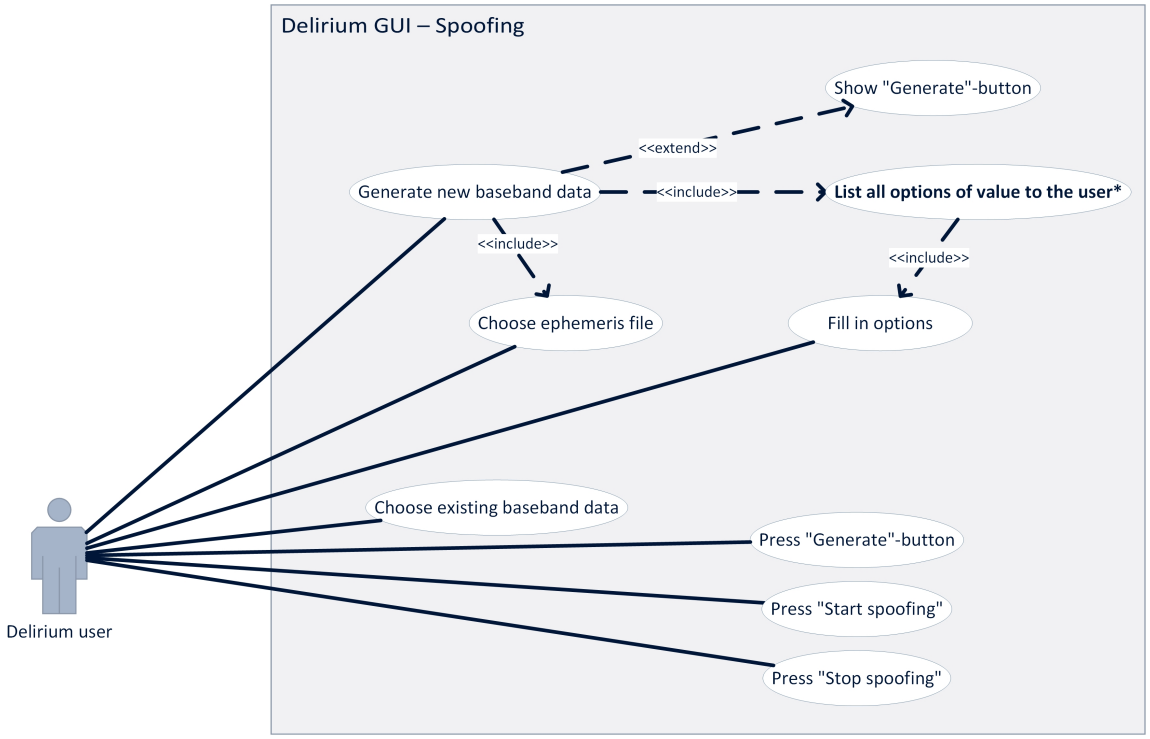


Figure N.4: Use case diagram for the Delirium GUI when jamming is chosen. Created by AM



***Delirium must allow the user to specify the following options:**

```
#   -e <gps_nav>     Ephemeris bin-file
#   -l <location>    Lat,Lon,Hgt (static mode) e.g. 30.286502,120.032669,100
#   -t <date,time>   Scenario start time YYYY/MM/DD,hh:mm:ss
#   -T <date,time>   Overwrite TOC and TOE to scenario start time
#   -d <duration>    Duration [sec] (dynamic mode max: 300 static mode max: 86400)
#   -o <output>      I/Q sampling data file (default: gpssim.bin ; use - for stdout)
```

Figure N.5: Use case diagram for the Delirium GUI when spoofing is chosen. Created by AM

12 Sequence Diagrams

The following are UML sequence diagrams which show the behavior of Delirium in certain scenarios. The legend for these diagrams can be seen in figure N.6.

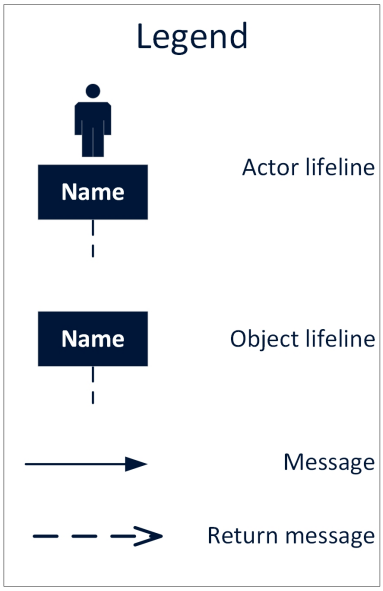
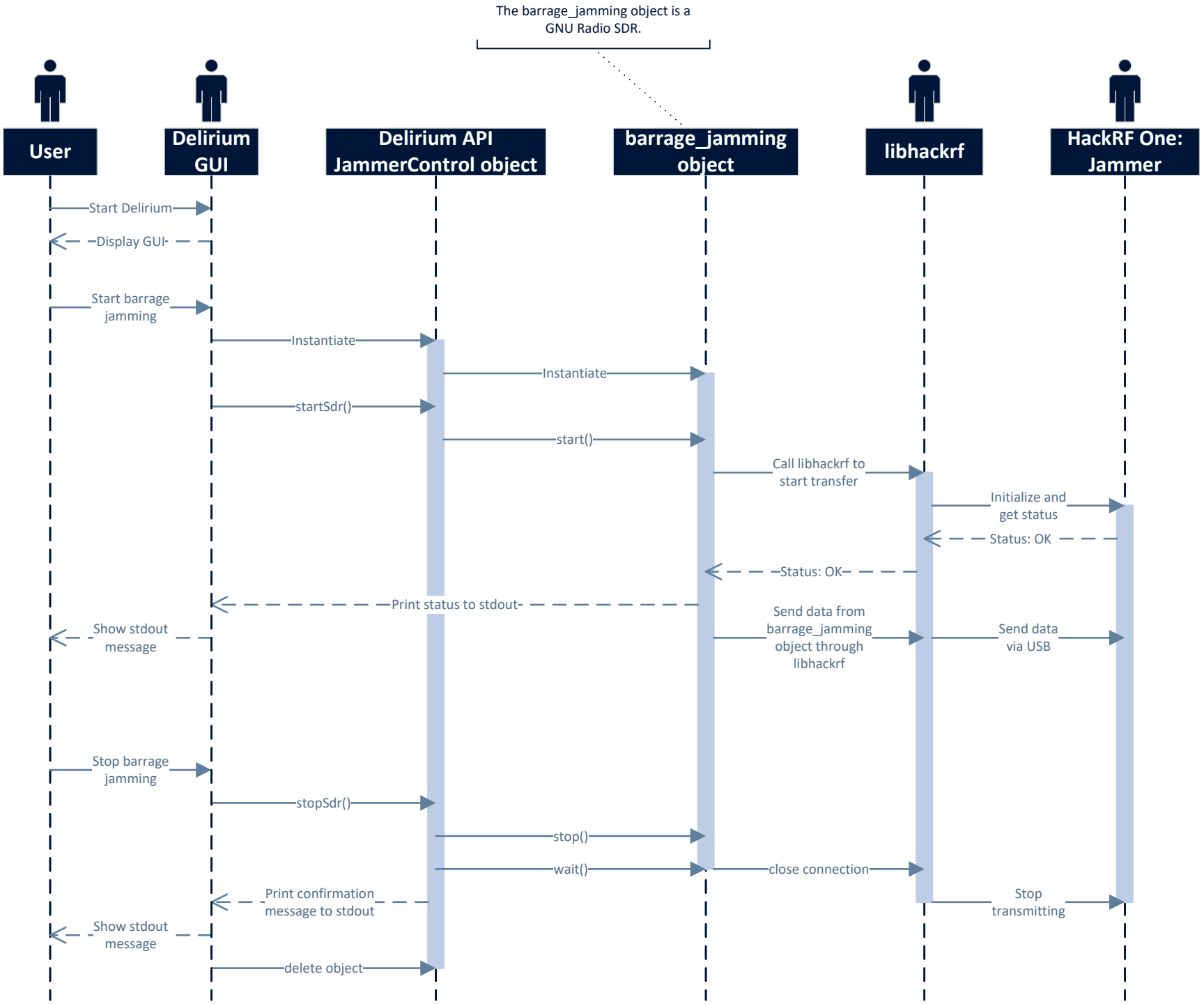


Figure N.6: Legend for the Delirium sequence diagrams, created by AM.

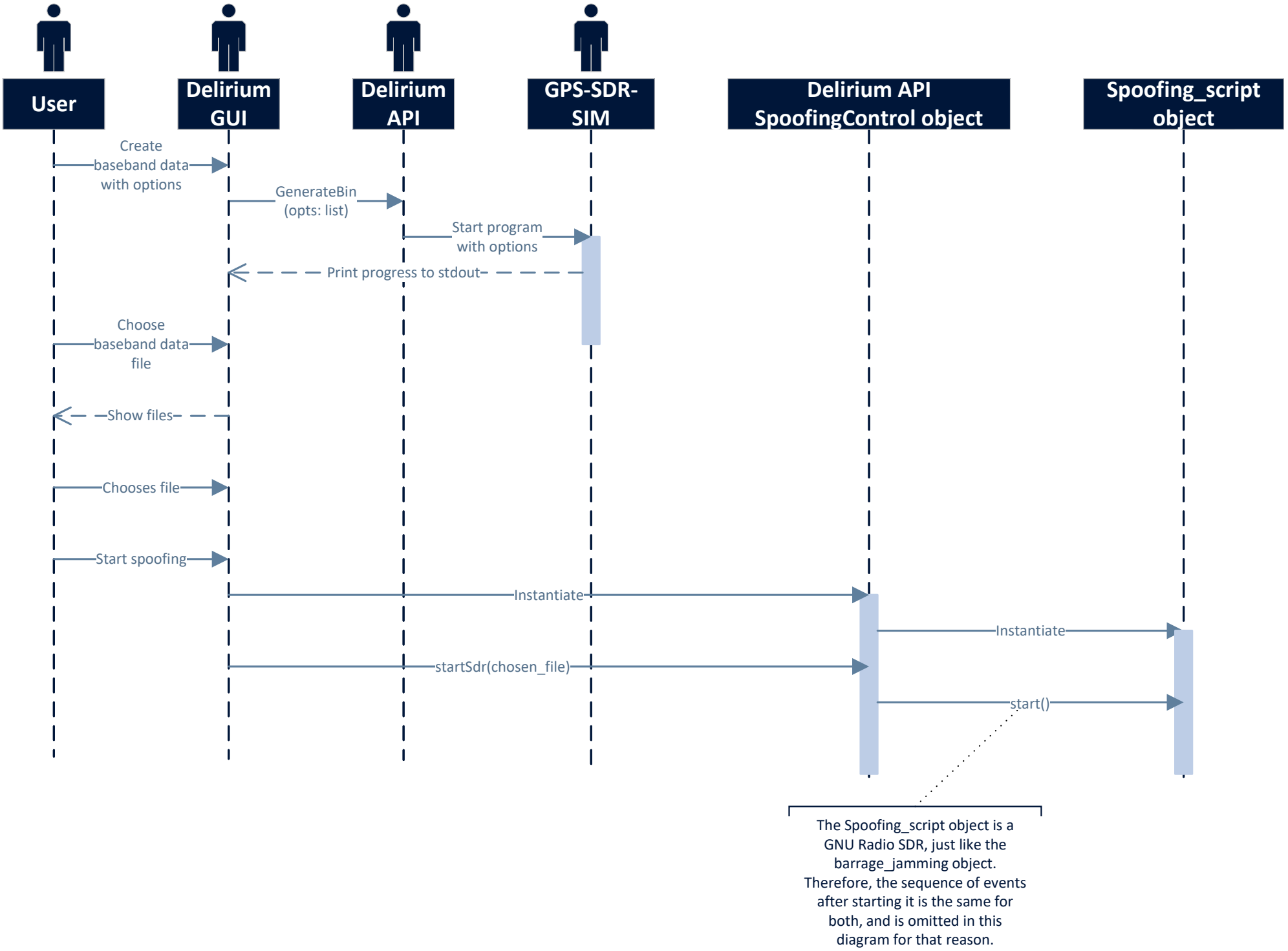
12.1 Sequence Diagram - Barrage Jamming

The following diagram shows the sequence of events from when a user starts Delirium, then starts a barrage jammer, and then finally stops it. The sequence of events presented in it are the same for all jammers except for the sweep jammer, which is presented later. Created by AM.


The barrage_jamming object is a GNU Radio SDR.
User
Delirium GUI
Delirium API JammerControl object
barrage_jamming object
libhackrf
HackRF One: Jammer
Start Delirium
Display GUI
Start barrage jamming
Instantiate
Instantiate
startSdr()
start()
Call libhackrf to start transfer
Initialize and get status
Status: OK
Status: OK
Print status to stdout
Send data from barrage_jamming object through libhackrf
Send data via USB
Show stdout message
Stop barrage jamming
stopSdr()
stop()
wait()
close connection
Stop transmitting
Print confirmation message to stdout
Show stdout message
delete object

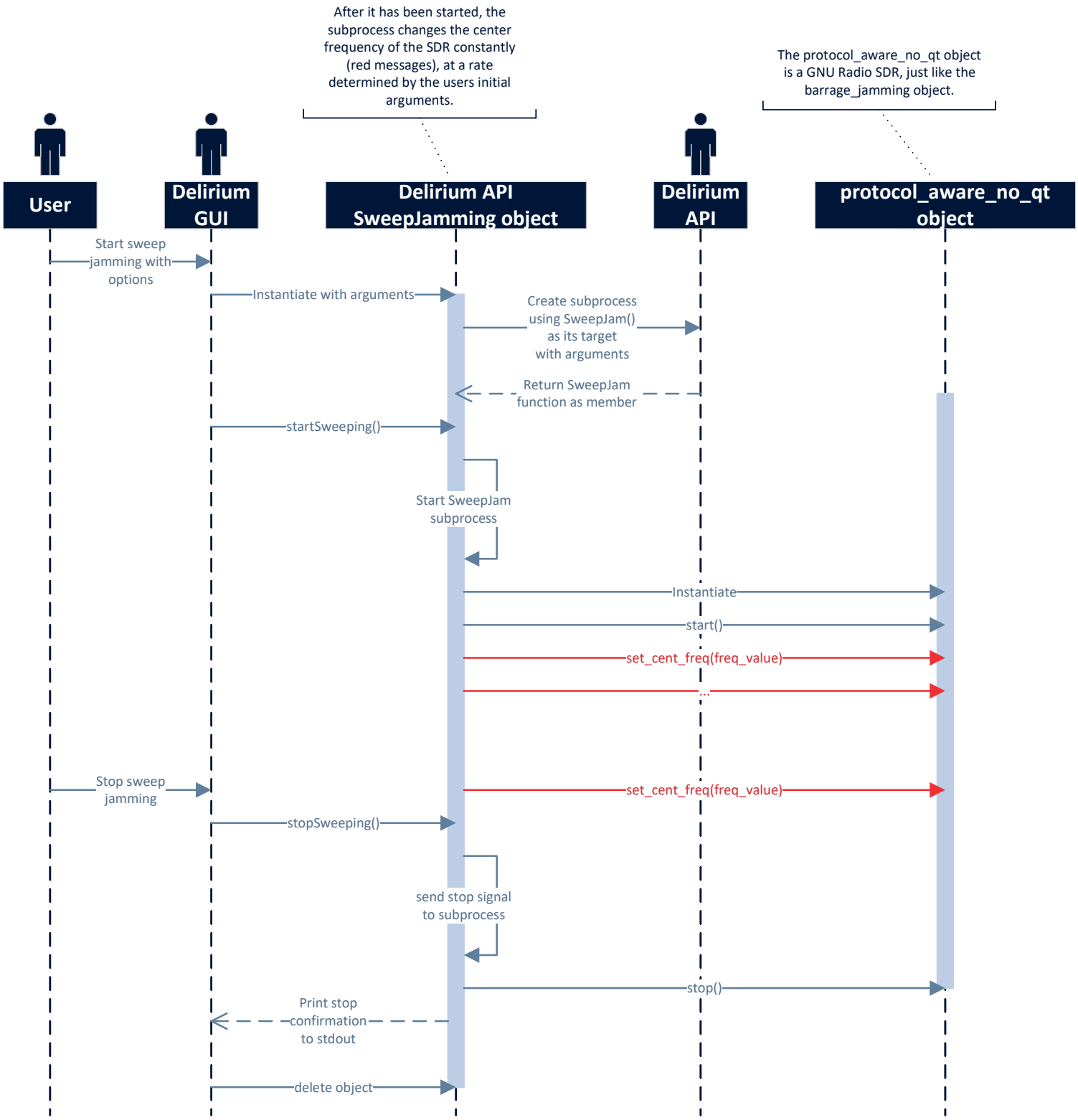
12.2 Sequence Diagram - Spoofing

The following diagram shows the sequence of events that must take place for the user to create a new GPS baseband data file using gps-sdr-sim (see section 15.3) and then start an SDR which broadcasts it. Created by AM.


User
Delirium GUI
Delirium API
GPS-SDR-SIM
Delirium API SpoofingControl object
Spoofing_script object
Create baseband data with options
GenerateBin (opts: list)
Start program with options
Print progress to stdout
Choose baseband data file
Show files
Chooses file
Start spoofing
Instantiate
Instantiate
startSdr(chosen_file)
start()
The Spoofing_script object is a GNU Radio SDR, just like the barrage_jamming object. Therefore, the sequence of events after starting it is the same for both, and is omitted in this diagram for that reason.

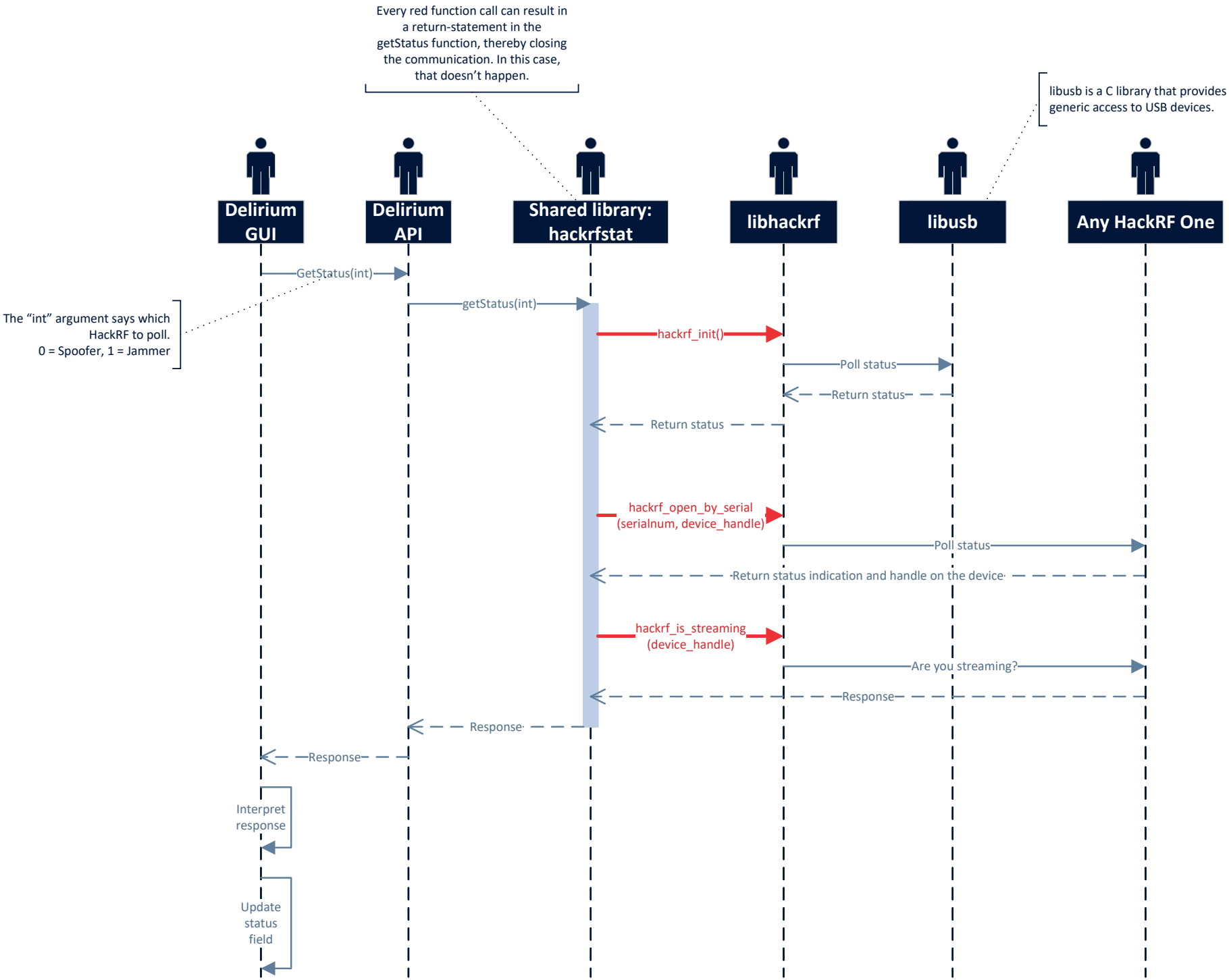
12.3 Sequence Diagram - Sweep Jamming

The following diagram shows the sequence of events that must take place for the user to sweep jam (see section 5.1.3 for more information) using Delirium. Created by AM.


After it has been started, the subprocess changes the center frequency of the SDR constantly (red messages), at a rate determined by the users initial arguments.
The protocol_aware_no_qt object is a GNU Radio SDR, just like the barrage_jamming object.
User
Delirium GUI
Delirium API SweepJamming object
Delirium API
protocol_aware_no_qt object
Start sweep jamming with options
Instantiate with arguments
Create subprocess using SweepJam() as its target with arguments
Return SweepJam function as member
startSweeping()
Start SweepJam subprocess
Instantiate
start()
set_cent_freq(freq_value)
...
Stop sweep jamming
set_cent_freq(freq_value)
stopSweeping()
send stop signal to subprocess
stop()
Print stop confirmation to stdout
delete object

12.4 Sequence Diagram - Status Polling

The following diagram shows the sequence of events that must take place for the Delirium GUI to poll a single HackRF's status using the C++ extension module (see section 16.4 for more information). Created by AM.


Every red function call can result in a return-statement in the getStatus function, thereby closing the communication. In this case, that doesn't happen.
libusb is a C library that provides generic access to USB devices.
Delirium GUI
Delirium API
Shared library: hackrfstat
libhackrf
libusb
Any HackRF One
GetStatus(int)
The "int" argument says which HackRF to poll.
0 = Spoofer, 1 = Jammer
getStatus(int)
hackrf_init()
Poll status
Return status
Return status
hackrf_open_by_serial (serialnum, device_handle)
Poll status
Return status indication and handle on the device
hackrf_is_streaming (device_handle)
Are you streaming?
Response
Response
Response
Interpret response
Update status field

Appendix O

NKOM Application Procedure

Nasjonal kommunikasjons-myndighet

Til.: GNSS samordningsforum
Kopi:

Fra: Nkom

Arkiv ref:

Notat

Dato: 16.06.21

Søknadsrutiner GNSS støysendinger

1. Introduksjon

All bruk av frekvenser i Norge krever tillatelse fra Nasjonal kommunikasjonsmyndighet (Nkom). Støysendinger[1] i frekvensbånd avsatt til satellittnavigasjonssystemer har potensielt store konsekvenser for berørte brukere som er avhengig av korrekt tids- og navigasjonsinformasjon. Det krever at informasjonen i søknader om tillatelse til GNSS-støysendinger gir et godt bilde av planlagt frekvensbruk, utstyr som skal brukes, geografisk lokasjon samt teoretisk beregning av påvirkning for andre brukere av GNSS-signalene. Denne informasjonen blir brukt til å vurdere de potensielle samfunnsmessige påvirkningene som støysendingene vil gi, samt til å gi informasjon om hvem Nkom skal varsle om støysendingene. Dette notatet beskriver søknadsrutinene.

[1] *Frekvensbruk som har til hensikt å hindre, påvirke og manipulere elektronisk kommunikasjon.*

Nasjonal kommunikasjonsmyndighet
Besøksadresse:
Nygård 1, Lillesand

Postadresse:
Postboks 93
4791 LILLESAND

Tel: 22 82 46 00
Fax: 22 82 46 40
firmapost@nkom.no

NO 974 446 871
www.nkom.no

2. GNSS-frekvenser

Rutinene gjelder for alle GNSS-frekvenser, det vil si frekvenser brukt i satellittnavigasjonssystemer under RADIONAVIGATION-SATELLITE allokering i RR (1164-1300 MHz / 1559-1610 MHz).

3. Søknadsfrister

For å få tilstrekkelig tid til saksbehandling og høring hos berørte myndigheter/aktører må søknaden være mottatt av Nkom senest 8 uker før planlagt støysending. Dette gir nok tid til utarbeidelse av vilkår i samarbeid med berørte aktører, varsling gjennom etablerte kanaler og tilstrekkelig tid for tillatelsesinnehaver til å gjennomføre eventuelle pålagte varslingsrutiner minst 1 uke før aktiviteten (for eksempel i form av «Notice to airmen» (NOTAM) og 2 uker før for «Etterretninger for sjøfarende»).

4. Krav til søknad

Søknaden må minst inneholde informasjon om:

- Testens formål.
- Geografisk plassering av støysendingsutstyr.
- Tidsperiode for testing, samt aktuelle tidsrom innenfor perioden hvor testing vil foregå
- Teknisk informasjon om utstyr og støysignal, herunder minst:
    - Antennekarakteristikk
    - Utgangseffekt/EIRP
    - Båndbredde
    - Signalkarakteristikk
- Konservativ vurdering av støysignalets geografiske utbredelse i form av påvirkning på kommersiell GNSS-mottaker (J/S-betraktninger) ved maks utsendt effekt fra bakkenivå og opp til 30.000 fot. Det skal vises utbredelse for 5 fot (AGL), 100 fot (AGL), 1.000 fot (MSL), 5.000 fot (MSL), 10.000 fot (MSL) og 30.000 fot (MSL).
    - Utbredelsen av signalet skal presenteres i form av kart for de ulike høydene nevnt i punktet over.
- Kontaktinformasjon til testleder for å kunne ivareta muligheten til å avbryte støysending umiddelbart om det oppstår livstruende situasjoner.

5. Fareområde

Søker må selv vurdere om aktiviteten er til fare for lufttrafikk i utbredelsesområdet for støysignalene, og om nødvendig søke om opprettelse av fareområde i luftrommet etter forskrift om luftromsorganisering. Søknad om opprettelse av fareområde sendes til Luftfartstilsynet.

6. Høringsrutiner

Når søknaden er mottatt vil Nkom vurdere behovet for høring av søknaden og vilkår med andre myndigheter, spesielt Luftfartstilsynet og Kystverket. Ved potensiell påvirkning av luftfart vil også Avinor Flysikring bli kontaktet. Det vil bli gitt 2 ukers frist for tilbakemeldinger til Nkom med

en vurdering av om påvirkning i aktuell sektor er akseptabel og eventuelle tilleggsvilkår som bør være i tillatelse.

7. Tillatelse

Tillatelse blir utstedt senest 3 uker før start av omsøkt støysending.

Tillatelsen vil som standard ha vilkår om:

- Varsling av lokale nødetater i god tid før øvelsen starter.
- Varsling ved oppstart av øvelse til Nkom for videre varsling inn i samordningsforum for GNSS.
- Varsling av lokale flyklubber som kan bli berørt samt Norges Luftsportsforbund 1 uke før oppstart av øvelse.
- Varsling av Norsk Luftambulanse på grunn av mulig påvirkning av innflygingsrutiner basert på GPS til baser og sykehus uten tradisjonell flynavigasjon.

Tillatelsen vil gjelder for eller flere spesifikke geografiske lokasjoner for test samt mulige restriksjoner på tidspunkt for testing innenfor omsøkt periode.

Tillatelse kan inneholde spesifikke vilkår som følge av krav fra høringsinstanser.

Nkom kan avslå søknad etter en helhetsvurdering, eller dersom søknaden ikke inneholder nødvendig informasjon etter punkt 3.

8. Annet

De som får tillatelse til støysending, skal unngå at støysendinger kan berøre tjenester i naboland. Støysendinger som har potensiale til å påvirke andre lands luftrom, land- eller havområder vil som utgangspunkt bli avslått fordi det normalt vil kunne kreve en nabolandskoordinering, med mindre den samfunnsmessige nytten anses som stor nok.

Utsendt effekt og tidsperiode for testing skal holdes til et absolutt minimum.

Det skal så langt det er mulig søkes om støysendinger i geografiske områder hvor potensialet for påvirkning på luftfart og sjøfart er minst mulig. Det er nødvendig med tilstrekkelig avstand til flyplasser, havner, skipsleder og større byer

Søker må også argumentere for at tester som gjennomføres i friluft ikke kan gjøres i skjermet rom (ekkofritt rom), eller gjøres som kablede tester.

Appendix P

NKOM Application

Fag: Bachelorprosjekt

Søknad: GNSS støysendinger

Helge Kopland, Andreas B. Sørensen

16/5-2024

Sammendrag

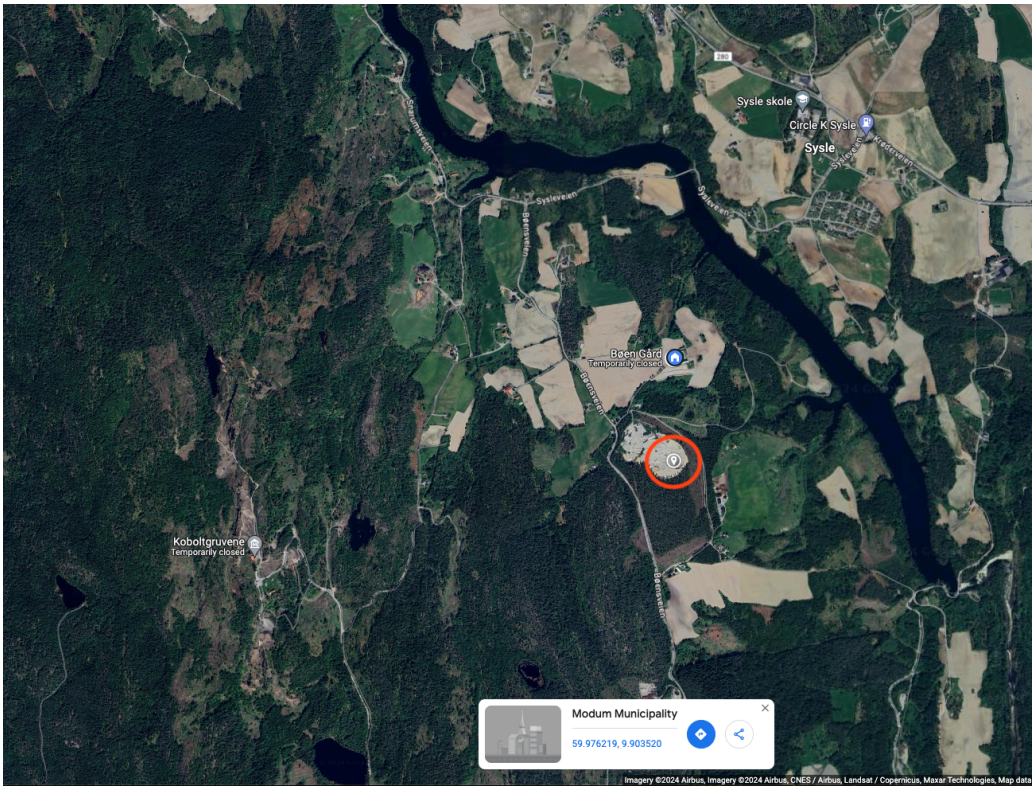
Vi ønsker i forbindelse med vårt bachelorprosjekt å sende støysignaler og egenproduserte GPS-signaler for å verifisere vårt portable anti-drone system.

Innhold

1 Testens formål

I forbindelse med vårt bachelor-prosjekt har vi fått i oppgave av vår oppdragsgiver, Kongsberg Defence & Aerospace, å teste flight-controlleren Navio 2 og dens evne til å motstå angrep via jamming og spoofing. Systemet er rettet mot GPS-signaler, og test i friluft vil gi oss ekte signaler å teste mot. Testen vil gå ut på å jamme de ekte GPS-signalene dronen (Navio 2) mottar, for så å sende egne GPS-signaler til den (spoofing).

2 Geografisk plassering av støysendingsutstyr



Figur 1: Plassering av støysendingsutstyr.

3 Tidsperiode for testing

Vi ønsker å gjennomføre testing den 31. mai, fra 1000-1600.

4 Teknisk informasjon om utstyr

Utstyret vi bruker er en HackRF One [1], sender og mottaker, med konfigurering ved hjelp av software defined radio (SDR). Systemet inneholder to stk. HackRF One, slik at vi kan sende støysignaler og egne GPS-signaler samtidig.

Informasjon om HackRF One fra produsent (Great Scott Gadgets) [1]:

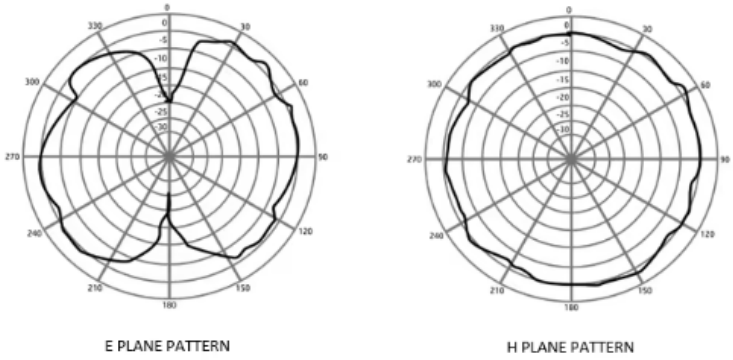
- 1 MHz to 6 GHz operating frequency
- Half-duplex transceiver
- Up to 20 million samples per second
- 8-bit quadrature samples (8-bit I and 8-bit Q)
- Compatible with GNU Radio, SDR, and more
- Software-configurable RX and TX gain and baseband filter
- Software-controlled antenna port power (50 mA at 3.3 V)
- SMA female antenna connector
- SMA female clock input and output for synchronization
- Convenient buttons for programming
- Internal pin headers for expansion
- Hi-Speed USB 2.0
- USB-powered
- Open source hardware

(Vi vil i tillegg bruke en attenuator (demper) for å senke styrken på signalene vi sender.)

4.1 Antennekarakteristikk

ANT500: omnidirectional (rundstrålende), dipol

Antennen vi skal bruke er en teleskopisk antenne designet for operasjon mellom 75 MHz og 1 GHz. Total lengde er justerbar fra 20 til 88 cm. Antennen er konstruert av rustfritt stål og består av SMA-konnektor, roterende skaft og justerbar ”albue” [2].



Figur 2: karakteristikk for rundstrålende dipol antenne.

5 Utgangseffekt/EIRP

Støysignalet vi sender er i form av additive white gaussian noise (AWGN), med unntak av "Protocol Aware jamming", hvor vi sender en "tilfeldig streng av bits (random stream of bits)" som er faseforsøvet 90° i forhold til det originale GPS-signalet.

5.1 Målte resultater: effekt og båndbredde

Tabell 1 viser resultater fra måling med signalgenerator for alle våre støysendere. Testen foregikk med kablet forbindelse mellom signalgenerator og HackRF One.

Type støysending:	Peak (dBm)*	Quasi peak (dBm)	EMI average (dBm)	Båndbredde:
Spot	-35.99	-35.50	-42.00	1 MHz
Sweep	-35.00	-42.01	-58.91	15.3 MHz**
Barrage	-44.84	-44.52	-52.23	14 MHz
Protocol Aware	-36.11	-36.34	-44.95	15.3 MHz
Spoofing	-29.03	-29.25	-32.84	2.6 MHz

Tabell 1: Målinger med signalgenerator (kablet).

**EIRP = Peak*

***Båndbredden er 15,3 MHz, men flyttes mellom de forskjellige senter-frekvensene for GPS, GLONASS, Beidou og Galileo.*

6 Konservativ vurdering av støysignalets geografiske utbredelse

Tabell 2 viser resultatene av beregninger for maks utsendt effekt (J/S) fra vår sterkeste støysender. Beregningene tar hensyn til GPS-satellittenes høyde, effekt og siktlinje tap. Beregningene kan ses i sin helhet i formlene 3, 4, 5 og 6.

Avstand:	J/S
5 fot:	52.94 dB
100 fot:	26.57 dB
1000 fot:	6.57 dB
5000 fot:	-7.41 dB
10000 fot:	-13.43 dB
30000 fot:	-22.98 dB

Tabell 2: Maks utsendt effekt ved forskjellige høyder.

6.1 Beregninger

6.1.1 EIRP

$$EIRP = P_T - L_C + G_A, \tag{1}$$

hvor EIRP (Effective Isotropic Radiated Power) er utgangssignalet til et signal når det blir konsentrert inn på et bestemt område av antennen, der P_T er utgangseffekten til senderen (output power of transmitter), L_C er tap i kabel (cable-loss) og G_A er antenne forsterking (antenna gain).

I vårt tilfelle vil peak utsendt effekt (ref. tabell 1) være lik EIRP, da alle målinger er gjort på signalgenerator, med neglisjerbart tap i kabel og uten gain.

$$EIRP = P_{T(peak)}, \tag{2}$$

6.1.2 J/S betrakninger

Fra Friis formler for overføring i fri sikt (free space):

$$\frac{J}{S} = \frac{\frac{P_J G_J G_R \lambda^2}{(4\pi d_J)^2}}{\frac{P_T G_T G_R \lambda^2}{(4\pi d_S)^2}}, \tag{3}$$

Forenklet:

$$\frac{J}{S} = \frac{P_J G_J d_s^2}{P_T G_T d_J^2}, \tag{4}$$

Omregning til desibel:

$$\frac{J}{S} = P_J + G_J - P_T - G_T + 20\log(d_S) - 20\log(d_J), \tag{5}$$

hvor J/S står for jammer til signal (GPS) rate,
P_J = jammerens utgangseffekt [dBW],
G_J = jammerens forsterking [dBi],
d_s = avstand fra sender til mottaker [m],
P_T = senders utgangseffekt [dBW],
G_T = senders forsterking [dBi]
d_J = avstand fra jammer til mottaker [m].

	Effekt:	Forsterking:	Siktlinje tap:	Høyde:
GPS-satellitt	44 dBm	13 dBi	-182 dB	ca. 20200 km

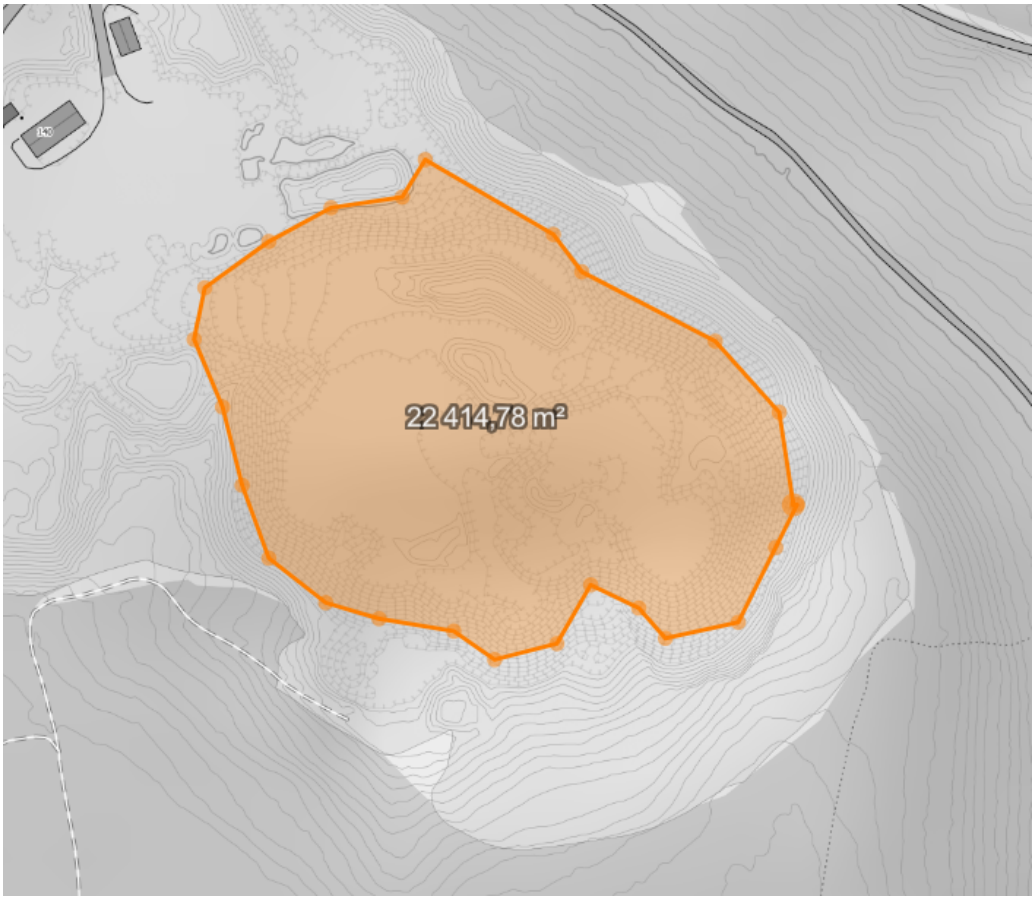
Tabell 3: Informasjon om GPS.

For 30000 fot med vår sterkeste støysender:

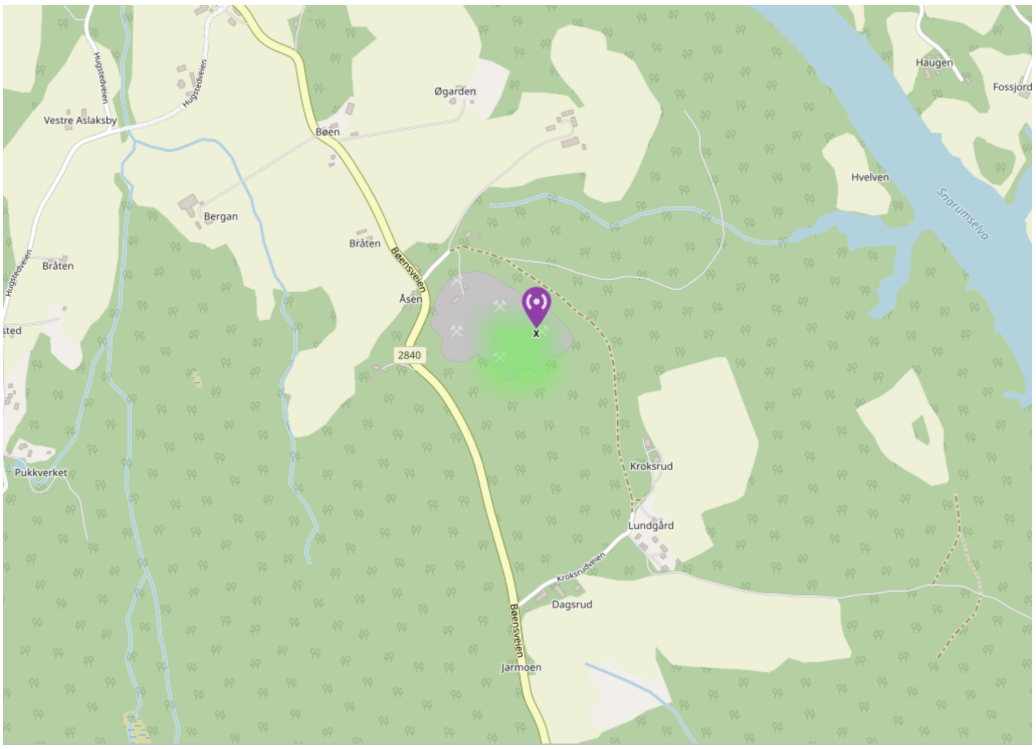
$$\frac{J}{S} = -62 + 0 - 14.08 - 13 + 20\log(20190000) - 20\log(10000) = -22.98\ dB, \tag{6}$$

6.2 Geografisk fremstilling:

Figur 3 og 4 viser geografisk utbredelse av vår støysending. Som man kan se av figur 4 er signalet svært begrenset av terrenget rundt, da støysenderen er plassert på laveste punkt i et grustak. Dette begrenser signalet til en mindre del av grustaket, og signalet vil ha høyest effekt oppover. Ved denne plasseringen begrenses signalets effekt på omkringliggende bygninger (og annet) til et absolutt minimum.



Figur 3: Begrensning av signal for mottakere på bakkeplan rundt støysender.



Figur 4: Beregning av signalutbredelse for mottakere på bakkeplan rundt støysender. Her er det tatt hensyn til obstruksjoner (sand, trær osv)

7 Kontaktinformasjon

Testleder:

- Navn: Helge Kopland
- Telefon/mobil: 47636548
- Mail: helge.kopland@gmail.com

Øvrige medlemmer:

- Navn: Andreas Bondal Sørensen
- Telefon/mobil: 99026026
- Mail: andreas.bondal@gmail.com

Øvrige medlemmer:

- Navn: Stian Nordholm
- Telefon/mobil: 90556447
- Mail: stiannordholm@gmail.com

Øvrige medlemmer:

- Navn: Anders Minde
- Telefon/mobil: 40031757
- Mail: anders.minde@gmail.com

8 Fareområde

Vi ser det ikke nødvendig å opprette fareområde for vår testing, grunnet lokasjon og effektbetraktninger.

9 Annet

Vi har gjort testing i Faraday-bur og kablet inn mot receiveren, dette for å verifisere effekten av vårt system. Kablet testing utendørs har vi verifisert at vi kan jamme og spoofe ekte GPS-signaler. Testing i Faraday-bur har vist oss at vi kan jamme eller spoofe trådløst. Vi får derimot ikke testet begge deler, ettersom vi ikke har ekte signaler inn til receiveren i Faradayburet. Det er derfor behov for å teste effekten av vårt system utendørs (hvor vi kan få inn ekte GPS-signaler) uten kabling, for å se om virkningen er som vi ønsker. Det er også eneste reelle mulighet for å teste rekkevidden slik systemet er satt opp nå.

Referanser

[1] G. S. Gadgets, "HackRF One." `https://greatscottgadgets.com/hackrf/one/`.

[2] G. S. Gadgets, "ANT500." `https://greatscottgadgets.com/ant500/`.

Appendix Q

Digital Frequency Radio Memory

13 Digital Frequency Radio Memory

ABS | *SN*

We mentioned in section 5.1 that there are two main jamming techniques: noise-jamming and repeater-jamming. This appendix focuses on repeater-jamming. The most used method of repeater-jamming is Digital Frequency Radio Memory or DFRM. DFRM is a jamming method where a jammer will send fake signals to a radar. This is done by the jammer receiving a signal from a radar, modulating the signal by frequency, phase, or amplitude, and then sending the signal back to the radar. Since the "only" reason for the radar to get a hit will be reflections from a material (significantly different than air) [13], the radar will perceive the fake signal as a real "target". The jammer can then send multiple fake targets to the radar, and flood the radar with fake targets, the radar will then make false assumptions about material, velocity, and range.

In figure Q.1 below, one can see the operation of the DFRM. The signal is converted from an analog to a digital signal to make use of the RAM and slice repeater. The slice repeater is capable of making a jamming signal from only a part of the received signal [13]. The new signal is then reconstructed into an analog signal ready to be transmitted back to the radar.

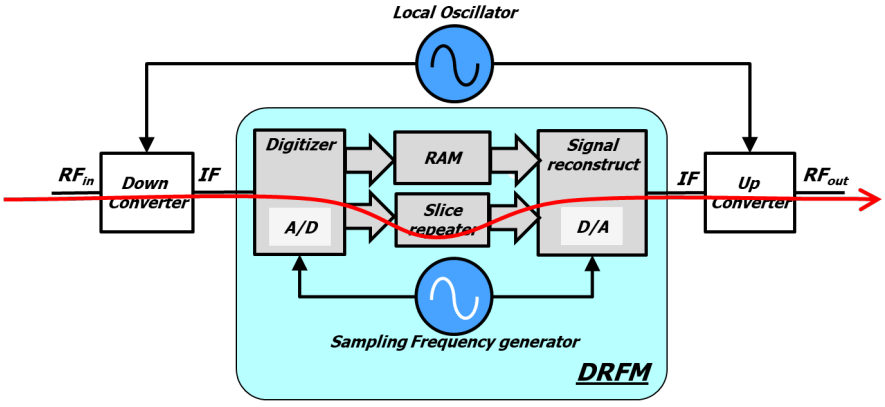


Figure Q.1: Digital Frequency Radio Memory operation [13].