Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Trygve Nerland

Radio tracking of sheep - Developing MAVLink enabled devices, MAVLink control and the basis for MAVLink enabled autonomous UAVs

Master's thesis in Master of Science in Informatics
Supervisor: Svein-Olaf Hvasshovd
June 2021

NTNU
Norwegian University of Science and Technology

Trygve Nerland

Radio tracking of sheep - Developing MAVLink enabled devices, MAVLink control and the basis for MAVLink enabled autonomous UAVs

Master's thesis in Master of Science in Informatics
Supervisor: Svein-Olaf Hvasshovd
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for ei betre verd

Norwegian University of
Science and Technology

Radio tracking of sheep - Developing MAVLink enabled devices, MAVLink control and the basis for MAVLink enabled autonomous UAVs

Trygve Nerland

June, 2021

Abstract

This is one out of three papers covering different parts of the same system. Our objective was to develop a system for locating animals with the use of unmanned areal vehicles(UAV) and radio communication technology. With the plan to use low cost and lightweight radio transceivers. The main focus of this paper is on the UAV and integrating the other components together.

We will take a look at the technologies used in our system. The relevant theory and science will be summarized. The relevant regulations is also be examined. Existing and alternative solutions will be explored. Different UAV systems will be compared.

The system architecture and the function of each component is explained and also how everything fits together into a fully functioning system. The reasoning behind why we chose to build a custom UAV and the process of getting it into a flight worthy state. Our tests, how they were conducted, the goals and any problems encountered during them will be explained.

The data from our tests will be gathered, visualized and analyzed. This analysis is going to have the main focus on the UAV, the distance measurements and the system as a whole. Using the results of the analysis to set our expectations, tweak the system or plan our path forward.

We will discuss the results of our analysis and examine the viability of our system. A comparison of our system with the existing solutions will be done.

We managed to create a system for surveying large areas autonomously and locating animals within them. This system could locate the animals with great accuracy, possibly with an average error of 15m. The system utilizing a custom built UAV and cheep radio transceivers. We could with minimal user input find animals and estimate their location. Our system does not provide any features except locating the animals, unlike other available systems with many new features. This combined with possible regulatory problems might cause our system to not be a viable competitor to existing solutions.

Sammendrag

Denne masteroppgaven er en av tre oppgaver som dekker det samme systemet. Hver av disse oppgavene har fokus på forskjellige deler av dette systemet. Oppgaven vår var å lage et system for å lokalisere sauer som ute på beite ved bruk av droner og radio teknologi. Her var planen å bruke små, lette og billige radio sendere. Denne masteroppgaven kommer til å fokusere på dronedelen av systemet og integrasjonen mellom resten av komponentene.

Vi kommer til å se nærmere på teknologiene som blir brukt i dette systemet. Relevant teori og kunnskap vil bli oppsummert. Relevante lovverk blir også gått igjennom. Vi ser også på eksisterende løsninger som har blir utviklet for å løse den samme problemstillingen. Detaljert kunnskap om forskjellige typer droner og oppbygningen av de blir går igjennom.

Systemarkitekturen blir gått igjennom og funksjonen til hver del i systemet blir forklart. Det blir også forklart hvordan alle delene av systemet passer sammen og lager et fullt fungerende system. Her blir valgene våre av drone begrunnet og hvordan vi bygde en spesialbygd drone for dette systemet. Vi viser hvordan vi gjennomførte testene våre. Og vi forklarer målet med de. Problemene vi støtte på underveis og løsningene på de blir også gått igjennom.

All data fra testene våre ble samlet, analysert og visualisert der det ga mening å gjøre dette. Denne analysen hadde hovedfokuset på dronen, avstandsmålingene og hvordan systemet fungerte som en helhet. Resultantene fra analysen ble brukt til å sette våre forventninger framover, gjøre justeringer på systemet eller planlegge videre utvikling.

I diskusjonen vi ser på resultatene fra analysen i sin helhet og vi ser også på praktikaliteten til systemet. En sammenligning av systemet vårt blir gjort med andre systemer som er laget for til å spore dyr.

Vi greide å lage et system som kan finne igjen dyr i store utendørsområder. Dette systemet er nesten helt automatisert ved hjelp av autopilot funksjonen til drona og våre egenutviklede applikasjoner. Systemet kunne lokalisere dyr med stor nøyaktighet, muligens med et gjennomsnittlig avvik på 15m. Dette systemer bruker en drone som er bygd spesifikt for dette formålet og billige radio sendere og mottakere. Systemer vår gir ikke noe ny funksjonalitet borsett fra enklere lokalisering av dyr. Andre systemer som blir brukt for å lokalisere dyr gir mye ekstra funksjonalitet i tillegg. Dette kombinert med mulige begrensninger pga. regelverk kan begrense nytten av systemet vårt sammenlignet med eksisterende systemer.

Foreword

At the end of summer when the grazing period ends, it's a large task for farmers to gather up all the animals and take them back from their grazing lands. Giving farmers better and more efficient tools can result in a significant reduction in the time and manpower needed for tasks like this. This study aims to determine if it's viable to automate the search for these animals using UAVs and radio technology. The animal we are focusing on in this paper is sheep and the radios in use is low-Cost energy-Efficient Bluetooth Transceivers.

We choose this project because it seemed interesting working with drones and other physical devices in a practical setting. The possibility of creating a system that could save farmers a lot of time and labour seemed intriguing.

Acknowledgements

We wish to thank Gard Steinsvik and Grzegorz Swiderski for their work on their related projects. Thanks to Nordic Semiconductors for providing us with nRF52833 development kits. We also wish to thank Svein-Olaf Hvasshovd for guidance and wisdom throughout the project.

Contents

List of Figures

List of Tables

Abbreviations

BEC ..... Battery eliminator circuit. In this case it's a constant voltage power supply powering onboard electronics.

BLE ..... Bluetooth Low Energy

BVLOS ..... Beyond visual line of sight

EKF ..... Extended Kalman Filter

ESC ..... Electronic Speed Controller, drives and controls the drone's motors using power from the battery

FC ..... Flight controller, the brain of the drone

GCS ..... Ground Control Station

GNSS ..... Global navigation satellite system

GPIO ..... General-purpose input/output

GPS ..... Global Positioning System

INS ..... Inertial Navigation System

MAC ..... Media Access Control

MAVLink .... Micro Air Vehicle Link

MTOM ..... Maximum take-off mass

RTL ..... Return to Launch

SITL ..... Software in the Loop

SoC ..... System on a chip

UART ..... Universal asynchronous receiver-transmitter

UAV ..... Unmanned aerial vehicle

Chapter 1

Introduction

At the end of summer when the grazing period ends, it's a large task to gather all the animals from their grazing lands. This is a long and tiresome process involving scouring both forests and mountains looking for the animals. A more efficient method to find all the animals would save a lot of time and manpower.

There is already some solutions to this problem being tested commercially by different companies. The solutions usually involves a collar containing a GPS receiver, mobile network communications equipment and a battery, this type of units are expensive, heavy, cumbersome and can not be used by younger and growing animals.

We are going to look into an alternative solution with a very different approach. This involves equipping animals with a tags containing a very small Bluetooth LE chip and a battery. A drone would also be equipped with a Bluetooth LE chip to communicate with the tag, retrieving information from it and calculating the distance between the tag and the drone. By using GPS data from the drone and the distance measurements can determine an approximate position for the animal. The small size also makes it possible to use this tag on younger animals.

In this report, we will first describe the relevant theory, and then detail the path we took in order to reach a solution. Finally, we will reflect on the results and compare them to existing solutions.

1.1 Original Task description

Our task was to develop a system capable of locating animals using a UAV and radio technology. The use of cheep radio transceivers for distance measuring opens up new possibilities to develop lightweight and cheap animal tracking. By combing the two technologies we can provide low cost animal localization without the need to have cell phone or satellite connectivity built into each tag.

1.2 Software Base

1.2.1 Python

We used the Python programming language[50] version 3.7.5 in this project. The major software libraries are listed below:

Library	Version	Description
pymavlink	2.4.14	Python MAVLink interface and utilities. (Custom fork [37])
matplotlib	3.3.4	Comprehensive library for creating visualizations in Python.

1.2.2 ArduPilot

In this project we used the ArduPilot open source autopilot software for both the flight controller and the SITL. We used version V4.1.0-dev of ArduPilot in this system. This was compiled on Linux(Ubuntu 18.04) from source code available on GitHub [2].

1.2.3 SheepRTT

The SheepRTT modules have software operating them. More information abou this in [45].

1.2.4 Radio Sheep GCS

Radio Sheep GCS is a ground control station for managing drone operations with sheep searching related purposes. [43] More information about this software is found in [44].

Chapter 2

Theory

In this chapter we will look at existing and alternative solutions. We will look at technologies we could use to build our system. Also we will go through the basis for UAVs and all relevant systems around them. The regulations around UAV usage in Norway will also be examined as this is very relevant for us. Relevant theory and science will also be mentioned here.

2.1 Existing solutions

We gathered information about existing solutions for the same problem we are trying to solve. We choose to take a look at the regular way to locate and gather sheep and two systems based around GPS collars.

2.1.1 Traditional/regular

The traditional way to gather animals from the grazing areas is to manually search for them. To help this manual search the animals is outfitted with bells. The bells make sound every time the animals move, and this sound can be heard and used to locate animals many hundred meters away. This process quite takes a quite lot of time, especially when covering large areas. We found bell collars for sheep available for 84 NOK before sales taxes.

2.1.2 GPS collars

A technology that have gotten popular the last few years is GPS collars. The some GPS collars are specially made for tracking the movement of farm animals. These GPS collars use cellphone networks or satellite communication to report the animal's movement at regular intervals. This means you can keep track of the animals the whole season without manually visiting them. Many of these collars can also be used as virtual fence or geofence to keep animals in the desired area. The virtual fences uses audio warnings and electric shocks to prevent animals from leaving the desired area.

A downside of the GPS collars is the size of the units themselves. The larger units are also not optimal for use with smaller animals.

We took a closer look at GPS collar already commercially available. We chose to focus on the products from Nofence and findmy. Data from the manufactures

themselves were used in this comparison. We focused on making a comparison of connectivity, battery life and costs.

Nofence

The collar from Nofence requires cell phone connectivity to report location or make changes to geofence settings. [38] On newer models 2G or LTE CAT M-1 can be used for connectivity. The battery life is on average around three weeks, but can vary between one week and three months depending on the use. Configuration and monitoring can be done through a mobile phone app. The form factor is similar to a normal collar with a bell, but a little bit larger. This system is commercially available.

The unit cost per GPS collar is 1850 NOK before sales taxes. [39] In addition to the upfront cost the collar requires an active subscription from Nofence. The cost of this subscription varies depending on the total number of animals and the length of the grazing season. Based on this factors the cost can be between 6 NOK and 1.5 NOK per animal per day.

findmy

The collar from findmy does not require cell phone connectivity, it instead uses satellite communication to exchange data. [28] The battery life estimated to last two to three seasons if the grazing season lasts half a year. The form factor is similar to a normal collar with a bell, but a little bit larger. This system also features a system to alert the user when the animals get in potentially stressfull or dangerous situations. This system is commercially available.

The unit cost per GPS collar is 1849 NOK before sales taxes. In addition to this the collar requires an active subscription from findmy. The cost of this subscription is 229 NOK per year.

2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless technology developed by the Bluetooth Special Interest Group (SIG). [30] It's designed as a low-power technology to control and monitor devices ranging everything from consumer devices to medical or industrial devices. It was integrated as a subset of the Bluetooth 4.0 specification in 2009. The main focus of this technology is to increase battery life on small low power devices designed for operating on batteries for long periods of time. The other focuses of this technology is low cost for the hardware itself. Because the technology is based on Bluetooth there is support for it on most modern devices supporting Bluetooth.

2.2.1 Long-Range Mode

With the introduction of Bluetooth 5.0 an optional feature known as long-range mode or Coded PHY were introduced. This mode allows longer range at the cost of speed. Both the original mode and the long-range mode use a sending rate of 1Mbps, but with long-range mode each bit is sent two or eighth times instead of

one time. Sending each byte multiple times increases the chance of it being received by the other device. But this also reduces the bandwidth to 500Kbps or 125Kbps depending on how many times each bit it retransmitted. This could increase the maximum range a BLE device can communicate at to 1km or more. Because of the reduced bandwidth when using this mode the radio itself needs to be active for a longer amount of time compared to the original mode, thus increasing the energy consumption. As this feature is optional it's important to make sure both devices you're trying to connect together support this feature.

2.2.2 nRF52833 development kit

Nordic Semiconductors have given us access to five nRF52833 development kits. It's easy to connect the development kit to a computer by using a micro-USB cable. The development kit have a built in JTAG/SW interface for programming the built in nRF52833 chip and also the possibility of programming an external nRF chip. The use of 2.54mm headers on the board makes it easy to connect anything to the GPIO pins for testing. The development kit includes an integrated antenna and an U.FL connector for attaching an external antenna.

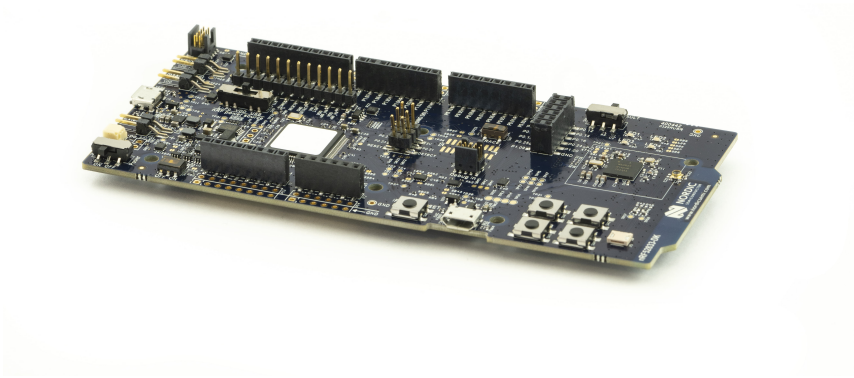


Figure 2.1: nRF52833 Development kit.
Source: `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52833`

2.2.3 MINEW MS88SF23

We also bought three MINEW MS88SF23, which is a Bluetooth LE module based on the nRF52840 SoC. The nRF52840 is very similar to the nRF52833, but features double the amount of RAM and flash storage. It's a small form factor module, ideal for use where weight and size are constrained. It measures 23 x 17 x 2 mm and have a weight of approximately 1g. The module have 28 soldering pads along the sides for power, GPIO, debug and programming interface and USB. The placement of soldering points along the side makes it easier to solder by hand compared to many comparable modules. It also features an U.FL connector for an external antenna.

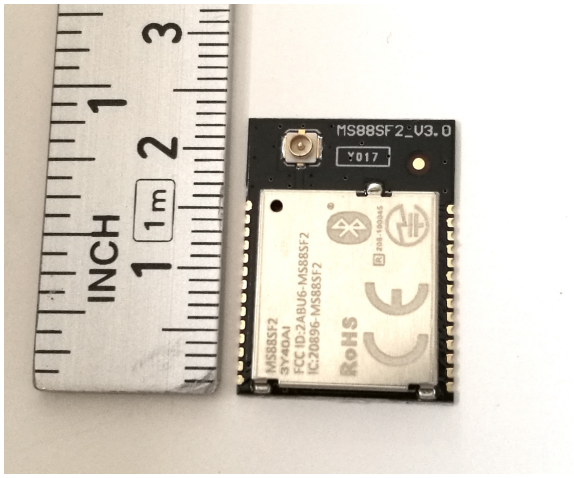


Figure 2.2: MINEW MS88SF23

2.3 Distance estimation and localization techniques

When we want to estimate the position of a target we can use different radio localization techniques. A common one you probably have already heard of is triangulation, but there is also alternatives like multilateriation.

2.3.1 Triangulation

Triangulation is a technique for determining where a new location is by forming a triangle using known locations. The first step is to determine the direction the signal from the target, also known as the angle of arrival(AoA). Then from the known locations we draw a line or a cone in that direction and where they intersect we can estimate the new location to be.

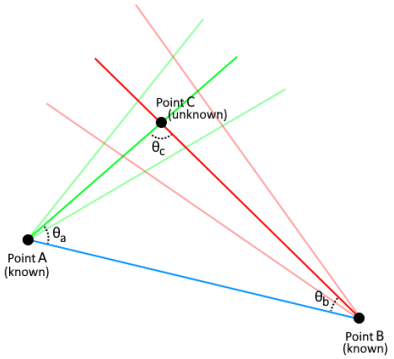


Figure 2.3: Triangulation illustration.

2.3.2 Multilateriation

Multilateriation is a technique for determining where a new location is by measuring the distance between a number of known locations. After we have the distances, we can draw a circle around each known location with the radius being the distance to the new location. Where the circles intersect we can estimate the new location to be. If we have too few distance measurements the circles can intersect at multiple points, but only one of the points is the real location. An example of this problem is shown in figure 2.4. The solution to this problem is to gather more distance measurements from different locations.

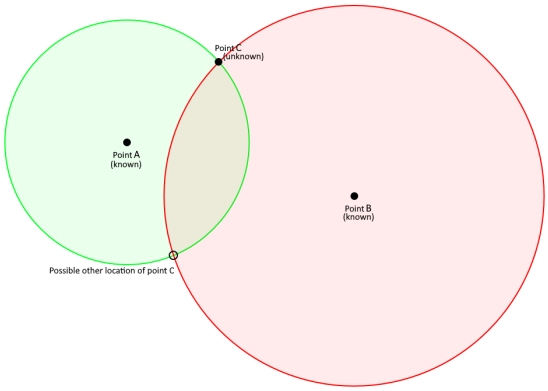


Figure 2.4: Multilateriation illustration

Particle filter

This is a method for using the distance measurements to estimate a new location. The particle filter method used in this paper is based on the Monte Carlo Localization. More information about the implementation of this localization technique is found in [44].

2.3.3 RSSI distance estimation

RSSI or Received Signal Strength Indicator can be used as an indicator of how far away something is. Usually a signal gets weaker the further away you go and stronger the closer you are. By measuring the signal strength we can estimate the distance between the radio sender and the receiver. However the signal strength can be influenced by many factors like destructive and constructive interference, obstructions, signal reflections and the antenna orientations. It can be hard to detect or compensate for these factors. Because of this the signal strength can vary a lot between measurements and be unreliable.

2.3.4 RTT distance estimation

In a similar way RADAR estimate distance by measuring the time it takes for a reflected signal to return we can measure the time it takes to get a reply to our radio message. We measure the Round Trip Time(RTT) as the time from us sending out a message and until the Time of Arrival(ToA) for the reply. By knowing the signal propagation speed and the RTT we can calculate a distance estimate.

This distance estimation technique requires the target to actively answer to pings from the sender. Variations in the signal strength would not affect the estimated distance. As long as the signal is strong enough to be received we can get a measurement. It requires the searcher to have accurate high resolution clock to measure the delay between sending and receiving. The radio signal can get reflected of a surface and thus take a longer alternative path, this reflected signal could use longer time to reach the searcher and make it appear to be further away. It's also important to factor in processing time and delays at each end.

As determined by Nyholm in [40], using a RTT based distance estimation is more accurate than RSSI based distance estimation.

2.4 UAV

2.4.1 Firmware alternatives

There is a wide selection of both open source and closed source UAV firmwares. We have considered a few of them and compared them below. A more comprehensive and in depth survey of open source firmwares is available at [26].

ArduPilot

ArduPilot is a advanced, powerful and reliable open source software system. [46] It's also very diverse and it can control almost any type of vehicle including but not exclusive to airplanes, multi-rotors, helicopters, boats, submarines and ground vehicles. There is built in support for MAVLink. It also have autonomous flight capabilities.

Betaflight

Betaflight was the experimental fork of Cleanflight with a focus on flight performance, leading-edge feature additions and supporting a wide range of flight controllers. [23] It can control both multi-rotor and airplane UAVs. Mostly used on racing UAVs and does not have autonomous flight capabilities. It does not have autonomous flight capabilities.

INAV

A fork of Cleanflight with a very heavy focus on GPS and autonomous flight capabilities. [31]

PX4

PX4 is an open source flight control software for drones and other unmanned vehicles. [25] The project provides a flexible set of tools for drone developers to share technologies to create tailored solutions for drone applications. PX4 provides a standard to deliver drone hardware support and software stack, allowing an ecosystem to build and maintain hardware and software in a scalable way.

DJI

DJI is the company with the largest marked share in the drone marked. [6] DJI drones have autonomous flight capabilities and support control through the DJI SDK. The firmware on the drones is proprietary and closed source. It does not support control over MAVLink.

2.4.2 Ground Control Station

A Ground Control Station(GCS) is a piece of software designed to communicate with with the drone, showing operating information, configuring the parameters of the drone itself or planning autonomous flight missions.

Mission Planner

Mission planner is a open source GCS made for ArduPilot, it can be used to configure everything on the UAV and help with setup of both hardware and software. There is also possibility to monitor the drone and change parameters in flight. It also contains an advanced log analyzer for both manual and automatic analysis. A powerful autonomous flight planning tool that can fetch maps and terrain information from multiple sources like Google maps or Bing Maps. Windows is required to run this software. It communicates with the drone over MAVLink.

Figure 2.5: Mission Planner

QGroundControl

QGroundControl support full flight control and autonomous mission planning for any MAVLink enabled drone. This means it would work with drone firmwares like PX4, ArduPilot, INAV and many more. The software is open source. QGroundControl runs on all major desktop and mobile operation systems including Windows, Linux, OS X, Android and iOS.

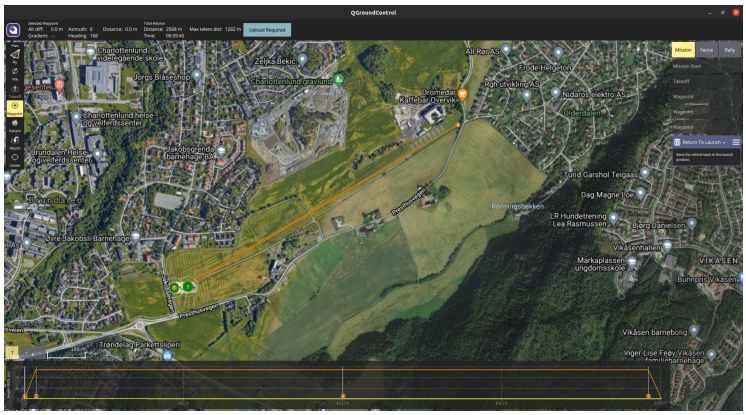


Figure 2.6: QGroundControl

DJI GS PRO

DJI have made a GCS for use with DJI drones. It's closed source and requires an iPad to run. The app itself is free but requires an expensive payments to unlocking more advanced features.

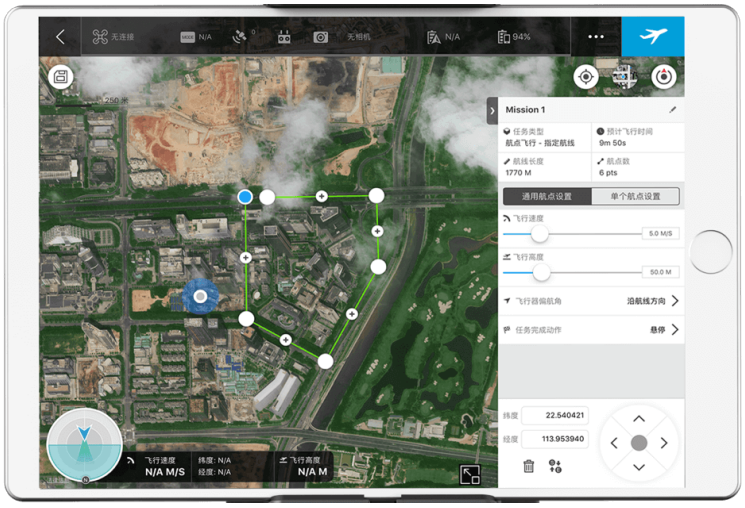


Figure 2.7: DJI GS PRO

Radio Sheep GCS

The Sheep GCS is our custom made GCS for this our system. It is developed with Typescript and Electron. It can fetch map and terrain data from Kartverket. This data can then be used to plan autonomous flight missions. It can also receive, process and visualize data from our SheepRTT module on the UAV. It communicates with the drone over MAVLink.

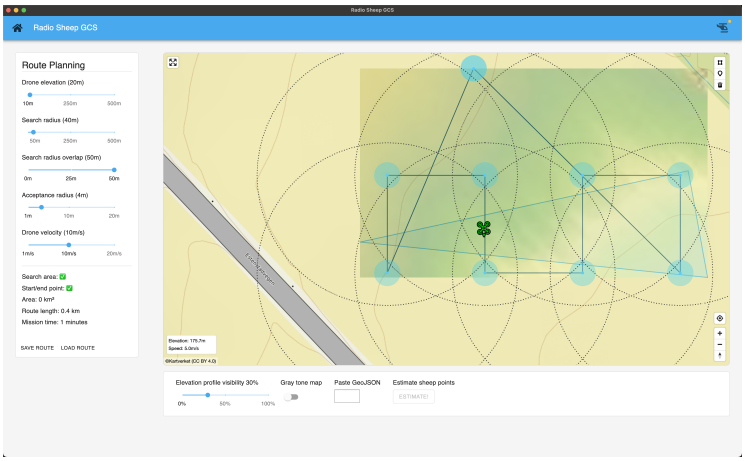
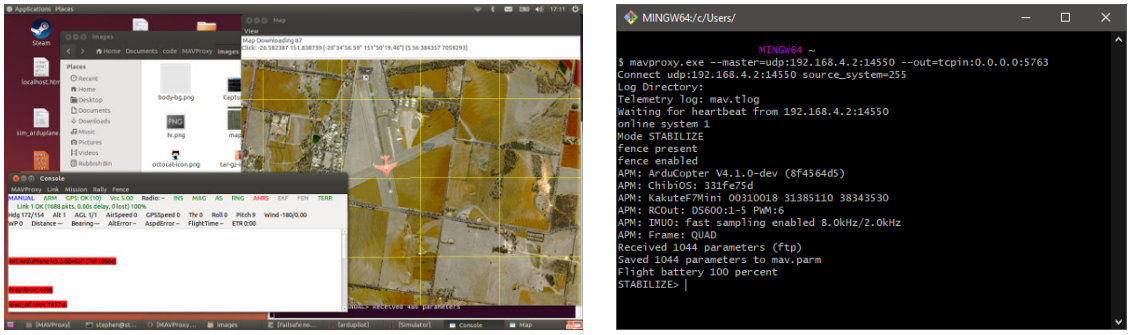


Figure 2.8: Radio Sheep GCS

MAVproxy

MAVproxy is a GCS for MAVLink based systems. [49] There is both a command line interface and a graphical interface. The original focus of MAVproxy was to handle multiple MAVLink links and have since evolved into a fully functioning GCS. One of the most powerful features in MAVproxy is the ability to route messages between multiple links. It can do this over many different protocols like UDP, TCP and serial connections. MAVproxy runs on Windows, Linux, OS X and some other operating systems.



(a) MAVproxy GUI in linux (b) MAVproxy CLI

Figure 2.9: MAVproxy

2.4.3 Simulated Vehicle (SITL)

A simulated vehicle or Software In The Loop (SITL) allows you to simulate a full UAV running the same software as a real UAV would. [4] The ArduPilot SITL simulator is built from the same code that runs on the drone. This allows you to run any version of the ArduPilot autopilot software without any physical hardware. It's running as a native executable on a computer, built using a ordinary C++ compiler. It can run natively on both the Linux and Windows operating system. A physics engine is also included to simulate the physics of the simulated UAV.

This makes testing and developing the communication with the UAV much faster and less risky. There is no risk of damage to the UAV while testing. Also we don't need to have the physical UAV available, so everyone can work on it anytime, anywhere.

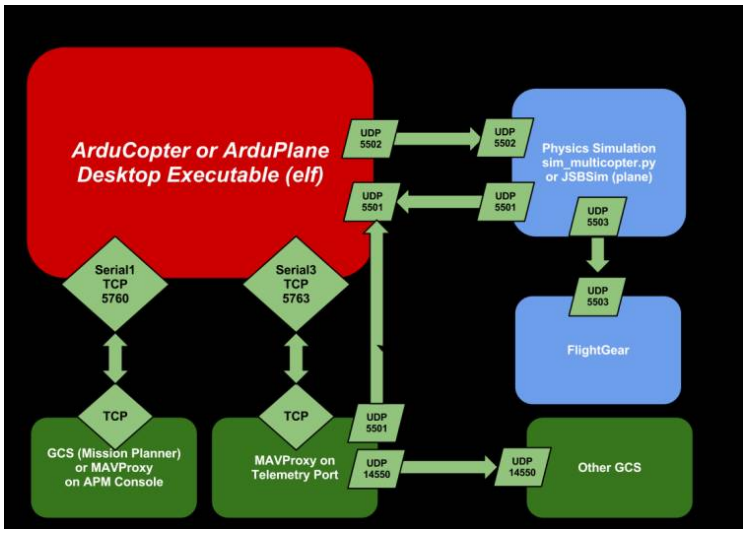


Figure 2.10: ArduPilot SITL Simulator structure [4].

2.4.4 Types of UAVs

UAVs come in multiple shapes and forms. The main types are multi-rotor, fixed-wing and hybrid-wing, with their marked share in that order. [32] Different types of drones have their own strengths and weaknesses and some are better suited for

certain kinds of tasks then others. [24] Below we go through the main types of drones listed above.

Multi-rotor

Multi-rotor is the kind of drone you usually think about when when drones are mentioned. This type of drone consist of a central body containing most electronics and usually four or six arms extending out horizontally with a motor and propeller on each of them. The advantages of this type of drone is easy control, takeoff and landings. High mobility in all directions and the ability to hover stationary is also some very large advantages. The main disadvantage multi-rotors that the UAV need to spend a lot of energy just to keep itself from falling out of the sky. [24]

Fixed-wing

A fixed-wing drone is more similar to a small airplane. This type of drone have fixed wings and usually one forward or backward facing propeller and flight control surfaces controlled by servos. The main advantages of this drone type is very good flight endurance resulting in long range and high flight time and can therefore cover large areas in one flight. The disadvantages is that landing and takeoff requires a suitable area and it cannot keep itself flying below a minimum speed. Autonomous takeoff and landing may require more sensors and be harder to do than on multi-rotor drones. [24]

Hybrid-wing

A hybrid-wing drone tries to combines the advantages of multi-rotor and fixed-wing drones. There are many different solutions for hybrid-wing drones with their own advantages and disadvantages, a survey comparing them can be found in the cited paper. [42]

2.4.5 UAV components

For the UAV we can either use a prebuilt and commercially available UAV or build our own tailored to this project. In both cases the UAV requires a set of standard components. With the exception of the GPS all other listed components are required for the UAV to fly.

Table 2.1: UAV main components and their functions

Part	Description
Flight Controller	The “brain” or main computer of the UAV.
ESC	Electronic Speed Controller, controls motor speed.
Motor	Converts electrical energy into mechanical energy. Makes propeller spin.
Propeller	Converts rotational power to trust.
Radio receiver	Communicates wirelessly with the radio transmitter.
GPS/compass	Used to find UAV position and orientation.
Frame	Holds everything together.
Battery	Provides electrical energy to the UAV.

2.4.6 Flight modes

Modern UAVs have multiple flight modes available for use. The many different flight modes are each suited to different usages and situations. The flight modes range from the normal radio transmitter controls you expect to find on a drone to GPS assisted modes or autonomous flight. There is even a flight mode for doing automated flip mid air. The selection of flight modes are either done through a switch on a radio transmitter, mission commands or over MAVLink from a GCS or companion computer. [1]

Table 2.2: Flight modes

Flight mode	GPS required	Description
Stabilize	No	Self-levels the roll and pitch axis. Full control by the user. May be hard to control for beginners.
Alt Hold Altitude Hold	No	Holds the current altitude, usually using a barometer combined with an accelerometer. The UAV self-levels when the controls are released. This mode is much easier to control than the Stabilize mode, here the user can change the altitude by moving the throttle and move in any direction by tilting the UAV. Altitude control and horizontal control are independent of each other. If the user let go of the control the UAV would come to a stop and hold that position.
Loiter	Yes	Same as Altitude hold, but also uses GPS data to hold the current position. This is even easier than Alt Hold mode. The use of GPS data nearly eliminates any drifting while the drone tries to hold a position. This mode is also less susceptible to wind affecting the UAV.
Land	Optional	Tries to perform a landing by slowly reducing the altitude until the drone reaches ground level. Usually triggered by a fault condition like low battery or the signal from the radio transmitter is lost.
RTL Return to launch	Yes	Goes to a predetermined altitude and then returns to the takeoff location.
Auto	Yes	Autonomously executes a predefined mission stored on the flight controller. A mission may include takeoff, landing, waypoints and other mission commands. No user input required in this mode.

The relevant flight modes for use in our system. Flight mode names and descriptions from ArduPilot. [1] [34]

Failsafes

When a major problem is detected by the UAV a failsafe is triggerd. This could be from the battery running low, UAV is breaking the configured geofence boundaries or the contact with the radio transmitter controller is lost. Which mode it goes into can depend on what failsafe was triggered. The goal of a failsafe is to avoid loosing or damaging the UAV when something does wrong. The pilot can take back control after a failsafe have been triggered if they wish. The most common failsafe modes are RTL or land, the first one causes the drone to fly back to it's launch site and the other causes it to slowly descend until it makes contact with the ground.

2.5 Global navigation satellite systems

A GNSS is a system using satellites to provide positioning all around the globe. [21] Each satellite transmitts a highly accurate time signal. By using the time signal from multiple satellite a receiver can then estimate its location. The minimum number of satellites required to get a positional fix is four, this means you need line of sight to at least four different satellites to find your location. Each GNSS consist of between 24 to 30 satellites in different orbits around the earth. The systems also consist of a operational control segment on the ground for monitoring, maintenance and corrections.

Today there is four GNSS in operation. The oldest of these systems is GPS(Global Positioning System), it has been operated by the United States of America since 1978. It was originally only intended for military use but were later opened up for civilian use. GLONASS was formerly operated by the Soviet Union and is now operated by Russia, it have been operating since 1982 and is being used for both military and civilian purposes. The BeiDou system have been operated by China since 2000 and is used for both civilian and military purposes. The Galileo system have been operated by the European Union since 2016.

2.5.1 Accuracy

The accuracy of GNSS positioning usually varies from a level of 30 m to 50 cm. [27] The higher levels of accuracy requires more work and increased time and processing. With a good signal reception the GPS system can achieve an absolute error of 2m in the east/west and and north/south directions and 6m in the vertical direction when used for real time navigation.

Using GPS together with another GNSS systems can increase its accuracy. Many GNSS receivers support using both GPS and GLONASS at the same time for increased accuracy. [36] Many newer receivers also support Galileo and BeiDou. Using all of these GNSS together can reduce the time required to get a position fix by almost 70% and increase the accuracy by around 25% compared to only using a GPS receiver.

2.5.2 Dangers and signal jamming

Today many systems depends on GNSS, everything from navigation systems on board aerial and nautical vessels to mobile phones and drones. As systems become

more and more dependent on GNSS the potential danger and damages from any disruption of these systems are increasing too. [29] A system that is primarily dependent on GNSS could encounter potentially catastrophic failure if it were to fail or become unreliable. As all wireless systems are vulnerable to radio jamming. When a radio transmitter starts sending garbage on the same frequency as the GNSS it could end up overpowering the signal form the GNSS. There have been cases all around the world where small jammers caused issues for GNSS. There have also been episodes where the GPS signal in large areas in northern Norway experienced signal jamming.

2.6 Extended Kalman Filter

The Extended Kalman filter(EKF) is a nonlinear version of the Kalman filter. [41] [48] It's a tool that can be used to solve estimation problems like combining and filtering data from multiple sensors. A very suitable use for EKF is in UAVs. Here the data from GNSS, accelerometers, gyroscopes, barometers need to be combined into stable and reliable position and velocity estimates. [33] The EKF does this by running continuous cycles of prediction and filtering. A big advantage of the EKF over simpler alternatives is it's ability to detect sensors with significant deviations and become less influenced by data from the unreliable sensor.

2.7 Regulations of drone use in Norway

The regulations regarding drone use in Norway falls under Luftfartstilsynet. As of 1. of January 2021 Norway adopted the drone usage regulations of the European Union. [22] Because this change is happening during the writing of this thesis we will only focus on the new regulations as they will be relevant in the future. Different types of drones usage falls under different categories, those are elaborated on below.

Open

The open is where most low risk and leisure drone activity happens. The main points of the category is:

- The drone must weigh less than 25 kg.
- The pilot needs to maintain visual line of sight (VLOS) with the drone.
- The drone must always fly within 120 meters from the closest point of the earth.
- No carriage of dangerous goods.
- No dropping of items.
- The drone must be marked with the operators registration number if it's equipped with a camera or the weight is 250g or above.
- Pilots of drones with a weight of 250g or above need to complete a basic online exam.

The open category consist of three subcategories, A1, A2 and A3. A1 is for smaller drones with a weight up to 900g and allows you to fly over uninvolved people but not over crowds. A2 is for small medium drones with a weight up 4kg and allows you to fly as close as 30m to uninvolved people to fly close to people or 5m if the drone is operating in a low speed mode of max 3m/s. A3 is for larger drones with a weight up to 25kg and requires you to fly far from uninvolved people and at least 150m away from residential, commercial, industrial or recreational areas.

Specific

The specific category is for riskier operations that do not in the open category. To operate in this category you need an operational authorisation from the National Aviation Authority where they are registered, unless the operation is covered by a Standard Scenario. In this category beyond visual line of sight (BVLOS) can be conducted if specific requirements are met.

Certified

The certified category is for operations with the highest level of risk. Passenger transport and autonomous cargo carrying drones fall into this category. Because of the high risk involved the drone need to be certified. Also the operator need an air operator approval. The person piloting the drone need a pilot licence.

2.8 MAVLink

MAVLink is a very lightweight messaging protocol for primarily communicating between a GCS and unmanned vehicles. [7] But also other MAVLink enabled devices can be used. MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern, data streams are sent or published while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission.

MAVLink as a binary protocol that can run over anything like physical wires, radio transmitters, mobile networks, satellite links or traditional IP networks.

2.8.1 System and component IDs

All MAVLink networks consist of devices with a system ID and one or more component IDs. [16] Every message sent contain the sender's system and component ID and some types of messages also contain the intended recipient. When there is a intended recipient the message contains the fields target_system and target_component. A message can also be intended for multiple recipients using broadcasts. You can read more about how this works in subsection 2.8.5.

Both the system ID and the component ID allow 256 unique values, this is because each ID is represented using 8 bits or one byte. Out of the possible 256 values, the ID 0 is reserved for broadcasts.

System ID

The unmanned vehicles usually have default system ID of 1. [16] If more unmanned vehicles are added to the MAVLink network, their system ID should increment to a unique unused value. GCS systems usually use the high end of possible system IDs, like 255. If multiple GCS systems are used in a MAVLink network each of them should have their own unique system ID.

Component ID

Component IDs are used for the different types and instances of onboard hardware or software on a unmanned vehicle. This might be autopilot, cameras, servos, GPS systems, avoidance systems etc. [10]. A components must use the appropriate component ID in their source address when sending messages, eg. the autopilot related messages are sent under the autopilot component ID while a MAVLink enabled camera sends messages under a camera component ID. Components can also use IDs to determine if they are the intended recipient of an incoming message. [16] The complete list of appropriate component IDs is available in the cited source.

2.8.2 Dialects

A MAVLink dialect is a set of messages, enums and commands that can be used for communication between MAVLink enabled devices. The dialect can contain both protocol and vendor-specific messages, enums and commands. The standard dialect is common.xml, it contains the "universal" messages used to control the drone and retrieve telemetry data. The ardupilotmega.xml dialect is an extension of the common.xml dialect that adds support for many ArduPilot specific messages and other MAVLink enabled devices like gimbals, cameras, rangefinders and more. A dialect is defined in a dialect XML file that can easily be modified by hand and then compiled to MAVLink libraries using mavgen from pymavlink. Pymavlink supports a wide range of programming languages, but only the C and Python languages are officially supported. [9] [11]

Messages

MAVLink uses message definitions to decide how to encode and decode the content of sent and received messages. The message definitions are from the configured MAVLink dialect. Only the messages found in the definitions can be understood and no new messages can be added at runtime. Each message definition have their own unique id, this is used to identify what message is being received and how to decode the payload. MAVLink 1 can support up to 256 unique message definitions, while MAVLink 2 can support up to 16 million unique message definitions. A list of all official message definitions can be found at [13]. An example of a message is shown in Table 2.3.

Table 2.3: MAVLink Message: GLOBAL_POSITION_INT

Field Name	Data type	Units	Description
time_boot_ms	unit32_t	ms	Timestamp (time since system boot).
lat	int32_t	degE7	Latitude (WGS84, EGM96 ellipsoid)
lon	int32_t	degE7	Longitude (WGS84, EGM96 ellipsoid)
alt	int32_t	mm	Altitude (MSL). Positive for up.
relative_alt	int32_t	mm	Altitude above ground(from takeoff) Positive for up.
vx	int16_t	cm/s	Ground X Speed (Latitude, positive north)
vy	int16_t	cm/s	Ground Y Speed (Longitude, positive east)
vx	int16_t	cm/s	Ground Z Speed (Altitude, positive down)
hdg	unit16_t	cdeg	Vehicle heading (yaw angle), 0.0..359.99 degrees.

This message provides a filtered global position (e.g. fused GPS and accelerometers). It is designed as scaled integer message since the resolution of float is not sufficient and consistent. The degE7 format gives an accuracy of 1.1cm. [13]

Enums

Enums are a data type used to represent a group of constants. Constants being permanent unchangeable variables. Usage of enums makes code easier to read and understand and reduces the chance of typos. An example of how useful enums are for improving code readability is instead of remembering that a 16-bit signed integer have the data type value 4, you can instead just write MAV_PARAM_TYPE_INT16 or for a 32-bit floating point number you can write MAV_PARAM_TYPE_REAL32 instead of 9. [13]

Commands

Commands are as the name suggests defined commands that can be issued to a MAVLink enable vehicles. [47] This includes setting waypoints, mission parameters like changing the flight speed or starting missions. Each command can have up to 7 parameters. There is two types of commands, the ones that can be executed imminently and the long running commands that takes time to complete.

2.8.3 MAVLink 1 vs MAVLink 2

The change from MAVLink 1 to MAVLink 2 expands the capabilities of the protocol. [19] [18] One of the largest changes is the use of 24 bit message IDs, this allows the use of over 16 million unique messages in a dialect, up from 256 with MAVLink 1. Another addition was the support for packet signing, this can be used to ensure the message was sent by a trusted system. The support message extensions was also added, this allows adding new optional fields to the message definitions without breaking binary compatibility for receivers. Compatibility and incompatibility flags was also added, the flags are used to indicate how incoming packets should be handled. Packets with a unsupported can still be handled in the standard way, while a packets must be dropped if the incompatibility flags are not supported.

2.8.4 Serialization

Even if the way to store MAVLink message in memory can vary greatly from system to system the MAVLink message sent over the wire(serialized) is strictly defined. There is two different formats for this, MAVLink v1 and MAVLink v2. The MAVLink v2 format have additional features over MAVLink v1, we will go into more details about this in subsection 2.8.3.

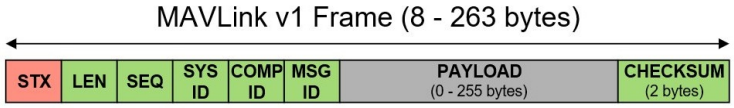


Figure 2.11: MAVLink 1 serialized packet format. [17]

Figure 2.12: MAVLink 2 serialized packet format. [17]

Magic byte(STX)

Every MAVLink frame starts with a magic byte, this is 0xFE(254) for MAVLink 1 packets and 0xFD(253) for MAVLlink 2 packets. This is also known as a Protocol-specific start-of-text (STX). This is used to detect the beginning of a MAVLink frame.

Length

This is the length of the payload. This is used for parsing the message. When MAVLink 2 is used this is the length of the payload after zero truncation.

Incompatibility and compatibility flags

MAVLink v2 also have one byte for incompatibility and one byte compatibility flags following the payload length [17]. If a receiver does not understand any incompatibility flag, it is required to drop the MAVLink packet. The compatibility flag can be used to indicate a packet should get a higher priority than other packets. It's not required for the receiver to understand any compatibility flag as the packet itself is not affected.

Packet sequence number

The packet sequence number increments for every sent message from sent from the specific component. This goes up to 255 and then loops back to 0. By checking if any sequence numbers are skipped on the receiver we can the detect packet loss on the MAVLink connection.

System id

The system id of a MAVLink message is used to identify the sender of the message. This is the same system id as described in subsubsection 2.8.1.

Component id

The component id of a MAVLink message is used to identify the sender of the message. This is the same component id as described in subsubsection 2.8.1.

Message id

This is the same message id as described in subsubsection 2.8.2.

Payload

The payload contains the message fields defined by the message definition. A field can have one of the following data types: uint64_t, int64_t, double, uint32_t, int32_t, float, uint16_t, int16_t, uint8_t, int8_t or char. During transmission the message fields are reordered by the data type size, with larger data types at the beginning of the payload [17]. A field can either be a single value or an array. When using MAVLink 2 optional fields called extension fields can also be included at the end of a message. On MAVLink 2 zero truncation is done, this involves removing all tailing zeros from the payload, this shortens the packet itself.

Checksum

The checksum is a two byte value that is computed using CRC-16/MCRF4XX from the content of the message. The sender computes it and adds it to the end of the message when sending. The receiver also computes this value after decoding the content of the message. If the received checksum does not match the computed checksum the content of message is most likely corrupted and therefore disregarded.

Signature

By attaching a signature to every message we can verify that the received messages are from a trusted source. [12] This is only possible on MAVLink 2.

Figure 2.13: MAVLink 2 message signing. [12]

The signature consist of three separate parts, the link ID, the timestamp and the signature. The link ID identifies over which link the MAVLink message was sent. This is only significant for multi-link MAVLink systems. The timestamp is the time since 1st January 2015 in 10th of ms. The signature is the first 48 bits of a SHA-256 hash of the entire packet. There is also a incompatibility flag set on every signed packet, indicating receivers not supporting signing most drop that packet. [12]

Number encoding

All numbers are encoded starting with the least significant byte and ending with the most significant byte. A benefit of encoding numbers in this way is that we can truncate the tailing zeros of potentially large numbers. An example of this is if the message ends with a U_INT32 containing the number 12, only the least significant byte of the number is being used, the three remaining bytes only consisting of zeros.

2.8.5 Routing

MAVLink network typically consists of the drone itself and the GCS, but a MAVLink network can be made of many systems all connected together, each system with one or more components [16]. The network is connected by links between different components, much like a modern computer network. As a modern computer networks, MAVLink networks also have routing protocols and rules to determine where to send a packet. When having multiple components in a MAVLink network there is a need to determine where each message should be sent, this is done by a set of routing rules.

Rules used to determine if a packet should be forwarded:

- Not forwarding packages it does not understand(not in the component's message definition.)
- Some MAVLink messages contain the fields target_system and target_component, if this is the case forward the message on a link leading to the target.t this is used to indicate an intended receiver for the message.
- A broadcast message where the target_system field is 0 or not included should be forwarded on every link(except the link it was received from if re-transmitting).

There is also rules that determine if a receiver should process a message:

- If the message is a broadcast message.
- If the target_system field matches the receiver's system id and target_component field is broadcast, matches the receivers component id or the component id in the target_component field is unknown.

Some MAVLink messages contain the fields target_system and target_component, this is used to indicate an intended receiver for the message. There is also a need to keep track of links to figure where to find another system or component.

An example of routing is when the autopilot have a link to the GCS and another link to an on board computer. A message from the GCS is not sent directly to the on board computer and vice versa, but instead routed through the autopilot component.

2.8.6 Microservices

Microservices are higher level protocols running over the MAVLink network using MAVLink messages. [15] The most important and relevant microservices used in our system is elaborated on below. A major point of the protocols is the built in retransmission of messages that can be lost over a unstable link.

Heartbeat Protocol

The heartbeat protocol is used advertise the existence of a component to the MAVLink network. [14] All components broadcast their heartbeat regularly and looks out for heartbeats from other components or systems. The heartbeat contains the system id, component id, vehicle type, flight stack, component type, flight mode, and system status of the sender component. It's also used to determine which links and components are healthy and used for routing. A heartbeat is usually broadcasted at 1 Hz. A heartbeat should only be broadcast if the component does not have major problem.

Mission Protocol

The mission protocol allows a GCS to upload, download and clear flight plans on the autopilot. This is critical for setting up autonomous flight missions. The protocol also sends notifications when the current mission item changes, eg. the UAV reaches a waypoint.

Parameter Protocol

The parameter protocol is used for both retrieving and setting parameters on a MAVLink component. A parameter can be everything from the max acceleration of the unmanned vehicle to how it should read the battery voltage. The protocol allows to both read a specific parameter or all parameters from a component. Parameter names can be up to 16 characters. Values can be 8, 16, 32 and 64-bit signed and unsigned integers, and 32 and 64-bit floating point numbers. [15]

Command Protocol

The command protocol guarantees the delivery of MAVLink commands to a MAVLink enabled vehicle. This is done by using acknowledgements and retransmission when no response is received. It supports both commands that can be executed imminently and long running commands. Each command can have up to 7 parameters. The result from the execution of the command is sent in return. The progress of long running commands can be tracked or the long running command can be cancelled.

There are two types of MAVLink command messages, COMMAND_INT and COMMAND_LONG, in the former parameter 5 and 6 is a scaled integer and allows higher precision when using commands containing coordinates. The rest of the parameters are floats.

2.8.7 Security Threats

The MAVLink protocol have multiple security issues, this is not a focus or very relevant for our research and we wont discuss it in detail. You can read more about this topic in [34, p. 8].

2.9 Signal attrition

The strength of a wireless signal is most often measured in dBmW, this is a logarithmic scale where the reference value is one mili watt. One of the reasons for using a logarithmic scale it that it makes the calculations much simpler you stupidly large or small numbers.

2.9.1 Free-space path loss

Free-space path loss is signal attrition in air without any obstacles. [51] The formula for calculating the free-space path loss is given as

$$L_{FS} = 100 + 20log_{10}(d) \tag{2.1}$$

with d as the distance between the transmitter and the receiver. An easier way to calculate this to remember a doubling of distance equals to a reduction in signal strength of 6dB.

2.9.2 Near-Ground Path Loss at 2.4 GHz

When a radio signal travels close to the ground it behaves differently compared to when traveling in free-space. [51] The signal can get reflected of the ground may partially cancel out the line of sight signal through destructive interference. This makes calculating the path loss a lot more complicated. The plane earth model can be used to calculate the path loss near the ground and it is given as

$$L_{PE} = 40log_{10}(d) - 20log_{10}(h_t) - 20log_{10}(h_r) \tag{2.2}$$

With d as the distance in meters, h_t as the height of the transmitter and h_r as the height of the receiver in meters. An easier way to calculate this to remember a doubling of distance equals to a reduction in signal strength of about 12dB. As shown by [51] this model is not always as accurate and my vary based on the terrain and other factors.

2.9.3 Path loss through vegetation and animals

There is also path loss when the signal is passing through anything on the path between the transmitter and the receiver. In our case the most relevant obstacles would be vegetation or the animals them self. Radio signals at 2.4GHz are easily absorbed by water and turned into heat in a process called dielectric heating, this is the same process happening in your microwave at home. Animals and plants are the main obstacles we could encounter with our system. As they consist mostly of water the graph in Figure 2.14 could be a close enough approximation for calculating the path loss through this type of obstacles.

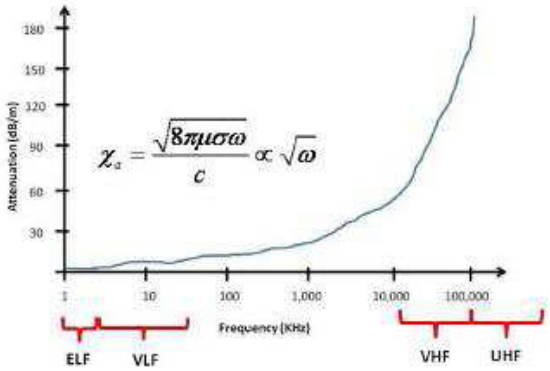


Figure 2.14: RF attenuation in sea water. [35]

2.10 Signal interference

Wireless communication have always had the issue where it could be affected by other radio signals. This happens when another radio transmits on the same frequency simultaneously. The amount of interference depends on how strong the signals are compared to each other. If the other signal is strong enough it could overpower the desired signal and cause enough interference that we can't decode it anymore.

The 2.4GHz spectrum is used by a lot of different devices. After the Federal Communications Commission decision in 1985 to open up the ISM band for unlicensed use the number of devices utilizing the band have exploded. [20] It's now used by everything from WiFi and Bluetooth to microwave ovens and wireless controllers.

Chapter 3

Method

In this chapter we will look at how the system functions and the system architecture. We will go though the separate components of system and explain their function. Then we will show how everything fits together as a whole. We will explain the challenges and solutions we found while designing this system. The reasoning for our choice of UAV and its components will also be here.

We will also explain how our testing was conducted, what we wanted to achieve with the tests and any problems we encountered.

3.1 System Architecture

Our system consist of the GCS, the UAV and SheepRTT modules. Each of these components consist of multiple sub component in either software or hardware. Our main focus now will be on the UAV and the MAVLink communication but we will also discuss the other components being interacted with. The system architecture is shown in the figures below. The system requires the GCS, UAV and SheepRTT modules all working together to locate the animals.

The same system architecture description is also found in [44] and [45].

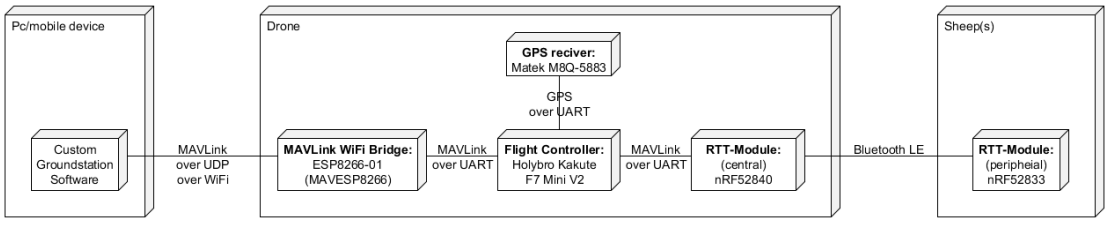
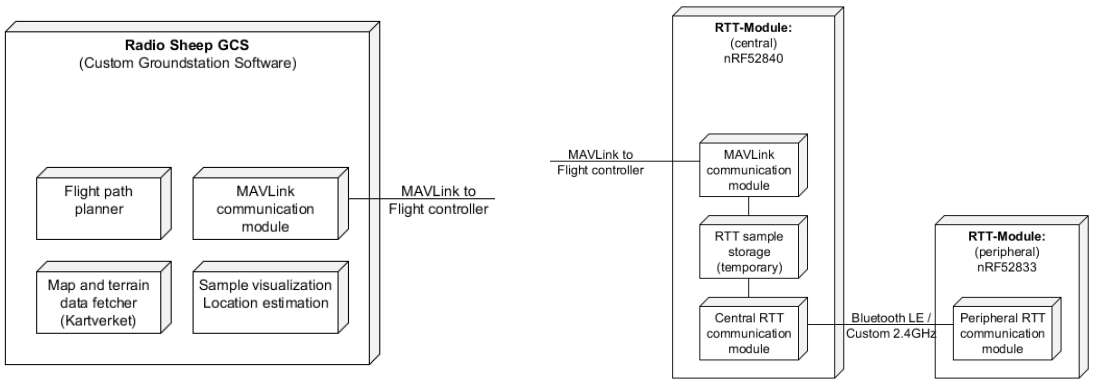


Figure 3.1: The main components of the system.



(a) GCS sub-components (b) SheepRTT sub-components

Figure 3.2: GCS and SheepRTT sub-components

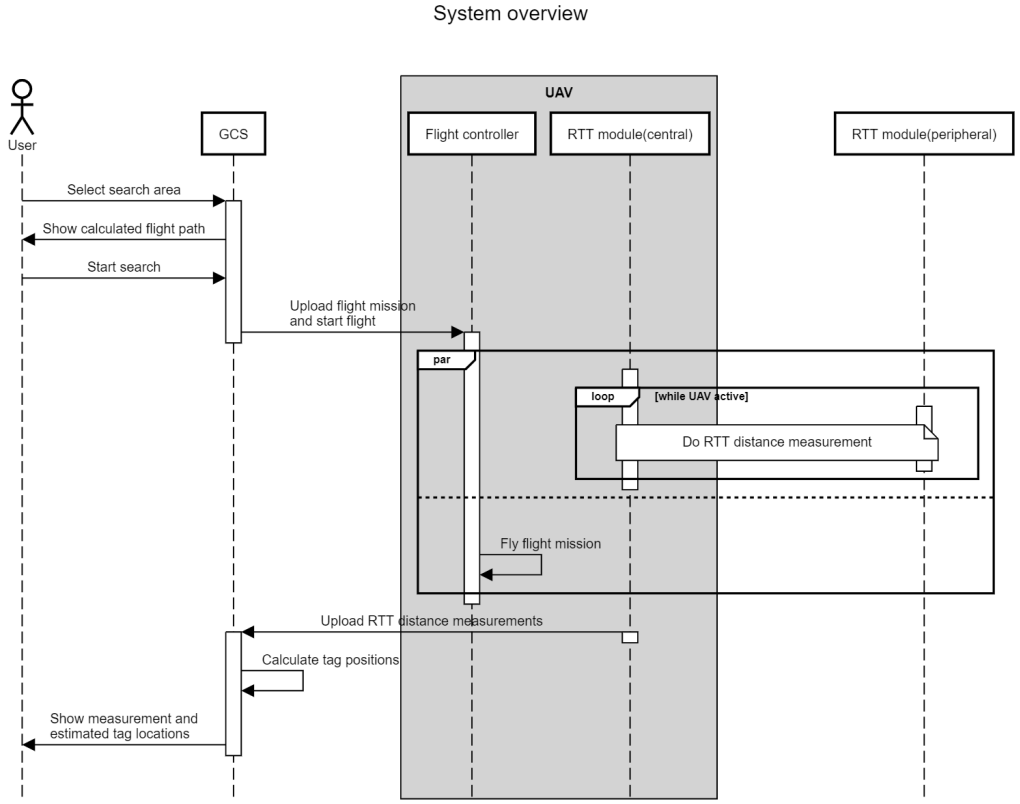


Figure 3.3: Overview of system.

3.1.1 SheepRTT

The SheepRTT system provides distance measurements between the UAV and the tags worn by the animals. It consists of two different types of modules, each with different a purpose and function. One of the two types are the central module, located on the UAV. And the other is the peripheral modules located on the animals. When a central module is in the range of a peripheral module it can calculate the distance the peripheral module by measuring RTT. The distance measurement is then saved together with positional data and can later be used to determine the

position of a tag or peripheral module. The positional data is provided by the GPS on the UAV.

The main focuses of this solution is providing small size, low weight and long battery life to the tags worn by the animals.

3.1.2 UAV

The purpose of the UAV is to act as a platform to transport the SheepRTT central module. The UAV also provides a communication channel for the SheepRTT module to connect with the GCS. The UAV also provides GPS data to the module. We will look closer at the UAV later in this chapter.

3.1.3 Radio Sheep GCS

Radio Sheep GCS is the piece of software the user will interact with. [44] It's used to control the rest of our system and it's an integral part of it. It does everything from flight planning to calculating the positions of the SheepRTT tags. When opening the software the user is greeted with a map and connection options for the UAV.

Connecting to UAV

The UAV creates it's own WiFi network for MAVLink communication. To connect to this network you simply connect to it as you would with any other WiFi network. After this the GCS can connect to a UDP port on the UAV to establish communication. Radio Sheep GCS will handle all the communication with the UAV and the nRF module.

Area survey planning

When Radio Sheep GCS is connected, the UAV's position will be shown on the map. Radio Sheep GCS allows us to select an area of the map to search for animals and will plan an efficient flight plan to cover this area. Flight height and other parameters for the flight plan can also be tweaked. When the user is satisfied with the flight plan it can be uploaded to the UAV with the click of a button, then the UAV is ready to take off. With the click of another button it will take off autonomously and fly the planned path, while searching for SheepRTT tags. After this the UAV will return and land automatically.

Terrain data is used when planning the flight path, this allows us to fly over uneven terrain with a somewhat constant height above ground.

Measurement visualization and position estimation

When the UAV have returned and the MAVLink connection have been reestablished the SheepRTT module aboard the UAV will upload all the RTT distance measurements to Radio Sheep GCS. The distance measurements can then be visualized on the map and used to calculate the position of each tag or animal. An example of a visualization of the distance measurements is Figure 4.2. An example of the calculated tag positions is Table 4.6.

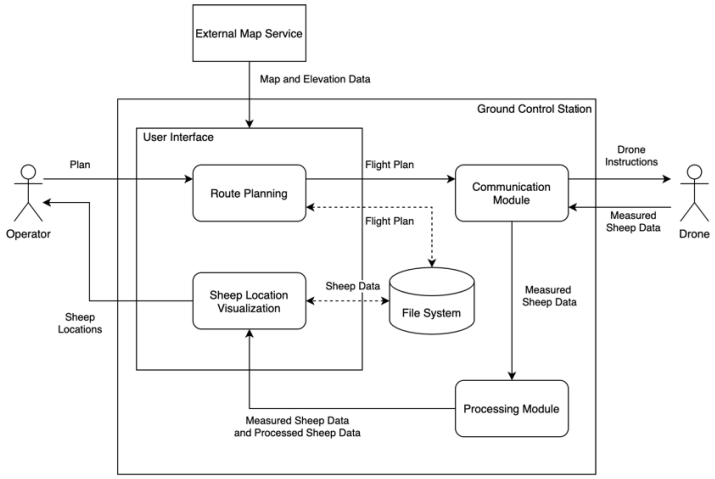


Figure 3.4: Overview of Radio Sheep GCS. [44]

3.2 SheepRTT

The SheepRTT modules is the central part of our system. The SheepRTT modules performs the actual distance measurement and exchange them with other parts of the system.

3.2.1 Central and peripheral boards

The system consist of two different RTT modules. The first is a nRF module known as a tag or a peripheral module, this advertises it's existence at regular intervals and responds to initiation of RTT distance measurements. The second is a nRF module programmed to listen for advertisements from the peripheral modules and initiate a RTT distance measurement when found. The second module will be referenced as the central module.

3.2.2 RTT distance measurements

A short explanation of RTT distance measuring with can be found in subsection 2.3.4. The details about our RTT distance measurements is covered in Grzegorz report [45]. A sequence diagram of a RTT distance measurement is shown in Figure 3.5.

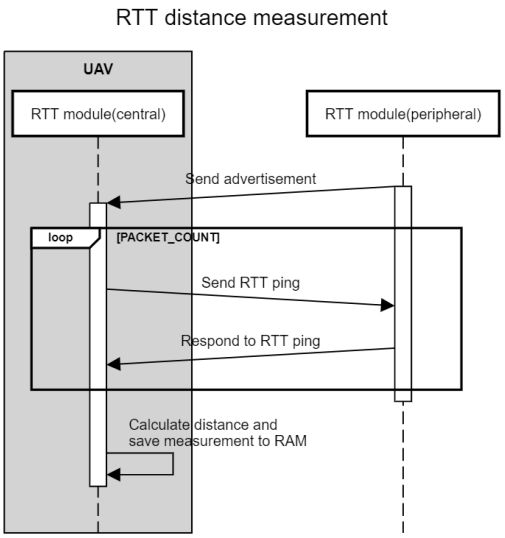


Figure 3.5: RTT distance measurement sequence diagram.

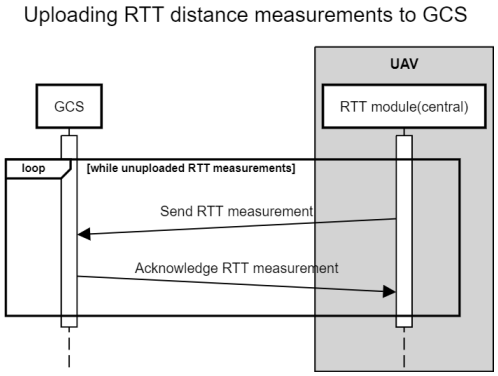


Figure 3.6: RTT distance measurement upload sequence diagram

3.2.3 RTT Implementations

We have developed two different implementations with their own strengths and weaknesses. The implementations are not compatible with each other. This means implementation #1 can't connect to implementation 2# for RTT distance measurements. There is no difference between the implementations on the MAVLink side. To change between the different implementations the SheepRTT module is flashed with the desired implementation.

#1 - BLE based

Our first implementation utilizes BLE protocol. This makes the implementation connection-oriented. Compared to our other implementation this have a higher power consumption and lower accuracy. The use of BLE makes this implementation compatible with BLE-enabled customer devices like mobile phones and laptops.

#2 - Connectionless

This implementation is provides lower power consumption and higher accuracy. This in turn gives better battery life on the tags and more accurate distance measurements. It is not compatible with BLE customer devices.

3.2.4 System design

Early development

In the beginning our plan was to program the storage of RTT distance measurements as a software module in ArduPilot itself. This software module in ArduPilot would communicate with the SheepRTT module over UART and also communicate with the GCS over MAVLink. The SheepRTT module would just output RTT distance measurements to the UAV. This would have the SheepRTT module acting the same

way as a normal sensor like a temperature sensor or a barometer. During our work we encountered many downsides with this solution, mainly the large effort required to integrate this into the ArduPilot codebase. This would also require a custom data format to communicate between ArduPilot and the SheepRTT module. But the biggest problem would be the need to compile our new software module and flash ArduPilot to any UAV in order for it to function with our system.

New MAVLink focused design

Working on implementing the first design we found a new solution. This new solution would be easier to implement and more robust. The big changes from the previous design was to implement MAVLink communication directly on the SheepRTT module. By designing the system this way the recompilation of ArduPilot would no longer necessary and no custom data format between ArduPilot and the SheepRTT module would be required. This would also make our system compatible with any ArduPilot powered UAV without making any modifications to it. With this design the flight controller would route MAVLink messages between the GCS and the SheepRTT module. Another major advantage from this design was the ability to store multiple distance measurements in the SheepRTT module while the connection to the GCS was unavailable and then send all the measurements once communication was reestablished.

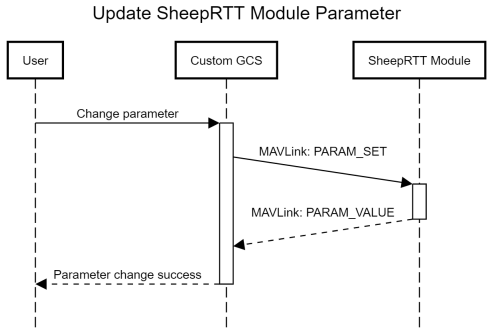
3.2.5 MAVLink Implementation

The communication between the different components in the system is vital to everything functioning. This communication happens over the MAVLink messaging protocol. Here the module requests and receives and the required data, transmits all distance measurements and the module's parameters can be updated.

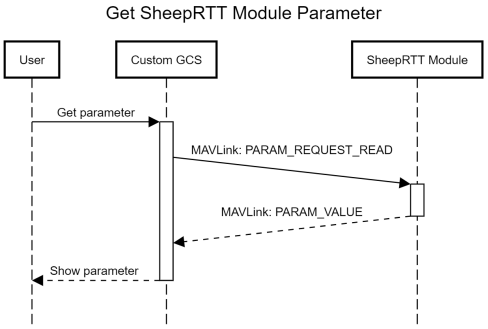
The supported features for the MAVLink implementation on the SheepRTT module:

- Transmit RTT distance measurements to GCS.
- Retrieve GPS information from the UAV in real time.
- Ability to tweak RTT module parameters.
- Support for the MAVLink Heartbeat protocol.

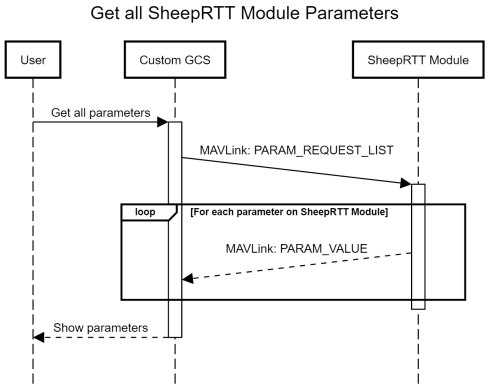
MAVLink communication sequence diagrams



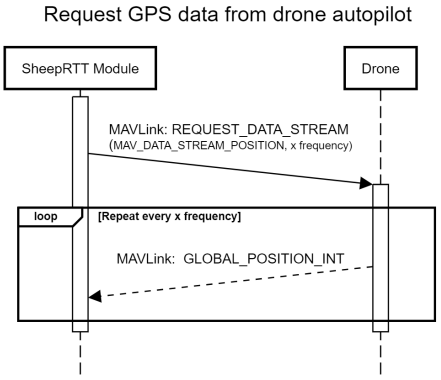
(a) Sequence diagram for setting a parameter on the SheepRTT module.



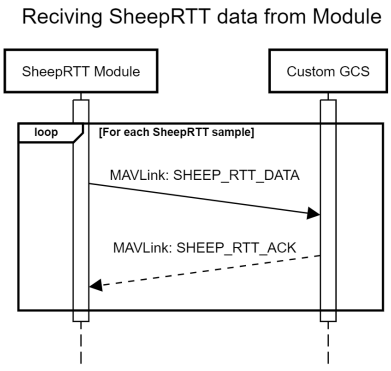
(b) Sequence diagram for getting a parameter from the SheepRTT module.



(c) Sequence diagram for getting all parameters from the SheepRTT module.



(d) Sequence diagram for the SheepRTT module requesting GPS data from the drone autopilot.



(e) Sequence diagram for the SheepRTT module sending SheepRTT data to the GCS.

Figure 3.7: SheepRTT module MAVLink communication.

MAVLink dialects

For the purpose of transmitting the RTT distance measurement data from the SheepRTT module we decided to add two custom MAVLink messages. The most orderly way to accomplish it to add a new MAVLink dialect with our MAVLink messages. We named our new dialect **sheeprtt** and added our custom messages here. To combine this dialect with the ardupilotmega dialect used by ArduPilot we created

a new dialect named **ardupilotmega_sheeprtt**, this dialect inherited everything from the **ardupilotmega** and **sheeprtt** dialect. With this we could easily generate MAVLink libraries for any programming language supported by MAVLink's library generator.

Added MAVLink Messages

We added two new messages, a message for RTT distance measurement itself and a message for acknowledging the reception of this distance measurement. The former is sent from the SheepRTT module to the GCS. The format of this message is partially based on the position format in MAVLink Message: GLOBAL_POSITION_INT. The other message is sent by the GCS to the SheepRTT module to acknowledge the reception of a RTT distance measurement.

Table 3.1: MAVLink Message: SHEEP_RTT_DATA

Field Name	Data type	Units	Description
seq	uint32_t		Sequential sample id (from power on).
timestamp	uint32_t	ms	Time since module power on.
lat	int32_t	degE7	Latitude (WGS84, EGM96 ellipsoid)
lon	int32_t	degE7	Longitude (WGS84, EGM96 ellipsoid)
alt	int32_t	mm	Altitude (MSL). Positive for up.
dis	uint16_t	m	Distance between the drone and the tag.
tid	uint16_t		Identifier for the tag.
rssi	int8_t	dBmW	Signal strength.

Message for transmitting SheepRTT distance measurements. Contains the position of the drone when the measurement was taken, the distance to the tag, the tag's id, RSSI(signal strength), timestamp and Sequential sample id.

Table 3.2: MAVLink Message: SHEEP_RTT_ACK

Field Name	Data type	Units	Description
seq	uint32_t		Sequential sample id (from power on).

Message used to acknowledge the reception of SHEEP_RTT_DATA packets at the GCS. This is sent from the GCS to the module. Allows the module to free up memory by not keeping already received measurements.

MAVLink Data encapsulation

The MAVLink routing used by ArduPilot does not forward MAVLink messages it does not understand. [3] This means our newly added MAVLink messages wont be forwarded by the flight controller to the GCS or other components in the MAVLink network. One way remedy this problem is to recompile the ArduPilot software with our **sheeprtt** MAVLink dialect added.

Another way to solve this problem is by encapsulating our messages in generic data messages. The messages DATA16, DATA32, DATA64 and DATA96 are able to hold up to 16, 32, 64 or 96 bytes of binary data. Because those messages are part of the **ardupilotmega** MAVLink dialect, any component supporting this dialect will

also support forwarding our encapsulated messages without any additional configuration. [8] We choose the DATA64 message to encapsulate our SHEEP_RTT_DATA message and the DATA16 message for our SHEEP_RTT_ACK message. DATA64 and DATA16 was chosen because they are the smallest data messages that can fit our messages.

Encapsulation introduces overhead when transferring data. The following overhead calculations assume the use of MAVLink 2 with zero truncation. This overhead comes from the additional header, checksum and the other fields in a DATA message. The overhead for using encapsulation is 14 bytes (10B MAVLink header, 2B checksum, 1B payload.type, 1B payload.length) per packet on MAVLink 2. If the DATA packets are sent over MAVLink 1 the amount of data changes. Because there is no zero truncation in MAVLink 1 the SHEEP_RTT_DATA increase in size. The total overhead for using encapsulation is a 65% assuming one SHEEP_RTT_DATA and SHEEP_RTT_ACK is sent per distance measurement on MAVLink 2.

Table 3.3: Encapsulation overhead

Message	**Native size**	**MAVLink 1**		**MAVLink 2**	
		Encapsulated size	**Overhead**	**Encapsulated size**	**Overhead**
SHEEP_RTT_DATA	33 bytes	74 bytes	124.24%	47 bytes	42.42%
SHEEP_RTT_ACK	16 bytes	26 bytes	62.50%	30 bytes	87.50%
Average			93.37%		64.96%

Encapsulation makes it possible to use the SheepRTT module with any ArduPilot drone, avoiding the hassle of doing modifications to UAV except attaching the nRF module to any MAVLink or MAVLink 2 enabled UART port.

SheepRTT module power up sequence

When the module powers up it starts sending out a MAVLink heartbeat. This is while listening and waiting for a heartbeat from the UAV. This is defined as an autopilot component with same system id as the module. When a heartbeat from the UAV is received the module would start sending REQUEST_DATA_STREAM messages containing stream_id MAV_DATA_STREAM_POSITION to the UAV. When this is received by the the UAV it would start sending GLOBAL_POSITION_INT message to the SheepRTT module at regular intervals. This GLOBAL_POSITION_INT contains information about the current time and position and also information about the movement of the UAV. The content of this message is detailed in Table 2.3. This is important as the SheepRTT module depends on positional data from the UAV to save the distance measurements with accurate position data.

We needed to find a simple and robust way to upload the RTT distance measurements from the module to the GCS. At the GCS the measurements could be analyzed the position of the tags could be estimated. To make our data formats more consistent with already existing formats we chose to base our messages on the already existing GLOBAL_POSITION_INT message, keeping the latitude, longitude and altitude field and their format. We then added our own data from the distance measurements, containing the estimated distance between the UAV and

the tag. The identifier of the tag and the RSSI was also included. Furthermore we added a Sequence field to keep track of what messages have been received by the GCS. When the GCS receives a SHEEP_RTT_DATA message it will respond with a SHEEP_RTT_ACK message containing the same sequence number. The exact message format in the SHEEP_RTT_DATA message can be found in Table 3.1.

Module actions after power on:

1. Start sending out heartbeats.
2. Wait until a heartbeat is received from the autopilot.
3. Request the drone to send it GPS data at regular intervals.
4. Wait until a valid GPS position is received.
5. Wait until the heartbeats from the drone indicates it's armed(active).
6. Start taking measurements and save them to RAM.
7. Start sending the oldest saved measurement to the GCS at regular intervals.
8. When an acknowledge message is received for the measurement it's deleted from RAM and the next measurement gets transmitted.

MAVLink: Planning, testing and component simulation

Before we started integrating MAVLink into our GCS and SheepRTT module we simulated all the components of our system working together. We did this to learn more about using MAVLink and to find the optimal solution. We utilized Python and the pymavlink library to simulate each component with simplified internal logic but with the full MAVLink implementation. The flexibility and ease of programming with Python made this prototyping process much faster.

Only the SheepRTT module had all of it's functions implemented in the simulator. While the simulator for the GCS and UAV did only implement the basic functions required to test communication with the SheepRTT module. All simulated components implemented the Heartbeat protocol. The GCS simulator had the functionally receive RTT distance measurements, acknowledge them and visualize them. The UAV simulator can move around and respond to requests to share its position with other components. The position is shared by sending "request_data_stream_position" messages after a request for them is received. While the simulator for the SheepRTT module implemented all the functionally to simulate the real module, even simulating RTT distance measurements to virtual tags.

The use of pymavlink and flexible command line parameters allowed us to connect the simulators in whatever way we wanted. This could be to another simulator, the ArduPilot SITL or the real UAV. This flexibility allowed us to test any part of our system with a combination of simulated and real components. With this setup there was no need to have the of the other components available when developing on GCS or SheepRTT module.

The simulator for the SheepRTT module was used as a blueprint when implementing MAVLink on the real SheepRTT module. We also made a test suite for

the SheepRTT module and it's simulator. This test suite was used to ensure all components were functioning as intended and make sure any update did not break anything.

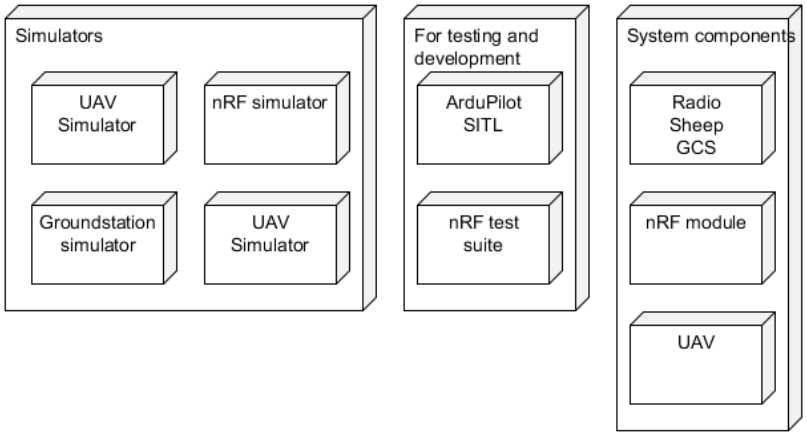


Figure 3.8: Overview of MAVLink components, with simulators and development components. Note: nRF = SheepRTT

3.2.6 Moving from development kit to smaller module

The Nordic Semiconductors nRF52833 development kit have a large size and high weight weight compared to our UAV. The weight of the development kit is 50g and this would be a large portion of our weight limit of our 250g for the UAV. A size comparison between the development kit and the UAV can be seen in Figure 3.9. We decided early that it would be more practical with a smaller module fulfilling the same role as the development kit. After researching multiple alternatives we opted for the MINEW MS88SF23 module. The deciding factors was the small size, low weight, low price and more accessible soldering points compared to other available modules.

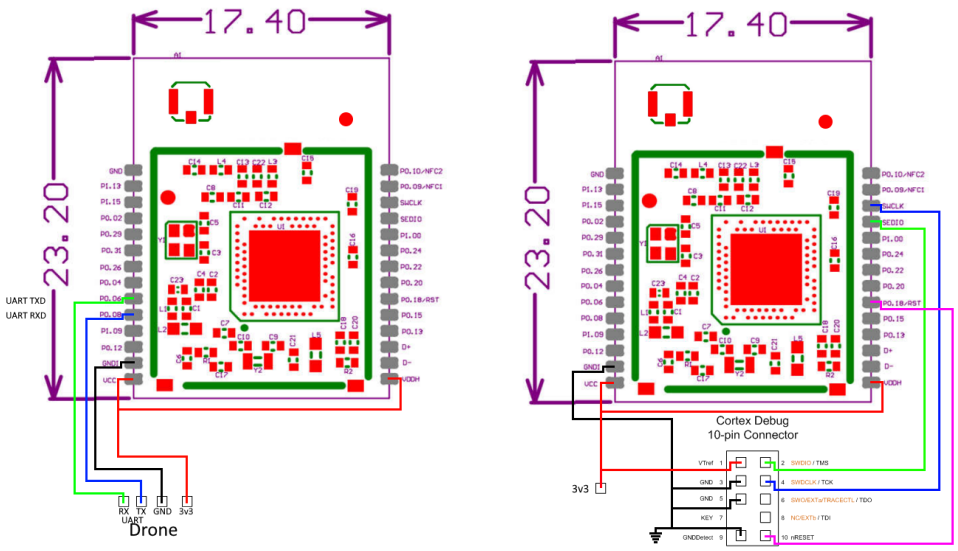
There was number of challenges when moving from the development kit to a smaller module. The largest of the challenges was the need to for an external programmer for the MINEW module. Luckily we could use the development kit as an external programmer for the module.

Figure 3.9: The UAV with a development kit connected over UART nRF52833 Development kit connected to the drone, powered by the drone and communicating over MAVLink over UART.

Wiring and diagrams

We used official documentation for both the development kit and the MINEW module to figure out the required wiring. Both the wiring required to connect them to the UAV and the wiring for programming the MINEW module. This can be found in Figure 3.10. The module gets powered by the 3.3V BEC on the flight controller. Connecting 3.3V to both the VCC and VDDH pin causes the nRF52840 to go into something called Normal Voltage mode instead of opposed to High Voltage mode. The Normal Voltage mode disables an internal voltage regulator and reduces the power consumption. The UART RX and TX from them module is connected to the UART TX and RX on the flight controller through the logic level converter. This was done because the flight controller uses a 5V logic voltage, while nRF52833 uses a 3.3V logic voltage.

The wiring for connecting the MINEW module to an external programmer is using the Cortex 10-pin debug connector at the programmer end and a custom connector for interfacing with the module. As the Cortex Debug connector does not provide power we needed to supply 3.3V from another source. The Serial Wire Debug (SWD) interface was used for programming the module and required the bi-directional data pin SWDIO and a clock with the SWDCLK pin. Also the reset pin RST were required for resetting the module and getting it into a programmable mode.



(a) The wiring for the MS88SF23 module when connected to the drone.

(b) The wiring for the MS88SF23 module when programming or debugging.

Figure 3.10: The wiring for the MS88SF23 module.

We decided to use male and female header connectors for connecting our cable to the module. This allowed us to connect and disconnect it easily. This decision allowed us to quickly change between connecting it to the UAV or an external programmer when making changes to the software on the module.

The module had a pitch of 1.1mm between each pin, meaning the distance each soldering point was 1.1mm. We did not have access to male and female header connectors with a pitch of 1.1mm. We instead decided to use headers with a pitch of 1.27mm as this was close enough for this particular use.

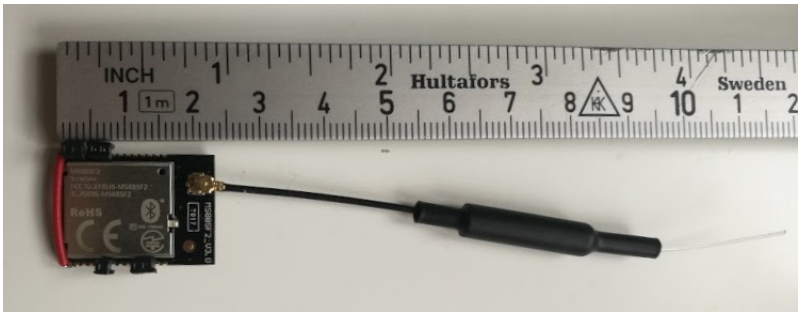
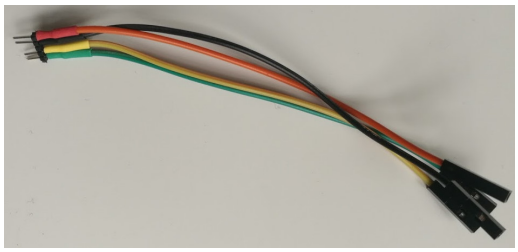


Figure 3.11: nRF module (MINEW MS88SF23) with connection headers and antenna attached.



(a) Custom cable between MINEW MS88SF23 module and the drone. Providing power and UART communication.

(b) Custom cable between MINEW MS88SF23 module and external programmer. For programming or debugging.

Figure 3.12: Custom cables for the MINEW MS88SF23 module.

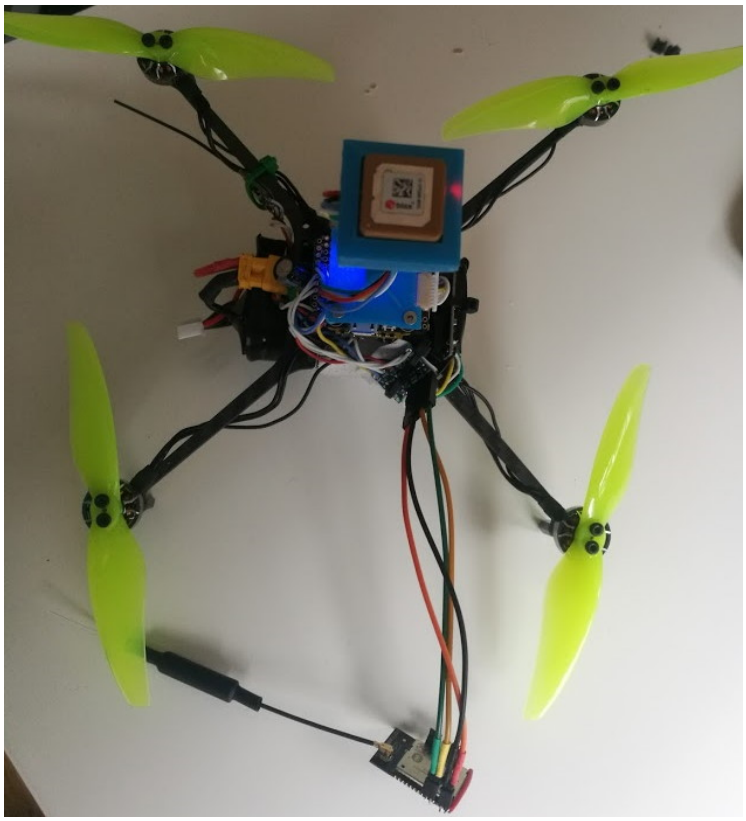


Figure 3.13: The drone with a MINEW module connected over UART

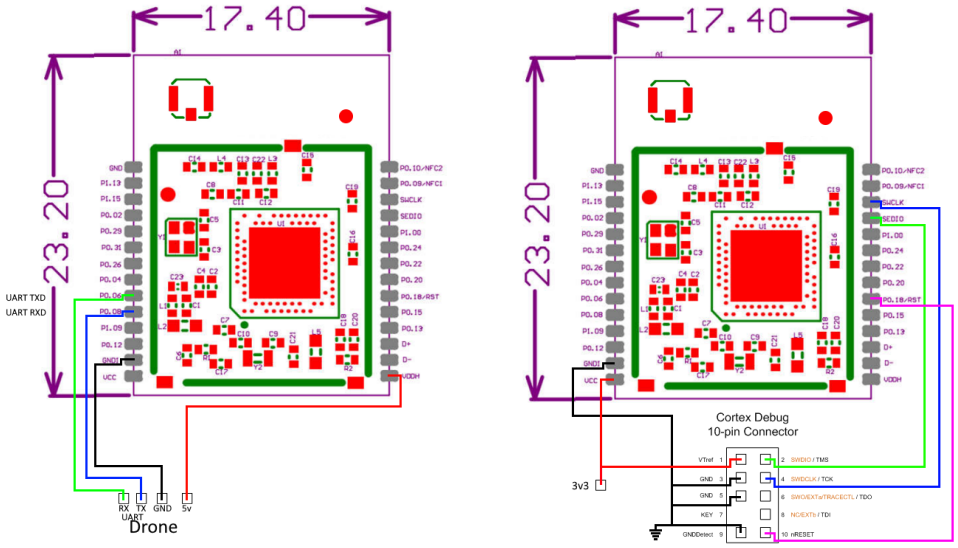
Troubles during testing and upgraded wiring configuration.

During our testing we encountered a problem where the ESP8266 module would sometimes stop working during flights. This prevented us from connecting the UAV to the GCS over WiFi. This problem appeared after installing the nRF module on the drone, but it happened fairly infrequent and could be solved with a power cycling of the drone. This problem became more severe and frequent when running longer tests as it would stop us from completing our tests.

We suspected the problem could be the power delivery from the 3.3v BEC. The 3.3V BEC was shared between the nRF module and the ESP8266. The maximum current the BEC can supply is 200mA and we suspected both devices combined could exceed this and cause the voltage from the BEC to drop to a level that caused problems with the electronics, resetting the nRF module and sometimes causing the ESP8266 to hang in a failed state.

To negate this problem we decided to power the nRF module from the 5V BEC instead of the 3.3V BEC. This would reduce the load on the 3.3V BEC. The 5V BEC can supply 1A, this is more than the 200mA from the 3.3V BEC. Also the 5V is only shared with the combined GPS receiver and magnetometer.

Further testing revealed this change did not remedy the problem and the changes were unnecessary.



(a) The wiring for the MS88SF23 module when connected to the drone.

(b) The wiring for the MS88SF23 module when programming or debugging.

Figure 3.14: Version 2 of nRF MS88SF23 module wiring.

3.3 UAV

Immediately after we started working on our master thesis we started to look into the requirements for the UAV to use in our system. We decided a set of requirements that was used when looking for a suitable UAV. One of our requirements was the possibility of running open source software. Another constraint was to keep the MTOM (Maximum take-off mass) under 250g, this includes the weight of the UAV, the batteries and our hardware. This weight constraint was introduced because any UAV heavier than this requires registration and certification for the pilot flying it. The possibility to use MAVLink for communication is also a requirement. This is because MAVLink is a flexible, well supported and open source protocol for communication with unmanned vehicles. If possible the UAV would also have a flight time of at least 30 minutes.

- Open source
- Weight under 250g
- MAVLink support
- Flight time of 30 mins

As we could not find any UAV on the marked fulfilling our requirements we chose to build a our own UAV to use as a platform for our system. We decided a lightweight multi-rotor quadcopter with a focus on maximizing flight endurance. Was the optimal choice for test as this allowed us to easily control it and hover when necessary. We choose to use ArduPilot as it's well tested piece of software with a large ecosystem and good community support.

Figure 3.15: Our UAV

3.3.1 Drone components

This section contains the parts of the UAV and a description of them. The reasoning for the component decision.

A short description of each component and it function is found in Table 2.1.

Table 3.4: Our UAV components

Part Type	Part	Weight
Flight Controller	Holybro Kakute F7 Mini V2	6g
ESC	T-MOTOR CINEMATIC F30A 4IN1 BLHELI_32	5g
Motors	4x FLYWOO ROBO RB 1204 ULTRALITE 5150KV	4x 5g
Propellers	4x Gemfan Hurricane 4024 2-Blade	4x 1.5g
Radio receiver	FrSky Archer M+ ACCESS	1g
GPS/compass	Matek M8Q-5883	7g
Frame	Custom carbon fibre frame	12g
Battery	Sony/Murata VTC6 18650 3000mAh 3S 33.3Wh	148g
Telemetry/MAVLink	ESP8266-01	2g
Logic level converter	4 channel Logic level converter	1g
RTT module	MS88SF23(nRF52840)	2g
Misc	Wires cables and other fastening hardware	22g
Total	-	232g

Table 3.5: Auxiliary and system specific components and their functions.

Part Type	Description
Telemetry/MAVLink	Provides a connection to a GCS/MAVLink network.
Logic level converter	Convert the signal voltage between components.
BLE RTT Module	Perform RTT distance measurements over BLE.

Flight controller

The flight controller we chose to use was the Kakute F7 Mini V2. The reasons for this choice was:

+ Well tested and good documentation.
+ Small footprint.
+ Low weight.
+ Six UART ports.
+ Built in 5v 1A BEC and 3v3 200mA BEC to power on-board devices.
+ Supports the ArduPilot, PX4 and Inav firmwares.
− Only the V1 and V2 boards are supported. The V3 board changed microprorcessor type and is no longer capable of running ArduPilot. [5]
− The V2 revision is hard to find.
− A little bit more expensive than other flight controllers.

ESC

We decided to use the T-MOTOR CINEMATIC F30A ESC for our UAV. Our reasons for choosing it was the low weight and it would work well with our S3 battery configuration. In retrospect this may not been the best choice as it required rewiring of the JST cable going between the ESC and the flight controller.

Motors

The ROBO 1204 motors from Flywoo was chosen for our UAV. It was chosen because of they are lightweight and the availability of performance charts. The performance charts contained data about thrust generation and energy consumption with different propellers, voltages and throttle levels. The sweet spot for efficiency appeared to be near the planned total weight of the UAV. Efficiency is defined as the radio between generated thrust and energy consumption.

Propellers

For our propellers we decided to have multiple options. We went for a set of Gemfan Hurricane 4024 2-blade propellers and a set of Gemfan 2540 flash 3-blade propellers. The Gemfan 4024 was chosen because of their high efficiency while the 2540 was chosen because it was one of the best performing propellers on the ROBO 1204 motor's performance chart. We also had a backup set of each of them in case the propellers suffered any physical damage.

Radio receiver

The FrSky Archer M+ ACCESS were chosen because of it's low cost and weight combined with decent range. It supports the FrSky ACCESS protocol. Despite being called a radio receiver it's actually a radio transceiver and it's capable of sending telemetry data from the UAV.

GPS/compass

The Matek M8Q-5883 is a combined GNSS and compass module. It supports GPS, GLONASS and Galileo. It was chosen because it's a well tested component and it delivers good accuracy at a low price. It's also confirmed to be working well with ArduPilot.

Frame

The frame is the main structure of the UAV. All other components are fastened to this. We chose to use a carbon fibre frame designed by the user brad112358 on the ArduPilot forum. Our reasons for choosing this design was the low weight and low drag. This frame weights 12g and have a thickness of 4mm. It's designed for propeller sizes up to 4 inches.

Battery

Choosing the right battery for the UAV is very important as it's one of the main factors determining the UAV flight endurance. It exists are two relevant battery

technologies, lithium-ion and lithium-polymer. Lithium-ion batteries usually have higher energy density per gram but lower max discharge current. Lithium-ion batteries also have a hard metal shell protecting them from damage and reducing the chance of a punctured battery compared to lithium polymer. While it's easy to find lithium polymer batteries targeted for use with UAVs, it's much harder to find lithium-ion batteries for this purpose. Unless you can find somewhere to buy preassembled lithium-ion battery packs, you need spot wielder to assemble them.

We have considered one lithium polymer battery, Gens Ace 3s and two lithium-ion batteries, the Sony VTC6 and the Samsung 35E. The UAV hovering is estimated to require 5-6A while wind or the UAV moving around would increase the power consumption. This would bring the Samsung 35E close to it's limits as the maximum continuous discharge current is 8A. This left only the Sony VTC6 and the Gens Ace 3s, here the Sony VTC6 was chosen because of the much larger energy capacity.

Table 3.6: Considered battery alternatives

Model	Nominal Capacity	Nominal voltage	Energy	Weight	Discharge Current (maximum Continuous)
Gens Ace 3s	1800mAh	11.1V	20Wh	150g	81A
Sony VTC6 3S	3000mAh	11.1V	33.3Wh	148g	15A
Samsung 35E 3S	3500mAh	11.1V	38.9Wh	154g	8A

Figure 3.16: Sony VTC6 3S with a XT30 connector and JST balance charge connector. A spot wielder was used to assemble the lithium-ion battery pack.

Telemetry/MAVLink

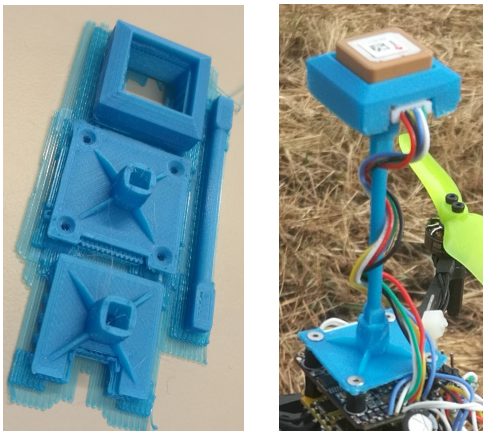
The ESP8266-01 was chosen as the telemetry radio for connecting the UAV to a computer using WiFi. Allowing MAVLink communication between the GCS software and the UAV.

+ Extremely cheep.
+ Easy to connect from computer or other device.
+ Lightweight.
+ Well tested.
− needs 3.3V and a 3.3V signal level converter.
− Relatively low range.

GPS mount

Keeping the GPS and magnetometer away from the rest of the electronics is important to reduce interference and increase the accuracy of the sensors.

Our first GPS mount consisted of nylon standoffs, these proved to be very fragile and had a high chance of breaking under uncontrolled landings. This was later replaced with a custom designed and 3D printed GPS mount. The mount consist of four parts that snap together and does not require any adhesives. The base connects with threaded holes using 4 M2 screws to a 20x20mm mount. It's easy to disassemble for transport.



(a) GPS mount parts.

(b) GPS mount in use.

Figure 3.17: 3D printed GPS mount for Matek M8Q-5883. For 20x20mm mount.

SheepRTT module mount

We also designed a custom mount for our SheepRTT module. This mount would clip on to the carbon fiber frame of the UAV and would also be secured with a zip tie. The mount would safely keep the SheepRTT module in place while in the air. By mounting the SheepRTT module way from the other component on one of the arms it would suffer less interference and give the antenna better coverage.

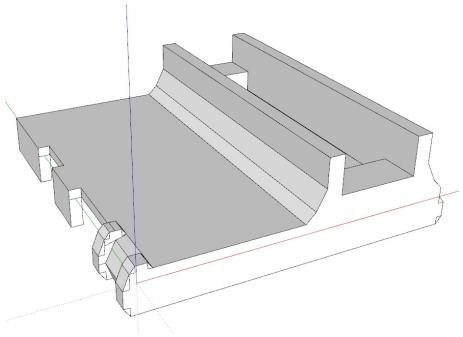


Figure 3.18: Design for mounting the SheepRTT module.

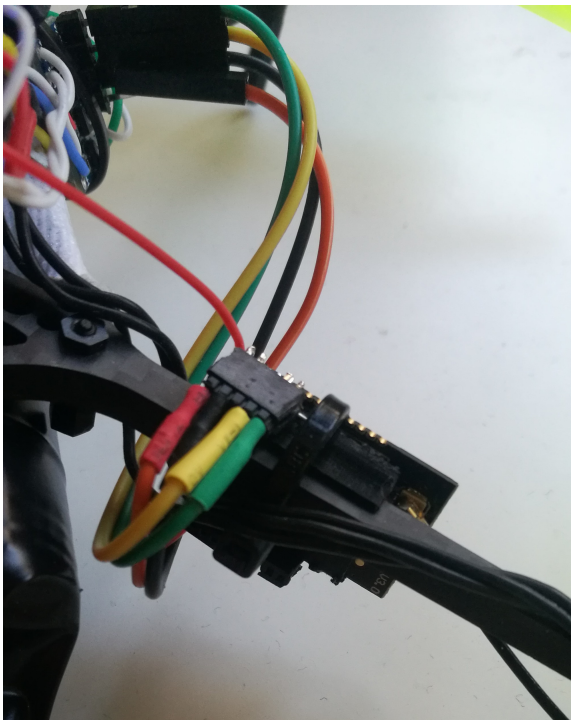


Figure 3.19: Custom 3D printed mount for SheepRTT module.

3.3.2 Radio transmitter

The radio transmitter takes input from the pilot and transmits them to the radio receiver on the UAV. This allows the pilot to steer the UAV and also do other actions like arming the it, changing the flight mode or starting autonomous missions. We chose to use the FrSky Taranis X9 Lite radio transmitter as this was the cheapest compatible radio transmitter fulfilling our requirements.

Despise its name this radio transmitter is in reality a radio transceiver and it's capable of receiving telemetry data from the radio reciver on the UAV. This telemetry data is in the FrSky passthrough format.

The radio transmitter is running the open source OpenTX firmware. This is accompanied with the Yappu telemetry script for displaying telemetry data and OpenTX sound for audio alerts and notifications.

Figure 3.20: FrSky Taranis X9 Lite radio transmitter

Yaapu FrSky Telemetry Script

The Yaapu FrSky Telemetry Script a LUA script for displaying telemetry data on a radio transmitter screen. [52] It receives FrSky passthrough telemetry data from the radio reciver on the UAV. Here the pilot can see the important telemetry data from the UAV, examples of this is the battery level, error messages, GPS information, RSSI, height above ground and current heading of the UAV.

The Yaapu FrSky Telemetry Script requires the FrSky radio transmitter to use a recent version of the OpenTX firmware.

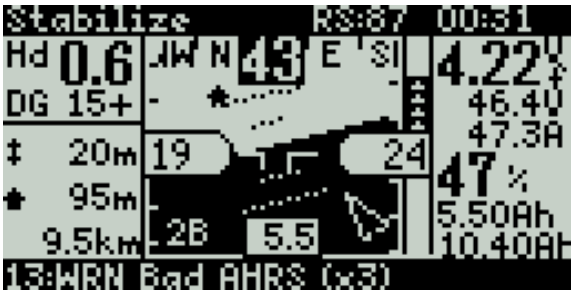


Figure 3.21: Yaapu FrSky Telemetry Script screen. [52]

3.3.3 Assembly

After we received all parts for the UAV we could start assembling them. The process adding for each component is detailed below.

Flight controller

Before doing any assembly we flashed the flight controller with ArduCopter, the multi-rotor version of ArduPilot. After this we tested the flight controller by connecting it to a computer running Mission Planner. Here we could verify the flight controller was working as intended.

ESC

The next step was to connect the ESC to the flight controller. This proved to require some effort as the pin-out at the flight controller end was different from the pin-out on the ESC. Using documentation from both manufacturer we rewired the DST connector in a suitable way for connecting these two components.

Radio receiver

The radio receiver was connected to 5v and a UART port on the flight controller. After this the receiver power up. We had problems binding the receiver to our radio transmitter. This was solved by updating the firmware on both the transmitter and receiver. The UART port in use was then configured for SPORT.

Battery

The battery was connected to the battery pads on the ESC. The ESC distributed power from this to the flight controller over the DST cable. A XT30 connector was used on the cable between the ESC and the battery. The ESC also contains voltage and current sensors to monitor our battery.

GPS/compass

The GPS module was connect to 5v, I2C and UART port configured for GPS on the flight controller.

Frame

The ESC and the flight controller was mounted on the frame using nylon M2 standoff screws. Later vibration dampening was added between the UAV and the flight controller. This was done to stop the vibrations from the frame reaching the accelerometer on the flight controller.

Motors

A motor was mounted on each arm of the frame using M2 screws. Then the three wires from each motor was soldered to the motor connection pads on the ESC. Some of the wires needed to be reordered to ensure each motor was spinning in the correct direction.

Propellers

The propellers was pushed down on the shaft of each motor and then secured with M2 screws.

Logic level converter

The high voltage reference pin on logic level converter was connected to 5v on the flight controller and the low voltage reference pin to 3.3v. Both RX and TX of two different UART ports was connected to the signal pins on the high voltage side. The PCB for the logic level converter was mounted on the frame with a zip tie.

MAVLink telemetry

The ESP8266 ESP-01 was flashed with the ArduPilot specific version of MAVESP8266. The ESP was then connected to the flight controller through the logic level converter. This was done because the ESP module can only handle 3.3v signals.

RTT module

The RTT module was connected to the flight controller through the logic level converter. This was done because the ESP module can only handle 3.3v signals. To make further changes to the RTT module easier we decided to use a header and removable jumper wires between the module and the logic level converter.

3.3.4 Configuration and calibration

Some of our components required further configuration or calibration.

Flight controller

The accelerometer built into the flight controller required a calibration, this was done using the setup guide in Mission planner.

On our first flight we noticed strong oscillations in the UAV. This was caused by the UAV overcompensating when trying to stabilize itself. The was corrected by tuning down the p-gain parameter on the UAV.

ESC

To get the ESC working we needed to use one of the supported protocols to communicate with it. We choose to use the DSHOT300 protocol as this started working immediately after configuring it through Mission planner.

Radio receiver

At first we set up the SPORT protocol between the radio receiver and the flight controller. We could not get the receiver to transmit telemetry data back to the radio transmitter. This was solved by updating ArduPilot to 4.1 and using FPORT protocol between the flight controller and the radio receiver.

Battery

The correct parameters for the battery was changed in ArduPilot. The parameters involved the expected battery voltage and capacity.

GPS/compass

The compass required a calibration before it could be used. The setup guide in Mission planner was used.

Motors

The motors required the correct configuration and ordering in ArduPilot. If the motor ordering was not correct the UAV would flip over right after takeoff. Incorrect ordering would cause this because the location of motors on the frame did not correspond to the expected and configured locations for ArduPilot.

3.4 MAVLink communication

Ensuring the MAVLink communication between all components in our system went smoothly required a bit of work. More details about this below.

3.4.1 Dialect modifications

For sending the SheepRTT distance measurements we needed to add two messages to the MAVLink protocol. More information about this in subsubsection 3.2.5.

3.4.2 Radio Sheep GCS & Node-MAVLink: implementing missing features

Radio Sheep GCS was implemented as an Electron app. This added the requirement to have a MAVLink packer and parser for Typescript.

The existing node-mavlink module was missing many features required to work fully with our system. This was the support for arrays and strings sent as char arrays. To remedy this problem we made a lot of changes in the node-mavlink libary where we added for arrays, strings sent as char arrays, zero truncation, support for extension fields, more tests and the ability to parse multiple incoming packets at once. This required some changes to the pymavlink libary too, this was the addition of the array length property when generating files for use with Typescript.

3.4.3 SheepRTT MAVLink implementation

More information about the SheepRTT MAVLink implementation can be found in subsection 3.2.5.

3.5 Practical tests

The following tests were performed to set our expectations and determine the capabilities of the system. We used the results to detect possible problems and tune the system for a more optimal performance.

Some of the tests below were in reality the same test performed multiple times or multiple similar test all summarized together. They are described as one test for more coherent writing and to contain less repetitions.

3.5.1 Range test

The with our first test the plan was measure packet loss, signal strength and distance estimation at different distances. We determined to do our testing in the southern parts of Høyskoleparken as it gave us direct line of sight up to around 240m. In Figure 3.23 the test location is shown with markings indicate where we would stand during the test. The marking with the number zero is where the stationary central board would be located, this can be seen in Figure 3.22. The other markings indicate where the peripheral board would be located for each measurement, with a higher number indicating further away. The computer would run JLinkRTTViewer allowing it to show and save the debug terminal output from the board. The log of the debug terminal output could later be analysed.

The settings for this test was a packet count of 500, this means 500 RTT distance measurements was performed per measurement shown. Both board were manually held at a height of 70cm.

Figure 3.22: Development kit held manually at a height of 70cm. Connected to the computer for logging the results.

Figure 3.23: Locations used for each measurement.

3.5.2 Antenna orientation and signal strength test with obstacle

Our next test had the goal of determining how antenna orientation and obstacles impacted the signal strength. The obstacle was a 5 litre storage container filled with

water, this simulates a sheep head blocking the path between the two boards. A storage container filled with water was chosen because most animals consist primarily of water and it would be a close enough substitute for our test. The storage container have a width of 12cm, meaning the signal have to travel through the container or be reflected around it by the environment. measure path loss from 12cm of water between sender and receiver. Simulate the sheep body blocking the signal.

Measurements were originally planned to be done at 35m and 225m with both boards at a height of 70cm above the ground. The 225m distance was later changed to 133m as we had trouble getting any measurements at 225m with a water container blocking the path. The test was repeated with all antenna orientations in Figure 3.25 both with and without the water can in between. The distance between the water can and the board was 5cm.

Figure 3.24: Point 0 is where it was measured from, Point 1 is at 35 meters distance. Point 2 is at 130 meters distance.

(a) Up, antenna pointing towards the "UAV"

(b) Left, antenna pointing towards the "UAV".

(c) Right, antenna pointing towards the "UAV".

(d) Down ,antenna pointing towards the "UAV".

(e) Up, antenna pointing away from the "UAV".

(f) Towards the "UAV", antenna pointing upwards.

(g) Towards the "UAV", antenna pointing left.

Figure 3.25: Different orientations of the development kit used during this test.

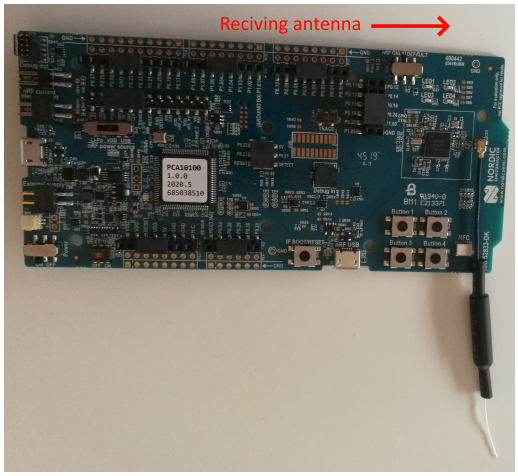
3.5.3 Range and signal strength tests with different antennas and antenna orientations

The antenna is one of the most important parts of a wireless system and its performance will affect the performance of the entire system. We decided to investigate the performance of different antenna configurations and orientations. The development kits included a U.FL connector for connecting an external antenna, and bypassing the internal antenna. We choose to use the external antenna pictured in Figure 3.26. We choose to do this test in two parts, one short range test investigating a multitude of different antenna configurations and a longer range test with the best performing antenna configurations from the short range test.

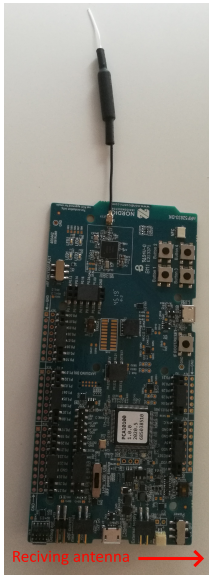
Figure 3.26: External 2.4GHz antenna with U.FL connector. Gain: 2.5dBi

Short range

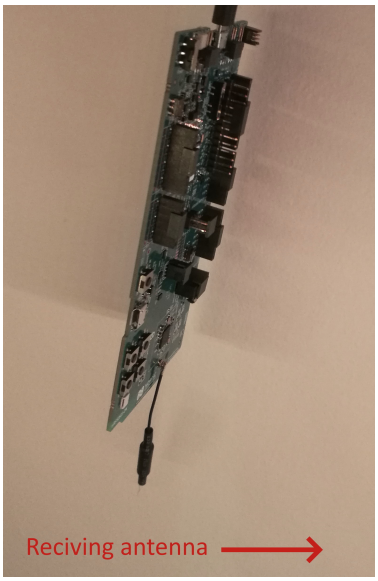
The short range test were preformed indoors with a distance of 2.5m between the central and peripheral board. All antenna configurations in Figure 3.27 was used on both the central and peripheral radio. The results was averaged over three measurements.



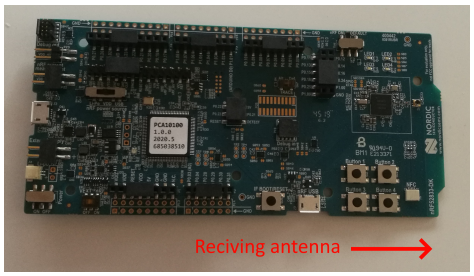
(a) External, antenna along the edge of the board.



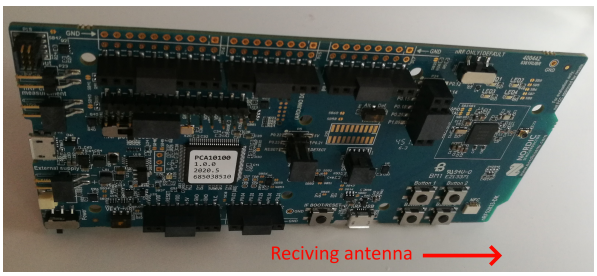
(b) External, antenna perpendicular to the edge of the board.



(c) External, antenna perpendicular to the edge of the board, oriented vertically.



(d) Internal, antenna side of the board pointing towards receiving antenna.



(e) Internal, antenna side of the board pointing towards receiving antenna, oriented vertically.

Figure 3.27: Antenna configurations and indication of the direction of the opposing antenna.

Long range

The long range signal strength tests were performed with the antenna configurations shown in Figure 3.27b and Figure 3.27c. The goal of this test was to determine the maximum range of our system. We planned to measure the RSSI at ranges from 0m to 800m in intervals of 50m. The test were performed near Dragvoll in Trondheim, starting at the parking lot near Bekken. This site was chosen because it provided a long stretch with continuous line of sight. The results were averaged over three measurements.

Figure 3.28: Planned long range test path. Starting at the Bekken parking lot and going in the direction of Dragvoll. Visualized in Google Earth Pro.

3.5.4 Manual flight test with drone

Our next objective was to have a functioning and flight worthy UAV platform for our remaining tests. This test consisted of manually flying the UAV in loither mode using the FrSky Taranis X9 Lite. The motors on the UAV got armed and throttle increased to 60%, this caused the drone to take off and gain altitude. When the UAV reached a height of a few meters the throttle was reduced to 50%, causing it to hoover perfectly in the same position. ArduPilot uses GPS data fused with other sensor data using EKF to achieve this. The UAV was then flown around a bit until it landed again at the same spot it took off from.

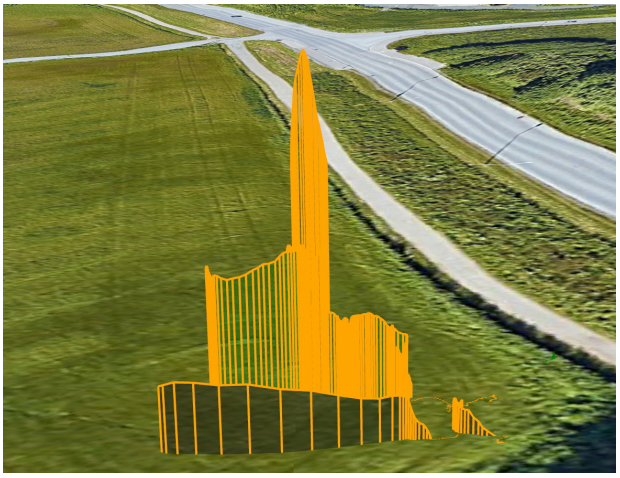
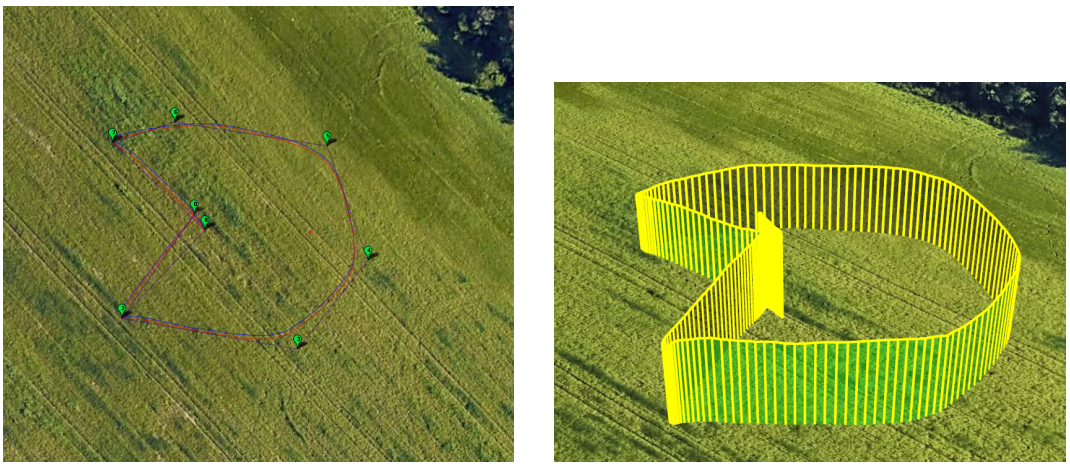


Figure 3.29: First manual flight test. Visualized in Google Earth Pro.

Figure 3.30: UAV used in this test.

3.5.5 Autonomous flight test

We performed the autonomous flight test learn more about the autopilot or autonomous flight features in ArduPilot. And also observe them in action. We decided to do this before running any tests with Radio Sheep GCS. This was to know more about how the drone would behave with a mature and well tested GCS. This test was performed by creating a flight plan in Mission Planner and then uploading it to the UAV. The UAV was then armed and manually flown to a height of around 5m in loither mode. The flight mode on the UAV was then changed to auto mode where it imminently started executing the flight plan that was uploaded Mission Planner. The UAV passed within 2m to all waypoints in the flight plan in correct order. When the last waypoint was reached the UAV started hovering there until it was landed manually.



(a) Flight path and flight plan shown in Mission Planner.

(b) Flight path in 3D. Visualized in Google Earth Pro.

Figure 3.31: Autonomous flight test flight path.

Problems

During one of our autonomous flight tests the UAV started climbing extremely fast to an altitude much higher than was it was supposed to. The altitude graph on Figure 3.33 shows how rapid this climb was, reaching speeds of 8m/s vertically, climbing 40m in 5 seconds. When it reached an altitude of 95m the UAV stabilized itself and triggered the geofence failsafe. This failsafe changed the flight mode to RTL and the UAV come back down and it tried to get back and land in the exact same spot it took of from. During this process it stopped descending at a height of around 10m, and it then started drifting towards some trees and crashed into them. This can be seen on Figure 3.32 where the black wall is the supposed path and the other is the path of the UAV ending in some trees.

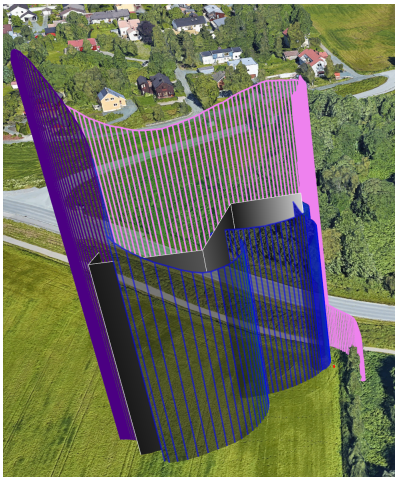


Figure 3.32: Flight path in 3D. Visualized in Google Earth Pro.

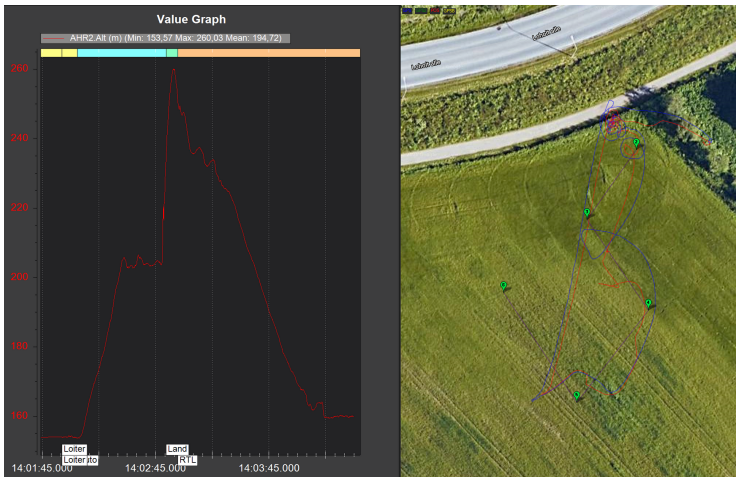


Figure 3.33: Altitude graph and flight path.

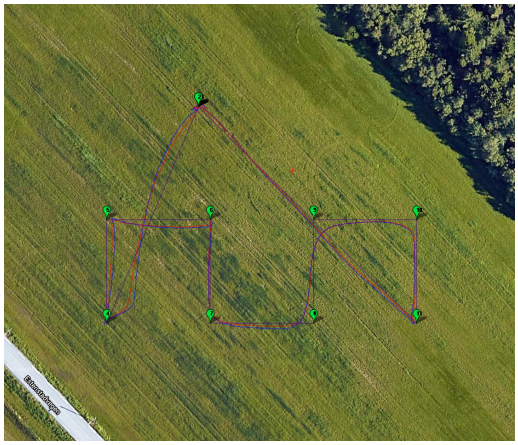
We later found that vibrations from the UAV caused the accelerometer to clip. This means the accelerometer was exposed to acceleration much higher than the maximum value it could measure. This in turn triggered the vibration failsafe making the UAV ascend rapidly to avoid crashing into the ground while the UAV tried to recover from this. When it recovered the UAV started to return to launch.

This problem could be mitigated by reducing the vibrations the accelerometer was subjected to. The easiest way to reduce this vibrations was to install better vibration dampening around the flight controller where the accelerometer was installed.

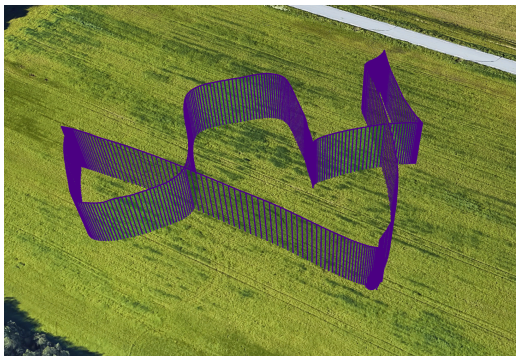
The other problem where the UAV started drifting was found to be human error. It was uncovered that while in RTL mode the pilot could take control manually and stop the RTL sequence by adjusting the throttle. This caused the UAV to start drifting without the pilot's knowledge. This problem can be mitigated by having better knowledge about the system in use.

3.5.6 Small scale full system test

The full system test is supposed to have the GCS, UAV and SheepRTT modules all working together to locate the animals. We started the Radio Sheep GCS and connected to the UAV, this allowed us to select an area to search and the GCS made an effective flight plan to cover this area. When our flight plan was ready it was uploaded to the UAV over MAVLink and the UAV was given the command to arm the motors and execute the flight plan. The UAV then took off autonomously, followed the planned path, then came back and landed. When connection was reestablished Radio Sheep GCS retrieved all the measurements from the SheepRTT module by communication over MAVLink.



(a) Flight path and flight plan shown in Mission Planner.



(b) Flight path in 3D. Visualized in Google Earth Pro.

Figure 3.34: Small scale system test flight path.

Problems

When executing our first autonomous flight plan by Radio Sheep GCS we encountered a problem where the UAV would rapidly climb right after takeoff. This lasted until the geofence height limit of 35m was reached, and a geofence failsafe was triggered. This failsafe made the UAV RTL and the UAV came back down and landed in the exact same spot it took of from. This also proves how important Ardupilot's geofence system, the failsafe handling and RTL is to prevent failures.

After downloading the flight plan from the UAV to Mission Planner we found that the altitude for each way point in the flight plan was given in relative height instead of absolute height. The launch site was around 160m above sea level and gave the waypoints height of 170m above ground instead of 10m. A quick modification to Radio Sheep GCS were we changed it from relative height to absolute height fixed this issue.

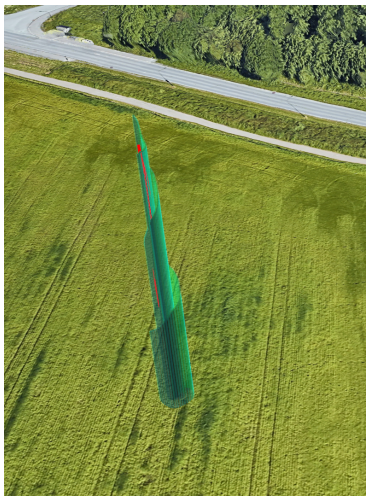


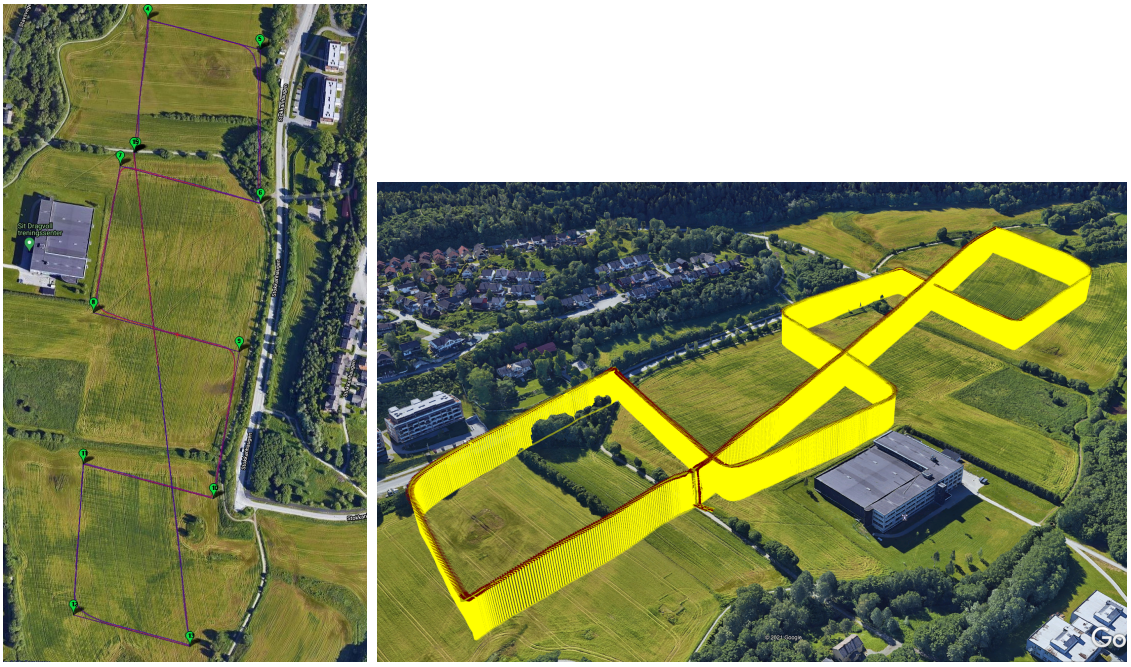
Figure 3.35: Flight path in 3D showing when the geofence failsafe was triggered. Visualized in Google Earth Pro.

Figure 3.36: Altitude graph and event timeline around the Geofence failsafe triggering.

3.5.7 Large scale full system test

This test is the same as the small scale test, but with newer versions of our software and a new and larger test area. The new test area is the fields between Dragvoll and Stokkan in Trondheim. This area was chosen because the large open fields would provide clear line of sight and the walking paths along the fields would make moving around easier.

The test needed to be repeated several times because of technical difficulties. Most of them being problems with the ESP8266, preventing us from connecting to the UAV and retrieving data from the SheepRTT module. More information about this in subsubsection 3.5.7.



(a) Flight path and flight plan shown in Mission Planner.

(b) Flight path in 3D. Visualized in Google Earth Pro.

Figure 3.37: Large scale system test flight path.

where d_m is the distance measured between the nRF module on the UAV and the tag, h is the height difference between the UAV and the tag. d is the measured distance with compensation for the flight height of the UAV.

Problems

Our main problem were troubles with the ESP8266 not working after long flights, leaving us unable to retrieve data from the SheepRTT module. Because of this problem we decided to connect the GCS to the UAV over USB. Connecting to the drone over USB provides us a serial port to communicate with the UAV. As Radio Sheep GCS does not support this we used mavproxy route MAVLink packages between the serial port and the UDP port.

Our other problem was two bad soldering joints between the SheepRTT module and the connector leading to the UAV. One of there soldering joints were the RX pin on the nRF module, leaving it unable to receive any MAVLink messages. The other soldering joint was connecting VCC pin to 3.3V, this caused the module to only be powered by the VDDH pin. When powered only by the VDDH pin it go into High voltage mode and use an internal regulator for the incoming power, leading to increased power consumption.

3.5.8 Range and signal strength test with module on drone

The goal of this test was to gather data about the accuracy over longer distances and maximum range. This data could then be used to tweak our implementations for optimal performance and provide information about the system's capabilities.

For this test the location of the central and peripheral board was switched around. This was done to provide additional debug information about each RTT measurement. The SheepRTT module on the UAV was programmed as peripheral board. The SheepRTT module located at the starting location was mounted at a height of 1m and programmed as a central SheepRTT module and connected to a computer to log the additional debug information. In this configuration a RTT distance measurement was done every 7.5ms. When everything was ready we started UAV with an autonomous flight plan as shown in Figure 3.38.

As the peripheral board is only meant to be located by the central board it did not posses any ability to locate itself by using the GPS on the UAV. Because of this we extracted the GPS data from the flight log and matched this to the debug data from the central board using timestamps. After the test we could compare the measurements from the central board with the GPS data from the UAV. This allowed us to calculate the real distance, measured distance and the inaccuracy of the measured distance. the RTT measurements.

This test was repeated multiple times with two different parameters. The first parameter was the implementation in use. And the second parameter was the transmit power setting on the central board.

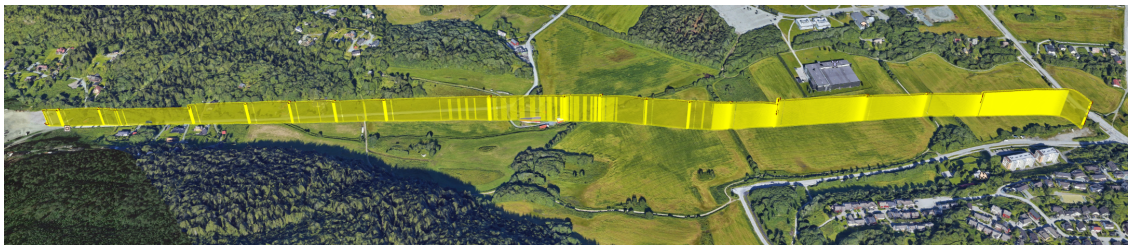


Figure 3.38: Flight path for long range testing. Starting near Stokkan and heading in the direction of the Bekken parking lot. Visualized in Google Earth Pro.

Problems

During our first test of implementation #2 we did not get any measurement at distances over 400m. Our data showed the strength signal was not the problem. We tracked this down to the radio of the SheepRTT peripheral board not staying awake long enough after the while waiting for a response form the central board. Increasing the waiting time fixed our issue.

3.5.9 Optimal speed, power consumption and range

The goal of this test was to determine the optimal speed to archive the longest range possible. With this data we could also estimate the maximum flight time and range.

We tested the UAV hovering and traveling with different speeds, all the way up to the legal speed limit for our UAV. This included speeds of 2.5m/s, 5m/s, 7.5m/s, 10m/s, 15m/s and 19m/s. Any wind at the test site would impact the results. This is because as having tailwind or headwind would decrease or increase the power consumption respectably. To reduce the impact the wind would have on our tests we did two runs, one going with the direction of the wind and one in the opposite direction and then taking the average of this.

In our case this was north to south and south to north. The test itself was performed at the fields between Dragvoll and Stokkan on a day with little wind. The flight plan was aligned to wind direction at the test site.

The speed the UAV was traveling at was measured by the on board GPS module. Both voltage and current was measured by the ESC. All of this data was saved to the flight log on the flight controller and later analyzed.

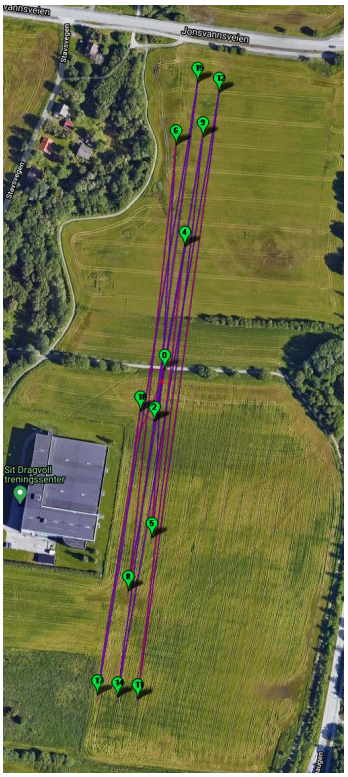


Figure 3.39: Optimal speed test flight path. At the fields between Dragvoll and Stokkan. Visualized in Mission Planner.

3.5.10 General problems and considerations

Bluetooth CRC and battery problem

While preparing for the last few tests we started having a new problem where we didn't get any measurements and upon further inspection we would get the message "malformed" very consistently in the debug console on the central SheepRTT module. A "Scanning..." message followed by a "malformed" message indicates Bluetooth CRC error. This could indicate a problem with the SheepRTT modules and could be caused by low battery voltage. The battery voltage was measured to be 3.01V, witch should be within the normal parameters for the module. Plugging in a powered USB cable for a brief moment fixes it temporarily and allows measurements until the module is turned off for a few minutes. Changing the batteries fixed the problem. The problem was most likely caused by voltage sag when drawing current from a near depleted battery.

Development kit unstable micro USB connection

During some of our tests we had problems with the development kit disconnecting from the computer in the middle of the test, this forced us to repeat a few of the tests multiple times. This was most likely caused by small movements loosening the micro USB cable from connector on the development kit.

Sources of interference

The Bluetooth LE modules we are using for testing are utilizing frequencies at 2.4GHz. In our system the Bluetooth channel 50 at the frequency 2453MHz is used. There is a large number of devices utilizing the 2.4GHz band, causing interference and loss of range. This is more of an issue when testing more urban areas and would be less of a problem for real world scenarios, far away from the cities. The most relevant sources if interference is by own own equipment on the drone, followed by personal devices and then by wireless devices like WiFi routers near the test site. Near our test site there is residential buildings and a large university buildings. Interference from these sources is hard to do anything about and should be taken into consideration, especially at long ranges. To limit interference from personal devices we turned of WiFi and Bluetooth on our mobile phones during testing.

On the drone itself there is three sources of interference, the wireless receiver for controlling the drone, the ESP8266 used for MAVLink communication and then the drone electronics itself. The wireless receiver uses the FrSky ACCESS protocol to both receive controller inputs and send telemetry data back. The FrSky ACCESS protocol operates at around 2.4GHz, it's a proprietary protocol so we don't have any details about specific frequencies used by it. The ESP8266 is acting as a wireless access point and have been configured to use the WiFi channel 1. This channel is at 2412MHz and should not overlap with the frequencies used by the Bluetooth LE modules. The power electronics on the drone could also generate interference as this handles currents of around 5 to 15 Ampere just a few centimeters away from the module. The interference from the electronic components have not been measured.

Chapter 4

Analysis

The analysis have a focus on the UAV, the SheepRTT distance measurements and our system as a whole. Analysis and solutions for smaller problems regarding the flights themselves are kept in section 3.5.

4.1 Practical tests

This section contains the analysis of data from our practical tests.

4.1.1 Range test

The purpose of this test was to determine the accuracy of the RTT distance estimations, the signal strength and packet loss at longer distances. As we can see from Table 4.1 the distance estimations from the RTT measurements are close to the distance measured by GPS and would be good enough for our purpose. The loss of signal strength and the increased packet loss around measure point 3 could come from the proximity to residential buildings, the nearby parking lot or that it was the lowest points in the terrain. At measure point 4 and 5 you can see the signal strength and the packet loss improving again even if the distance between the central board and the peripheral board increases, this could be because the measuring points where higher up in the terrain and thus having a better line of sight. The uncertainly of the GPS data is +-3.2m.

Table 4.1: Range test results

Measure point	Measured distance (m)			RSSI (dBmW)	Packet loss
	GPS	nRF52833	Difference		
0	0	0	0	17	0%
1	32.8	35.0	2.2	61	0.0%
2	72.7	83.0	10.3	69	0.6%
3	133.3	136.3	3.0	84	68.0%
4	168.8	175.5	6.7	69	2.4%
5	221.2	227.5	6.3	73	0.0%
	Mean		5.7		

4.1.2 Antenna orientation and signal strength test with obstacle

The purpose of this test was to determine how obstacles and different antenna orientations would affect the effective range of our system. We planed to run this test at distances of 35m and 225m, but we faced problems during the test at 225m with a water container as an obstacle and we decided to reduce the distance to 133m and continue the test.

By analyzing our results we found that when the water container blocked the path, the signal strength were reduced by between 4dBm and 7dBm. This is with the signal traveling through 12cm of water. This data corresponds with the expected path loss based on the path loss of a signal going through seawater as shown in the grapth in Figure 2.14.

Table 4.2: Range test 2 results

Position (direction the top side Of the devkit is facing) Figure 3.25	Signal, RSSI (dBmW)				
	35m		133m		**Average**
	With water container	W/o water container	With water container	W/o water container	**(position)**
Up (antenna pointing towards «drone»)	-74.1	-62.6	-73.3	-71.9	**-70.5**
Left (antenna pointing towards «drone»)	-73.4	-72.3	-79.7	-82.2	**-76.9**
Right (antenna pointing towards «drone»)	-80.3	-74.6	-81.0	-76.5	**-78.1**
Down (antenna pointing towards «drone»)	-73.2	-61.8	-75.3	-68.6	**-69.7**
Up (antenna pointing away from «drone»)	-77.5	-67.4	-73.2	-69.4	**-71.9**
Towards «drone» (antenna pointing upwards)	-74.4	-61.0	-80.5	-74.1	**-72.5**
Towards «drone» (antenna pointing left)	-69.4	-69.9	-86.4	-80.4	**-76.5**
Average (distance/water container)	**-74.6**	**-67.1**	**-78.5**	**-74.7**	

4.1.3 Range and signal strength tests with different antennas and antenna orientations

The purpose of this test was do determine if there was any difference between using the integrated antenna or an external antenna and also figure out what antenna configuration could give us the best range. This test was split into two different test with the later one building on the results from the first.

Close range

As we can see from the result on Table 4.3 the use of the "External, antenna perpendicular to the edge of the board." configuration on both boards and the "External, antenna perpendicular to the edge of the board, oriented vertically." combined with "Internal, antenna side of the board pointing towards receiving antenna, oriented vertically." configuration had the best signal strength, -40.2dBmW and -40.1dBmW respectively. The difference in signal strength between some of the best performing configurations and the worst performing configurations were 16.4dBmW, this difference could in worst case reduce the range to $\frac{1}{6}$ compared to the best performing configuration one according to the free-space model. If we used the plane-earth model this is reduced to $\frac{2}{5}$. The results from this test were very important for planing future tests.

Table 4.3: Short range antenna configuration test results.

	Signal, RSSI (dBmW)				
Antenna configuration ↓Central/Peripheral→	External, antenna along the edge of the board.	External, antenna perpendicular to the edge of the board.	External, antenna perpendicular to the edge of the board, oriented vertically.	Internal, antenna side of the board pointing towards receiving antenna.	Internal, antenna side of the board pointing towards receiving antenna, oriented vertically.
External, antenna along the edge of the board.	-54.1	-46.4	-53.3	-52.2	-50.1
External, antenna perpendicular to the edge of the board.	-46.9	-40.2	-51.7	-45.2	-55.2
External, antenna perpendicular to the edge of the board, oriented vertically.	-53.9	-46.1	-43.4	-56.5	-47.0
Internal, antenna side of the board pointing towards receiving antenna.	-53.6	-46.0	-50.7	-52.0	-55.4
Internal, antenna side of the board pointing towards receiving antenna, oriented vertically.	-52.0	-51.8	-40.1	-51.8	-46.8

Long range

During this test we noticed the path loss affecting our signal were much higher than expected. It was decreasing at 10dB per doubling of the distance instead of the expected 6dB. This resulted in a much weaker signal and shorter range. At a distance of 250m we could no longer initiate a RTT distance measurement even if we could sometimes detect the advertisement packet from the peripheral board.

We also noticed when cars were passing on the road, and blocking the path between the central and peripheral board it would impact the measurement by either making the signal weaker or blocking it completely at longer ranges. This factor was not considered when choosing a location to run this test.

Even if we didn't get the test results we expected, the results appeared to be consistent with a very close to 10dB weaker signal per doubling of the distance. This got us into reading about near-ground path loss and this paper Wang et al. [51], exploring this type of path loss. This much higher path loss could be from near-ground path loss as the signal would travel very close to the ground. This could have caused destructive interference resulting in the loss of signal strength. Neither the free-path model or the plane earth model fits our data very well. Our results are somewhere in between those two models. In the paper by Wang et al. [51] they experienced the same results and their proposed models could be used as a better approximation when performing low altitude signal strength tests.

Table 4.4: Long range antenna configuration test results.

	Signal, RSSI (dBmW)	
Antenna configuration Distance ↓	External, antenna perpendicular to the edge of the board.	External, antenna perpendicular to the edge of the board, oriented vertically.
0m	-35.4	-38.4
50m	-72	-67
100m	-81	-77
150m	-86	-83
200m	-90	-86.5
250m	–	–

4.1.4 Manual flight test

We encountered no problems during this flight test. The UAV obeyed the pilots commands perfectly. As this test did not include the SheepRTT module, there is not a lot to analyze.

4.1.5 Autonomous flight test

We encountered no major problems during this flight test. The UAV took off autonomously and then followed the planned flight path perfectly before coming back and landing by itself. As this test did not include the SheepRTT module there is not a lot of data to analyze.

4.1.6 Small scale full system test

During the flight the SheepRTT module gathered RTT distance measurements. When connection to Radio Sheep GCS was reestablished the distance measurements was uploaded. The distance measurements was coupled with the correct GPS data. As all components of the system functioned as intended, this was a successful test. We could continue our work with the knowledge that our system functioned as intended.

4.1.7 Large scale full system test

When test flight was finished and all the RTT distance measurements was uploaded to Radio Sheep GCS we could start analysing them. The analysis of the RTT distance measurement done by a combination of Radio Sheep GCS and manual work. We received 1032 distance measurements in total. Of there 91 of the measurements did not contain a distance, only indicating the tag was in the proximity. So far the location estimation methods in radio Sheep GCS does not make use of these 91 measurements. We are then left with 941 usable RTT distance measurements with accompanying GPS locations.

Table 4.5: Number of RTT distance measurements per tag.

Tag	Measurements		
	Usable	Unusable	Total
1	171	39	210
2	270	13	283
3	245	20	265
4	255	19	274
Total	941	91	1032

Prepossessing

The 941 distance measurements was then further processed into four different datasets. Each dataset was processed in another way. One of the dataset contained all RTT distance measurements without any modifications. Another dataset used only distance measurements with a distance under 100m. While another one used the high difference compensation explained below. And the last dataset was a combination of both the preprocessing steps.

When the UAV was right above the tag the distance measured was around 20m instead of 0m. This is because of the height difference between the UAV and the tags. We needed to find a way compensate for the height difference. This would also be useful for adapting the distance measurements to a two dimensional map and for calculating the tag position. We need to separate the vertical and horizontal distance vectors between the UAV and the tag. If we assume the ground is flat or the height difference between the UAV and the tag is constant we can use Equation 4.1 as an approximation to compensate for the height difference.

$$d = \begin{cases} \sqrt{d_m^2 - h^2}, & \text{if } d_m \geq 0 \\ 0, & \text{if } d_m < 0 \end{cases} \tag{4.1}$$

where d_m is the distance measured between the nRF module on the UAV and the tag, h is the height difference between the UAV and the tag. d is the measured distance with compensation for the flight height of the UAV.

The last preprocessing step was to split each of the four datasets into 10 smaller datasets. The reason for doing this is that during testing the frequency of SheepRTT distance measurements was set to 10 times the normal frequency. And by splitting up the dataset we reduce the number of samples to a realistic amount and we also have multiple pseudo tests to average out.

Results and accuracy

Results from analyzing the accuracy of the full SheepRTT system is shown in Table 4.6. Here "Max 100m" means we remove all distance measurements with a distance over 100m. "w/ height difference compensation" means we used Equation 4.1 to compensate for the height difference between the UAV and the tag.

Table 4.6: Large scale full system location estimation accuracy comparison

Data used	Method			
	Particle		Intersection	
	Average distance error(m)	Average uncertainty(m)	Average distance error(m)	Average uncertainty(m)
All	163.4	390.8	15.9	39.3
All, w/ height difference compensation	163.4	387.9	14.4	46.2
Max 100m	37.0	147.8	19.2	34.6
Max 100m, w/ height Difference compensation	37.8	143.2	18.8	35.3

Here we can see the best performing method is the intersection method using all measurements and with compensation for the height difference. As we can see the particle method struggles with the higher amount of measurements when using all measurements.

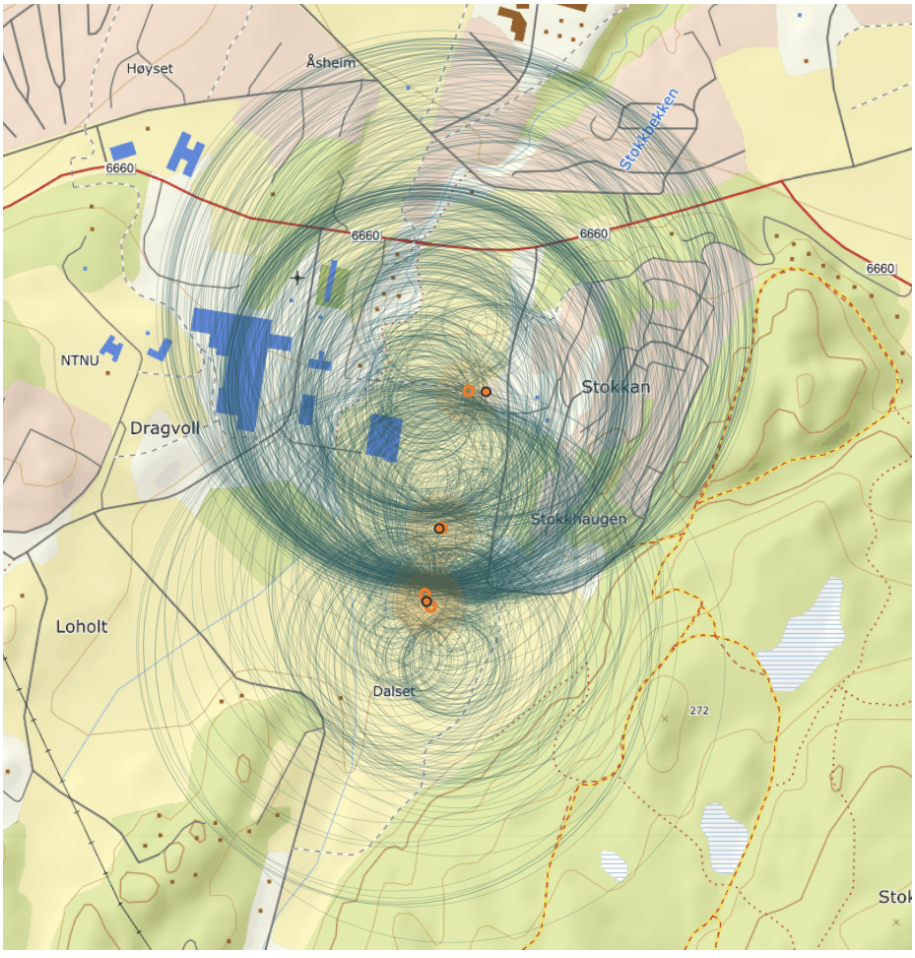
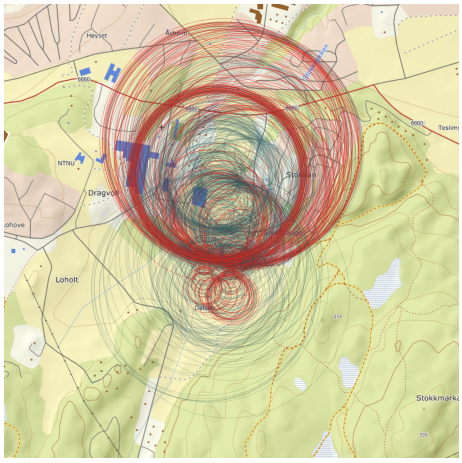
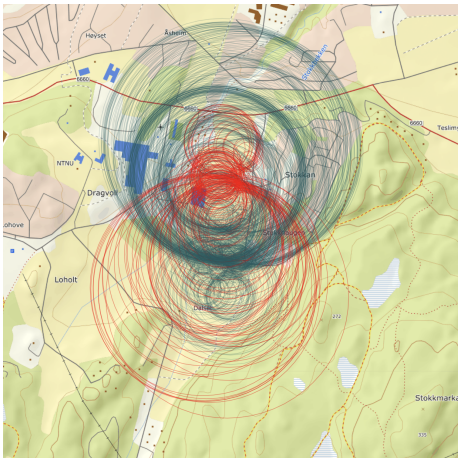


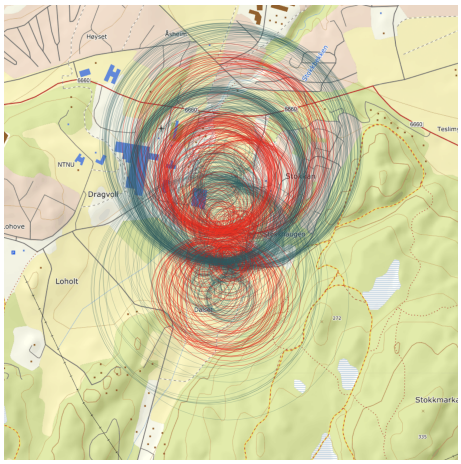
Figure 4.1: Location estimations compared to the real locations. Using all measurements and the intersection method.



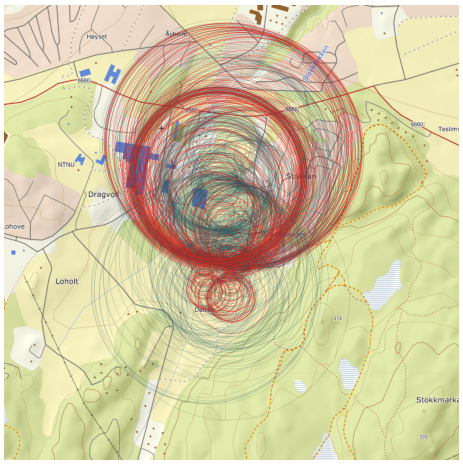
(a) Distance measurements for tag 1.



(b) Distance measurements for tag 2.



(c) Distance measurements for tag 3.



(d) Distance measurements for tag 4.

Figure 4.2: RTT distance measurements visualized, red is the measurements for one specific tag. The center of each circle is the position of the drone when measuring the distance and the radius is the measured distance.

4.1.8 Range and signal strength test with module on drone

The goal of this test was to determine the maximum range for the SheepRTT and to gather more information about the accuracy of the system.

For the analysis the distance measurements was provided by a detailed debug log from the central board, and this was then compared to the GPS data from the UAV. This was used to generate the figures below. Note that the shown figures is only a subset of the results. More about this results can be found in [45].

As we can see from the figures the accuracy improves significantly with implementation #2. This combined with a lower power consumption makes this look like the superior implementation.

The range of both implementations should be the same. The reason we can see a maximum range difference between Figure 4.3 and Figure 4.6 is the use of a lower transmit power during that particular test of implementation #2.

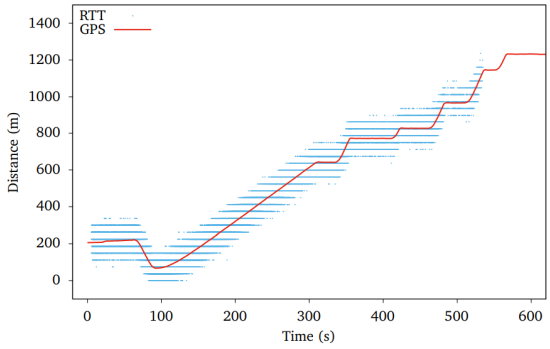


Figure 4.3: RTT distance measurement compared to GPS distance for SheepRTT implementation #1.

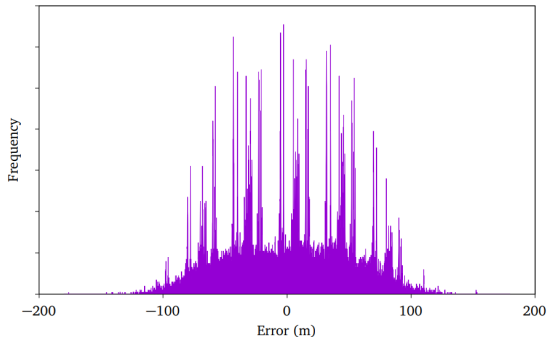


Figure 4.4: Histogram of the error distribution for SheepRTT implementation #1.

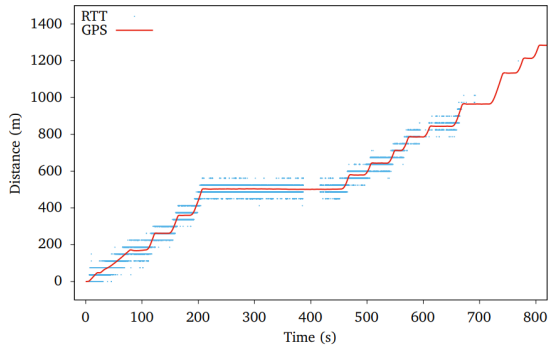


Figure 4.5: RTT distance measurement compared to GPS distance for SheepRTT implementation #2.

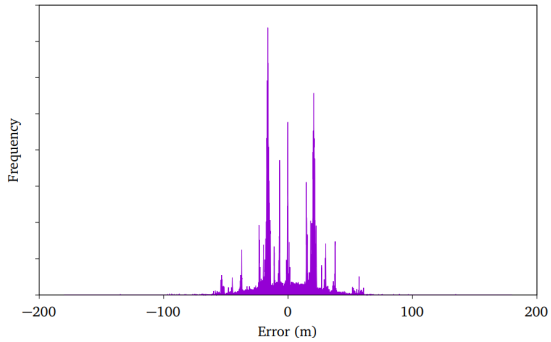


Figure 4.6: Histogram of the error distribution for SheepRTT implementation #2.

4.1.9 Optimal speed, power consumption and range

After the testing was finished we analyzed the flight log in Mission Planner and extracted the recorded speed, battery voltage and battery current. This was further processed to get the data in Table 4.7.

As you can see the power consumption falls sightly as the speed increases until we reach 7.5m/s, this could be from oscillations in the UAV, changing wind conditions or inaccuracies from the voltage and current sensors. All these values are very close to each other and could be within the margin of error. After this the power consumption starts to rise again as the speed continues to increase. This can be seen better on Figure 4.8. The important takeaway from this is that the energy consumption stays roughly the same for speeds up to 10m/s. Almost all of the energy used by the UAV is expended keeping it in the air, while only a small portion of the energy is used to move in the desired direction. When flying at speeds over 10m/s the increased air resistance begins to impact the energy consumption.

With these results we can find the optimal speed for the UAV to maximize range or flight time. The estimated range at different speeds is plotted in Figure 4.7. From this we find that the equilibrium point where increasing speed does not increase range is over 19m/s. This is because the increasing air resistance is not high enough to lead to a decreased range. The reason we did test at higher speeds is the legal speed limit of 19m/s for our UAV. Therefore to maximize range we should fly the

Table 4.7: UAV speed/power consumption and estimated range and flight times.

Speed (m/s)	Power consumption(w)			Wh /km	Range(km) w/ 33.3wh battery	Flight time (min)
	Heading south	Heading north	Avg			
0			59.5			33.58
2.5	59.2	54.6	56.9	6.32	5.27	35.11
5	53.6	55.2	54.4	3.02	11.02	36.73
7.5	49.5	56.7	53.1	1.97	16.93	37.63
10	50.0	61.3	55.7	1.55	21.54	35.90
15	59.9	79.0	69.5	1.29	25.89	28.77
19	64.2	83.2	73.7	1.08	30.91	27.11

UAV at the highest speed possible. The estimated range at a speed of 19m/s with the 33.3Wh battery is almost 31km.

The wind speed during the testing was 2-4m/s and was blowing from north to south. The weather information was fetched from yr.no.

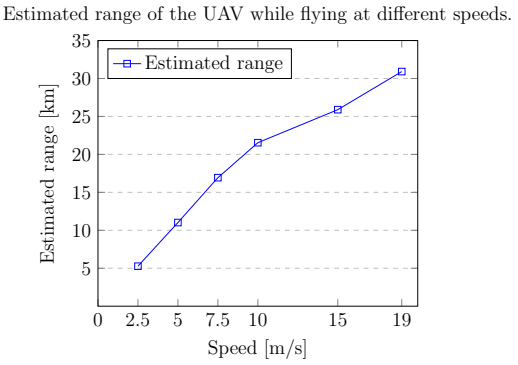


Figure 4.7: UAV estimated range at different speeds.

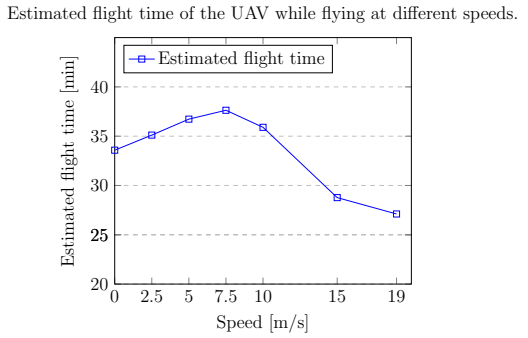


Figure 4.8: UAV estimated flight time at different speeds.

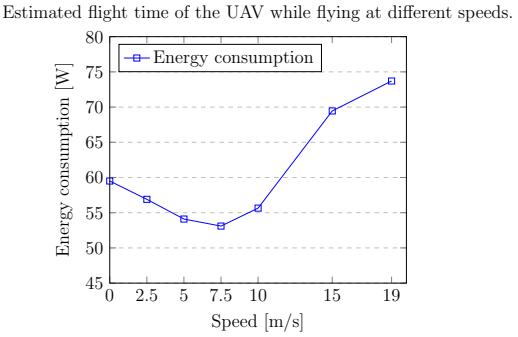


Figure 4.9: UAV energy consumption at different speeds.

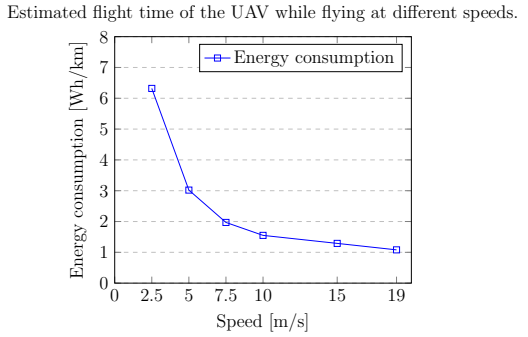


Figure 4.10: UAV energy consumption per km

4.1.10 Theoretical prototype tag

By using the data we have gathered while developing this system we can make a rudimentary design for a smaller prototype tag. Designing a smaller SheepRTT peripheral module or tag was not within the original scope of the project. By doing this we could estimate some of the properties our system could have as it gets closer to a finished product.

If we combined a MINEW MS88SF23 module, an antenna and a CR2032 battery with a battery holder we could have a small prototype tag. This would have a weight under 10g, a size around 20x25x12mm excluding the antenna and a total cost under 100 NOK. The estimated battery life of this prototype could be between 46 weeks and 67 weeks when using implementation #2 of SheepRTT. [45]

Chapter 5

Discussion

This chapter is discussing the results from our analysis and looks at viability of our system.

We are going to compare our system to other alternative solutions and weighing up the pros and cons. We are also going to look at the possibilities and challenges our system could face in the future.

5.1 SheepRTT range and accuracy

During our testing we gathered data about the range and accuracy of the SheepRTT system. We found the system could find tags at a distance of at least 1.2km and measure the distance with a uncertainty under 50m. For more information see [45].

5.2 System MAVLink integration

The integration of MAVLink into the SheepRTT module itself allows plug and play functionally with ArduPilot based UAV. This could be expanded with some compatibility modifications to include all MAVLink based UAVs.

5.3 Full system performance

Our full system tests shows promising results for locating animals. The system estimated the position of the tags within 15m on average. This is more than good enough for our needs. The tag locations would only be used as guidance when gathering the animals from the grazing area.

5.4 UAV flight speed, range and efficiency

One of our objectives was to find the maximum range of our UAV. To find this me measured the efficiency of the UAV flying at different speeds. We found that the optimal speed to maximise range was 19m/s. With this speed we could achieve a range of up to 30km.

5.5 Comparison to existing solutions

We are going to compare the traditional way of gathering, the GPS collars and our system with the theoretical prototype tags.

Ease of use

The traditional way can be done by anyone able to traverse the terrain and recognize the sound of the bells. Both the solution from Nofence and findmy offers a mobile phone app for configuration and monitoring. Our system can be operated through Radio Sheep GCS.

Connectivity

The traditional way requires no connectivity. The GPS collar from Nofence requires 2G or LTE CAT M-1 connectivity in the grazing area. While the GPS collar from findmy connects a satellite network and should work anywhere. Our system does not require any connectivity in the grazing area but Radio Sheep GCS requires an internet connection to fetch maps and terrain data.

Battery life

The traditional way requires no batteries at all. The average battery life for the GPS collar from Nofence is 3 weeks. The average battery life for the GPS collar from findmy is about two to three seasons. The estimated battery life of our system is between 46 weeks and 67 weeks with a CR2032 battery.

Costs

The cost of the bell collars is 84 NOK. The cost of the GPS collars from Nofence and findmy is around 1850 NOK per unit. There is also a need for an active subscription from the manufacturer to use the GPS collars. The estimated cost of a prototype tag for our system is around 100 NOK, but our system also requires a UAV to function.

Form factor

The form factor of the collars is the collar itself with a big rectangular cube hanging under the animals head. For our system the size is a small package of 20x25x12mm and a weight of around 10g.

Geofence and other capabilities

The traditional way limit the areas where the animals could go was to set up fences or use natural barriers. Nofence allows the user to set up allowed or disallowed areas for the animals, the collar gives the animal audio warnings or electric shocks when nearing the border of these areas. The system from findmy will give the user alerts if the animal is outside its allowed area. Our system does not provide any geofence functionality.

The system from findmy also features notifications when the animals get into potentially dangerous situations.

System maturity

The traditional bell collar is well tested and have been in use for many centuries. The systems by Nofence and findmy have been commercially available for a few years. They have been field tested performed satisfactory in real world conditions. Our system have only been tested at larger scale a few times and the system is not ready for commercial use.

Summary

The traditional way is the simplest and cheapest, but it requires a lot of labour. Our system have around the same unit cost as the bell collar, but we also requires a UAV to function. The systems from Nofence and findmy allows tracking of the animals remotely all season, but they are also much more expensive and both of them requires an active subscription. Nofence have built in "herding" functionally but also much shorter battery life than our system or the system from findmy. Our system does not have any kind of geofence functionality. The other systems are already available commercially.

Our system is not a replacement of any of the other system, but functions more as a auxiliary tool for the traditional way to gather up the animals. It's more like tool to locate the areas where the animals reside. It does not provide any new features unlike the systems from Nofence and findmy.

5.6 Possible regulatory obstacles

Our system is well suited for use cases where the UAV could operate autonomously to survey large areas and spend a long time beyond visual line of sight(BVLOS). In Norway and the European Union any UAV operating BVLOS is categorized in the specific category. This means it could require operational authorisation from the National Aviation Authority. This could be a huge limitation for adoption of our system. Also the speed limit of 19m/s is not a major problem but it could increase the time required to survey an area.

Chapter 6

Conclusion

When we started working on this master thesis we got a task to solve, locating animals with the use of UAVs and radio technology. The task was split into three parts, each focusing on a separate component of the full system. One part was focused on detecting and finding the animals using radio technology. Another part focused on developing a UAV platform for this system and then integrating it with the rest of the system. And the last part was focused on developing a GCS application to control this system and process the data generated. This paper focuses on the UAV platform and integration with the rest of the system.

Even if our main focus here was the UAV and integration we could not ignore the other components of the system. Therefore discussion about the other components is included and how the system functions as a whole.

The UAV platform we decided to use in our system was a multi-rotor quadcopter running the ArduPilot software. This UAV was capable of autonomous flight, a requirement to allow automated surveying of areas. This means the UAV could execute given a flight plan without any input from a pilot. From our testing we estimate the UAV could have a range of 30km and capable of holding speeds of 19m/s.

Our tests of the SheepRTT modules shows they can measure distances up to least 1.2km with an uncertainty of $\pm 50m$. The uncertainty does not increase with distance. By doing multiple RTT distance measurements and averaging them we can reduce this uncertainty. The SheepRTT module can then be used with the rest of our system, the UAV and GCS to survey an area and locate animals within it. With the full system surveying an area we should be able to locate animals fairly high accuracy. In the testing of our system it managed to locate the tags with a high accuracy, only being off by $15m$ on average.

Compared to already existing solutions our system provides geolocation with a very lightweight and low cost unit. The major downside of your system is the need fly a SheepRTT equipped UAV nearby to locate the tags. The tags do not posses any ability to do this on their own. Our system does not provide any geofence capabilities as is provided by some GPS collars. Another feature of most GPS collars is the ability to check the animals' location at any time.

Our system is based around the UAV flying autonomous flights and surveying remote areas. The areas could be far away and beyond visual line of sight. The regulatory requirement for operating a UAV beyond visual line of sight(BVLOS) could pose a big challenge for our system and could limit it's practical use.

Our system should be compatible with any ArduPilot based UAV. Compatibility

can later be expanded to include all MAVLink based UAVs with some modifications.

Compared to other solutions this system adds extra complexity with the addition of the UAV and does not provide any new features. This combined with the possible regulatory obstacles could make the whole system less practical. The knowledge gained may be useful for developing future solutions to other problems. Examples being the SheepRTT system being used to measure distances and the knowledge about MAVLink being used to develop other MAVLink enabled devices.

6.1 Further Work

Further work can be done to expand the capabilities of this system. The UAV platform used by this system can be changed to one with a longer range, possibly using fixed or hybrid wing designs. Also a better MAVLink connection between the UAV and the GCS. Better stability and longer range would make large scale tests and real world scenarios much more practical.

Modifying the system to allow multiple SheepRTT modules on a UAV could increase the flexibility of the system.

Adding support the UAVs running the PX4 and other MAVLink based firmware would require fairly little work and would increase the number of UAVs compatible with our system.

Developing the planned prototype tag a smaller design for the tag with integrated batteries would be a big step forward as this would make testing with real life animals much easier.

We also need to look deeper into the regulatory obstacles and determine if this system can be viable under the current regulations.

Bibliography

[1] Ardupilot copter: Flight modes. https://ardupilot.org/copter/docs/flight-modes.html, . Accessed: 2021-03-12.

[2] Github - ardupilot. https://github.com/ArduPilot/ardupilot, . Accessed: 2020-11-29.

[3] Mavlink routing in ardupilot. https://ardupilot.org/dev/docs/mavlink-routing-in-ardupilot.html, . Accessed: 2021-02-03.

[4] Ardupilot - sitl simulator (software in the loop). https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, . Accessed: 2021-02-14.

[5] Holybro kakute f7 mini. https://ardupilot.org/plane/docs/common-holybro-kakutef7mini.html, . Accessed: 2020-09-29.

[6] Drone manufacturer market shares: Dji leads the way in the us. https://droneii.com/drone-manufacturer-market-shares-dji-leads-the-way-in-the-us. Accessed: 2021-01-06.

[7] Mavlink developer guide. https://mavlink.io/en/, . Accessed: 2021-02-06.

[8] Dialect: Ardupilotmega. https://mavlink.io/en/messages/ardupilotmega.html, . Accessed: 2021-02-03.

[9] Mavlink dialects. https://mavlink.io/en/messages/, . Accessed: 2021-02-05.

[10] Mavlink messages: Mav_component. https://mavlink.io/en/messages/common.html#MAV_COMPONENT, . Accessed: 2021-03-01.

[11] Mavlink: Using pymavlink libraries (mavgen). https://mavlink.io/en/mavgen_python/, . Accessed: 2021-03-05.

[12] Mavlink: Message signing (authentication). https://mavlink.io/en/guide/message_signing.html, . Accessed: 2021-03-22.

[13] Mavlink common message set. https://mavlink.io/en/messages/common.html, . Accessed: 2021-01-16.

[14] Mavlink microservice: Heartbeat. https://mavlink.io/en/services/heartbeat.html, . Accessed: 2021-04-15.

[15] Mavlink messages: Microservices. https://mavlink.io/en/services/, . Accessed: 2021-03-06.

[16] Mavlink routing. https://mavlink.io/en/guide/routing.html, . Accessed: 2021-03-01.

[17] Packet serialization. https://mavlink.io/en/guide/serialization.html, . Accessed: 2021-02-03.

[18] Mavlink 2. https://mavlink.io/en/guide/mavlink_2.html, . Accessed: 2021-02-03.

[19] Mavlink versions. https://mavlink.io/en/guide/mavlink_version.html, . Accessed: 2021-02-03.

[20] Federal communications commission record: Before the federal communications commission washington, d.c. 20554. https://docs.fcc.gov/public/attachments/FCC-94-213A1.pdf, 1994. Accessed: 2021-05-12.

[21] Current and planned global and regional navigation satellite systems and satellite-based augmentations systems. https://www.unoosa.org/pdf/publications/icg_ebook.pdf, 06 2010. Accessed: 2021-05-13.

[22] European Union Aviation Safety Agency. Easy access rules for unmanned aircraft (regulations (eu) 2019/947 and (eu) 2019/945). 2020-12-03. https://www.easa.europa.eu/sites/default/files/dfu/Easy%20Access%20Rules%20for%20Unmanned%20Aircraft%20Systems%20November%202020.pdf.

[23] Team Betaflight. Betaflight about. https://betaflight.com/. Accessed: 2020-10-18.

[24] Marinus Boon, A. Drijfhout, and Solomon Tesfamichael. Comparison of a fixed-wing and multi-rotor uav for environmental mapping applications: A case study. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W6:47–54, 08 2017. doi: 10.5194/isprs-archives-XLII-2-W6-47-2017.

[25] Dronecode. Software overview: Px4 autopilot. https://px4.io/software/software-overview/. Accessed: 2020-10-18.

[26] Emad Samuel Malki Ebeid, Martin Skriver, Kristian Terkildsen, Kjeld Jensen, and Ulrik Schultz. A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems*, 61, 05 2018. doi: 10.1016/j.micpro.2018.05.002.

[27] Bernd Eissfeller, GERALD AMERES, Victoria Kropp, and DANIEL SANROMA. Performance of gps, glonass and galileo. 01 2007.

[28] findmy. findmy. https://www.findmy.no/.

[29] Oeystein Glomsvoll and Lukasz Bonenberg. Gnss jamming resilience for close to shore navigation in the northern sea. *Journal of Navigation*, -1, 06 2016. doi: 10.1017/S0373463316000473.

[30] Carles Gomez, Joaquim Oller Bosch, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors (Basel, Switzerland)*, 12:11734–53, 12 2012. doi: 10.3390/s120911734.

[31] iNavFlight. iNAV wiki. `https://github.com/iNavFlight/inav/wiki`. Accessed: 2020-10-21.

[32] Fortune Business Insights. Small drone market size, share & covid-19 impact analysis, by type (fixed wing, rotary wing, and hybrid wing), by end-use (military & defense, consumer, and commercial & civil), by maximum take-off weight (less than 5 kg, 5-25 kg, and above 25-150 kg), and regional forecast, 2020-2027. `https://www.fortunebusinessinsights.com/small-drones-market-102227`, 11 2020. Accessed: 2021-02-25.

[33] Pedro L Jimenes, Jorge A Silva, and Juan S Hernandez. Experimental validation of unmanned aerial vehicles to tune pid controllers in open source autopilots. In *European Conference For Aeronautics And Space Sciences (Eucass)*, 2017.

[34] Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgui. Micro air vehicle link (mavlink) in a nutshell: A survey. *IEEE Access*, PP:1–1, 06 2019. doi: 10.1109/ACCESS.2019.2924410.

[35] Marco Lanzagorta. Underwater communications. *Synthesis Lectures on Communications*, 5(2):1–129, 2012. doi: 10.2200/S00409ED1V01Y201203COM006. URL `https://doi.org/10.2200/S00409ED1V01Y201203COM006`.

[36] Xingxing Li, Maorong Ge, Xiaolei Dai, Xiaodong Ren, Mathias Fritsche, Jens Wickert, and H. Schuh. Accuracy and reliability of multi-gnss real-time precise positioning: Gps, glonass, beidou, and galileo. *Journal of Geodesy*, 89, 03 2015. doi: 10.1007/s00190-015-0802-8.

[37] Trygve Nerland. Github: trygve55/pymavlink. https://github.com/trygve55/pymavlink.

[38] Nofence. Nofence. https://www.nofence.no/, .

[39] Supernode Nofence. Nofence: Regn ut hva nofence vil koste for akkurat ditt behov. https://nofence.supernode.no/sheep.html, .

[40] Henrik Nyholm. Localizing sheep using a bluetooth low energy enabled unmanned aerial vehicle for round-trip time of arrival-based multilateriation. Master's thesis, NTNU: Norwegian University of Science and Technology, 07 2020.

[41] Maria Ribeiro and Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. 04 2004.

[42] Adnan Saeed, Ahmad Bani Younes, Chenxiao Cai, and Guowei Cai. A survey of hybrid unmanned aerial vehicles. *Progress in Aerospace Sciences*, 03 2018. doi: 10.1016/j.paerosci.2018.03.007.

[43] Gard Steinsvik. Github: Gardsteinsvik/radio-sheep-gcs. https://github.com/GardSteinsvik/radio-sheep-gcs.

[44] Gard Steinsvik. Radio-tracking of sheep — ground control station. Master's thesis, NTNU: Norwegian University of Science and Technology, 06 2021.

[45] Grzegorz Swiderski. Radio tracking of sheep: Low-cost energy-efficient coarse distance estimation using bluetooth low energy transceivers. Master's thesis, NTNU: Norwegian University of Science and Technology, 06 2021.

[46] ArduPilot Dev Team. ArduPilot about. `https://ardupilot.org/index.php/about`, . Accessed: 2020-10-16.

[47] ArduPilot Dev Team. ArduPilot command protocol. `https://mavlink.io/en/services/command.html`, . Accessed: 2021-05-20.

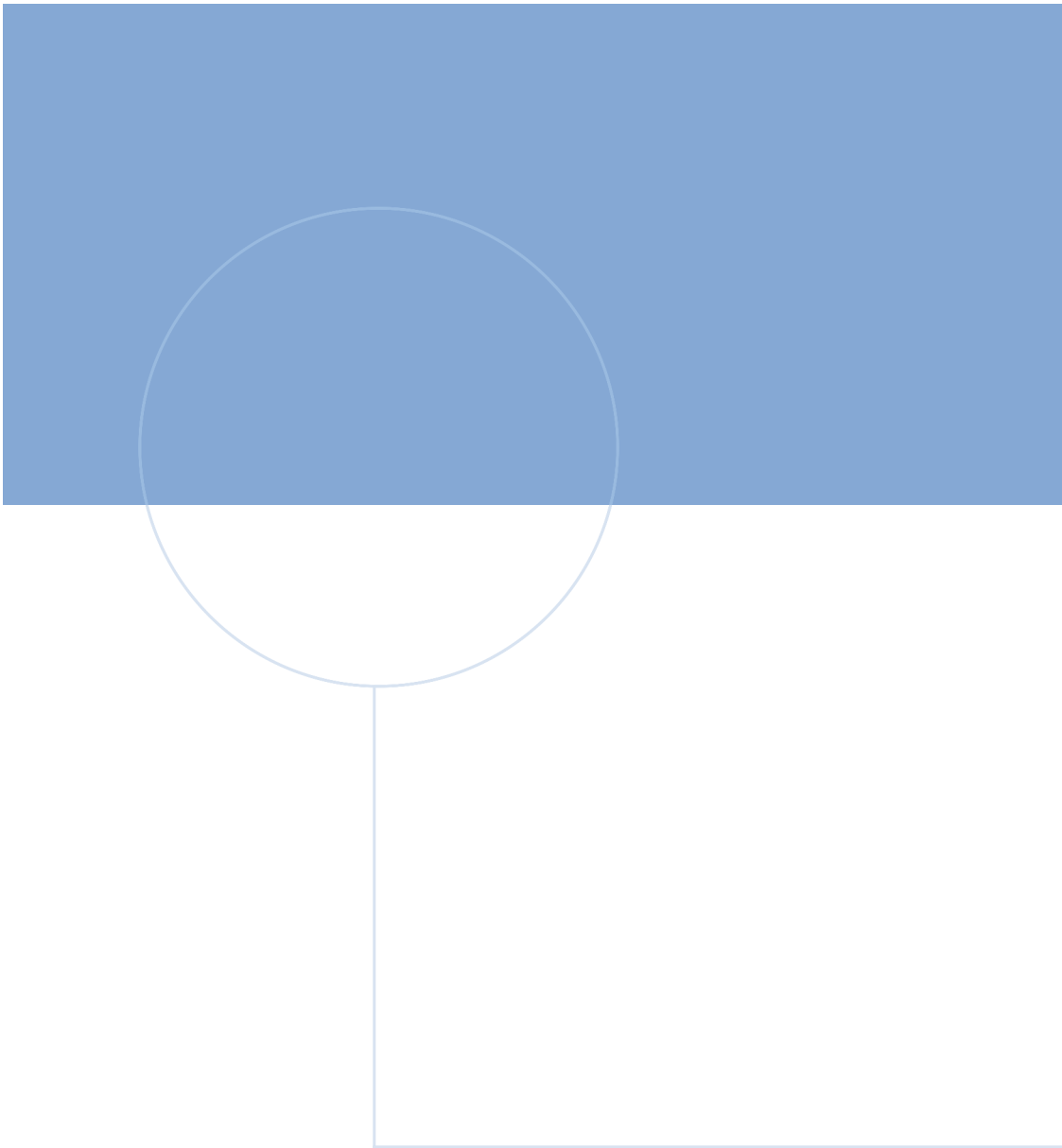
[48] ArduPilot Dev Team. ArduPilot extended kalman filter (ekf). `https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html`, . Accessed: 2021-04-18.

[49] ArduPilot Dev Team. ArduPilot mavproxy. `https://ardupilot.org/mavproxy/`, . Accessed: 2021-05-28.

[50] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[51] Daihua Wang, Linli Song, Xiangshan Kong, and Zhijie Zhang. Near-ground path loss measurements and modeling for wireless sensor networks at 2.4 ghz. *International Journal of Distributed Sensor Networks*, 2012, 08 2012. doi: 10.1155/2012/969712.

[52] yappu. Yaapu Frsky Telemetry Script wiki. `https://github.com/yaapu/FrskyTelemetryScript/wiki`. Accessed: 2021-05-27.



NTNU

Norwegian University of
Science and Technology