Jonathan S. Linnestad & Ole Håkon K. Ødegaard

Utilizing Drones to Automatically Cover Outfield Pastures

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2020

Master's thesis

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

NTNU
Norwegian University of Science and Technology

Jonathan S. Linnestad & Ole Håkon K. Ødegaard

Utilizing Drones to Automatically Cover Outfield Pastures

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2020

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

Abstract

This paper represents a master thesis at NTNU in Trondheim. The overall goal is to prove the concept of using consumer-available technology to automatically cover an area for finding animals grazing in outfield pastures. With an iterative approach, a complete system for automatic flights covering a user-defined search area is implemented using a drone.

The user defines a search area using a mobile application. From this search area, the system produces a covering flight path. This flight path considers changes in the terrain and allows the drone to fly at a consistent altitude. The height adjustments are made by using official height data from Norway. The proposed solution utilizes polygonal shapes. It is verified that path planning in a convex environment is easier than in a concave; hence, convex decomposition is performed on concave polygons.

The feasibility of the system was verified by conducting multiple tests, both for components in isolation and for the overall system. Test results show that the system works as intended, but that more work is needed to make the system production-ready and safe in different environments. The conclusion is that such a system is feasible.

Sammendrag

Dette dokumentet representerer en masteravhandling ved NTNU i Trondheim. Det overordnede målet er å verifisere brukbarheten til et system som dekker over et område for å overvåke dyr som beiter på utmarksbeite. Dette skal gjøres ved hjelp av forbrukerteknologi. Gjennom en iterativ prosess er det implementert et komplett system for å dekke over et område ved hjelp av en autonom drone.

Brukeren definerer et søkeområde i en mobilapplikasjon. Fra dette området produserer systemet en flyrute. Denne flyruten tar hensyn til høydeforskjeller i terrenget og dronen flyr med en konstant høyde over bakken. Høydejusteringene gjøres ved hjelp av offisiell høydedata fra Norge. Den foreslåtte løsningen benytter seg av polygoner. Det er verifisert at stiplanlegging av konvekse polygoner er lettere enn konkave. Løsningen deler derfor opp konkave polygoner i mindre, konvekse polygoner ved hjelp av konveks dekomponering.

Det har blitt utført en rekke tester, både for hver komponent og for den overordnede løsningen. Testene har verifisert brukbarheten av systemet og viser at det fungerer som tilsiktet. Det stilles derimot krav til ytterligere arbeid før løsningen kan benyttes i ulike andre miljøer. Det konkluderes med at et slikt system er brukbart.

Preface

This paper represents a master's thesis conducted at the Norwegian University of Science and Technology (NTNU), at the Department of Computer and Information Science (IDI). The thesis is written as a cooperative work between the authors Linnestad and Ødegaard. Guidance is provided by IDI through supervisor Svein-Olaf Hvasshovd.

The thesis is written as an independent master's thesis. All work described in this paper, except for referenced work, is done by the authors from January to June 2020.

Acknowledgement

We want to thank NTNU for setting the basis of this work, through five years at Gløshaugen. The process has been truly developing for us, both when it comes to knowledge and personal manners.

We would like to pay our special regards to Svein-Olaf Hvassovd for being our supervisor for this thesis. His advice has been of huge help throughout the process, from not knowing what we were going to write about to the final word in the last chapter.

Ole Håkon Ødegaard: I would like to give a special thanks to my family. My mom, Marit, and my dad, Pål Håkon, have supported me through all years and have always reached out in times of need. Your interest and devotion to what I do have motivated me to learn new things and aim high. I would like to thank Mia for being beside me for the last couple of years and supporting me. I am really grateful, and I look forward to many years together with you. My two brothers, Lars Kristian and Per Marius have stood beside me from the day I was born – thank you for all the good times. I would also like to thank Jonathan for being my partner through this project. What a journey it has been!

Jonathan Linnestad: I would like to give appreciation to NTNUI Nerds and Abakus Linjeforening for the community and fellowship you have given me through my time in Trondheim. I would also like to give thanks to my family for supporting me. My dad, Niels, has nurtured my interest in technology and helped me find my passion for building and creating things, and my mom, Live, has been a pillar of emotional support and always listened and given advice, both on good and bad days. And a special thanks to Sunniva for making the last year in Trondheim the best year. Lastly, I would like to thank my co-author Ole Håkon Ødegaard for letting me beat him in ping-pong every day.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), commonly referred to as drones, have become more prevalent in the last couple of years. It started as a military invention, but has later been taken into use both in the industry and for other special operations. Examples of applications are search and rescue operations [1], wildlife monitoring [2], and medicine and equipment delivery services [3]. As drone technology has evolved through the years, they have become smaller, easier to produce, and, hence, cheaper.

Throughout the last decades, agriculture has become more industrialized and centralized. It has gone from many small to few, but large, farms that are capable of producing a more considerable amount of food and crops. To be able to monitor, manage, and run large areas of crop more efficiently, farmers apply innovative and technological tools – among them, drones. The use of drones makes the job easier, safer, and time-saving.

The ability to monitor farms from a bird's-eye view is beneficial to farmers, especially in rough terrains. Drones can access vast areas of ground efficiently with a low amount of risk. Human pilots can control the aircraft along its path, but may have difficulties observing the drone's complete environment. When the drone is out of sight, it is reasonable to assume that the drone itself is better suited to sense the surrounding environment and act accordingly to preserve a safe and efficient flight. To do so, the drone will have to be equipped with the necessary tools to allow for automatic flights.

From Linnestad and Ødegaard [4], we have that sheep farmers are struggling to fulfill the authorities' requirements on the amount of animal supervision. At the same time, many animals are injured or killed by predators. This paper describes a proof of concept where the overall goal is to verify if drones safely can be utilized to cover an area for monitoring animals in outfield pastures. Since a majority of animals in the outfields of Norway are sheep, the focus lies on, but are not restricted to, sheep herding.

To isolate underlying concerns, the paper focus on answering the following research questions:

1. *How can a consumer drone be programmed to safely follow a predefined pattern automatically?*
2. *How can a path be defined so that it efficiently covers a large range of polygons?*
3. *Is a system for automatically covering a user-defined search area efficiently feasible, using consumer available technology?*

Each research question is in fact dependant on the previous one. Number three represents the final goal. If any of the former research questions fails, the system is unlikely feasible. In other words, the drone has to be able to fly automatically and a path covering an area must be derived from a predefined polygon.

The project is done iterative, and the paper is written chronologically. Chapter 2 introduces how sheep herding are done today and common challenges. It also introduces different drone designs and what environmental challenges one have to consider when flying a drone. Chapter 3 gives an introduction to the drone used in this paper, Chapter 4 implements automatic flights into a drone, Chapter 5 describes how a user can interact with a map in a mobile application, and Chapter 6 and 7 find a suiting path to cover a predefined search area. Chapter 8 proposes the final solution and puts components from the previous chapters together. Each component is tested independently, and three full-scale tests are conducted and described in chapter 8. Chapter 9 discuss results and the proposed solution, and a final conclusion is given in Chapter 10.

Chapter 2

Background

Many Norwegian farmers place their livestock in forests or on mountains during the summer. The animals feed on natural grass fields and nearby water resources while farm pastures are left to grow. This makes animal farming less expensive and time-consuming.

On the other hand, as animals are left in the wilderness, chances of accidents and predator attacks increase. Rough terrain and harsh weather conditions can endanger the animals' physical and mental health, especially for young animals. To cope with these challenges, farmers, individually or through cooperation, take walks in grazing areas, making sure the animals are healthy and unharmed. Such operations often involve long walks over hills and valleys and might be dangerous for the farmer. Farmers need a better way to watch over their stock in outfield pastures.

A possible solution is to take advantage of a drone's elevated view and use this to monitor the livestock in the terrain. Drones can cover large areas of ground in a short time without endangering farmers or animals. At the same time, the aircraft can be equipped with tools, such as radio receivers and thermal cameras, to aid animal surveillance.

2.1 Sheep Herding on Outfield Pastures

Animals' health is protected by Norwegian laws and regulations, for instance, through the Animal Welfare Act [5] and Regulations of Welfare for Sheep and Goats [6]. All animal farmers are obligated to treat their livestock well and protect them from dangers and unnecessary stress. This also applies for animals on outfield pastures.

2.1.1 Statistics

According to statistics from 2019, more than 2,2 million of the Norwegian livestock graze on outfield pastures during the summer season [7]. A majority of these animals are sheep. At the same time, there are several predatory animals residing in the same areas. From the year 2000 to 2019, statistics show a yearly average of 291 animal injuries caused by bears, 248 by wolfs, 187 by lynx, and 91 by wolverines [8]. It is estimated that a total of more than 100 000 sheep died on outfield pastures during the 2016 season [9]. More than 65% was assumed deceased from other reasons than predators, such as illness, accidents, or not found during sheep gathering. From these statistics, it is clear that a large number of animal deaths and injuries can be avoided. By introducing new tools and technology for better and more efficient animal monitoring, we hopefully can see a decrease in animal injuries on outfield pastures.

2.1.2 Today's Situation

To investigate how sheep herding can be improved, we need to understand today's situation, hence procedures and tools in use. According to Linnestad and Ødegaard [4], shepherds do walks on outfield pastures multiple times a week. Trips are usually planned and carried out as a cooperative arrangement between multiple farms. Farmers with animals grazing in the same areas often have some kind of partnership to increase the efficiency of the shepherding. Such methods improve the quality of shepherding. Still, as animals usually are widely spread over large areas of woods and hills, it can go several days before injured animals are discovered and brought back for treatment. Sheep also tend to hide whenever injured or frightened. Because of this, they are often hard to find – especially in areas with rough terrain and thick forest.

Linnestad and Ødegaard states that most of the farmers use binoculars when they are watching over their sheep. With binoculars, sheep can be seen from a longer distance, which makes it more efficient and less exhausting for the farmers. At the same time, it seems that many are struggling to fulfil the authority's recommendations on how often one should watch the animals. Can drones be used to accomplish this goal?

2.2 Drones

As mentioned, drones are unmanned aerial vehicles. This section will investigate common types of drones, compare them, and go into greater details about the most common consumer drone, namely the quadcopter.

2.2.1 Different Drone Designs

There are several types of drones. The definition of a drone may include several things, from weather-balloons to drones used in military operations, such as the MQ–9 Reaper with a wingspan of 66 meters [10]. As this paper focuses on covering an area, specifically in the wilderness, where farm animals graze, it makes sense to look at common types of drones used in the consumer market. The following are some common drone types [11]:

- **Multi Rotor Drones:** These drones have multiple rotors. The most common is the quadcopter having four rotors. Quadcopters are relatively cheap to make; they can take off from a standstill, and hover. It has a relatively high energy consumption as it has to use energy constantly to stay elevated.
- **Fixed-Wing Drones:** These look like ordinary aeroplanes with a wing and a propulsion system. They need a runway or a launcher to take off and are not able to hover. Since fixed-wing drones use air to generate lift, they are more energy-efficient than drones. They use energy to move forwards, rather than upwards.
- **Single Rotor Drones:** These are more akin to helicopters with a single rotor blade on the top, and usually, one at the tail, spinning vertically to keep it in balance. They are harder to manufacture and, hence, more expensive compared to other types.
- **Hybrid Vertical Take-Off and Landing (VTOL) Drones:** These are hybrids of fixed-wing and rotor based designs and inherit the advantages of taking off and landing anywhere while being energy efficient in the air.

They all have different capabilities and are used for various purposes. Due to the drone used in this paper being a quadcopter, this is the drone design that will be highlighted.

2.2.2 Quadcopters

Quadcopters have four propellers. In helicopters, there is a single rotor with multiple blades. Each blade can tilt around itself and helps stabilize and move the aircraft. In quadcopters, however, the blades are rigid, meaning movement and control come from the difference in uplift, between the four propellers. The aircraft will move based on the velocity of each propeller. Propellers move in different directions to keep the quadcopter stable. This is illustrated in figure 2.1. If all propellers were spinning in the same direction, the drone itself would spin in that direction as well. On single-rotor helicopters, this effect is handled by a tail rotor. On quadcopters, the rotors are moving in opposite directions, and the turning forces on the drone nullifies.

A quadcopter moves by changing its pitch, roll, and yaw as well as throttle. The throttle is simply how fast the propellers spin. Decreasing the throttle on a hovering aircraft will cause it to descend. If the throttle increases, the quadcopter ascends. The following list shows how the aircraft acts to changes in pitch, roll, and yaw:

- *Pitch* makes the drone move forward and backwards. The aircraft will tilt either forward or backwards and move in the direction it tilts. To pitch the drone in one direction, propellers on the opposite side of the aircraft will have to increase their thrust.
- *Roll* is a rotation around the drone itself. The aircraft will roll either to the right or to the left and will move in the rolled direction. When the thrust on the left propellers increases, the aircraft rolls to the right.

- *Yaw* is a rotation around the vertical axis. A change in yaw makes the aircraft's direction (or heading) change. An example: the aircraft is hovering and face north. It yaws clockwise and stops when it has turned 90 degrees. It is now facing east. Yaw changes by increasing the throttle on propellers spinning in the same direction as the yaw.

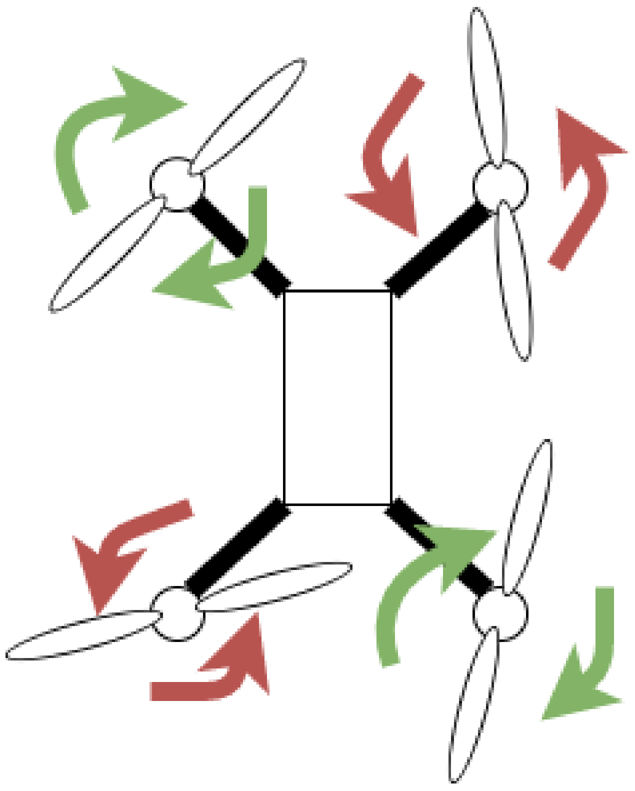


Figure 2.1: Alternating directions of propellers on quad copters. Green is clockwise, red is counter-clockwise.

2.3 Environmental Considerations

To evaluate the use of drones for animal surveillance, we need to consider different environments in which the drone may operate. Such environments may change based on weather conditions, time of the year, time of the day, locality, or other factors. For the drone to be able to fly safely and perform its objective, we present important considerations and evaluate them with respect to the present problem.

2.3.1 Changes in Wind and Weather Conditions

In this paper, we only consider the use of drones in outdoor environments. In an outdoor environment, changes in weather and wind conditions can happen quickly. For small and light drones, heavy wind can potentially cause the drone to lose control and make it crash, hit an object, or damage the aircraft's components. It is primarily the responsibility of the operator on the ground to make sure the weather conditions are ideal for flying the drone. Strong wind or heavy rain might suddenly appear and damage the drone, so the pilot must take the necessary precautions before taking off.

Studies [12] have shown that when the wind is heading in the opposite direction as the flight direction, the drone will use less power. The battery consumption will be lower when flying against the wind because of the increased lift the drone gets when it is moving horizontally. When the drone hovers in the same position, it does not gain any lift from the wind – the propellers have to do all the work. As more air flows against the flight direction, less power is required to keep it elevated.

2.3.2 Stationary Objects

There are many kinds of stationary objects one has to consider when flying a drone outside. Some examples are houses, buildings, poles, antennas, towers, and power lines. The drone should keep a safe distance away from such objects to avoid a crash.

Many drones use a built-in compass to pinpoint the location and movement of the aircraft. High voltage power lines produce electromagnetic fields. If magnetic fields interfere with the drone's magnetic compass, it can affect the drone's manoeuvrability [13]. Depending on the distance to the power line, the results can be catastrophic. Radiofrequency towers and other steel and electronic objects can also produce the same kind of interference.

Outdoor environments often have vast vegetation. Trees and bushes in all kinds of shapes might appear in the drone's flight path. If this is the case, the drone needs to take action to avoid crashing into the trees. If a rotor from the drone hits a branch, this can be catastrophic for the aircraft. It will not just damage the rotor, but a free fall from a considerable height down to the ground can also damage other components of the drone. Personnel and animals on the ground can also be hurt if they are hit by the drone.

Many drones have onboard obstacle avoidance systems. Such systems use sensors to measure the distance between the aircraft and the surrounding obstacles. If the distance gets too small, the aircraft's flight controller will take necessary actions to avoid a crash. How well these systems work and how accurate they are might vary based on the drone and the object.

2.3.3 Moving Objects

Drones should fly at a safe distance from other flying objects, like birds, other drones, and crewed helicopters and aircraft. Hitting other flying objects can cause a hazard both to the objects themselves, and any personnel, both on the ground and in the air.

There have been multiple incidents [14, 15] where birds, and especially eagles, have attacked flying drones. One can assume that bird attacks are more likely to happen if the drone is flying close to a bird's nest. To avoid such incidents, one should avoid flying near nesting areas.

2.3.4 Regulations

Regulations on the use of drones vary depending on the country. Since this is a Norwegian study, we focus on laws and regulations conducted by the Norwegian government and aviation authorities in Norway.

The Norwegian Civil Aviation Authority (CAA)[16] distinguishes between *commercial use of drones* and *drones for leisure*. A commercial user is a person or organization making money from flying drones. If the drone is used as a hobby or a sport, we define it as leisure. Different rules and regulations may apply to the user depending on several parameters, like the drone type, the size of the drone, the weight of the drone, and the purpose of flying the drone.

Commercial drone operators are required to apply for a flight certificate. The CAA operates with three different certificates. Each of them has different restrictions and requires different applications (see Table 2.1).

Top Five Rules

The CAA has a list on their web page containing the top 5 most important rules for flying a drone as a hobbyist. These rules are developed based on *Regulations Concerning Unmanned Aircraft etc.* [17] and are considered as good guidance on how to operate a drone. The top five rules are as follows:

1. The drone should always be kept within your sight and operated in a careful and considerate manner. Never fly near accident sites.
2. Never fly closer than 5 km from airports unless you have explicit clearance to do so.
3. Never fly higher than 120 meters off the ground.
4. Never fly over festivals, military facilities or sporting events. Keep a distance of 150 meters.
5. Be considerate of others privacy. Take note of the rules concerning photos and films of other people.

The drone is kept within sight during all tests described in this paper, as stated by rule number 1. Rules 2, 3, and 4 are fulfilled by performing tests on extensive grasslands, football fields, or in the woods. Any photos or videos containing

Type of certificate	**Restrictions**	**Application**
RO1	• Max weight: 2.5 kg • Max speed: 60 knots • Requires a visual line of sight to the drone • Restricted to flight in daylight	Letter to the CAA with information about the organization and drone type
RO2	• Max weight: 25 kg • Max speed: 80 knots • Requires a visual or extended visual line of sight to the drone	• Pass online exam • Receive approval on an application containing a risk analysis and an operation manual for the use of the drone
RO3		• Pass online exam • Receive approval on an application containing a risk analysis and an operation manual for the use of the drone

Table 2.1: Comparison of three different operator classifications. RO = RPAS (Remotely Piloted Aircraft Systems) Operator.

personal information captured by the drone during any of the tests are handled confidential and deleted after each test. No photographic material containing personal information is stored or shared on public channels.

Privacy

Many drones have a camera device in addition to other sensor components. If any of these components capture or record information, e.g., images, sound recordings, or videos, about a person, privacy regulations apply. The Norwegian Data Protection Authority (DPA) has developed five recommendations [18] regarding data privacy and drones:

- Minimize the amount of recorded information about humans and other identifiable objects.
- Make yourself visible when operating the drone.
- Do not use pictures or information captured by the drone for other purposes than originally intended.
- Make sure all information captured by the drone is safely stored. Evaluate the security mechanisms in the software on the drone and any device storing

the information.
- In commercial operations, make sure to have a valid agreement with clients regarding the ownership of the captured information.

Chapter 3

The Drone

The work described in this paper utilizes a DJI Mavic 2 Enterprise Dual to test automatic flights. The Mavic 2 Enterprise Dual is an all-purpose foldable drone, with both a regular camera and thermal vision. It has four electric motors, a lithium rechargeable battery, as well as obstacle detection in all directions. This chapter highlights features of the drone used in this thesis, both in terms of hardware and software. All specifications are taken from the official user manual [19]. Table 3.1 highlights key specifications that are relevant to this project.

Specification	**Value**
Takeoff weight	899 grams
Max takeoff weight	1100 grams
Dimensions(Length/Width/Height in cm	32.2/24.2/8.4
Max accent speed	5 m/s in S-mode 4 m/s in P-mode
Max descent speed	3 m/s
Max speed	70km/h in S-mode 52km/h in P-mode
Max flight time	31 minutes at a consistent 25km/h
Operating temperature	-10°C to 40°C

Table 3.1: Relevant specifications of the DJI Mavic 2 Enterprise Dual.

3.1 Introduction

Given the difference in drones, both in capabilities, customizability and availability, it is important to look at the drone used for the tests and in the overall system. Although the system is built to be general, some parts depend on the drone itself. This chapter gives an overview of the drone used in this paper. Specifications of the drone are investigated and we discuss how these affect the proposed system.

3.2 DJI Drones

Dà-Jiāng Innovations (DJI) is the largest consumer-based drone manufacturer in the world. The Shenzen based company grossed over $2.7 billion in 2017 [20]. Drones produced by DJI are used in everything from photography and film-making to surveillance and agriculture. They offer a large variety of drones for different purposes, both for general consumers, as well as enterprise drones with specialized purposes. The different drone series they offer are as follows:

- Mavic – Foldable all-purpose drones
- Spark – Small relatively cheap drones
- Phantom – Drones with advanced aerial mapping capabilities
- Matrice – Highly durable and configurable, increased load capability

There are also several variations within each series. An example would be the Mavic series where the smallest drone – Mavic Mini – has a takeoff weight of 249 grams compared to the Mavic 2 Enterprise with a max takeoff weight of 899 grams.

3.3 Modes of Flying

The DJI Mavic 2 Enterprise Dual offers three different modes of flying. The modes affect how the aircraft responds, both in terms of speed and other features such as obstacle avoidance. The different modes are as follows:

- *P-mode* – Positioning mode (default). This mode limits the maximum flight speed, enables obstacle avoidance, and good GPS reception is key. This mode must be enabled to program the drone.
- *S-mode* – Sport mode. This mode disables obstacle avoidance and has a much higher maximum flight speed. It gives more direct control of the drone, compared to the other modes.
- *T-mode* – Tripod mode. Similar to P-mode, but with much slower maximum speeds. Used for stable shooting of images and precision flights.

As this paper aims at automatic flights, P-mode is the obvious choice.

3.4 Electric Motors and Propellers

The combined maximum takeoff weight of all four motors is 1100 grams. As the drone itself weigh 899 grams, this allows for up to 201 grams of additional equipment. The ascent speed has a maximum of 5 m/s and the descent speed a maximum of 3 m/s. This means that from hovering at ground-level to a maximum height of 120 meters, the drone would use 24 seconds, and 40 seconds back down. This is actually a significant time when the maximum flight time with one battery is roughly 31 minutes. Takeoff and landing will, in other words, steal ~3% of the total time of one flight with a fully charged battery, given an altitude of 120 meters.

The propellers featured on the Mavic 2 Enterprise are detachable. Each one connects to a specific motor, given by a colour-coding. This is important because the motors, and, hence propellers, rotate in different directions.

3.5 Battery

The battery can be detached from the aircraft. This means one can have multiple batteries and easily replace it if one battery runs out. As stated in the previous section, the battery pack lasts for approximately 31 minutes in ideal conditions. This is at a moderate speed of 24 km/h. With these metrics, the drone should be able to fly around 12 km, flying in a straight line, at the same altitude, under optimal weather conditions.

3.6 Remote Controller

The DJI aircraft is manually controlled by a remote controller. The controller has two sticks for controlling the aircraft, buttons to issue commands such as to start filming or take a picture, as well as a screen used to show the state of the aircraft, such as possible errors, collision detection, and altitude. There are two antennas attached to the controller. The transmission distance is 5000 meters unobstructed.

The controller can be connected to a smartphone to issue commands from the phone, change settings and see a video feed from the drone's camera. The connected phone can communicate with the drone through either the official DJI application or a custom-made application. Figure 3.1 shows an overview of how the different hardware components communicate.

3.7 Object Detection and Avoidance

The DJI Mavic 2 Enterprise Dual also has a vision system consisting of sensors around the drone. With the vision system, the aircraft can detect obstacles in its path. Depending on the mode of the aircraft, it will try to avoid the obstacle. This is done either by stopping, flying over, or around the obstacle. An operator can change the settings for which action to take. This is a built-in feature, and will not have to be implemented.

3.8 Software Development Kits

DJI offers Software Development Kits (SDK) to create custom functionality and applications. The SDKs provide functions that access the built-in features of the drone. The different official SDKs provided by DJI are as follows:

- Mobile SDK – Includes SDKs for both iOS and Android.
- UX SDK – Includes UX drop-in components to integrate functionality swiftly.
- Onboard SDK – Used to add functionality directly to the aircraft.
- Payload SDK – Allows for third-party hardware to communicate with the aircraft.
- Windows SDK – Allows Microsoft Windows applications to access the aircraft.

The SDKs have access to several parts of the drone. An example is the iOS SDK having access to the following:

- High and low-level flight control.
- Aircraft state through telemetry and sensor data.
- Obstacle avoidance.
- Camera and gimbal control.
- Live video feed.
- Access to drone media files.
- Route planning for automatic flight.
- State information from the battery and remote controller.

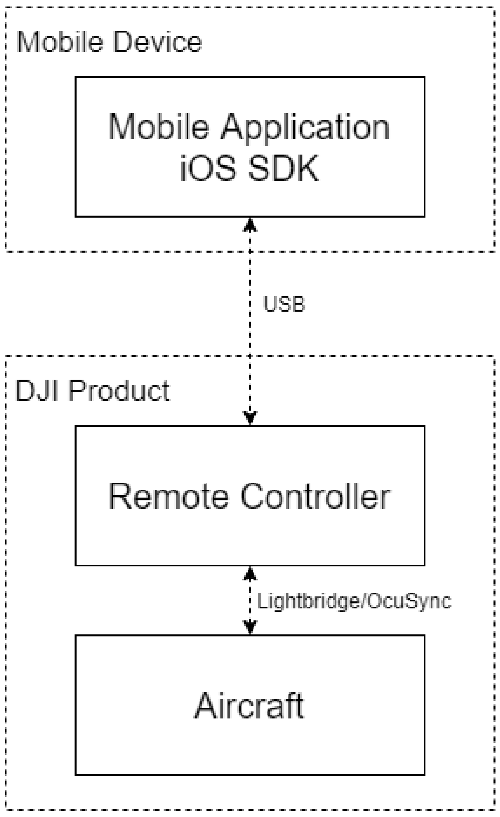


Figure 3.1: Overview of the communication between the mobile application using the iOS SDK, the remote controller, and the drone.

Route planning, flight control, obstacle avoidance, and state information is of high relevance to this project. Figure 3.1 shows how the SDK is used in an application to communicate with the aircraft through the remote controller. The use of the SDK in this project is described in greater detail in Chapter 4.

Chapter 4

Automatic Flight

4.1 Introduction

This chapter describes the choices and implementation of automatic flights with the DJI Mavic 2 Enterprise Dual. It is divided into sections from initial research to implementation details, automatic flight testing, results, discussion, and a conclusion. The goal behind the test in this chapter is to verify the possibility of automated flights along a defined path, as well as gather metrics, such as completion time, for different tasks. The results provide a foundation for the other parts of the system, such as path planning and user-defined search areas.

4.2 DJI SDKs

As the drone used in this thesis is a DJI Mavic 2 Enterprise Dual, some considerations are made due to custom functionality limitations. As described in chapter 3, DJI offers SDKs to control the drone programmatically and receive feedback from sensors. This section looks at the different SDKs provided by DJI and highlights the pros and cons in regards to this project. Features supported by the SDKs are shown in table 4.1.

4.2.1 Mobile SDK

The mobile SDK communicates with the aircraft through a mobile application connected to the remote controller. There exist versions both for iOS and Android. Combined, iOS and Android hold 100% of the market share in the world, with Android at ~87% and iOS at ~13% [21]. In Norway, the two operating systems are more evenly shared, with ~55% for iOS and ~45% for Android [22]. In other words, building a system for both Android and iOS will, in practice, cover the whole smartphone market. The iOS SDK is available in the programming languages Objective-C and Swift, while the Android SDK is available in Java.

Pros

A mobile application gives the apparent advantage of having a touch screen to display information on and to register input from the user. Medienorge says that 95% of all Norwegians have access to a smartphone [23]. This means that a system built for smartphones is highly available to the public. In addition to the high availability, a smartphone is usually compact in size and easy to transport. Other hardware, such as laptops, are more cumbersome to use in remote areas, due to their size and low battery capacity. The Mobile SDK also includes a lot of functionality relevant to this project, such as defining automatic flights.

Cons

As can be seen in Table 4.1, the Mobile SDK lacks a lot of the features of the Onboard SDK. Another limitation is the need for a smartphone to use the system. The Mobile SDK will require the use of three hardware parts: the phone, the controller, and the drone itself. This means the total cost of the system increases. However, as most people have a smartphone, this is not a practical limitation.

4.2.2 UX SDK

The UX SDK is a collection of premade UI elements using the Mobile SDK. As such, the UX SDK has a subset of the features of the Mobile SDK. These features are linked to UI elements, and the UX SDK aims to make application development easier.

Pros

The UX SDK allows for rapid development of standard functionality. The components in the UX SDK do not require a lot of programming, and you gain access to many common UI elements, such as maps, buttons, and video feed. The components use the underlying Mobile SDK and, as such, work on a higher level of abstraction.

Cons

As the components in the UX SDK are already built, they lack customizability. Adding additional functionality to a component is therefore limited. As the UX SDK builds upon the Mobile SDK, all the cons of the Mobile SDK also apply to the UX SDK.

4.2.3 Onboard SDK

The Onboard SDK is a set of different libraries to be used onboard the DJI drone, and this means the code runs directly on the drone (or controller). In other words, a mobile phone is not needed – only the drone and the controller – when using the Onboard SDK.

Pros

Table 4.1 shows that the Onboard SDK is the most feature-rich SDK. It shares the same features as both the Mobile SDK and the Windows SDK but has also access to more advanced features.

Cons

Due to being able to communicate with the Mobile SDK, there are almost no obvious disadvantages to the Onboard SDK. However, when looking at the ease of development, the Onboard SDK uses low-level languages and requires more in terms of setup and development expertise. The documentation is also poor in terms of examples.

Feature	**Windows SDK**	**Mobile SDK**	**Onboard SDK**
High and low level flight control	√	√	√
Aircraft state data	√	√	√
Obstacle avoidance	√	√	√
Camera and gimbal control	√	√	√
Live video feed	√	√	√
Pre-defined missions	√	√	√
State and control of battery and remote controller	√	√	√
Access to media stored on camera		√	√
Stereo vision video feed			√
Real-time disparity map			√
Custom local-frame navigation			√
Mobile SDK communication			√
Multi-function I/O ports			√
Synchronization with flight controller			√

Table 4.1: Features of the different software development kits (SDKs) from DJI.

Payload SDK

The payload SDK is used to program communication between third-party hardware and the drone.

Windows SDK

The Windows SDK offers almost all the features of the Mobile SDK but runs exclusively on a Windows Operating System. As such, it has a clear disadvantage, as the most common Windows systems being laptops and, thus, being less portable than a smartphone.

4.2.4 Conclusion

The Mobile SDK, specifically the iOS SDK, is found to be the best option for this project. There are several reasons for this choice. Firstly, as this system includes both a user interface and automatic flight, the choice of using the Mobile SDK means everything can be packaged into one application. If additional features provided by the Onboard SDK is needed, this can be added by using the Onboard SDK-to-Mobile SDK communication feature. Secondly, both authors are in possession of iOS devices, and for convenience, iOS is chosen over Android.

4.3 Implementation

This section discusses the initial setup of the mobile application, what features are required from the Mobile SDK and how the SDK is implemented.

4.3.1 Mobile SDK Details

Flight Control

Flight Control holds a set of functions that controls the aircraft and reads aircraft statuses. The Mobile SDK does not support all functionality in flight control. Most of them are documented, but during implementation, many of them return the following error message: *This feature is not supported by the SDK*. Therefore, such features can not be utilized.

Missions

Missions are used to automate flights. Missions are either managed by the application or uploaded and managed by the aircraft. Different types of predefined missions are offered by the SDK. The following are predefined mission types:

- *Waypoint Missions* use a series of coordinates (waypoints), automatically fly between each waypoint, and may execute an action at each. Waypoint Missions are uploaded and executed by the aircraft.

- *Hotpoint Missions* are fixed points that the aircraft will fly around at a constant radius. Adjustable parameters include speed, altitude, and direction.
- *FollowMe Missions* instruct the aircraft to follow a GPS-positioned object (e.g. mobile phone) at a fixed distance. This can be used when the object (e.g. a person with a phone) is moving, and the aircraft can not be remote-controlled by an operator.
- *ActiveTrack Missions* uses the aircraft's vision system to track a subject. The user defines the subject with a rectangle and confirms the subject to be followed.
- *TapFly Missions* allows the aircraft to move to an object marked by the user on the screen. It uses the visual system to determine the location and flies towards it.
- *Panorama Mission* turns the camera around while taking pictures: The pictures are then combined to create a panorama picture. This feature is not supported by the Mavic 2 Enterprise.

All these mission types inherit from the `MissionControl` class. The `MissionControl` class can be used to create custom missions. Missions are created by adding *Actions* to a *Timeline*. An *Action* is an operation for the aircraft to perform. A *Timeline* is a series of *Actions* to be executed sequentially. The timeline is uploaded to the aircraft, and the aircraft performs the first action when the timeline is started. Upon completion of an action, the aircraft performs the next. There are several types of actions. The most relevant ones are listed here:

- `GoToAction` – defines a location and altitude for the aircraft to move to. It also allows the flight speed to be set.
- `TakeOffAction` – makes the aircraft start its propellers and take off from the ground.
- `Landaction`: Makes the aircraft land and turn off its propellers.
- `GoHomeAction`: The aircraft will fly to its home location. The default home location is the point of takeoff.
- `ShootPhotoAction`: The camera will take a photo.
- `GimbalAttitudeAction`: the gimbal will change direction. This is used to move the camera.
- `AircraftYawAction`: Changes the direction of the aircraft.

Together these actions can be used to automate a flight, from takeoff to landing. While in the air, the `ShootPhotoAction` makes it possible to take a picture automatically. This, combined with `GoToAction`, can be used to survey an area.

4.3.2 Implementation of iOS Application

As the goal behind the first iteration is to get the drone to fly automatically in predefined patterns, as well as to know the capabilities of the drone, the first implementation focuses on a small part of the system, namely automatic flight in predefined paths. Thus two features are implemented: connecting to the drone and creating custom missions.

Connecting to the Drone

To use any of the features of the Mobile SDK, one first has to connect to the drone. This requires the phone to be connected to the remote controller. When the phone is physically connected to the remote controller, the application has to be registered using the `DJISDKManager`. Listing 4.1 shows how the mobile application is registered with the DJI Mobile SDK. A *SDK key*, provided by DJI, is stored in a configuration file and automatically read by the `DJISDKManager` upon registration.

```swift
//Example of registration app with the SDK

import DJISDK

class DJI: NSObject, DJISDKManagerDelegate {
   override init() {
      super.init()
      self.registerWithSDK()
   }

   func registerWithSDK() {
      DJISDKManager.registerApp(with: self)
   }

   // MARK: DJISDKManager Delegates

   func productConnected(_ product: DJIBaseProduct?) {
      NSLog("Product connected")
   }
}
```

Code listing 4.1: Swift implementation on how to register the application with the DJI Mobile SDK.

Listing 4.1 does not consider errors, such as not being connected to the drone and other failures. This is an example of how to register the application. The SDK manages most of the registration, leaving error handling and overhead control to the developer. Error handling is included in the final implementation [24]. In this case the application invokes `registerApp` which upon completion calls `productConnected`. When `productConnected` has been called, features of the Mobile SDK are available.

Mission

Once the application has been registered with the SDK and the product is connected, missions are available. Missions are explained in section 4.3.1. In the initial implementation, custom missions are used. Listing 4.2 shows a very simple use of custom missions.

```
1  func addTimelineElements() {
2     let missionControl = DJISDKManager.missionControl()
3     missionControl?.scheduleElement(DJITakeOffAction())
4     missionControl?.scheduleElement(DJIGoToAction(coordinate:
          CLLocationCoordinate2D(latitude: 63.418337, longitude:
          10.402769), altitude: 5)!)
5     missionControl?.scheduleElement(DJILandAction())
6  }
7
8  func startTimeline() {
9     DJISDKManager.missionControl()?.startTimeline()
10 }
```

Code listing 4.2: Swift implementation of a custom mission.

The *addTimeLineElements* function adds actions to the timeline. The following actions are scheduled between line 3 and line 5:

- `DJITakeOffAction` – returns an action for the aircraft to take off
- `DJIGoToAction` – returns an action for moving to a certain GPS-coordinate with a defined altitude
- `DJILandAction` – returns an action for landing the aircraft

When `addTimeLineElements` is called all these actions are uploaded to the aircraft. The aircraft then awaits the timeline to start. This is done by calling the `startTimeLine` function. Calling the function starts the execution of the timeline on the aircraft. By using custom missions, and these actions, all tests defined in 4.4 may be performed.

4.4 Test Plan

4.4.1 Introduction

This section describes a test plan for the first iteration. The goal of the test is for the drone to fly in predefined patterns and land automatically upon completion. Several tests are performed to verify the implementation of the iOS application. They also confirm that the current implementation works and satisfy all the requirements, and give metrics on the speed at which the drone can complete different actions. The test is mainly to get an idea of the capabilities of the drone, which are used as a base for further development. For every test, the time to complete each action and the time between each action are measured. This is used to determine the cost of different types of actions, as well as the cost of having multiple actions.

4.4.2 Automatic Flight Tests

This section describes and justifies each test. Figure 4.7 shows the tests #3-5 with a birds-eye view. Test #1 and #2 are excluded due to not changing coordinates, only altitude.

Test #1

Test #1 is simply a takeoff and a landing. The aircraft should take off and fly to approximately 70 cm, followed by landing safely in the same spot. This test shows whether the aircraft can land and take off automatically. These actions are essential for the rest of the system. Figure 4.1 shows each step and what will be measured for test #1.

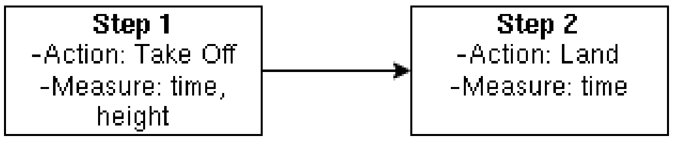


Figure 4.1: Steps in automatic flight test #1.

Test #2

Test #2 consists of taking off, increasing altitude to 2 meters, and landing. This test shows if the land action can be performed from a higher altitude than a finished takeoff altitude. Figure 4.2 shows each step and what will be measured for test #2.

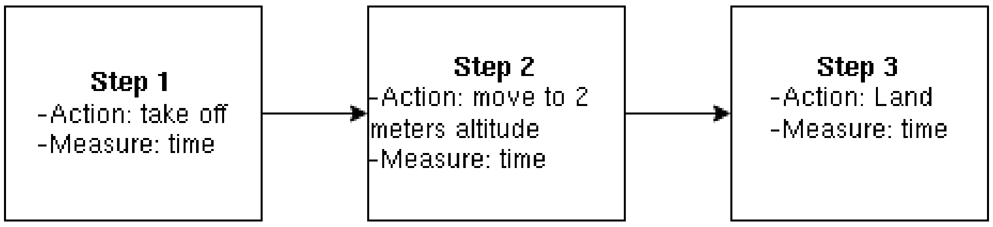


Figure 4.2: Steps in automatic flight test #2.

Test #3

Test #3 includes the following actions: takeoff, increase altitude to 5 meters, move to a defined location approximately 40 meters from the starting point, return to the takeoff location and land. This test shows if the drone can move to a defined location and how fast it moves. It also shows how fast the drone can do a 180° turn. Figure 4.3 shows each step and what will be measured for test #3.

Test #4

Test #4 is as follows: take off, move to a location while increasing altitude to 20 meters, return to the takeoff location while decreasing altitude to 1 meter, and land. This test is performed to check how the drone moves between two points with different altitudes. Figure 4.4 shows each step and what will be measured in test #4.

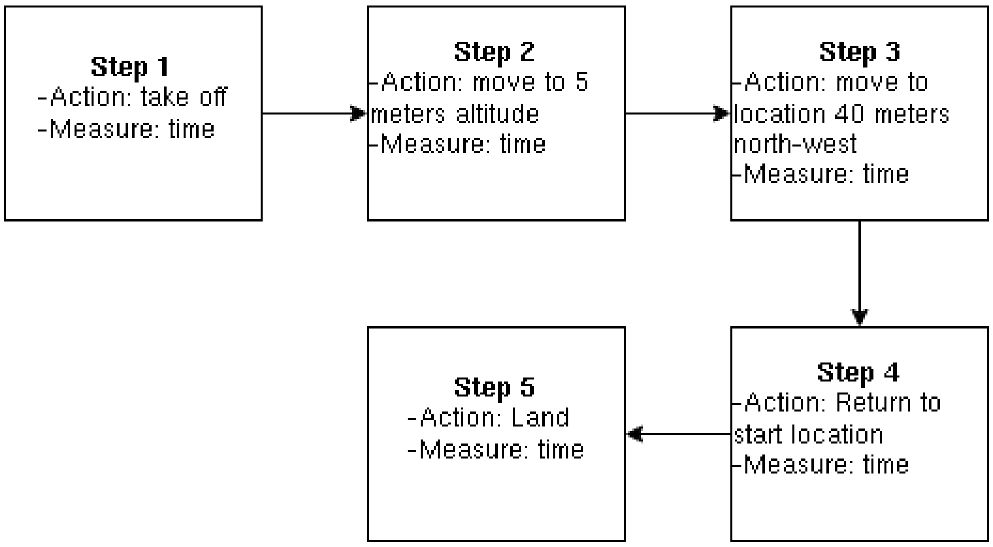


Figure 4.3: Steps in automatic flight test #3.

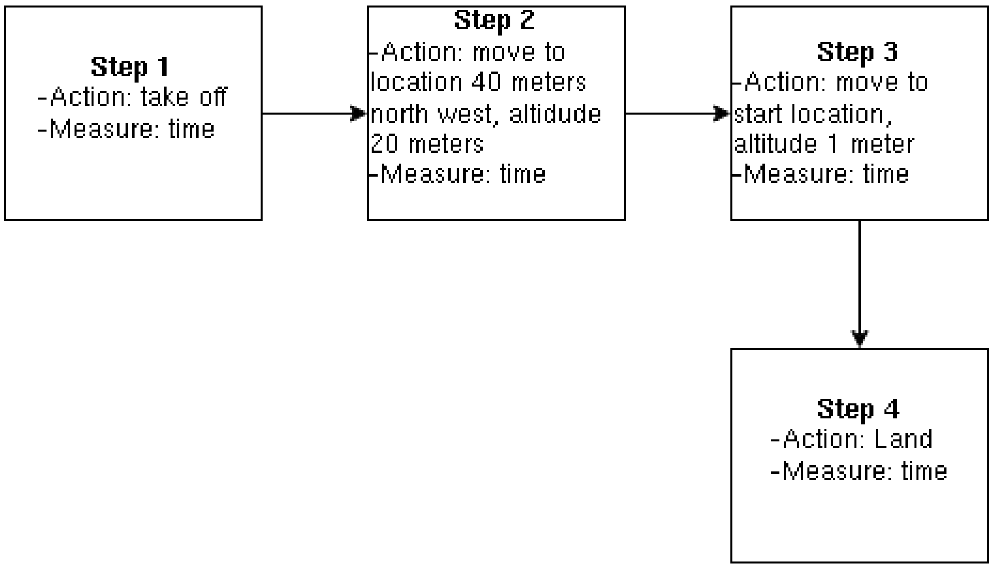

Step 1
-Action: take off
-Measure: time
Step 2
-Action: move to location 40 meters north west, altidude 20 meters
-Measure: time
Step 3
-Action: move to start location, altitude 1 meter
-Measure: time
Step 4
-Action: Land
-Measure: time

Figure 4.4: Steps in automatic flight test #4.

Test #5

Test #5 is as follows: take off, move to an altitude of 10 meters, move 20 meters, move another 20 meters in the same direction, return to start point, and land. This test is performed to see how much delay there are between consecutive actions. Figure 4.5 shows each step and what will be measured in test #5.

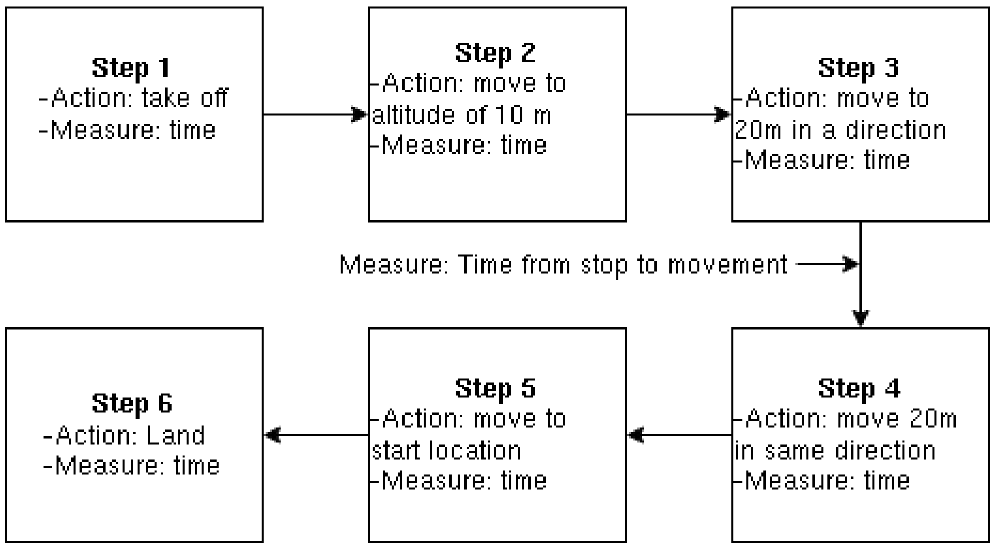

Step 1
-Action: take off
-Measure: time
Step 2
-Action: move to altitude of 10 m
-Measure: time
Step 3
-Action: move to 20m in a direction
-Measure: time
Measure: Time from stop to movement
Step 4
-Action: move 20m in same direction
-Measure: time
Step 5
-Action: move to start location
-Measure: time
Step 6
-Action: Land
-Measure: time

Figure 4.5: Steps in automatic flight test #5.

Test #6

Test #6 includes the following actions: take off, move in a square roughly ten by ten meters at an altitude of 5 meters, return to takeoff location, and land. This test verifies if the drone can move between several defined locations and return to the starting point. Figure 4.6 shows each step and what will be measured in test #6.

4.5 Results

The test was performed on the 10th of March at Dødens Dal in Trondheim, Norway. The time of the test was between roughly 14:30-14:40, and the wind in Trondheim at that time was between 4.3 m/s and 7.6 m/s [25]. This section presents results from the tests. Table 4.2 shows the total time for each test.

Test #	Time (s)
1	15.93
2	26.67
3	60.75
4	55.20
5	74.42
6	95.29

Table 4.2: Time from starting takeoff to finished landing for each of the 6 automatic flight tests.

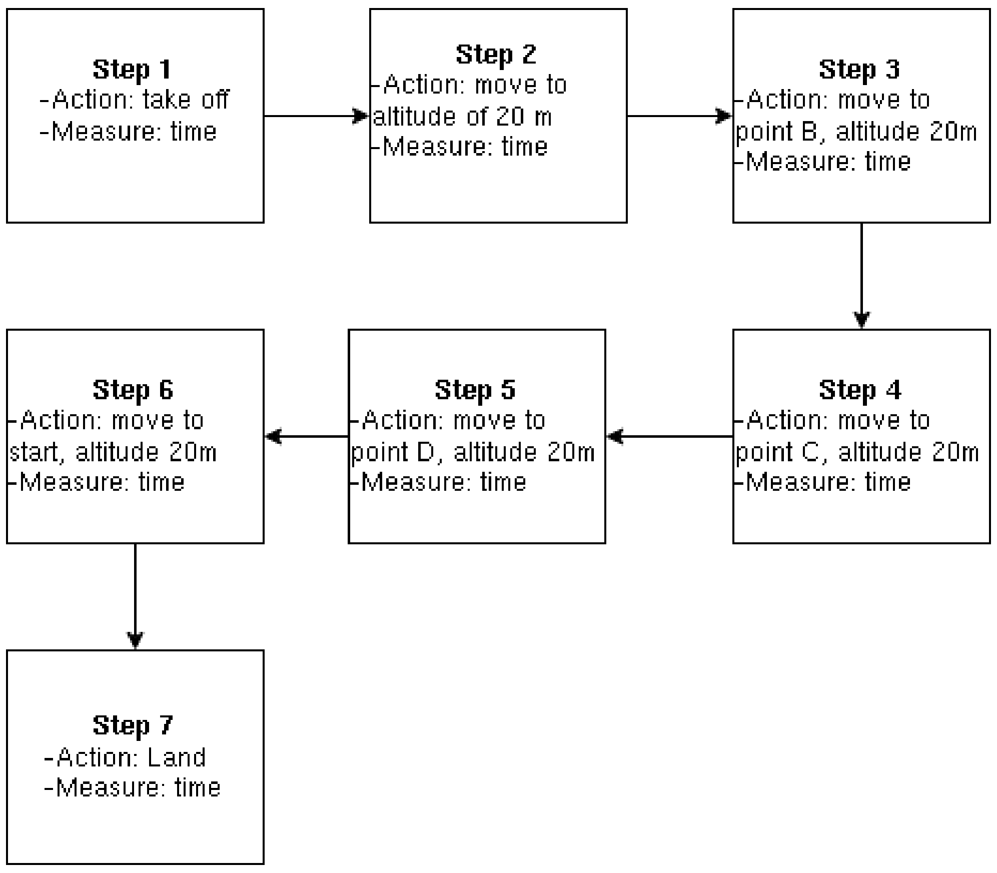

Step 1
-Action: take off
-Measure: time
Step 2
-Action: move to altitude of 20 m
-Measure: time
Step 3
-Action: move to point B, altitude 20m
-Measure: time
Step 4
-Action: move to point C, altitude 20m
-Measure: time
Step 5
-Action: move to point D, altitude 20m
-Measure: time
Step 6
-Action: move to start, altitude 20m
-Measure: time
Step 7
-Action: Land
-Measure: time

Figure 4.6: Steps in automatic flight test #6.

4.5.1 Takeoff

As every test includes a takeoff step, we can find out the average time used on a takeoff action. The tests indicate that the drone hovers at an altitude of 1 meter at the completion of the takeoff step. The average time is $\tilde{4}.93$ seconds to take off.

4.5.2 Intermediate Steps

Between two actions is an intermediate step where the drone hovers. This step averages to 5.39 seconds. Results show that the time used on the intermediate step depends on the preceding action. Table 4.3 shows different averages of time in intermediate steps.

4.5.3 Relevant Results of Each Test

Test #1

The total time of the test was 15.93 seconds. 7.06 seconds were used on the intermediate step between takeoff and landing.

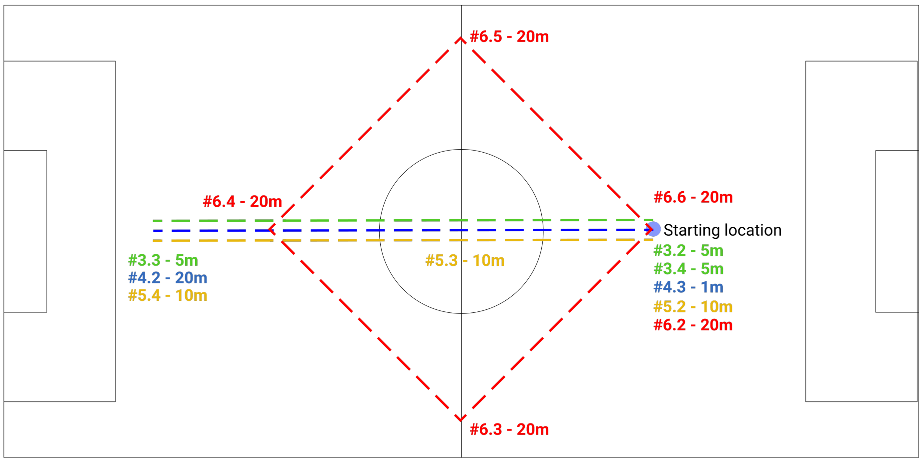


Figure 4.7: Illustration of steps in tests #3-6. Does not include steps for landing and takeoff. Green is test #3, blue is test #4, yellow is test #5, and red is test #6.

Intermediate step	Average time (s)
All intermediate steps	5.40
Intermediate step after takeoff	8.08
Intermediate step excluding takeoff	4.37

Table 4.3: Averages of intermediate steps through all automatic flight tests.

Test #2

The aircraft moved to a height of 1-meter altitude and hovered. This was followed by increasing the altitude to 2 meters and from there, directly landing. The total time used on test #2 was 26.67 seconds.

Test #3

Results from test #3 show that the aircraft used 10.24 seconds to the first point and 8.46 to move back. That is, two equal actions were performed, but the results show a different time usage. The total time used on test #3 was 60.75 seconds.

Test #4

The aircraft flew 20 meters away while increasing altitude. It flew in an inclined line to the designated point. It also flew inclined back to the takeoff location. Test #4 used a total time of 55.20 seconds.

Test #5

The aircraft used 4.94 seconds at step 5, hence, with no change in direction or altitude. Total time used was 74.42 seconds.

Test #6

The average time to turn 90° and fly 20 meters was 7.61 seconds, not including intermediate steps. Total time used was 95.29 seconds.

4.6 Discussion

There are several things to discuss, both in regard to the choices of implementation, the tests, and the results of the tests. This section discusses each, in turn, starting with the choices of implementation, followed by the test as a whole, finishing with the results of the tests.

In regards to the implementation, the choice fell on using MissionControl from the iOS SDK. This choice is the one with the most freedom, considering that most of the FlightControl features are not supported. This implementation does not utilize other actions than GoTo, TakeOff, and Landing. As such, it might have been easier, development-wise, to use Waypoint Missions instead. This will be a consideration when implementing further features of the system. Another point is that a custom mission uses a long time on each intermediate step. Other types of missions, e.g. Waypoint Missions, may be optimized internally to limit this.

The goal of the tests is, as stated, to see the capabilities of the aircraft and how it behaves in the air. As such, the tests were performed to find these capabilities. This means that although the tests were timed, the focus does not lie on the accuracy of these timings, nor the travelled distances. The results of the tests may have varied if performed under different conditions. And although time was measured under each test, the strength of these measures may be considered weak. They do however give several indications on how the aircraft performs and gives an estimate on the expected maximum flight distance.

Results from each test show that the aircraft performs as expected – all tests completed from start to finish. The aircraft travels to designated points and does so in an inclined line when altitudes differ. It also lands from any height given a land action. The most interesting results from the tests are the intermediate steps, averaging around 5.40 seconds between each action. This is a long time where the aircraft is not moving. The aircraft uses more time in intermediate steps directly following takeoff than in other intermediate steps. Intermediate steps still average out to 4.37 seconds, disregarding the intermediate step after takeoff. This means that for each additional action performed, another 4.37 seconds of flight time can be added. For an entire flight covering an area, this is a considerable amount of time. If the mission consists of 50 locations, that is an additional 3 minutes and 38.5 seconds where the aircraft is doing nothing. Long intermediate steps is also a consideration for actions that do not change direction or altitude, as shown by test #5.

As all tests were done under heavy wind, there is reason to believe that the times would improve under better conditions. The aircraft was visibly fighting the wind to get to the right position, and this could be one of the reasons for the long intermediate steps.

4.7 Conclusion

The overall purpose of this chapter is to verify that the drone can fly automatically to predefined locations, how to implement this on a DJI Mavic 2 Enterprise, and test the capabilities of the drone when flying automatically. DJI offers several SDKs, and the choice falls on the iOS SDK, due to the system requiring a user interface. The features of the Onboard SDK is also available by communication through the Mobile SDK. Using features from the Onboard SDK falls outside the scope of this thesis, but might come in handy in future work.

The implementation uses missions. This allows the developer to define actions for the drone to perform, such as taking off, flying to a location, and landing. Using these actions, one can define flight paths for the aircraft, and the aircraft will fly to them automatically.

The implementation was tested to chart the capabilities of the aircraft when flying automatically. The tests focus on basic flying, such as taking off, landing, moving to locations, and changing altitude. The results of the tests show that a big factor for the time the aircraft uses is the intermediate steps between actions. Limiting the number of actions will reduce the overall completion time significantly. Due to the high cost of each GoTo Action, Waypoint Missions are assumed to be better suited. Further development will discuss and investigate the use of Waypoint Missions.

Chapter 5

The Application

5.1 Introduction

This chapter describes the implementation of a mobile application for user interaction. The goal is for the user to be able to define an area on a map. This area represents the search area in the field where the drone should perform its search. The search area will then be sent to a dedicated server, which calculates a suited path for the drone. The application will compose a mission based on the path in the response and transfer this to the drone, which will then execute the mission.

The first sections of this chapter will give a quick introduction on how users can interact with a map and how a map can and should be implemented into an iOS application. Based on the current problem description, we compare map frameworks and find the one that is best suited. Next, we describe the implementation of the map in the application before we verify our design through usability tests.

5.2 Design Considerations

A map can be used in many ways. The map's design must respect and satisfy how a user is supposed to use it. A map can show roads and railroad tracks, or it can show contours and hiking tracks. Which map representation to choose depends on the user's goal. If the map is interactive, possible controls should be visible for the user as well. This section describes design considerations when implementing a map view into an application.

5.2.1 Guidelines from Apple

Apple has implemented a number of *Human Interface Guidelines* [26] for Apple developers. These guidelines are specially designed for the development of iOS applications and ensure a more persistent user experience across different applications. The guidelines include a section [27] about how maps should be used in an application and how users should be able to interact with maps. The most important pinpoints are described below.

Keep the Map Interactive

Most maps found in mobile applications and websites are interactive. A map is interactive if a user can interact with the map, e.g., move the map, zoom in and out, or get directions. In a mobile application, one usually takes advantage of what is called *gestures*. Gestures are actions a user can perform on the screen using its fingers. Modern devices often support both multi-finger touch and force-touch. This means that the number of fingers and the force of the touch respectively can be used to perform different actions. In other words, a soft press can trigger a different reaction than a hard press.

Consistency With the Rest of the Application

Map service providers, like Google Maps and Apple Maps, have their own layouts and themes. The respective provider sets colours, buttons, and typography, and the map layout may not be consistent with the rest of the application. Apple states in their guidelines that developers should implement the application's theme into the map's layout. This ensures consistency and enjoyable user experience. One example is the colour of annotations pins placed on the map by the user. If the pins represent some data displayed to the user outside the map view, the pin's icon and colour should be equally styled.

Keep Map Controls Visible

It is common to add custom controls to a map. Custom controls are used to interact with the map or other services in the application. To avoid custom controls to blend in with the map, one should choose colours wisely. Controls that look similar to objects on the map tend to be harder to see by the user. This can cause frustration and poor user experience.

5.2.2 Axis Maps' Cartography Guide

Axis Maps is a company formed in 2006 in the USA. Their goal is to provide custom maps that conform to the user's requirements. They focus on design and intuitive user interfaces, rather than algorithms. On their homepage, we find a *cartography guide* [28] that is further described below.

Medium

When deciding on map design, the medium on which the map will be displayed matters. On a higher level, a decision has to be made whether the map will be displayed on paper or on a screen. A map on a paper is not interactive and can not be customized by the user in the field. All the information needs to be displayed on the paper, or it will lose its usability for the user. A screen can be either an ordinary display screen or a touch screen. Touch screens are usually smaller, but can often support a more significant number of gestures, compared to a display screen. If the map should support several different media, maybe multiple designs have to be made.

Audience and Purpose

What the users are using the map for, and *who* the users are, play an important role when it comes to map design. Professional or enterprise users usually require a more complex map design than ordinary users. For an ordinary user, a plain map with little interaction might be sufficient. Professional users, on the other hand, might have different requirements, based on their work and goals. The design can vary, based on profession and field of work.

Map-worthiness

Axismaps states that *just because data can be mapped, doesn't mean it should be mapped*. Maybe there are more suited ways of representing the data than on a map, e.g. in a table. You do not need a map if you know the area and have an address.

Interactivity

Static maps represent a cartography state at a given point in time, for example, a printed map on a sheet of paper. Static maps can also be digital, often displayed as an image, but the level of interaction is still minimal, compared to what we call interactive maps. Interactive maps are usually web-based and can be displayed on different digital media. They can hold more information than a static map, and give more control to the user. Careful consideration has to be given to the design, the flow of user experience, and the overall user interface.

5.2.3 A Map to Fulfil the User's Needs

Most people with a smartphone have used a map application to find a place, give road directions, or locate oneself or others. A user always has objectives or goals he or she wants to achieve when using an application. If the application does not fulfil the user's needs, a more suited application will often replace it.

Just like an application is designed for some purpose, a map has to be designed to fulfil the user's needs. For example, if a person wants to use a map for road directions, it would not make sense to show railroads and hiking trails. A map with too much information, or wrong information, will be frustrating for anyone using the map.

5.3 Framework

Digital maps have been on the market for a while now. Since this project concentrates on developing a mobile application for iOS devices, a requirement is that the map service can be implemented directly into an iOS application.

Two popular map services that fulfil this requirement are Google Maps [29] by Google and Apple Maps [30] by Apple. As seen in Figure 5.1, Apple Maps and Google maps look similar when it comes to the theme and the look of the map layer. On the other hand, we note that Google Maps offer a more rich layout, with more annotations, buttons, and controls. Other map service providers exist, like MapQuest [31], Open Street Map [32], and Maps.me [33], but to keep the implementation effort to a minimum, we focus on Google Maps and Apple Maps. Both of these services provide support for iOS and documentation on how to implement them.

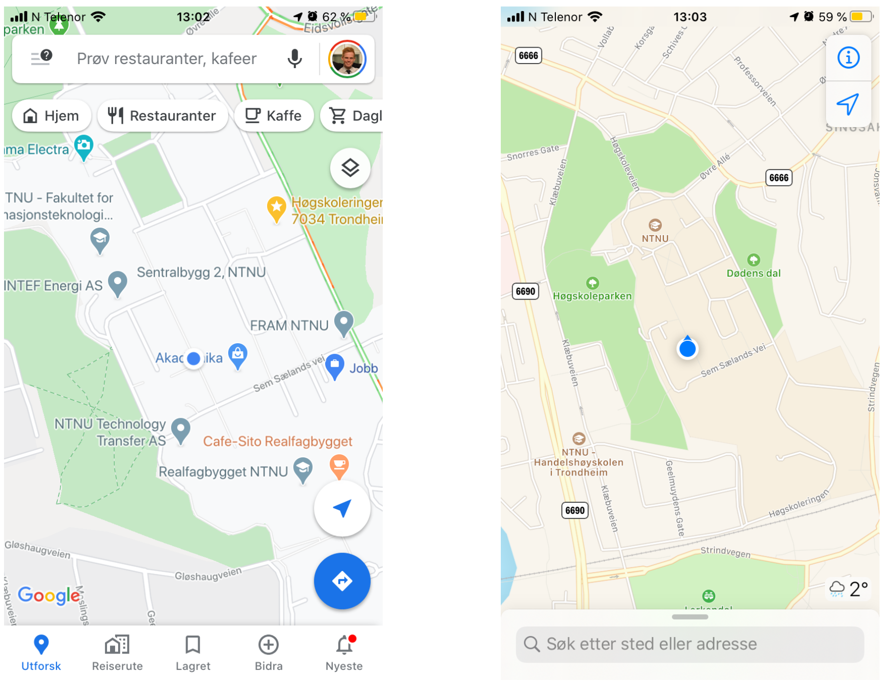


Figure 5.1: Graphical comparison between Google Maps (left) and Apple Maps (right).

5.3.1 Comparison

To make a decision on what map provider is best suited for the application, we need some evaluation criteria.

First, we require the service to be implementable into the iOS environment, either through a framework or a software development kit (SDK). The former is often easier to implement since it does not rely on third-party libraries. The latter

Criteria	**Google Maps**	**Apple Maps**
Can be implemented into iOS application	✓(Google Maps SDK for iOS)	✓(Apple MapKit Framework)
Map objects	✓	✓
Gestures	✓	✓
Ease of implementation	Moderate	Easy
Quality of documentation	Good	Good/moderate
Free?	✓[1]	✓
Storage/processing overhead	None/small	Moderate
Integration into iOS environment	Good/Moderate (separate SDK)	Good (Apple framework)

Table 5.1: Comparison between features of Google Maps and Apple Maps.

requires the developer to install a library into the application environment but results in more options.

Secondly, the map should support annotations (pins, markers), to be used as demarcations around the search area. It is a plus if the map can visualize the search area for the user in a good way. Either as a coloured overlay, a polygon with corners at the annotations, or both. Other criteria are ease of implementation, rich set of gestures, intuitive action handling, the ability to customize the user interface and add custom controls. The full comparison is summarized in Table 5.1.

Both of them have pretty much the same set of features and functionality. Some of Apple's documentation is outdated, but forums and development communities provide answers to common questions. Google Maps has a starter plan, which is free. Expenses will grow if you plan to use more advanced features and not be restricted by a request rate limit. Apple is free forever and has no rate limit. Since the application is implemented in an iOS environment, the implementation effort is smaller for Apple Maps than Google Maps.

Since Apple's MapKit is easier to implement and integrates better with the rest of the app, it is probably the most suited framework when it comes to implementation effort. On the other hand, after pilot testing the two frameworks, we find that many of the required features are easier to implement with Google Maps. MapKit is also more restricted when it comes to gestures, especially drag-and-drop versus tapping. Based on this, we found that Google Maps are more suited for the task and is more likely to fulfil the requirements of the mobile application.

[1]Up to $200 worth of usage. See https://cloud.google.com/maps-platform/pricing

5.4 Implementation

In this section, we will walk through the implementation of the map view step-by-step. To install Google Maps into our application sandbox, we use *cocoapods*. Cocoapods [34] is a *dependency* or *package manager* for Swift and Objective-C projects, much similar to *pip* for Python and *npm* for Node JS. A *package manager* is useful when working with open-source code and third party libraries because the install process is a lot easier than doing it manually.

5.4.1 Privacy

To use the GPS location of the device, the user has to allow this in the application. This is handled by iOS under the hood. We need to provide a text that describes what the location is used for. We can choose to ask for location *All the time* or *When in use*. For simplicity, we provide a description for both of them. Global settings like this are configured in a special file, called *info.plist* (information property list), which also holds other important information about the application and the overall environment.

5.4.2 Architecture

Swift uses a *model–view–controller* (MVC) architecture. The model represents the data that is stored in the application. This can be a database, a simple object, or a file. In great extent, we do not consider the model yet. The Google Maps SDK handles all data displayed on the map. Later on, a backend server will act as our model to the application (see Chapter 6).

A Swift controller is responsible for displaying a view on the mobile screen. In the proposed solution, the view is the actual map. The controller acts as a consumer of gestures and actions performed on the view, by the user. The controller will also be responsible for performing requests to the backend server and accept responses with data to be displayed.

5.4.3 Application Flow

To make changes to the map, we define a Swift class called `MapController` in which holds a reference to the map view. The map controller handles gestures, adds objects to the map, or changes the layout. When the user taps at a location on the screen, the controller finds the corresponding coordinate location on the map, create a marker and add the marker to the map view.

The markers make up the search area polygon. Every element in the polygon is connected consecutively to each other, and the last element is connected to the first to form a closed graph. In Figure 5.2, we see an example of a polygon, created by doing six press-and-holds on the screen. This polygon will later be used on the backend server to calculate the path.

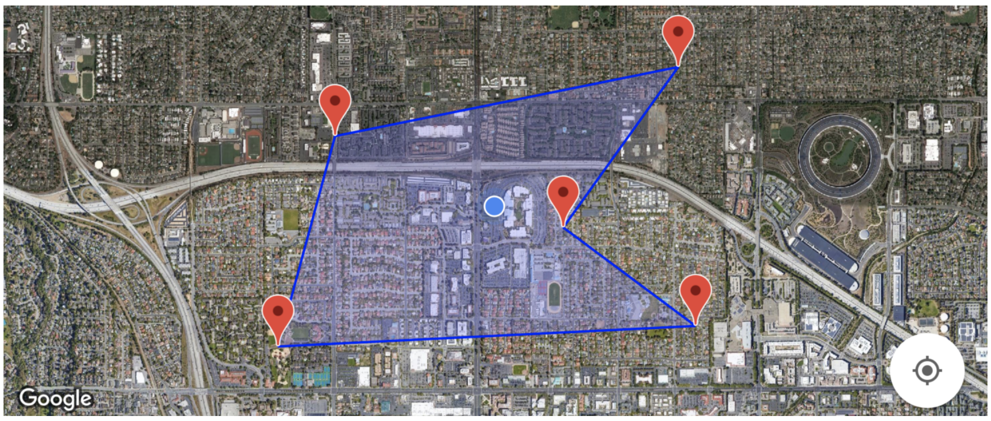


Figure 5.2: Making a polygon shape on the map. Red pins represents the outer bounds of the polygon. Blue marker represents the user's location.

5.5 Test

We implement a test to verify that the user interface works as intended. The overall goal for a person using the application is to draw a polygon defining the search area. And this is exactly what we want to test. Even though the map is just a small part of the application, we perform an isolated usability test to validate and detect possible faults with the design. By isolation the functionality in this manner, the user is not biased by other features. This gives a better foundation for taking further actions on the design of this particular feature. Usability tests are performed by non-biased users, who have never seen the application before. This simulates how the application would work in real life – when a user installs it and use it for the first time.

5.5.1 Preparations

It is essential to be well known with both the product itself and the test [35]. We execute pilot tests in advance to discover major errors in the application and the test itself. This ensures that the application does not crash during the test, and the user is able to complete the tasks – i.e. reach its goal.

We define test objectives based on the overall goal of the application. These objectives make up the tasks the test subject will try to perform during the actual test. For each objective, we define parameters – both qualitative and quantitative – from which we evaluate the final test result. We define a numeric score for each parameter whenever this is possible, which will give a good measure for comparison. When using quantitative metrics, comparing the tests becomes easier, since we can derive values like average, mean and standard deviation if necessary. The polygon accuracy is given by Equation 5.1. The denominator – 7 –comes from the number of points in the reference figure.

$$\frac{\text{Deviation from number of points}}{7} \cdot \frac{\text{Angles matching reference figure}}{7} \tag{5.1}$$

5.5.2 Objectives

Table 5.2 lists the test objectives. Each objective has one or more parameters as a measure on the success of the objective.

Test objective	**Objective parameters**
O1: The user should be able to define a polygon on the map, based on a predefined area given to the user.	• Time to complete task • Mistakes • Polygon shape accuracy
O2: The user should be able to change the polygon, according to a new predefined area given to the user.	• Time to complete task • Mistakes • Polygon shape accuracy (similarity to predefined area)
O3: The user should be able to delete vertices/markers from the polygon. Remove all markers	• Time to complete task • Mistakes

Table 5.2: Test objectives for the usability test.

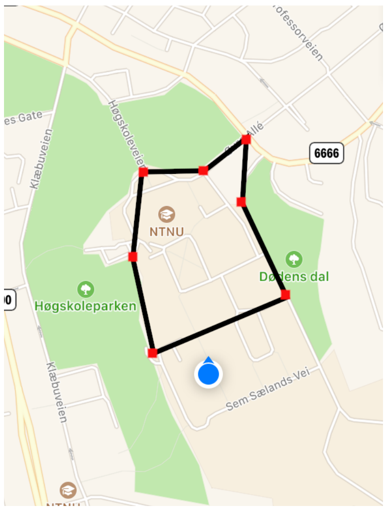
5.5.3 Test Implementation

To simulate a farmer who is familiar with the area and know exactly where the search area should be defined, we draw a map in advance (see Figure 5.3a). The test subject's objective is then to replicate this drawing as good as possible. For Q2, the aim is to change the shape accordingly to fit the shape in Figure 5.3b. For the last objective, the test subject is asked to remove one of the vertices from the polygon, i.e. delete a marker.

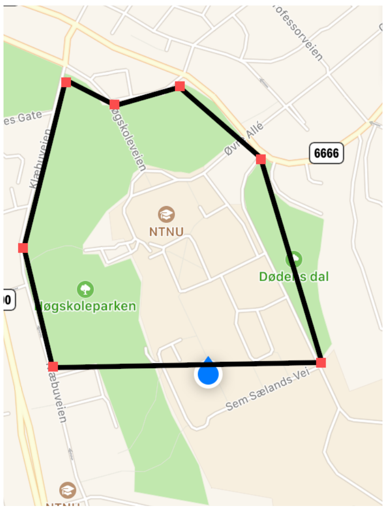
The application does not give any textual description on how to act, so the user will have to figure this out by himself. The critical thing to investigate is if the test subjects are able to use prior knowledge about maps in mobile applications to figure out how to complete the task. If the application design is consistent with the user's intuition, he or she should be able to complete the tasks.

5.5.4 Test Results

The result of the test is shown in table 5.3. Raw test results can be found in Appendix A. All tasks where completed by all participants. There were a total of five participants. Ranging between the ages of 23-56 with various degrees of tech skills.



(a) Shape to be drawn by test subject in O1.



(b) Shape to be drawn by test subject in O2 after editing.

Figure 5.3: Maps the test subject are to draw in objectives 1 and 2 in the usability test.

Test objective	Averaged results of all tests
O1: The user should be able to define a polygon on the map, based on a predefined area given to the user.	• Time to complete task: 83.05 seconds • Mistakes: 3.4 • Polygon shape accuracy: 0.83 • Completed: 5/5
O2: The user should be able to change the polygon, according to a new predefined area given to the user.	• Time to complete task: 82.18 seconds • Mistakes: 2.6 • Polygon shape accuracy (similarity to predefined area): 0.73 • Completed: 5/5
O3: The user should be able to delete vertices/markers from the polygon. Remove all markers	• Time to complete task: 72.34 seconds • Mistakes: 1.6 • Completed: 5/5

Table 5.3: Averaged results from all usability tests.

5.6 Discussion

Without any knowledge about either of the two types of map – Apple Maps and Google Maps – the solution was to implement both of them. This did take some time, but afterwards, it was clear which one was best suited to fulfil the application requirements.

The test conducted had five participants. According to Nielsen [36], one does not need to perform more than five usability tests. The reason is that one gains minimal amounts of new information when extending to six or more test subjects.

The main takeaways from the test are two things. One, the application works and all participants were able to complete all tasks. Two, some of the actions are not as intuitive as assumed. All participants said that they would be able to do the tasks without mistakes if they did them again, this means that while some struggled with the task the first time, they learned what was needed to complete the task. As such, while not everything is intuitive, it still provides value after it has been learned. Multiple participants commented on a few specific things in the application. Two participants said that *press and hold* to place points were not intuitive and should have been a simple *press*. Another issue commented on by all participants was that there was no indication if a marker was selected when deleting. Although all participants were able to complete Q3, this should be improved.

Chapter 6

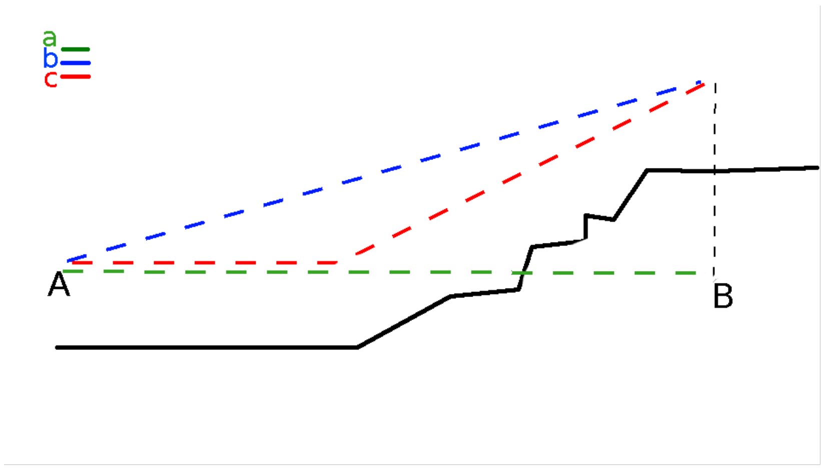
Altitude

The altitude changes a lot in many parts of Norway. About two-thirds of the land area of Norway is mountainous. A lot of the animals grazing in Norway graze in uneven and drastically changing terrain. This project aims at creating a fully automatic coverage of an area using a drone. This means that along a path, the altitude will most likely differ significantly. Therefore, two problems need to be addressed in regards to the terrain of an area.

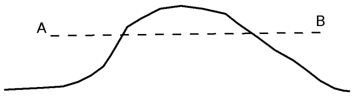
1. Obstacles in the flight path due to differences in altitude between points. See figure 6.1b for an illustration highlighting this problem.
2. Field of view of the camera due to the difference in altitude between points. See figure 6.1c for an illustration highlighting this problem. Point A shows the ideal height. In point B, details are lost in the picture due to the height, while point C inefficiently covers a relatively small area.

With regards to obstacles, this mostly involves an increase in the terrain leading to a hill or mountain getting in the way of the drone's flight path. When it comes to the field of view of the camera, pictures should be taken at an optimal altitude to ensure a balance between the size of the covered area and the images' resolution. These two problems mean that the final path needs to consider the altitude of each point in the path, as well as points between.

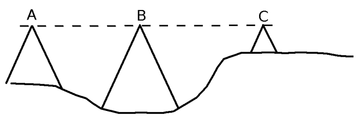
As a path is defined as a series of points, the terrain may differ significantly along a line between two points. A significant increase in altitude between two far points can easily cause the drone to crash to the ground. We add intermediate points at a defined interval along a path to ensure a relatively constant altitude above ground. Figure 6.1a shows possible scenarios when the path passes a hill. The green line disregards altitude completely. The blue line goes directly from point A to point B with the proper altitude increase over point B. The red line uses an intermediate point to keep a fairly constant altitude above the terrain.



(a) Shows different paths from a point A to point B. a, b, and c are different paths the drone may take.



(b) Shows the issue of disregarding terrain in regards to hitting objects.



(c) Shows the issue in differing altitudes with regards to the area covered by the camera.

Figure 6.1: Illustrations regarding difference in altitude of the terrain.

6.1 Height Data in Norway

As altitude needs to be considered when calculating the final path, the system should have access to height data of the area, and use the data to add altitude parameters to the path. This section evaluates different height data available in Norway and how they are added to the system. There are several height maps available in Norway. These depict the terrain in meters above sea level. They differ in accuracy and method of collecting, as well as some being a collection of multiple other data. All data is publicly available at *hoydedata.no* [37]. The data can be categorized into two: *Digital Terrain Model (DTM* and *Digital Surface Model (DOM)*.

6.1.1 DTM

DTM is collected height data that depicts the natural surface of Norway, and, hence, it does not consider obstacles such as trees or human-made structures [38]. *NN2000* is the current standard for altitude measurements in Norway [39]. It started in 2011 and was finished in 2018. It replaced NN1954, due to changes in the height of Norway as a whole. Norway rises slightly. NN2000, although started in 2011, uses heights data from the year 2000. Later, rising land is modelled and applied to these data. This means NN2000 should be accurate for the foreseeable future. DTM currently covers the entirety of mainland Norway.

6.1.2 DOM

DOM is a height model that also considers obstacles in the terrain. It includes trees and human-made structures and is usually made from *Local Point Clouds*. Local point clouds are part of the *Nation Detailed Height Model (NDH)* [40]. Point clouds are models of points in a 3D-space. When used as height models point clouds are detailed height measurements of the terrain in an area. NDH, as a project, aims to create a highly detailed model of Norway covering 253 000 km^2. It uses measurements from planes or helicopters with mounted laser scanners. The goal is an accuracy of 1m^2. Local point clouds are available, but due to the size of the data, it has to be downloaded as separate files for each local region. It is possible to get the height measurement of a single point, given that there exists a measurement of that point. DOM does not cover the entirety of Norway.

6.1.3 Conclusion

While the system could use both data sets, DTM was chosen due to currently being complete for the entirety of Norway. The system in this project should be usable in most of Norway, especially in areas where animals graze. DOM coverage in these areas is still lacking. As the focus in this project lies on grazing animals, that most of the time is at the terrain level and not on top of human-made structures or trees, there is none camera consideration that justifies using DOM. However, it may be used to check that the path of the drone does not cross any obstacles.

6.2 Coordinates

An important aspect to consider is the coordinates of where the aircraft is, and where it should go. The points at which the aircraft should fly needs some considerations, specifically how they relate to each other. Earth is a globe, and thus there are some challenges in regards to the final path. The drone uses longitude and latitude, but this does not translate well over large distances. The size of the areas where animals usually graze is not large enough to make a significant difference. On the other hand, if the proposed solution is to be adapted for other purposes, such as mapping large areas of ground, longitude and latitude problems arise.

6.2.1 Longitude and Latitude

Longitude and latitude together define coordinates on a globe [41]. Longitude is the west-east position, while latitude defines the north-south position. Longitude is given as degrees between 0°and 180°in either west or east. Latitude is also given as degrees, but between 0°and 90°north or south. The latitude is 0°at the equator, 90°North at the north pole, and 90°South at the south pole. An example coordinate is Trondheim lying at 3°25' 49.8" North, 10°23' 42.2" East. When doing calculations, it is common to use decimal numbers, where south of 0°longitude gives a negative number and north gives a positive number. Similarly, for latitude, where east is positive, and west is negative. As such, any position on earth can be defined with two numbers.

6.2.2 UTM

UTM is a projection of the earth surface [42]. It translates the curved earth into a flat two-dimensional surface using Gauss-Krügers projection. By sectioning the earth surface into smaller areas and making a Gauss-Krügers projection on each assures an accuracy of less than 40 cm per 1 km. There are 60 zones divided based on the latitude. These 60 zones are further divided into bands parallel to the equator. Figure 6.2 shows the different zones covering Europe. Note that zone 32V extends to the west to cover the whole of southern Norway. The algorithm described in Chapter 7 uses a two-dimensional coordinate system. Converting longitude/latitude coordinates to UTM allows the algorithm to do calculations without the need to regard the curvature of the earth.

6.2.3 Great Circle

Great Circle in regards to navigation is the practice of using the shortest line between two given points [43]. The line follows the globe around its curvature and ends up at the same point. Any two points on the earth's surface have a unique great circle except points being directly on opposite sides of the globe; they will have an infinite amount of great circles. Using Great Circle navigation, the system ensures that intermediate points are along the shortest path possible between points, while still following the curvature of the earth. Figure 6.3 illustrates a great circle derived from two points P and Q.

6.2.4 Conclusion

The final system requires the use of both UTM and longitude/latitude. The drone and the application use longitude and latitude coordinates to defined the search area. UTM is used to convert this area into a two-dimensional shape. This shape is later accepted by the path planning algorithm, and the final path is computed. The last step is to convert it back to longitude and latitude before the final path

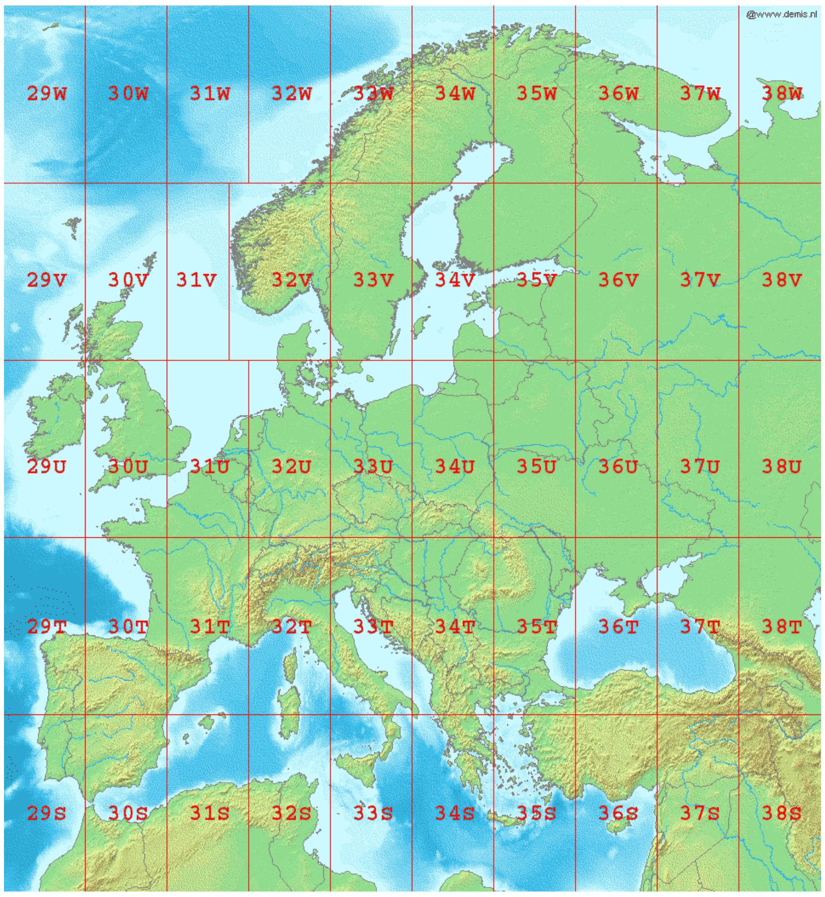


Figure 6.2: Illustration of UTM zones. "*LA2-Europe-UTM-zones.png*". *Image is public domain. No changes made.* *`https://upload.wikimedia.org/wikipedia/commons/9/9e/LA2-Europe-UTM-zones.png`.*

is sent to the drone. Great Circle is used to find intermediate points in the path, ensuring that they are on a straight line between two points.

6.3 Camera

A drone's altitude relative to the terrain of which the aircraft should fly depends on the purpose of the flight. If the purpose is spraying pesticides on a field, the height of the aircraft will be lower than if the purpose is to create a birds-eye view map of the area. As this project aims to create a solution for finding grazing animals, the height at which the aircraft should fly above the terrain also depends on the camera mounted on the drone. Aspects of the camera need to be taken into consideration, such as field of view (FOV), sensor, aperture, and focal length. The goal of this section is to find the properties of a camera and use this to find how large area a picture taken at a specific height will cover.

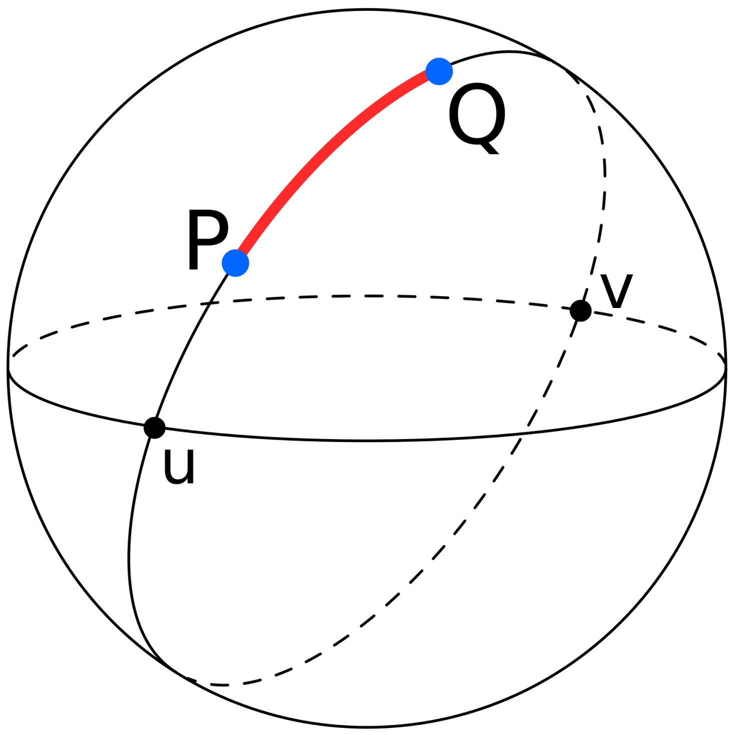


Figure 6.3: Illustration of a great circle line between points P and Q. *"Illustration of great-circle distance" by CheCheDaWaff. Licence: CC BY-SA [44]. No changes made.* *`https://upload.wikimedia.org/wikipedia/commons/c/cb/Illustration_of_great-circle_distance.svg`.*

6.3.1 Aspects of Cameras

As previously stated, several aspects of cameras need to be taken into consideration when determining the altitude of the aircraft. These aspects determine how well the focus is, the details of the image, how large area on the ground is present in the image, and more. Each relevant aspect is explained in this section, as well as how it affects the altitude the aircraft should travel.

Focus

Focus is directly correlated to the depth of the field. At the *plane of focus* an object is sharp and detailed [45]. The plane of focus is the plane perpendicular to the camera lens where the objects are at optimal sharpness. This plane is at a specific distance from the camera. Objects closer to and further away from the plane of focus get exponentially more blurry. Most lenses can adjust the focus so that the desired object is in the range where the image is still sharp. Modern cameras have auto-focus capabilities.

The drone has auto-focus, and as such, this does not need to be considered.

Sensor

The sensor of a digital camera is what actually captures the image [46]. It is a chip consisting of millions of light-sensitive elements called pixels. The light-sensitive elements translate the incoming light into digital values. Sensors differ in several aspects, but the two most relevant to this project, are sensor size and amount of pixels.

The sensor size determines the actual size of the sensor. The standard size is 36x24mm. With the same lens, a smaller sensor will capture a smaller subset of the image compared to a larger sensor. This is because the lens will focus the light around the smaller sensor, where a larger sensor will cover more of the light. Therefore the lens used on a camera will have different properties depending on the sensor size. A smaller sensor will have a smaller field of view with the same lens than a larger sensor.

The amount of pixels determines the resolution of the resulting image. As each pixel is converted to a digital value, more pixels results in images with a higher level of detail. The number of pixels is usually denoted as megapixels, meaning current cameras has a magnitude of millions of pixels.

The sensor is relevant to this project because it determines how much information the image holds, as well as the field of view.

Field and Angle of View

The *field of view (FOV)* is the section of the world observed from the camera [47]. *Angle of view(AOV)* is the angle determining the size of the field of view. A small angle of view, creates a narrow "cone" of what will be seen, while a large angle of view means more of the scene is covered in the image. Figure 6.4 shows how the angle and distance determine the size of what is covered in the scene and the difference in size depending on the distance. The angle of view depends on the sensor size and the focal length of the lens on the camera.

In photography, angle of view (AOV) and field of view (FOV) are often interchangeable. In this paper, for the sake of clarity, AOV is the angle, and FOV is the resulting coverage of the scene.

As images are mostly rectangular, the angle of view is different horizontally and vertically. As such, it is common to describe the different values of the angle of view as *horizontal angle of view* (HAOV), *vertical angle of view* (VAOV), and *diagonal angle of view*. The same applies for the different values of the field of view, respectively: *horizontal field of view* (HFOV) and *vertical field of view* (VFOV), and *diagonal angle of view* (DFOV).

To calculate the angle of view, the focal length of the lens and the sensor size is needed. Equation 6.1 calculates the angle of view where s is the sensor size, f is the focal length, and θ_{AOV} is the angle of view.

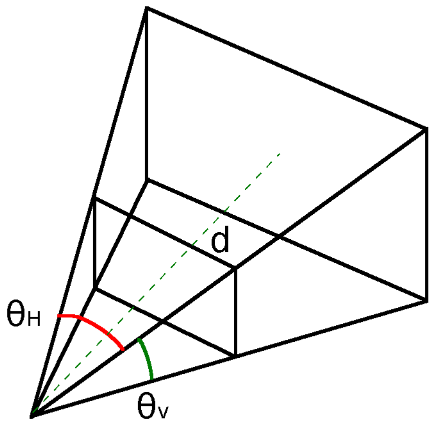


Figure 6.4: How the angle of view and distance determine the area covered by an image. Θ_V and Θ_H denotes the angle of view in vertical and horizontal direction respectively.

$$\theta_{AOV} = 2\arctan(\frac{s}{2f}) \tag{6.1}$$

Given the angle of view and distance, the coverage of the resulting image can be calculated. The formula for finding the field of view given the angle of view is shown in Equation 6.2, where d is the distance to the scene, and FOV is the resulting field of view.

$$FOV = 2\tan\frac{\theta_{AOV}}{2} \cdot d \tag{6.2}$$

It is also relevant to calculate the angle of view of one direction, given the angle of view of the other direction and the ratio of the sensor. Equation 6.3 and 6.4 show how to convert between HAOV and VAOV, where θ_{HAOV} is the horizontal angle of view, θ_{VAOV} is the vertical angle of view, and R is the ratio ($\frac{H}{V}$) of the sensor.

$$\theta_{VAOV} = 2\arctan\frac{\tan\frac{\theta_{HAOV}}{2}}{R} \tag{6.3}$$

$$\theta_{HAOV} = 2arctan(tan\frac{\theta_{VAOV}}{2} \cdot R) \tag{6.4}$$

To find the diagonal angle of view, we first need to find the length of the diagonal on the sensor. The relationship between the diagonal and the length of each side is given by Pythagorean Theorem and shown in Equation 6.5.

$$h^2 + v^2 = d^2 \tag{6.5}$$

6.3.2 Specifications

The camera specifications of the DJI Mavic 2 Enterprise Dual is shown in table 6.1 and are taken from the official manual [19]. The specifications are both for the thermal and the visual camera. Some things to consider when looking at the specifications for the thermal camera are:

- The sensor size and lens are not given. However, the sensor resolution is given.
- Only the HAOV is given, regarding the angle of view.

We can use the sensor resolution and the horizontal angle of view to find out the ratio. From this, we can also calculate the vertical angle of view of the thermal camera.

We have more information on the visual camera. Some things to consider when looking at the specifications for the visual camera are:

- 1/2.3" CMOS has a ratio of 4:3 [48].
- The effective pixels of 12 megapixels means there are 4000×3000 pixels.
- The direction of the AOV is not given.
- "35 mm format equivalent: 24 mm" means that the lens has the same focal length as a 24 mm lens on a 36x24 mm sensor.

Since the angle of view does not have a direction, it will need to be calculated using the focal length and sensor size. The lens is not able to zoom. The specifications are sufficient to calculate the area covered, given a specified height.

Thermal camera	
Sensor	Uncooled Vox Microbolometer
Lens	HAOV: 57° Aperture: f/1.1
Sensor resolution	160x120
Visual camera	
Sensor	1/2.3" CMOS Effective pixels: 12M
Lens	AOV: approx. 85° 35 mm format equivalent:24 mm Aperture: f/2.8 Focus: 0.5 m to ∞

Table 6.1: DJI Mavic 2 Enterprise Dual camera specifications.

6.3.3 Calculations

For this project, the coverage of an image from a given height is needed. The angle of view in both the horizontal and vertical directions can be used, with the height, to calculate the coverage. Using the specifications given in 6.1 we can use the formulas from 6.1-6.5 to find all relevant values. We need the AOV both horizontally and vertically for both the thermal and visual cameras.

Starting with the thermal camera, we already know the horizontal angle of view from table 6.1 as 57°. The sensor resolution of 160x120 indicates that the ratio is 4/3. Using formula 6.2, we get a vertical angle of view of $\tilde{4}$4.31°.

Finding the horizontal and vertical angle of view of the visual camera is more difficult, and some assumptions have to be made. The FOV from the manual does not give a direction. Using formula 6.1 and a sensor of 36mmx24mm, results in a diagonal of 84.11°. This is close to the approximation of 85°given in the manual, and it can be assumed that this is the diagonal angle of view.

The problem with this value is that it is a result of having a sensor with an aspect ratio of 3:2. The actual sensor has a ratio of 4:3. The most likely reason for this discrepancy is that all the information on the lens follows the 35 mm equivalent, and this has a sensor ratio of 3:2.

To convert the angle of view given a 3:2 ratio to the horizontal and vertical angle of view given a 4:3 format, we can change the sensor size of the 35mm equivalent to 36mmx27mm, which has a ratio of 4:3, and use this with formula 6.1. Using a f of 24mm and s as the sensor size for a given direction we have the following:

- Finding the vertical field of view:
  $2\arctan\frac{27mm}{2*24mm} = 51.72°$

- Finding the horizontal field of view:
  $2\arctan\frac{36mm}{2*24mm} = 73.74°$

As we now have the angle of views of both the thermal and visual camera, we can calculate the field of view in both directions given a height.

6.3.4 Implementation

As the different angle of views of both the thermal and visual camera is known, they can be used to find the field of view. Code listing 6.1 shows the implementation for finding the field of view. Equation 6.2 was translated to Javascript code and can be found on line 6 in code listing 6.1. Using function `getFov`, an object with all field of views is returned for a provided height. Table 6.2 shows the different field of views for heights 10, 50, and 100 meters.

Height / **Camera**	10 m	50 m	100 m
Thermal	10.86x8.14 m	54.30x40.72 m	108.6x81.4 m
Visual	15.27x9.69 m	76.37x48.47 m	152.7x96.9 m

Table 6.2: Field of views for the thermal and visual camera at 10, 50, and 100 meters altitude.

```
const degreesToRadians = (degrees) = {
   return degrees * (Math.PI/180);
};

const fov = (angleOfView, height) => {
   return 2 * Math.tan(degreesToRadians(angleOfView) / 2) * height;
};

const aov = {
   // Thermal
   thermal: {
      horizontal: '57',
      vertical: '44.31'
   },
   // Visual
   visual: {
      horizontal: '74.74',
      vertical: '51.72'
   }
};

const getFov = (height) => {
   return {
      thermal: {
         horizontal: fov(aov.thermal.horizontal, height),
         vertical: fov(aov.thermal.vertical, height)
      },
      visual: {
         horizontal: fov(aov.visual.horizontal, height),
         vertical: fov(aov.visual.vertical, height)
      }
   };
}
```

Code listing 6.1: JavaScript implementation for finding the field of view of the drone, given a height.

6.3.5 Conclusion

By looking at the components of a camera, like the sensor and the lens, the properties of the camera can be found. Formulas for finding the angle and field of view were used with the camera specifications from the DJI Mavic 2 Enterprise. After finding the angle of view of both the thermal and visual camera, a simple Javascript program was implemented that can be used to find the field of view of the camera on the drone, given a particular height. Due to the depth of view causing objects on the outer part of the image to be blurry, there should be an overlap between the images taken to ensure that all objects on the ground are captured.

Chapter 7

Path Planning

In this chapter, we investigate how to find a flight path covering a specified search area. The goal is to produce a flight path that covers the area efficiently with high accuracy. The problem can be divided into two sub-problems. First, a flight path has to be defined in the x/y-plane. The path makes up the point-to-point coordinates for which the drone should fly during its search. How to find these coordinates is described in detail in this chapter. Second, we need to take into consideration the height of the drone relative to the ground. This issue was discussed in detail in Chapter 6.

7.1 Introduction

We find drones as a suited tool in missions within inhospitable environment. They can perform automatic, semi-automatic or remote-controlled flights in places in which can be dangerous for humans to operate. For the drone to be able to autonomously find its way in such places, *path planning* has to be implemented into the aircraft's flight controller. Path planning is the process of finding a valid sequence of actions to get from point A to point B in a given environment. The environment can be observable, partially observable or not observable. If the drone behaviour is affected by outer factors, such as wind and temperature, the drone's actions might also be stochastic. This means that the outcome of an action is not fully deterministic, but influenced by some kind of randomness.

A commercial drone usually has a sensor or camera. The area on the ground captured by any such device is called a footprint. The footprint corresponds to the field of view, as described in Chapter 6. If we stack footprints side by side, as shown in Figure 7.1, we eventually cover the complete search area. If the aircraft is equipped with a camera, a footprint corresponds to an image, taken from a birds-eye point of view. The ground area captured by the camera is equivalent to the footprint and is dependent on the drone's altitude, as described in Chapter 6.

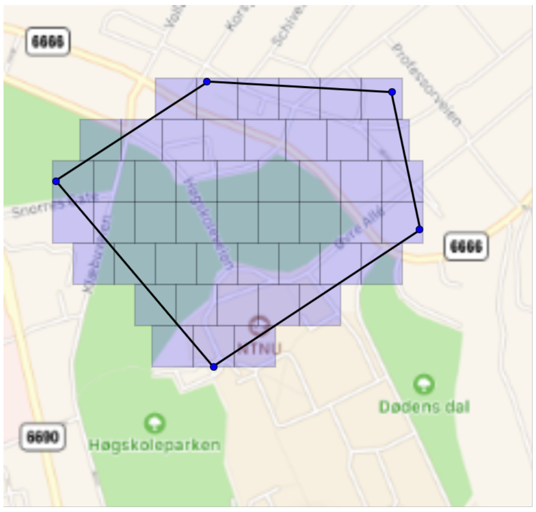


Figure 7.1: Footprints covering a search area. Black lines represents the search area. A light blue square represents a footprint.

7.2 Coverage Path Planning

Path planning can be sufficient in many cases. But when the targets are not defined as discrete locations, path planning algorithms come to short. This applies, for instance, if we want to investigate an area of ground instead of a single point. The coverage path planning (CPP) problem is defined as finding a path that completely covers an area of interest (AOI), based on some restrictions and a description of the environment. In the following, we will investigate how the area of interest is defined and what its restrictions are. Next, we will define an algorithm for finding a path that completely covers an area, solving the CCP problem.

7.2.1 Area of Interest

The AOI is the area where the drone performs its search, i.e. the search area. As defined in Equation 7.1 the AOI is represented by a sequence S of vertices, where each vertex v_i is defined as a pair of coordinates (v_i^x, v_i^y).

$$S = \{v_i = (v_i^x, v_i^y)\}_{i=0}^{N-1}, \tag{7.1}$$

where $N = |S|$ is the number of vertices. An edge e_i is the line between vertices v_i and v_{i+1}, where $0 \leq i \leq N-2$. Given vertex v_i and its neighbouring vertices v_{i-1} and v_{i+1}, we have that v_i's internal angle θ_i is given by Equation 7.2.

$$\theta_i = \arccos \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|}, \tag{7.2}$$

where $\vec{a}$ and $\vec{b}$ are vectors $[v_{i-1}^x - v_i^x, v_{i-1}^y - v_i^y]$ and $[v_{i+1}^x - v_i^x, v_{i+1}^y - v_i^y]$ respectively. Note the importance of calculating the vectors from the same point v_i.

To simplify our notation, given a vertex v_i, we denote the next vertex in the sequence as $v_{next(i)}$. If the sequence represents a closed polygon, we have that $next(i) = (i+1)(modN)$. It follows that $e_i = v_{next(i)} - v_i$. Similarly we have that $prev(i) = (i{-}1{+}N)(modN)$ for the previous vertex in the sequence, compensated for $i = 0$. Figure 7.2 shows an example of an AOI with vertices, edges and internal angles labeled accordingly.

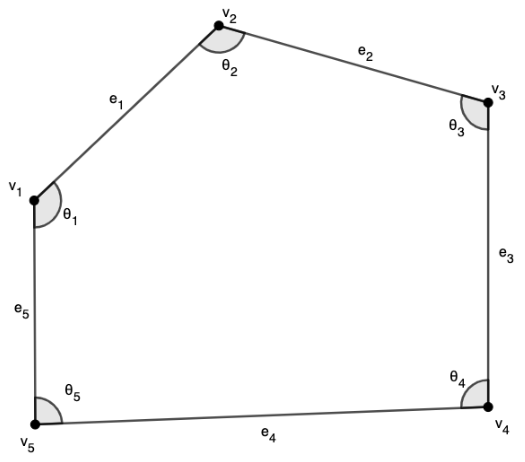


Figure 7.2: Example of AOI polygon with labels.

It is important to define how the AOI will look like. Different methods and concerns will apply, depending on the shape of the AOI. Andersen [49] has reviewed different flight paths for rectangular AOI, Coombes *et al.* [50] investigate different sweeping angles relative to the wind and extends the shape to convex polygons. More complex methods will apply when also considering concave methods. In the following, we will address different properties the AOI can have, by focusing on polygonal shapes.

Convex vs Concave

A polygon is either convex or concave. In practice, if a polygon is convex, any straight line drawn through the polygon will only intersect two edges. It also means that one can move in a straight line between any two points inside the convex polygon, without crossing any edges. Theoretic definitions are given below.

Definition 1: Convex Polygon
A polygon is convex iff all its interior angels are less than or equal to 180°. See Figure 7.4a.

Definition 2: Concave Polygon
A polygon is concave iff one or more interior angels are greater than 180°. See Figure 7.4b.

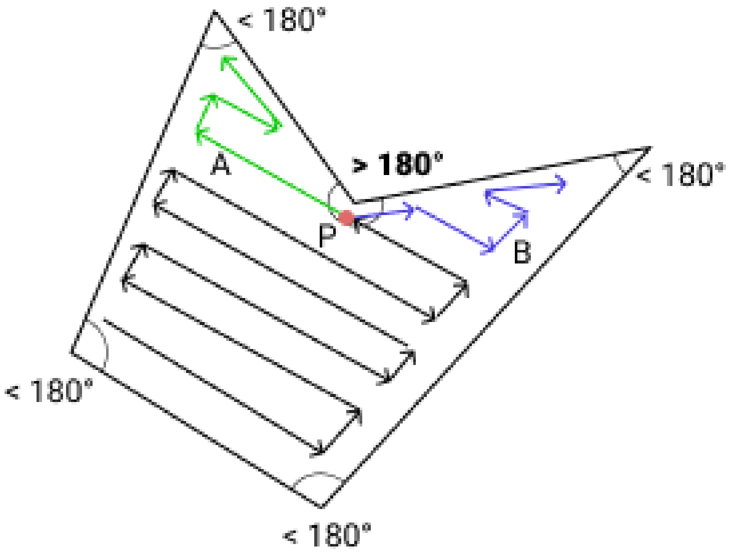
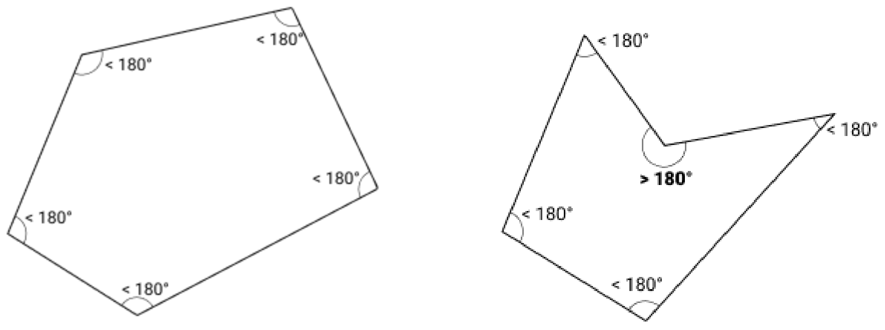


Figure 7.3: A concave polygon with a path planning problem.

As seen in Figure 7.3, a concave search area brings more complexity to the path planning problem. At point P, the planner has to make a decision on whether to choose path A or path B. Regardless of which is chosen, the aircraft will have to follow a path back to P to cover the other area. Hence, the drone will fly over an already covered area, which is undesirable.



(a) Convex polygon. **(b)** Concave polygon.

Figure 7.4: Comparison of convex and concave polygon.

Simple vs Complex

We classify polygons as simple or complex, based on the vertices' relative position. If vertices in a polygon are aligned so that edges intersect, we say that it is a complex polygon. Otherwise, it is simple.

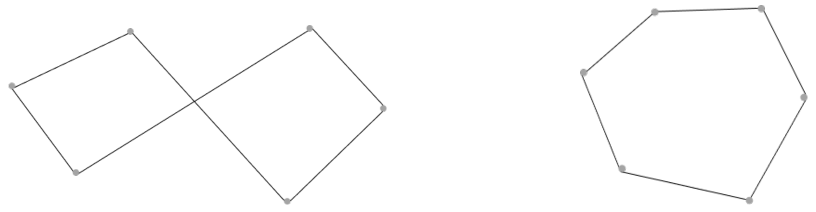

(a) Complex polygon. **(b)** Simple polygon.

Figure 7.5: Comparison of complex and simple polygon.

Definition 3:
A polygon is complex iff two or more of its edges intersect. Otherwise the polygon is simple. See Figure 7.5.

Complex polygons add further complexity to the path planning problem. To simplify, we require the search area polygon to be non-complex. Hence the application will have to make sure the user only submits simple polygons.

Regular vs Irregular

A polygon is *equiangular* if all its angles are equal in measure. A polygon is *equilateral* if all its sides are having the same length. A polygons regularity depends on these two properties and is defined below.

Definition 4: Regular Polygon
A polygon is regular iff it is both equiangular and equilateral; hence all angles and sides are equal respectively. If one or more of these conditions fail to hold, the polygon is irregular.

7.3 Evaluation Criteria

We define evaluation criteria to better target the desired outcome of the path planning algorithm. The criteria quantify the effectiveness and efficiency of the algorithm and the system as a whole. Furthermore, the system will be comparable to future work and similar systems.

The first thing we need to consider is the area coverage, that is, how much of the search area that is actually covered by the path. This value depends on properties like the drone's altitude, the pitch of the camera gimbal, and the camera's focal length. For comparison purposes, we express the area coverage as a percentage of the total AOI. High area coverage is desirable. On the other hand, we do not want to cover areas more than once. For very large AOIs, the drone might be forced to *go home* and recharge/replace its battery, but this is beyond the scope of this paper.

Another metric is the number of operations. The literature [51, 52] agrees that it is desirable to minimize the number of turns. This is because the flight time increases as the drone has to make more stops and accelerations during its flight.

Another useful metric is the average ground speed. This is calculated by dividing the total distance travelled by the total time spent executing the mission.

For the rest of this section, we will focus on finding a solution to the CPP problem that completely covers the AOI and minimized the number of turns. At the same time, we aim to minimize overlapping paths, i.e. not cover areas more than once.

7.4 Background

As shown above, a concave polygon introduces more complexity to the CPP problem. We find it feasible to perform a convex decomposition to minimize the number of overlapping paths during a search. This is further described in section 7.5. In this section, we define terms that form a ground for the rest of this chapter.

7.4.1 Width

As done by Jiao *et al.* [52], in the following, we define the width of a polygon:

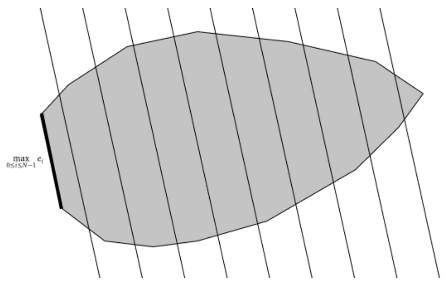
Definition 5: Width of polygon
The width (W) of a polygon (P) is the minimum span (D) between two parallel lines of support (l_1, l_2).

l_1 and l_2 are parallel lines that intersects a polygon P at opposite sides, so that P lies to the left (or right) of l_1 and to the right (or left) of l_2. It is given by intuition and mathematically proven by [52], that at least one of the support lines of a minimum span lies *on top of* an edge of P. That means that the support lines can be expressed by three vertices in the polygon. The first support line – l_1 – passes through the first two vertices. The second – l_2 – passes through the third vertex and lies parallel to l_1. To find the minimum span, we find the three vertices with the minimum width.

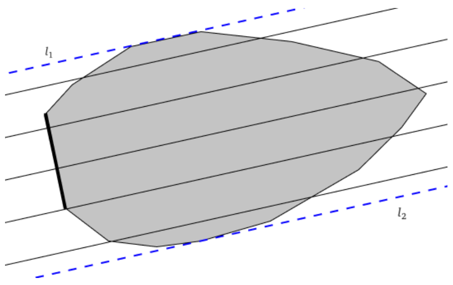
7.4.2 Sweeping Direction

Di Franco and Buttazzo proposes a solution using back-and-forth (BF) motions. The authors claim that high energy efficiency is achieved by setting the scanning direction to be parallel to the longest edge in the polygon. Jiao *et al.*, on the other hand, fly along the vertical direction of the width, that is, parallel to the lines of support.

Figure 7.6 demonstrates these two flight directions on a convex polygon. For simplicity, paths from one sweep to another are not included. Furthermore, if we count P sweeps across a polygon, we will have a minimum of $P \cdot 2 - 2$ turns to cover the entire area.



(a) Flight direction parallel to the longest edge.



(b) Flight direction parallel to the lines of support.

Figure 7.6: Comparison of the number of sweeps in different flight directions.

A flight direction parallel to the longest edge (see Figure 7.6a) results in more sweeps across the polygon than using the direction of the support lines (see Figure 7.6b). We count 9 sweeps and 16 turns for 7.6a, and 5 sweeps and 8 turns for 7.6b. Hence, when it comes to the number of turns, a flight direction parallel to the support lines is preferred over a direction parallel to the longest edge.

A flight direction parallel to the support lines is generally the best solution to minimize the number of turns. To prove this, we assume that the number of turns is proportional to the number of sweeps. In fact, it is also proportional to the width of the polygon. With this, we define the number of sweeps in Equation 7.3.

$$P = \lceil W/L_x \rceil, \tag{7.3}$$

where P is the number of sweeps, W is the width of the polygon and L_x is the width of the footprint. $\lceil x \rceil$ denotes the ceiling function, which maps a number x to the smallest integer greater than or equal to x. Since L_x is a constant and W is at its minimum, we have that P also has reached its minimum.

7.4.3 Convex Hull

To find convex properties of a concave polygon, it comes in handy to investigate the polygon's convex hull. In the following, we define the convex hull of a polygon.

Definition 6: Convex Hull
A convex hull is defined as the minimum convex polygon that completely circumscribes an object, as shown in Figure 7.7.

7.4.4 Greedy Approach

The basic idea behind a greedy approach is to always choose the solution that seems best at the moment [53]. To choose an optimal solution to a given problem, we need a comparison function. A comparison function takes two proposed

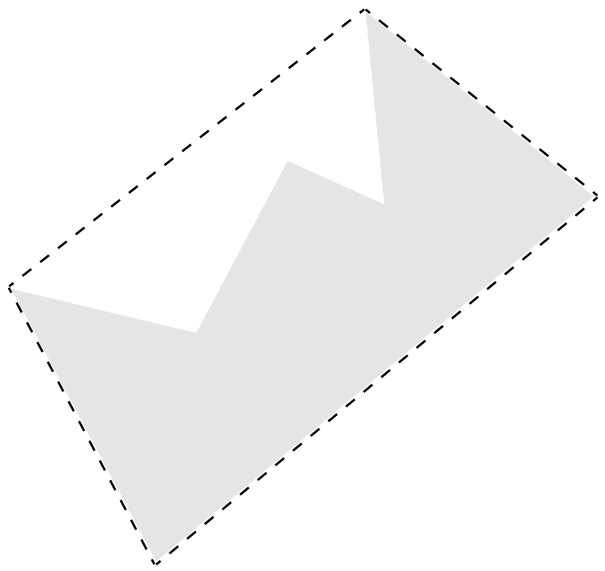


Figure 7.7: A convex hull (dashed line), circumscribing a concave polygon (gray).

solutions as input, compare them to each other, and greedy chooses and outputs the best solution.

7.5 Solution to CPP Problem

Convex decomposition is a popular method to reduce the CPP problem into smaller individual parts. It takes a (possibly concave) polygon as input, performs a convex decomposition (if necessary) on the polygon and outputs one or more *convex* polygons. The motivation for a convex decomposition is the assumption that it is easier to find a covering path for each of these convex polygons than for a concave polygon.

Algorithms for convex decomposition are proposed in several papers [54–56]. This section will focus on putting together a solution based on previous work. The goal is to develop an algorithm that fits the requirements and problem description in this project. At the same time, we aim to minimize the number of sweeps, and, hence, minimize the number of turns.

7.5.1 Related Work

Tor and Middleditch [54] defines an algorithm that traverses the edges of a polygon A. Each edge e_i is successively appended to either the current convex hull or one of the inner regions of the hull. The current hull H eventually grows into the final hull, where $H = Hull(A)$. An edge is categorized based on its position relative to the current hull, and the appropriate action is taken. The algorithm performs quadratic in the worst case and linear in the best case. It is expected that polygons, in this case, lie closer to the best case than the worst-case regions discussed by Tor and Middleditch.

Jiao *et al.* [52] presents a greedy algorithm that divides a concave polygon into convex sub-regions. The algorithm chooses the solution that minimizes the sum of sub-regions' widths. Some of these sub-regions are combined to produce a more efficiently flight path. As shown in Figure 7.8, on the left-hand side, the path consists of one more edge than the combined version on the right. For performance purposes, combinations like this might be beneficial to minimize the number of sweeps, and hence, the number of turns.

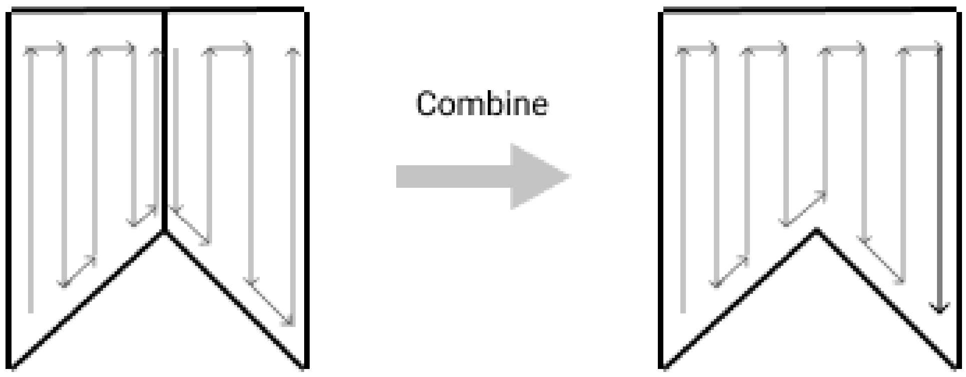


Figure 7.8: Fewer sub-regions are often better. Combining sub-regions can reduce redundant sweeps.

7.5.2 Approach

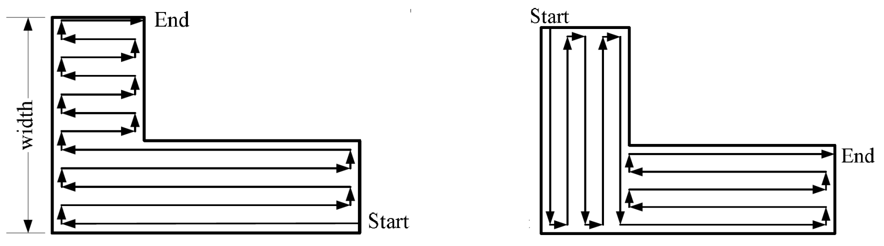
The cover path planning algorithm described in this chapter follows a greedy approach, inspired by Jiao *et al.*'s work [52].

We proved in Section 7.4 that setting the sweeping direction parallel to the support lines minimized the number of turns. Figure 7.9 demonstrates how plain path planning on a concave polygon would produce a sub-optimal solution. To deal with this, we propose a solution based on convex decomposition.

Convex Decomposition

Jiao *et al.* propose a simple yet effective algorithm for convex decomposition. The authors use a greedy recursive method that divides concave polygons into two sub-polygons by minimizing the sum of their widths. The first step is to split a concave polygon into smaller sub-polygons. For a starter, we know the following:

1. A concave polygon is defined as a polygon containing at least one concave vertex.



(a) Sub-optimal path. **(b)** Optimal path.

Figure 7.9: Sub-optimal coverage path with sweep direction based on support lines (left) compared with an optimal coverage path (right). ©2010 IEEE [52].

2. By splitting the polygon into two sub-polygons at a concave vertex, we may reduce the concavity.

We define a line splitting a polygon into two sub-polygons during convex decomposition as a decomposition line. We need to determine in what direction the decomposition line are to be drawn from a concave vertex. Jiao *et al.* shows that a decomposition line producing sub-polygons with minimum width sum is always parallel to one of the polygon's edges. For each concave vertex, the algorithm will have to consider all edges on that polygon and calculate the sum of widths. The decomposition line producing the minimum width sum is considered an optimal solution for a polygon. A split is not guaranteed to produce two convex polygons. If this is the case, the procedure is recursively repeated for all concave sub-polygons.

Path Planning

The proposed decomposing algorithm outputs one or more convex polygons. Since they are all convex, it is now possible to run a path planning algorithm and find a path on each of them. In the following, we define a path planning algorithm inspired by Jiao *et al.*'s work.

There are multiple dependencies when calculating the path of a convex polygon. A path within each sub-polygon is dependent on the drone's entry point to that particular polygon. An entry point is dependent on the sub-polygons' arrangement in relation to each other since an exit from one polygon leads to an entry to another. The order of the polygon is also dependent on the drone's start and finish position, which we can assume is at the same location for simplicity.

Inputs to the path planning algorithm are the location of the drone (start position), a list of all sub-polygons, and the width of the footprint. The first task is to find in what order the sub-polygons should be visited, i.e. searched. This problem is a version of the *traveling salesman problem*, which tries to find the shortest route through a list of locations, given the distance between each pair of locations. This problem is NP-hard and trying to solve this in polynomial time lies outside the scope of this project. We define an algorithm that starts at the vertex closest to the starting point and then visits that vertex's children. This is recursively repeated until a tree, with root at the drone's starting point, is found containing all sub-polygons. It is a fairly expensive algorithm, but solves the problem, and is feasible for a small number of polygons.

With the order of the sub-polygons, a path inside each of them can easier be determined. As described in section 7.4.1 the width of a polygon is determined by three vertices. In fact, either of these three vertices can be the starting point of a path inside the polygon. Figure 7.10 shows three possible starting points a path covering a polygon can have – 1a, 1b or 2.

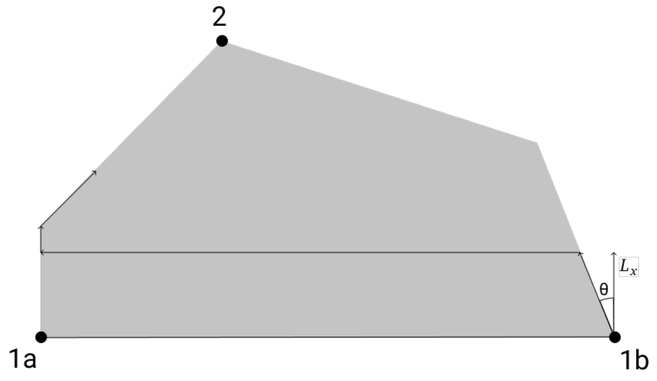


Figure 7.10: $1a$, $1b$ and 2 are three possible starting points of a path inside a polygon. We use θ and L_x to determine the next point in the path.

Note that a line passing through points $1a$ and $1b$ is a support line of the polygon. The second support line passes through point 2 and is parallel to the first.

The point closest to the exit point of the previous polygon will be the first point in the current polygon's cover path. The exit point will be located along the opposite support line. This project's proposed path planning algorithm starts at point $1a$ and $1b$. The path grows up towards point 2 by alternating between left and right side. We determine the next point in the path by using the angle θ between the *current edge* and a vector perpendicular to the support edges. The distance to travel along the current edge is determined by Equation 7.4 and the direction is the same as for the current edge. At the left side of Figure 7.10, we add an intermediate point that passes through the vertex, and, hence, change the current edge. The next edge towards point 2 in the polygon will then become the *current edge*.

$$D = \frac{L_x}{\cos\theta} \tag{7.4}$$

When the path has reached all the way to the other side (point 2 in Figure 7.10), the algorithm finishes and returns the path. The path is represented as a list of points, going from one support edge to another using a back and forth motion.

7.5.3 Implementation

The CPP algorithm is implemented as a Python script, using scientific libraries, like $matplotlib$, $numpy$ and $scipy$. For simplicity, we define a dedicated $Polygon$ class. This makes it easier to compute and store properties for each polygon alongside the polygon's sequence of points. A library called $shapely$ is used to find relations, intersections and topological operations.

All vertices are stored as *numpy arrays*. This makes it easier to perform matrix operations, lite dot product, addition, and subtraction. For instance we use the $numpy.dot$ (dot product) method to find the angle between two vectors $A = [a_x, a_y]$ and $B = [b_x, b_y]$, as given by Equation 7.5.

$$\theta = cos^{-1} \frac{A \cdot B}{|A||B|} \tag{7.5}$$

7.6 Results

The CCP algorithm is split into three steps:

1. Decompose the concave search area into smaller convex polygons.
2. Find a path inside each convex polygon.
3. Pick start and end points for each convex polygon, order the convex polygons and put their paths side-by-side.

Each step happens consecutively. The output from one step is input for the next. In this section, we examine the implementation of decomposition and path planning in addition to how to communicate with the CPP component.

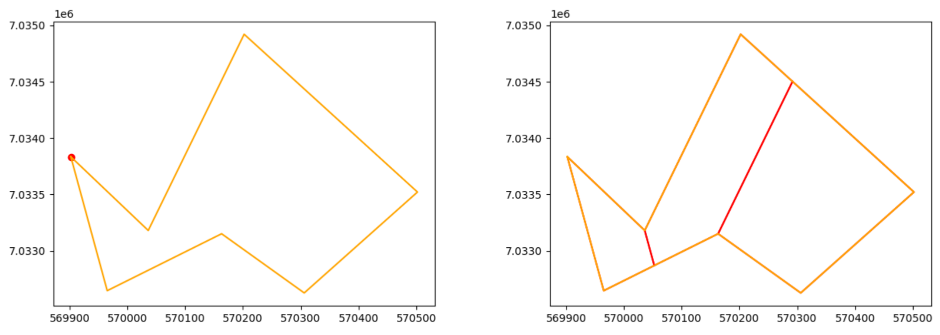
7.6.1 Convex Decomposition

In Listing 7.1, we see the implementation of the decomposition algorithm. Since we only decompose concave polygons, we stop execution in line 4 if the number of concave vertices is zero. The algorithm then enters a double loop. We have that `i` is an index for a concave vertex and `j` is an index for a vertex in the polygon. To draw a line from vertex `i` that is parallel to an edge, we use the edge's gradient. `split_line` on line 13 holds this line. Since the split line could be headed outside the polygon, we need to check if a successful split was found. The result will then be stored as a property on the decomposed polygon, and the sub polygons will recursively be decomposed.

```python
def decompose(polygon: Polygon, root_poly=None):
   number_of_vertices = len(polygon)
   if polygon.number_of_concave_vertices == 0:
      return

   sub_polygon_pairs = []
   for i in polygon.get_concave_vertex_indices():
      for j in range(number_of_vertices):
         vertex_i = polygon[i]
         if i == j:
            continue
         gradient = to_gradient([polygon[j], polygon[j + 1]])
         split_line = np.array([vertex_i, np.add(vertex_i,
             gradient * 1000)])
         split_result = polygon.split(split_line, i)
         if split_result is not None:
            sub_polygon_pairs.append(split_result)

   min_sum_of_widths_pair =
       find_min_sum_of_widths_pair(sub_polygon_pairs)
   polygon.sub_polygons = min_sum_of_widths_pair

   for poly in min_sum_of_widths_pair:
      decompose(poly, root_poly)
```

Code listing 7.1: Python implementation of the decomposition algorithm.

Figure 7.11 is an example of a successful decomposition of a concave search area. The decomposition outputs three convex sub polygons, as seen as red lines in Figure 7.11b. The red dot in Figure 7.11a represents the starting point of the search or the current location of the drone.



(a) **(b)**

Figure 7.11: Search area to be decomposed (a) and the decomposition result (b).

7.6.2 Path Planning

The next step is to calculate a back-and-forth path inside each sub polygon from the last step. The algorithm is shown in Listing 7.2. Line 7 alternates between `left` and `right` side so that the algorithm builds its path step-wise on each side.

```python
1   def find_path(self):
2
3       done = False
4       current_side = "left" if self.start_side == "right" else "right"
5       self.generate_side_edge(current_side, True)
6       while not done:
7           current_side = "left" if current_side == "right" else "right"
8           for forward_edge in [0, 1]:
9               done = self.generate_side_edge(current_side,
                    bool(forward_edge))
10              if done:
11                  break
12
13      current_side = "left" if current_side == "right" else "right"
14      self.generate_side_edge(current_side, True)
15
16      return self.path[::-1] if self.point_first else self.path
```

Code listing 7.2: Python implementation of the path planning algorithm.

The `generate_side_edge` method adds points alongside the polygon's left or right edge to the path. A variable (`forward_edge`) determines whether these points should be connected or not, as shown in Figure 7.12.

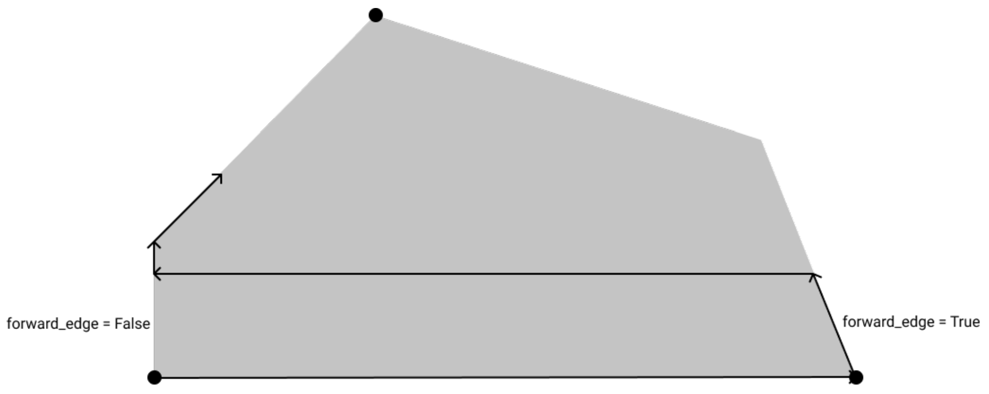


Figure 7.12: Variable `forward_edge` set to `False` on left and `True` on right.

The algorithm stops when the path has reached all the way to the other side of the polygon. The actual start and end points of the path through the polygon are given by the other sub polygons' paths in the final path planning result. This is determined by the `point_first` variable on line 16 in Listing 7.2. Figure 7.13 shows the result of a successful path planning. The result shows that the search area is completely covered, due to the fact that the distance from any given point in the area to a point on the path is less than or equal to the footprint width.

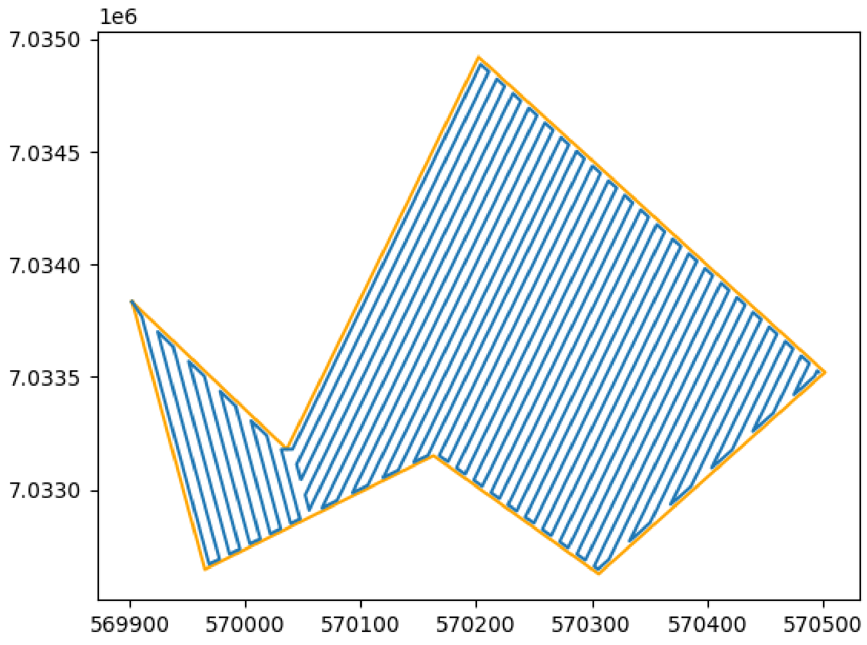


Figure 7.13: Resulting path (blue) from running the path planning algorithm on a concave search area (orange).

7.6.3 Interface

To be able to use the path CCP component, we need a way of communicating with the service. An application programming interface (API) is suited for this. An API is a web interface that accepts requests from a web client and responds with some data. The CPP service is set up to accept requests containing a search area and respond with a search path. This path is sent to the mobile application for mission execution.

To implement an API, we use a Python library called *Flask*. Flask runs as an ordinary Python script and listens to requests on one of the server's ports. A request to this port will run the CPP algorithm and return the path as a JSON response.

Chapter 8

Complete System

8.1 Introduction

A complete system utilizing the work done in the previous chapters was developed. The system allows for automatic flights along a path covering a user-defined search area. This chapter gives an overview of the system, describes each component, and how they interact. The features of the final system and how they interact is described in detail. Several tests are conducted to verify the feasibility of the final product – both under controlled conditions and in more realistic environments. Test results are set up against test goals and discussed in regards to the current problem description.

8.2 System Overview

8.2.1 Features

The complete system offers several features. The features can be categorized into features concerning user interaction, features concerning the flight path, and features concerning automatic drone flights.

Features Concerning User Interaction

From a user standpoint, the features are implemented as part of the mobile application. The features are as follows:

1. Interactive digital map, where the user can define an area.
2. Custom height at which the drone should fly. This allows for flights closer to the ground, resulting in more detailed pictures, or higher flights to cover larger areas.
3. Whether to fly at a specific altitude the entire path or follow the terrain.
4. Set the speed at which the drone should fly.

These features give the appropriate output for the subsequent parts of the system. In regards to the digital map, Chapter 5 describes the assumptions made and details the solution of the interactive map. All features listed above are present in the final map view of the application and assign more control to the user.

Features Concerning the Flight Path

A large part of the project is efficient coverage of an area. In the following, underlying features regarding the flight path are listed.

1. It creates a path given a polygon supporting different shapes, angles and amount of edges.
2. The flight path is calculated to cover the polygon efficiently, taking a minimal amount of turns.
3. Altitude parameters are added to each point, and the path is adjusted to account for changes in the terrain.

Features Concerning the Automatic Drone Flights

After a flight path is calculated and sent to the mobile application, the user is able to upload the path to the drone. When a path is executed, the following features are present during the flight:

1. The drone follows a path given by the uploaded mission.
2. Images are taken at intervals, covering the entire path.
3. The drone returns to the home location.

8.2.2 Architecture

This section gives a more detailed view of the different components of the system and how they interact. Much of the final system is based on the work done in previous chapters. Figure 8.1 shows all the components of the system. The software that was built in this project consists of an *altitude service*, a *path planning service* and a *frontend application*.

Frontend Application

The frontend application runs on an iPhone. It has three main purposes: providing a graphic user interface to the user of the system, sensing and receiving information from the remote controller, and communicate with the backend services. The frontend application is written in Swift and runs in an iOS environment. The map part of the application is described in detail in Chapter 5, while the communication with the drone is based on Chapter 4. Figure 8.2 shows the mobile application's default view. It shows a map and a number of actions the user may execute. The following are the different actions and what each does. Using these

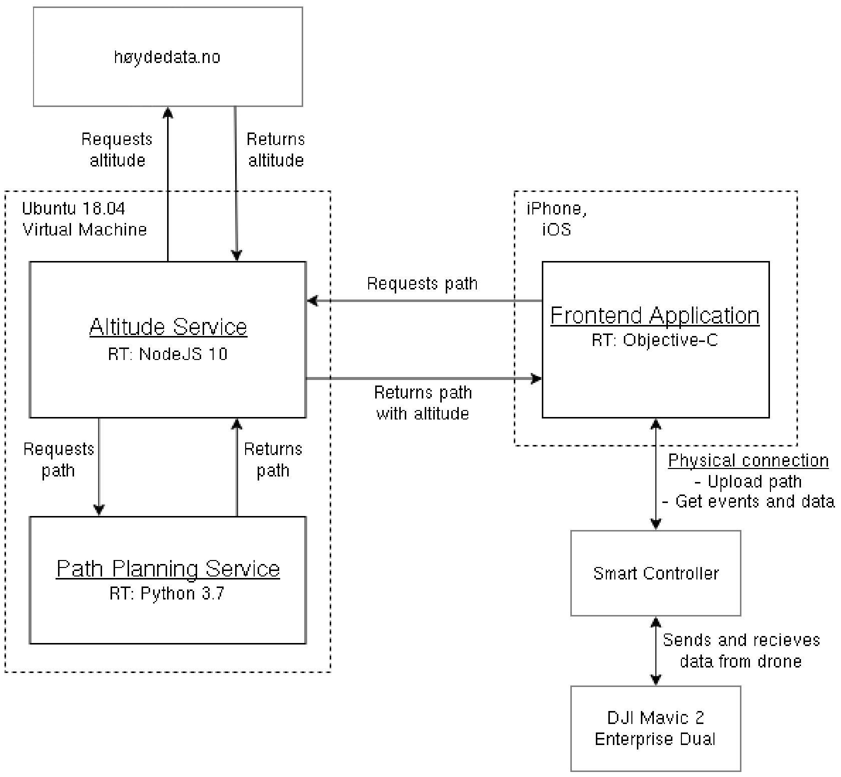


Figure 8.1: Overview of the complete system.

actions a user can perform a full scan of an area. The full code implementation can be found at `https://github.com/ohodegaa/SheepFinderApp` [24].

- **CHECK STATUS**: Checks the state of the drone. It shows as either *Disconnected, Ready to Upload, Uploading or Ready to Execute.*
- **Height**: Allows the user to define the height at which the drone should fly above the ground.
- **Use server height**: Enables or disables the use of height given by the altitude service. When enabled, the drone will follow the terrain. When disabled, the drone will fly at the same altitude disregarding the changes in the terrain.
- **Speed**: The speed at which the drone should fly.
- **Press and hold on the map**: When the user presses and holds on the map, a point is shown. Three or more points make a polygon.
- **Upload**: Uploads the polygon to the server.
- **Download**: Downloads the finished path from the server.
- **Prepare**: Prepares the mission and creates each point the drone should pass through.
- **Upload**: Uploads the mission to the drone.
- **Go**: Executes the mission. The drone will take off and fly automatically.
- **Stop**: Stops the mission. The drone will hover in place.

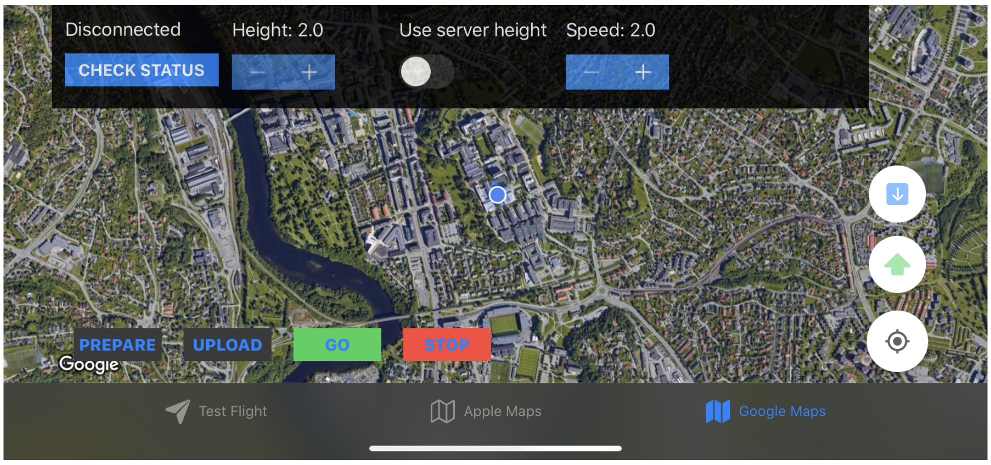


Figure 8.2: Default view of the front-end application.

Path Planning Service

The path planning service implements the algorithm described in Chapter 7. It receives a polygon and a footprint width from the altitude service and returns a path covering the polygon. The path planning service is written in Python and runs on an Ubuntu 18.04 Virtual Machine. The full code implementation can be found at `https://github.com/ohodegaa/Speepio` [57].

Altitude Service

The altitude service is written in JavaScript and runs on the same virtual machine as the path planning service. It has three main properties: it is the main entry point for the backend, it creates intermediate points, and supplies a path with altitude values.

At the endpoint of the backend, the Javascript program receives an array of points depicting a polygon as an HTTPS request. The points are given as objects containing latitude and longitude properties. These points are then converted into UTM, as the path planning algorithm assumes a flat two-dimensional surface. After converting the polygon to UTM, it is sent to the path planning service.

The path planning service responds with a covering flight path. Then, the altitude service creates intermediate points along the path at a fixed interval. The default is 20 meters. These points are used to make sure the drone follows the terrain at the same height through the entire path.

After creating intermediate points, the altitude service requests the altitude of each of these points from *hoydedata.no*. A point is not considered until the altitude differs with more than 10 meters. This is to minimize the number of intermediate points. The Altitude service is heavily based on the work done in Chapter 6. The full code implementation can be found at `https://github.com/jowies/SheepFinder-backend`[58].

Communication Between Components

As explained, there are multiple components that make the system. These communicate to create the final solution. Data is sent between the components using different protocols. In total, there are five links of communication.

1. **Drone ⟷ Remote Controller:** The communication between the drone and remote controller uses Wifi at either 2 GHz or 5Ghz frequency depending on the distance. Status messages are passed to the remote controller from the drone, and commands are sent the other way.
2. **Remote Controller ⟷ Application:** The remote controller and application are physically connected using USB.
3. **Application ⟷ Altitude Service:** The application and altitude service communicates over HTTPS – an encrypted, secure connection.
4. **Altitude Service ⟷ Path Planning Service:** The services communicate via HTTP. This is not a secure connection; however, as both services run on the same virtual machine, this is not an issue.
5. **Altitude Service ⟷ hoydedata.no:** hoydedata.no is a third party service. The height measurements of a point are requested from hoydedata.no over HTTPS.

Missions

In Chapter 4, missions were implemented to complete a test of automatic flight. It used `MissionControl` and specific actions such as `GoToAction` and `TakeOffAction`. The main takeaway from the test was that each addition action came at a high cost, as the drone would wait on average above 4 seconds in between each action. As this is highly inefficient, other solutions were looked at. There are more mission types available in the DJI SDK, as shown in 4.3.1. Waypoint missions fill all the requirements needed and was implemented instead of the custom mission used in Chapter 4. The main reason for this change is that Waypoint Missions allow the drone to take pictures between two points. Each waypoint is defined with a latitude/longitude coordinates as well as the altitude at which it should fly. The drone will take pictures at an interval of X meters, where the X is dynamically defined based on the height of flight as defined by the user. These pictures do not utilize the full potential of the camera, taking pictures and storing them as JPEG files. The reason for this is that JPEG is available at 2-second intervals. RAW files, which would be much more detailed, are only available every 10 seconds.

8.2.3 Process Model

The process of the system is shown in figure 8.3. It is divided into four: the drone, the user, the application/smart controller, and the server. The process model does not show the technical implementation but focuses on what happens, and in what order.

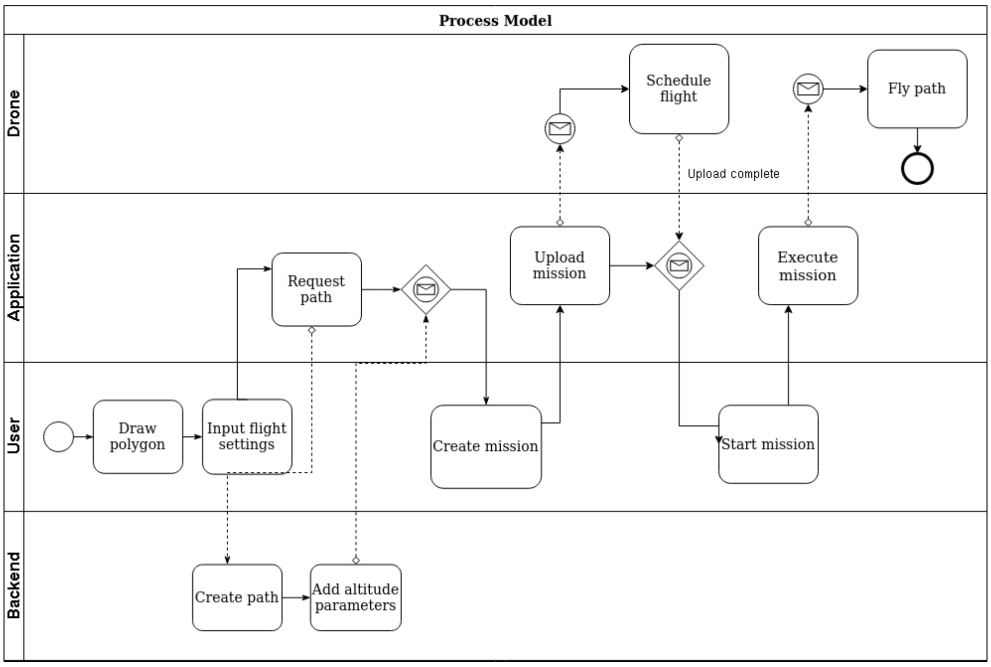


Figure 8.3: Overview of the process of the system.

8.3 Pilot Test

We perform a pilot test to verify that the complete system works as intended and does not contain errors. Figure 8.3 represents the intermediate steps for the test – from a user drawing the search area to successfully landing the drone. The overall goal is to verify the system components and the communication between them.

8.3.1 Test Setup

A pilot test is carried out in an environment with controlled parameters and with only the tools that are part of the complete setup. It, however, does not use the height parameters and stays at a fixed 2 meters altitude relative to the take-off point.

- Height: 2 meters
- Speed: 1 m/s
- Follow terrain: no

Test Equipment

The drone in use is a *DJI Mavic 2 Enterprise Dual*. This is controlled by a *DJI Standard Remote Controller* which has an *iPhone 6S* connected to it. The iPhone has a storage capacity of 32 gigabytes and is equipped with the latest software version (*iOS 13.5*).

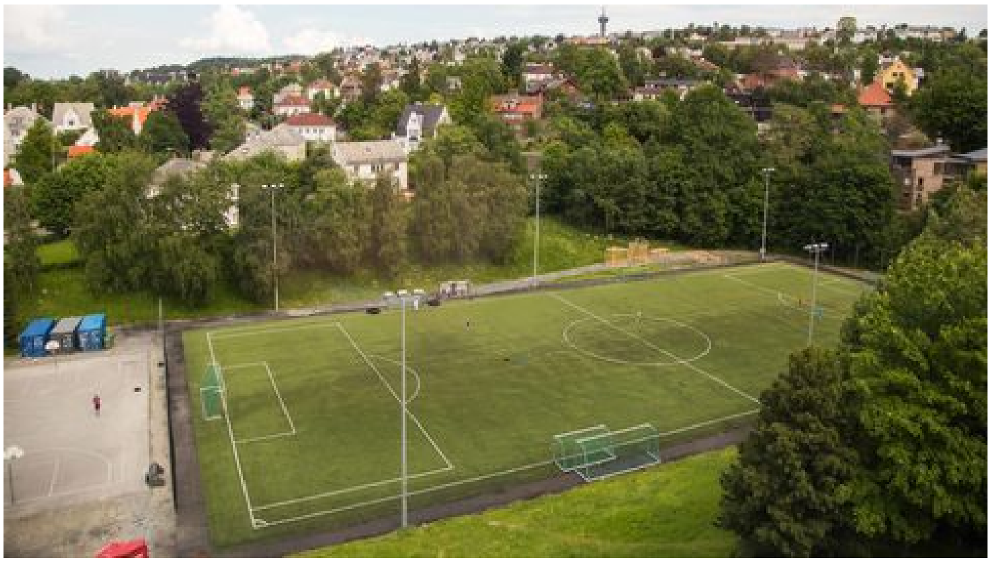


Figure 8.4: Overview of the safe test area in Dødens Dal.

Test Environment

The test is performed on a football field in Dødens Dal in Trondheim (see Figure 8.4). A group of people we're close to the test area during the execution of the test. They were notified about the situation and given a warning about possible risks. Also, there was a number of light poles around the football field. These could cause a potential hazard to the drone and involved personnel. The search area was drawn with a safe distance of approximately 20 meters away from any obstacle to prevent a crash.

8.3.2 Objectives

We define test objectives for the system to perform. The objectives represent the flow of the whole system and are to be executed in a given order. Each objective below is described as a requirement for a component of the system.

Open and Initialize the Application: A user should be able to open the mobile application without it crashing and see a front-page describing the state of the connected aircraft.

Navigate to the Map View: A user should be able to navigate to a screen containing a map view. In the map view, the user should find a number of possible actions to perform, in addition to its current location on the map.

Check Mission Status: The user should, at all times be able to check the status of the mission.

Draw a Search Area: The user should be able to draw a search area on the map. This includes being able to insert, remove, and move markers on the map as desired.

Submit the Search Area to the system: The user should be able to submit the drawn search area to the system (i.e. backend server).

Process the Search Area and Calculate the Path: The backend server should be able to accept incoming requests, containing a search area, process this and calculate a path. The path should include coordinates and an altitude for each intermediate step.

Download the Search Path: The user should be able to download the path from the system. When downloaded, the path should be visible as a pattern on the map, encapsulated by the drawn search area.

Prepare for Flight: The user should be able to prepare the drone for a flight. This involves setting up the mission and uploading the mission to the aircraft.

Execute the Mission: The user should be able to execute a mission, i.e. instruct the aircraft to start the mission and perform the search.

Search: The drone should perform its search when instructed by the user (application). This involves taking off, moving according to the search path and landing safely at the home location. During the search, the drone should take photos at a specific rate and save these in the aircraft's internal storage.

Stop the Search: The user should be able to stop the execution of a search at any time. The user should then be able to manually control the aircraft as desired, using the remote controller.

8.3.3 Results

The user successfully turned the aircraft and remote controller on and connected the phone to the remote controller. An indication that a connection between the aircraft and the remote was established was given by the controller. The user was then able to open the application and initialize the connection to the DJI SDK and the aircraft. This was done pressing a refresh button that updates the aircraft connection's state in the view.

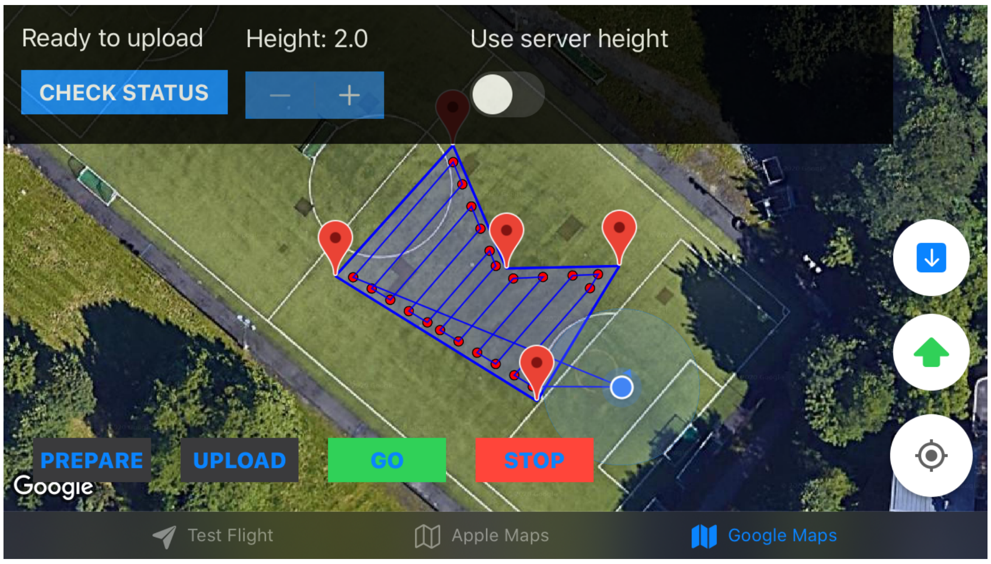


Figure 8.5: Path downloaded from server describing the search path.

By pressing *START*, the user entered the main view of the application and navigated to the screen containing the map. A search area was drawn on the screen and the *upload* button was pressed. Outputs from the backend server's log confirmed that the search area was received, but for a number of times, it ran in an infinite loop. The user made some adjustments to the search area on the map and resubmitted it to the server. The server ran successfully after three trials, and the user was able to download a correct path, as seen in Figure 8.5. The blue dot marks the user's position and is also the start and end location of the path.

The user successfully uploaded the mission to the drone and was able to execute the search mission without trouble. This led to the drone taking off and following the path from start to end. Instead of landing at the end, the drone did hover 3 meters above ground, so the user had to land manually using the remote controller.

Another test was executed with the intention of specifically testing the stop procedure. The system was not able to use the same mission as was already defined. The solution was to restart the application and perform the procedure again. After that, the user pressed *STOP*, and the aircraft stopped and hovered above the ground, as intended.

Switching from `GoToActions` to `Waypoints` led to a remarkable decrease in time used on intermediate steps. The test revealed that the time between waypoints was negligible, i.e. there was no clear distinction between one waypoint and the next.

8.4 Full-Scale Test

Two tests were performed under more realistic conditions. The tests followed the same pattern as the Pilot Test, but both the height and speed are more akin to what would be used in a practical setting. The tests were done in the same area, with a difference in altitude, namely 40 meters and 20 meters.

8.4.1 Test Setup

The tests require minimal additional setup from the pilot test. It uses the same application but with different parameters.

40 Meters Altitude

The 40-meter altitude test has the following flight parameters:

- Height: 40 meters
- Speed: 10 m/s
- Follow terrain: yes

20 Meters Altitude

The 40-meter altitude test has the following flight parameters:

- Height: 20 meters
- Speed: 10 m/s
- Follow terrain: yes

8.4.2 Test Environment

Both tests were done in an area with slight changes in altitude consisting of mostly of farm- and woodland. The starting point for both tests where 62°24'15.6"N 10°59'23.8"E. This is in Tolga, a county in Norway.

8.4.3 Objectives

The objectives are the same as in the pilot test.

8.4.4 Results

40-Meter Altitude

The 40-meter altitude test went without problems. The path of the flight can be seen in figure 8.6. While the changes in altitude are visualized in figure 8.7. The highest point relative to the starting point is at 82 meters, while the lowest is at 36 meters. In other words, the difference in altitude in the path is 46 meters. Figure 8.8 shows an image taken by the drone during the test. In the upper right quadrant, one can clearly see one sheep with two lambs.

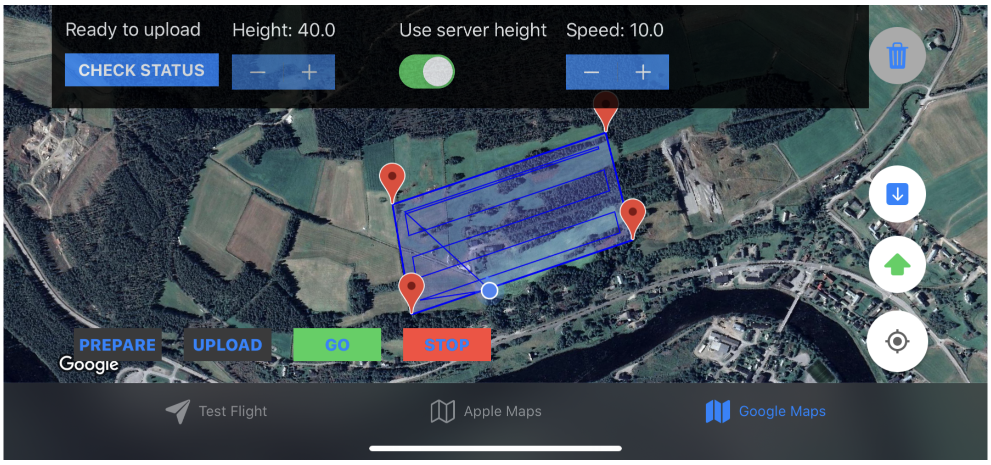


Figure 8.6: Area and path covered in test at 40 meters altitude.

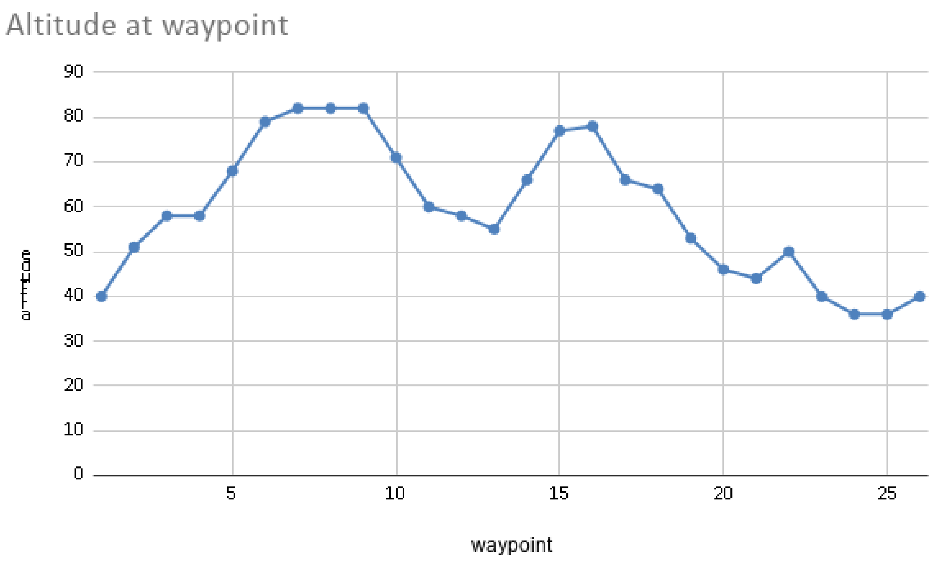


Figure 8.7: Height relative to takeoff altitude for each waypoint in test at 40 meters altitude.

20-Meter Altitude

The 20-meter altitude test resulted in a crash. The intended path of the flight can be seen in figure 8.9, while the changes in altitude are visualized in figure 8.10. The highest point relative to the starting point is at 30 meters, while the lowest is at 13 meters. In other words, the difference in altitude in the path is 17 meters. Figure 8.11 shows an image taken by the drone during the test. One can clearly see a small herd of sheep consisting of three sheep and 6 lambs.

The drone crashed between waypoint 8 and 9, in a thickly wooded area. The

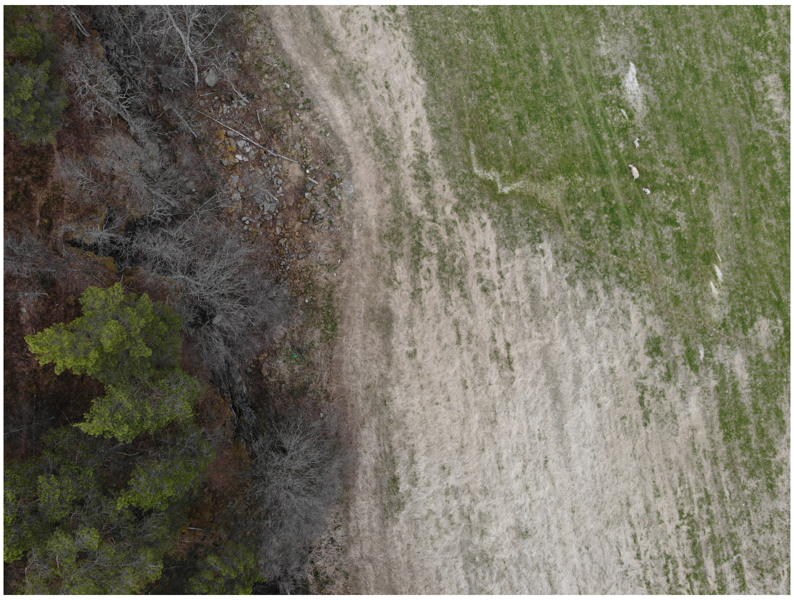


Figure 8.8: Image taken by drone during test at 40 meters altitude.

remote controller flashed "obstacle", and the drone operator tried to increase the height of the drone. No error message was shown, and the drone showed an altitude of 0 still connected to the remote controller. The recovered drone was partly buried in the ground, with one antenna loose, but no propellers were broken.

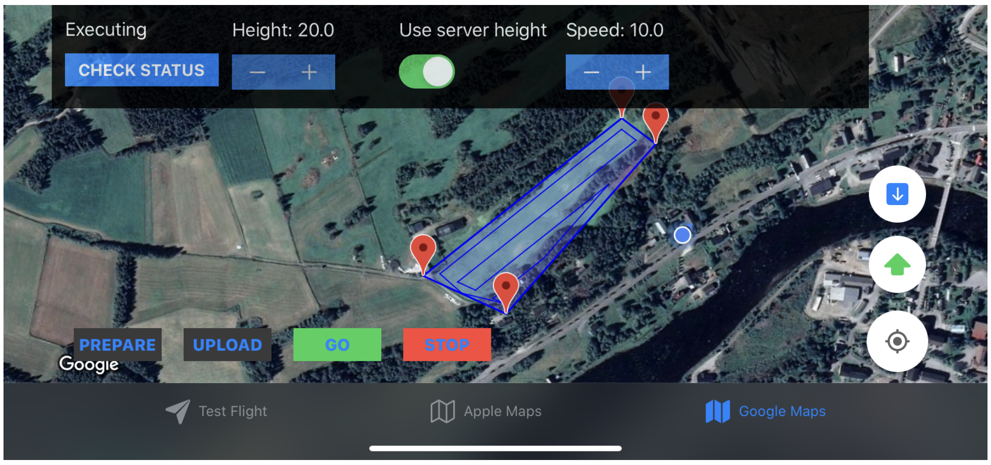


Figure 8.9: Area and path covered in test at 20 meters altitude.

8.5 Discussion

Multiple tests were performed, and each is discussed in this section, first the pilot test followed by the two realistic tests.

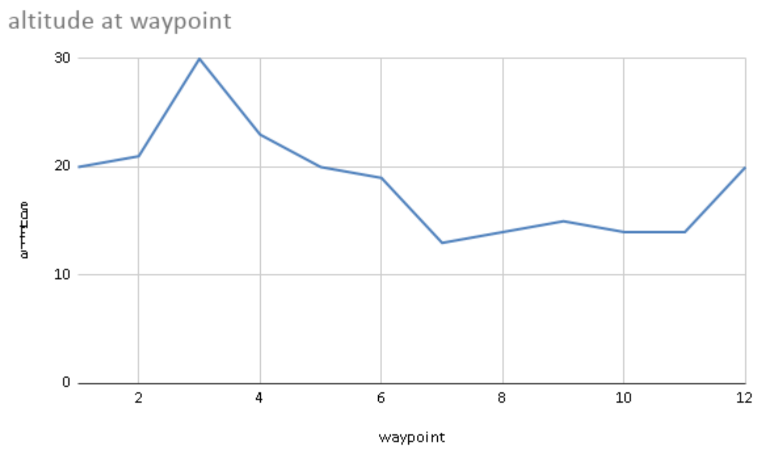


Figure 8.10: Height positions of each waypoint in test at 20 meters altitude.

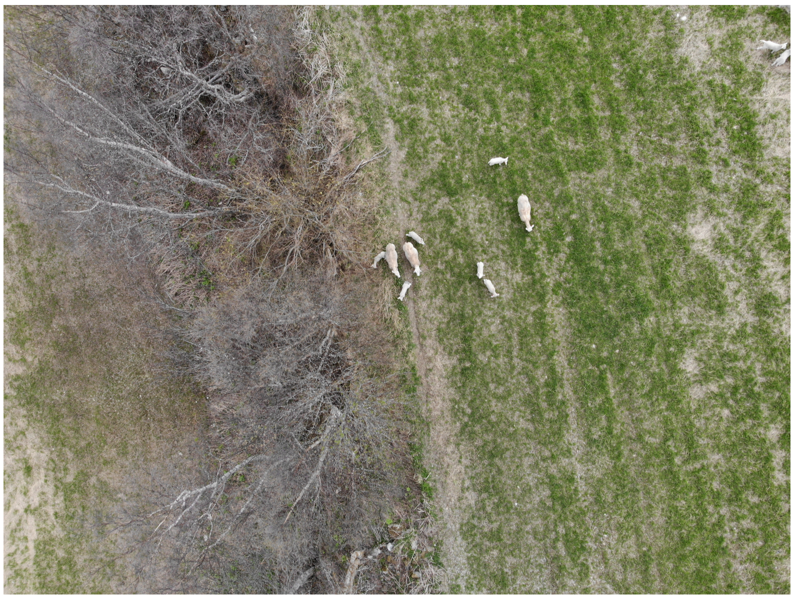


Figure 8.11: Image taken by drone during test at 20 meters altitude.

8.5.1 Pilot Test

The first problem occurred when the search area was uploaded to the server. The path planning algorithm was not able to exit from an infinite loop. It is not clear what causes this error, but development has revealed that floating-point errors can lead to unwanted results. This can happen when you store a floating-point number (decimal number representation) in Python. Because of limitations on the number of decimal places that can be stored in memory, round-offs are performed on number representations. This can, for instance, lead to the conclusion that a point is *not* located on a line segment when, in reality, it is.

When it comes to the user interface in the application, more feedback should be given to the user. The user does not know whether an action has been properly executed or not, and errors are not communicated to the user. The only way of telling the current state is to use the *CHECK STATUS* button and watch out for changes on the map. A suited solution would have been to notify the user with an alert telling if an action was successful or not.

As for the current system architecture, the user has to first upload the search area to the server and then request a result. If the processing task is not yet finished, the server will respond with an empty result when the user requests it. This leads to several requests to the server without any useful results being returned. It is designed in this manner to avoid connection time-outs with the server. A better solution is to make the server notify the application when processing is completed.

The server sometimes responded with strange paths. Some of them had paths going outside the search area, and some lacked paths in some parts of the search area. It is unknown what caused this issue, but it is assumed that it is one of several edge cases that has not yet been tested properly. Further testing on the path planning algorithm has to be done to prevent such errors.

Time used on intermediate steps was negligible, and the test therefore successfully score WaypointMissions higher than GoToActions when it comes to search time. The former also supports features where the camera can be set to take photos every x meters. For this to be possible in the later, an *Action* must have been added for each photo capture point – increasing the time used in intermediate steps.

8.5.2 Full-scale Tests

40 meters

The 40-meter altitude test can be considered successful, the test went as expected, covering the entire area, and taking images at even intervals throughout the test. By looking at Figure 8.8, the sheep can be seen clearly. However, there is reason to believe that a further height increase might make the sheep hard to see. Something to note is that the test takes pictures and stores them as JPEG and using RAW format would result in much more detailed pictures, at the cost of memory space and intervals at which pictures may be taken. The altitude difference in the waypoints was confirmed to work well, and the drone flew evenly over the terrain.

20 meters

The 20-meter test was a success until the drone crash. It took clear pictures and followed the terrain. Figure 8.11 clearly shows a herd of sheep. It followed the terrain up until the crash.

As to why it crashed, there are some possibilities. Figure 8.12 shows the altitude of every meter between waypoint 8 and 9. The blue line is the terrain, while the red line includes trees. The current version of the system does no use the surface model, but the terrain model, and as the figure shows, the flight path (yellow line), goes straight through the red line. This means the tree was in the way of the drone. This is not good enough, and the system should consider the surface of the area to make sure there is adequate clearance.

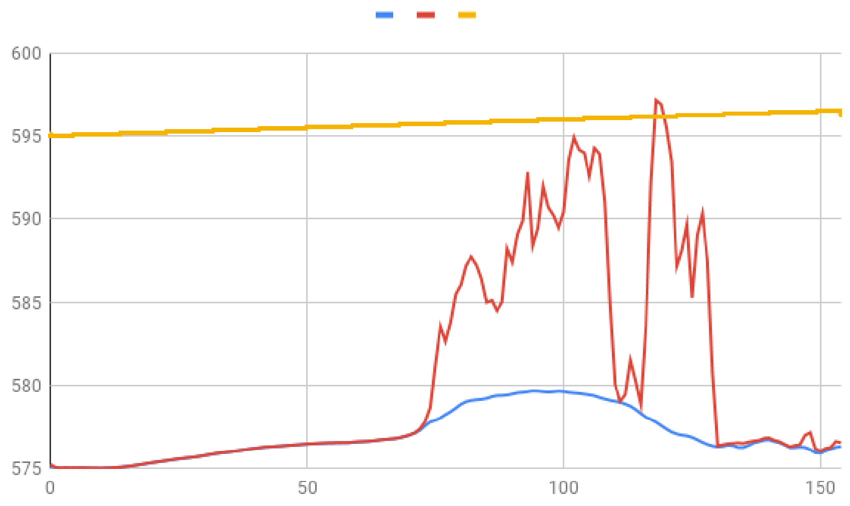


Figure 8.12: Height measurements between point 8 and 9 in test at 20 meters altitude. The blue line is the ground terrain, the red line includes trees, and the yellow line is the flight path.

It also has to be pointed out that the system only checks the altitude at every 20 meters, and if the height difference is not more than ten meters, it removes that point from the path. This would also cause inaccuracies and while deemed sufficient for flights at 40 meters above ground, should have been adjusted for the 20 meters test.

Another issue is the height measurements themselves. The height measurements are from 2018 and should be fairly accurate, while we can assume the terrain to be largely unchanged there might have been changes in vegetation, although it was not the key factor for the crash in this test.

Lastly is the strange case concerning the crash. No propellers were destroyed. This means that the drone did NOT hit anything in the air as this would break one or more propellers. As the remote controller flashed "obstacle", it is clear that something was near the drone at the time of the crash. No propellers broken points to the electrical engines simply stopping their rotation and the drone free-falling into the ground.

Regardless of what caused the crash, the work in preparation of the test should have been more thorough to prevent a crash from happening. A risk analysis of the test might have prevented the errors in the 20 meters altitude test.

Chapter 9

Discussion

9.1 Tests

Chapter 4 explore the DJI drone's capabilities of performing automatic flights. Some difficulties arose due to inadequate documentation for the DJI Mobile SDK. Many features responded with a "This feature is not supported by the SDK" message while the documentation stated that it was implemented. An initial test leads to the drone crashing into a branch and falling five meters to the ground. Luckily the aircraft survived, but this incident proves how bad things can happen. One can only imagine what would have happened if the same thing had happened from 30 meters altitude. It is clear that pilot tests should be conducted in areas without vegetation and other obstacles. A similar crash happened in the 20 meters altitude test. It was however different in circumstances as none of the propellers broke. From the experience of the first failed test, one would expect the propellers to break upon impact with an object. This makes the last crash especially interesting as all propellers were whole, indicating the crash was not coming from an impact with an object but from something else. What this could be is hard to say. But a theory is either that the drone automatically detected it was about to crash, and instead of hitting the tree head-on, turned off its engines and crash-landed. This, however, does not seem to be a common behaviour, and at this time, it is not possible to conclude what actually happened.

Section 8.3.3 describes a more or less successful pilot test. The system eventually worked as intended, and the drone searched through the whole search area. As small errors occurred, it was hard to locate the actual bug. It could be in several of the system components. Things get even harder with the altitude service as it relies on a third-party web service to fetch the altitude map. A better approach would have been to test smaller groups of components before the final test. In this way, it would have been easier to locate weak spots in the system and debug components and interfaces separately.

Several trials were made in the test before a path was returned. The error was the path planning component that ran in an infinite loop. More test with real data should have been conducted before the pilot test to ensure that it worked properly. Data that was used when implementing and testing locally was significantly different from the real data. One can assume that more errors could have been discovered with real data in the test set.

The accuracy of the real data was greater than for the test data, i.e. they include more decimal digits. Since the floating-point error (FPE) mechanism in the path planning algorithm was developed for less accurate numbers, it is assumed that it overcompensated with real-life data. As the algorithm is both highly iterative and recursive, the error possibly increased during execution. A solution to this could have been to analyze the accuracy of the incoming data and adjust the FPE mechanism accordingly.

The 40 meters altitude test in Section 8.3.3 was by and large successful. No problems occurred, and images were taken of the whole area. The test shows the potential of the system. It efficiently covered the area flying at a speed of 10 meters per second. It followed the terrain consistently and landed at the starting point. The images in the test clearly show grazing sheep.

While the 40-meter altitude test highlighted the potential of the system, the 20-meter altitude test highlighted the shortcomings. As already mentioned the cause of the crash is unknown, but from the results of the test show that the drone flew close to the top of the tree line, and the remote controller flashed "obstacle". The 20-meter altitude test suffered from decisions made early in the project when the intended height of the drone was above 40 meters. Forty meters gives a significant clearance of obstacles and trees. The 20-meter interval altitude adjustment between two points is not sufficient at lower altitudes and should have been much smaller.

As a result of the crash in the 20-meter altitude test, the altitude service was modified to accommodate these scenarios. Instead of checking every 20 meters, it now uses a different method of getting the altitude data from hoydedata.no. It now checks both the terrain and surface on the path every meter. It defaults to using the terrain, and if the clearance to an obstacle such as a tree or building is less than 5 meters, it uses the surface model and adjusts and flies higher in those areas. This implementation was tested in isolation using the same data as the 20 meters test, and the difference between the two implementations and how they relate to the terrain and surface of the area can be seen in Figure 9.1. It also adjusts itself so that the threshold for lowering its height is greater than that for increasing its height. The graph is the altitudes between waypoint 8 and 9, where the crash originally happened.

As a result, the system is now much more robust in relation to obstacles in the terrain. Instead of every 20 meters, every single meter is checked. The change also has a side effect of being much faster, as the data from hoydedata.no are received in batches.

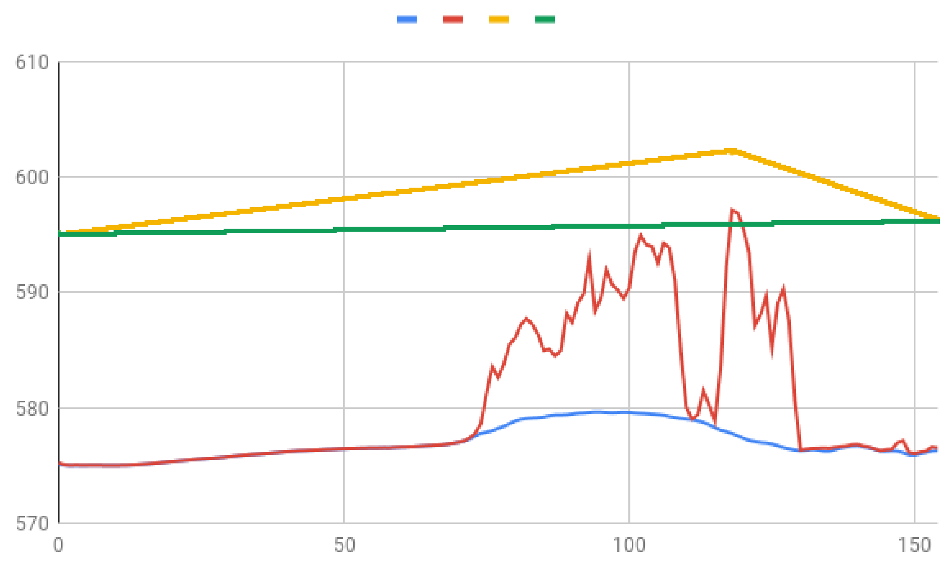


Figure 9.1: Results after modifying the altitude service and how it solves the crash problem. Blue is the terrain model, red is the surface model, green is the old flight path, and yellow is the new flight path.

9.2 The Complete System

The complete system is in a lot of ways a minimum viable product. It offers automatic flight, and interface for user-defined areas and settings, and takes consideration in elevation and terrain. Each part can be improved on for a more robust and secure system.

9.2.1 The Application

The application has evolved throughout this project. Design and layouts have changed since the first usability test, as more features have been applied. In other words, the system should have been tested with non-biased users to verify the final design. Due to the COVID-19 situation, this has not been possible. One can argue that this falls outside of the scope of this project since the main focus has been verification and feasibility testing. It is, however, still important to consider the users' needs and requirements when designing an application for practical use.

The test was performed on five participants. It can be argued that *the more the merrier*, but because of the COVID-19 situation, the number of tests is kept to a minimum. A possible approach could have been to do tests more iterative and test larger parts of the mobile application later in the project. From the test, it is clear that the map portion of the application works as intended, but with room for improvement.

9.2.2 Altitude

The altitude service is one area where improvement would directly lead to an increase in security and stability, as shown by the 20-meter altitude realistic test. The altitude service was improved after the 20-meter altitude test, and conditions leading to the crash has been solved. However, it still needs work. Although it now should be safe to fly in most areas, it is not recommended to fly at a high altitude. The new implementation uses both the surface and terrain models, but the security from the surface models is not available in the entirety of Norway. Therefore, the user should verify that the are they intend to fly at is covered by the surface model. This can be done at hoydedata.no. The new altitude service should be tested further, for a range of altitudes and areas to ensure that it is safe to use in a full-scale test.

9.2.3 Path Planning

Although many solutions to the CPP problem exist, every case has its pros and cons. Since the proposed solution does not have strict performance requirements, some shortcuts were made to decrease implementation time. The implementation did at first take advantage of the Shapely library to store shapes. The library came short later on as the system required calculations that were not supported by Shapely. Although third-party libraries can be handy to do quick implementations, one often realizes that some requirements are not met. Many Python libraries are large and come with an overhead. One should, therefore, weigh the benefits of using third-party libraries. In this case, implementing the functionality needed from the bottom was the right choice in the end.

Jiao *et al.* [52] implement an intermediate step in their algorithm where two sub polygons are combined based on some conditions. This is beneficial to minimize the number of turns but is not implemented in the proposed solution. Since the number of sub polygons is reasonably small, a decision has been made to skip this part. It should be a requirement in a production solution, but since the current solution focus on feasibility and testing, it is given less priority.

9.3 Future Work

Flying a drone outside can lead to hazardous situations, both for the drone operator, animals, buildings, and other people. This project focuses on testing and validating a system as a whole. Many security requirements have, therefore, been neglected. As a result, the proposed solution is not yet ready for production use and all use of the system and accompanying code without support from the authors' is therefore not recommended.

More work has to be done in able to use it in real-life situations. First of all, security has to be dealt with to a greater extent, as mentioned in section 2.3. This involves investigating mechanisms to handle power lines, urban areas, buildings, and other aircraft. Many DJI drones do have object avoiding features; however, as there has occurred two crashes, it can be concluded that the object avoidance is not sufficient for practical use. Any future implementation should not rely on object avoidance as the only measure to avoid colliding with objects.

9.4 Market and User Needs

It is up to debate whether or not this system can be used to benefit farmers. As the prices on sheep meat [59] is relatively low, it can get too expensive for a farmer to invest in such a system. If it is less expensive to loose sheep than to buy the proposed system, it would not be beneficial. Market analyzes, in addition to further development, have to be conducted in order to tell if the market is ready for the proposed solution.

It can be hard to introduce such a technological solution into a less digitized profession like farming. Many farmers might fear that *robots* will run them out of business and, hence, are sceptical to the proposed solution. This project is about proving a concept, and the motivation is the health of the animals. The hope is that it brings more innovation into farming and that farmers find it useful.

Chapter 10

Conclusion

The goal of this thesis was to prove a concept and investigate the feasibility of an animal monitoring system using drones. By conducting tests for each component, the system has iteratively evolved into a functioning solution. Answers to the underlying research questions are given in this chapter.

First, it is possible to program a consumer drone to fly automatically in user-defined patterns. This question was answered in Chapter 4, and shows how a DJI Mavic 2 Enterprise Dual using the official DJI SDK can be programmed for automatic flight. DJI is the largest manufacturer in the world of consumer drones. Moreover, although the code would not work out of the box on other drones, it should work on most DJI consumer drones.

The second research question relates to finding an efficient path covering an area. As described in Chapter 7, a path is found by first applying a convex decomposition to concave polygons, and then building back-and-forth paths perpendicular to the direction of the width. Despite some troubles during the pilot test (see Section 8.3), the algorithm, in general, produce successful paths, capable of covering a wide range of polygons. Further work has to be done to support edge cases fully.

The final research question is answered mostly in Chapter 8. The chapter shows the complete system made in this thesis. The success of the pilot test and the 40-meter altitude test shows that the system works as intended and fulfils its purpose under ideal conditions. The crash in the 20-meter test shows that more work is needed for the product to be used in real situations and provide value to the user. The root cause of the problem in the crash was fixed, and using the surface model of the area means it should be safe to use in areas with trees.

Further development and testing are needed. A survey should be conducted on the need for such a system, targeting sheep farmers. The focus of this thesis was feasibility testing and creating a proof of concept. It provides a base for further work in the digitization of animal husbandry and agriculture.

Bibliography

[1] M. Silvagni, A. Tonoli, E. Zenerino and M. Chiaberge, 'Multipurpose uav for search and rescue operations in mountain avalanche events', *Geomatics, Natural Hazards and Risk*, vol. 8, no. 1, pp. 18–33, 2017.

[2] L. F. Gonzalez, G. A. Montes, E. Puig, S. Johnson, K. Mengersen and K. J. Gaston, 'Unmanned aerial vehicles (uavs) and artificial intelligence revolutionizing wildlife monitoring and conservation', *Sensors*, vol. 16, no. 1, p. 97, 2016.

[3] V. Gatteschi, F. Lamberti, G. Paravati, A. Sanna, C. Demartini, A. Lisanti and G. Venezia, 'New frontiers of delivery services using drones: A prototype system exploiting a quadcopter for autonomous drug shipments', in *2015 IEEE 39th Annual Computer Software and Applications Conference*, IEEE, vol. 2, 2015, pp. 920–927.

[4] J. Linnestad and O. H. Ødegaard, *Manuell oppfølging av sau på beite*, Dec. 2019. [Online]. Available at: https://github.com/ohodegaa/manuell-oppfolging-av-sau-paa-beite/raw/master/Linnestad-og-Oedegaard-Manuell-oppfoelging-av-sau-paa-beite.pdf.

[5] Ministry of Agriculture and Food. (2010). Lov om dyrevelferd, [Online]. Available at: https://lovdata.no/dokument/NL/lov/2009-06-19-97 , visited on: 03/02/2020.

[6] Ministry of Agriculture and Food. (2008). Forskrift om velferd for småfe, [Online]. Available at: https://lovdata.no/dokument/SF/forskrift/2005-02-18-160 , visited on: 03/02/2020.

[7] SSB. (2019). Livestock grazing on outfield pastures, by region, contents and year, [Online]. Available at: https://www.ssb.no/en/statbank/table/12660/chartViewColumn/ , visited on: 12/05/2020.

[8] H. A. Solbakken and S. H. Berge. (16th Aug. 2019). Nedgang i tap av sau til ulv på beite i 2019, [Online]. Available at: https://www.nrk.no/innlandet/markant-nedgang-i-tap-av-sau-til-ulv-pa-beite-i-2019-1.14661400 , visited on: 24/02/2020.

[9] S. Tallaksrud. (23rd Jul. 2017). Ulven er ikke sauens største fiende, [Online]. Available at: https://www.nrk.no/innlandet/xl/ulven-er-ikke-sauens-storste-fiende-1.13566997 , visited on: 24/02/2020.

[10] U.S. Department of Defense (DoD), General Atomics Corp., Honeywell and Raytheon. Mq–1b predator / mq-1c gray eagle / mq–9 reaper, [Online]. Available at: `http://www.fi-aeroweb.com/Defense/MQ-1-Predator-MQ-9-Reaper.html` , visited on: 04/04/2020.

[11] JoJo. Types of drones – explore the different models of uav's, [Online]. Available at: `http://www.circuitstoday.com/types-of-drones` , visited on: 12/05/2020.

[12] C.-M. Tseng, C.-K. Chau, K. M. Elbassioni and M. Khonji, 'Flight tour planning with recharging optimization for battery-operated autonomous drones', *CoRR, abs/1703.10049*, 2017.

[13] Flytrap. (10th Mar. 2020). Drones and magnetic interference, [Online]. Available at: `https://www.viviscape.com/post/drones-and-magnetic-interference` , visited on: 01/04/2020.

[14] C. Clips. (22nd May 2020). Territorial bird attacks flying drone, [Online]. Available at: `https://www.youtube.com/watch?v=1NY3Df_Wuc4` , visited on: 05/11/2019.

[15] C. Clips. (7th Nov. 2015). Phantom 3 get kidnapped by two eagles, [Online]. Available at: `https://www.youtube.com/watch?v=FX3uOQiZs0A` , visited on: 05/11/2019.

[16] Luftfartstilsynet. (2020). Luftfartstilsynet (civil aviation authority), [Online]. Available at: `https://luftfartstilsynet.no/`.

[17] Stortinget (Supreme Legislature of Norway), Ministry of Transport, *Forskrift om luftfartøy som ikke har fører om bord mv.* 30th Nov. 2015. [Online]. Available at: `https://lovdata.no/dokument/SF/forskrift/2015-11-30-1404` , visited on: 05/02/2020.

[18] Datatilsynet. (20th Jun. 2018). Generelt om droner, [Online]. Available at: `https://www.datatilsynet.no/personvern-pa-ulike-omrader/overvaking-og-sporing/droner---hva-er-lov/droner/` , visited on: 10/03/2020.

[19] DJI, *Mavic 2 enterprise series - user manual*, Sep. 2019. [Online]. Available at: `https://dl.djicdn.com/downloads/Mavic_2_Enterprise/20190917/Mavic_2_Enterprise_Series_User_Manual-EN.pdf` , visited on: 18/02/2020.

[20] M. Borak. (3rd Jan. 2018). World's top drone seller dji made $2.7 billion in 2017, [Online]. Available at: `https://technode.com/2018/01/03/worlds-top-drone-seller-dji-made-2-7-billion-2017/` , visited on: 18/02/2020.

[21] IDC. (20th Apr. 2020). Smartphone market share, [Online]. Available at: `https://www.idc.com/promo/smartphone-market-share/os` , visited on: 08/06/2020.

[22] Statcounter. (Feb. 2020). Mobile operating system market share norway, [Online]. Available at: `https://gs.statcounter.com/os-market-share/mobile/norway` , visited on: 03/05/2020.

[23] SSB. (2018). Norsk mediebarometer, [Online]. Available at: `https://www.ssb.no/statbank/table/05244/tableViewLayout1/` , visited on: 04/03/2020.

[24] J. Linnestad and O. H. Ødegaard. (2020). Sheepfinderapp, [Online]. Available at: `https://github.com/ohodegaa/SheepFinderApp`.

[25] Yr (NRK and Meteorologisk institutt). (2020). Yr - 'trondheim' - statistics, [Online]. Available at: `https://www.yr.no/en/statistics/graph/1-211102/Norway/Tr%C3%B8ndelag/Trondheim/Trondheim?q=2020-03-10` , visited on: 11/03/2020.

[26] Apple Inc. (2020). Human interface guidelines - design - apple developer, [Online]. Available at: `https://developer.apple.com/design/human-interface-guidelines/` , visited on: 22/03/2020.

[27] Apple Inc. (2020). App and website maps - maps - human interface guidelines - apple developer, [Online]. Available at: `https://developer.apple.com/design/human-interface-guidelines/maps/overview/introduction/` , visited on: 22/03/2020.

[28] Axis Maps. Cartography guide, [Online]. Available at: `https://www.axismaps.com/guide/` , visited on: 22/03/2020.

[29] Google LLC. (2020). Google maps, [Online]. Available at: `https://www.google.com/maps` , visited on: 22/03/2020.

[30] Apple Inc. (2020). Apple maps, [Online]. Available at: `https://www.apple.com/ca/ios/maps/` , visited on: 22/03/2020.

[31] Mapquest. (2020). Mapquest - maps, driving directions, live traffic, [Online]. Available at: `https://www.mapquest.com/`.

[32] OpenStreetMap. (2020). Openstreetmap powers map data on thousands of web sites, mobile apps, and hardware devices, [Online]. Available at: `https://www.openstreetmap.org/about` , visited on: 23/03/2020.

[33] (2020). Maps.me (built on open street map), [Online]. Available at: `https://maps.me` , visited on: 23/03/2020.

[34] Cocoapods. Cocoapods, [Online]. Available at: `https://cocoapods.org/` , visited on: 25/01/2020.

[35] J. S. Dumas, J. S. Dumas and J. Redish, *A practical guide to usability testing*. Intellect books, 1999, p. 259.

[36] J. Nielsen. (Mar. 2000). Why you only need to test with 5 users, [Online]. Available at: `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/` , visited on: 16/04/2020.

[37] Høydedata. Høydedata, [Online]. Available at: https://hoydedata.no/LaserInnsyn/, visited on: 22/04/2020.

[38] Høydedata. Brukerdokumentasjon høydedata, [Online]. Available at: https://hoydedata.no/LaserInnsyn/help_no/index.htm?context=130, visited on: 22/04/2020.

[39] Kartverket. Nn2000, [Online]. Available at: https://www.kartverket.no/NN2000, visited on: 22/04/2020.

[40] Kartverket. Om nasjonal detaljert høydemodell (ndh), [Online]. Available at: https://www.kartverket.no/Prosjekter/Nasjonal-detaljert-hoydemodell/om-nasjonal-detaljert-hoydemodell/, visited on: 22/04/2020.

[41] K. Hofstad. (23rd Nov. 2018). Breddegrad i store norske leksikon på snl.no, [Online]. Available at: https://snl.no/breddegrad, visited on: 22/05/2020.

[42] L. Mæhlum. (11th Mar. 2020). Utm i store norske leksikon på snl.no, [Online]. Available at: https://snl.no/UTM, visited on: 22/05/2020.

[43] Natural Navigator. (4th Dec. 2018). Great circle, [Online]. Available at: https://www.naturalnavigator.com/news/2018/12/what-is-a-great-circle/, visited on: 26/04/2020.

[44] *Attribution-sharealike 4.0 international (cc by-sa 4.0)*, https://creativecommons.org/licenses/by-sa/4.0/, Apr. 2020. , visited on: 30/04/2020.

[45] S. Cox. (10th Aug. 2019). Understanding focus in photography, [Online]. Available at: https://photographylife.com/understanding-focus-in-photography, visited on: 02/05/2020.

[46] M. Golowczynski. (23rd Jun. 2016). Digital camera sensors explained, [Online]. Available at: https://www.whatdigitalcamera.com/technical-guides/technology-guides/sensors-explained-11457, visited on: 02/05/2020.

[47] D. Carr. (Jun. 2017). Angle of view vs. field of view. is there a difference and does it even matter?, [Online]. Available at: https://shuttermuse.com/angle-of-view-vs-field-of-view-fov-aov/, visited on: 02/05/2020.

[48] S. Crisp. (Mar. 2013). Camera sensor size: Why does it matter and exactly how big are they?, [Online]. Available at: https://newatlas.com/camera-sensor-size-guide/26684/, visited on: 02/05/2020.

[49] H. L. Andersen, 'Path planning for search and rescue mission using multicopters', Master's thesis, NTNU, Institutt for teknisk kybernetikk, 2014.

[50] M. Coombes, W.-H. Chen and C. Liu, 'Boustrophedon coverage path planning for uav aerial surveys in wind', in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2017, pp. 1563–1571.

[51] C. Di Franco and G. Buttazzo, 'Coverage path planning for uavs photogrammetry with energy and resolution constraints', *Journal of Intelligent & Robotic Systems*, vol. 83, no. 3-4, pp. 445–462, 2016.

[52] Y.-S. Jiao, X.-M. Wang, H. Chen and Y. Li, 'Research on the coverage path planning of uavs for polygon areas', in *2010 5th IEEE Conference on Industrial Electronics and Applications*, IEEE, 2010, pp. 1467–1472.

[53] Basics of greedy algorithms, [Online]. Available at: `https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/` , visited on: 20/04/2020.

[54] S. B. Tor and A. E. Middleditch, 'Convex decomposition of simple polygons', *ACM Transactions on Graphics (TOG)*, vol. 3, no. 4, pp. 244–265, 1984.

[55] J.-M. Lien and N. M. Amato, 'Approximate convex decomposition of polygons', in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 17–26.

[56] J. Li, W. Wang and E. Wu, 'Point-in-polygon tests by convex decomposition', *Computers & Graphics*, vol. 31, no. 4, pp. 636–648, 2007.

[57] J. Linnestad and O. H. Ødegaard. (2020). Sheepio, [Online]. Available at: `https://github.com/ohodegaa/Speepio`.

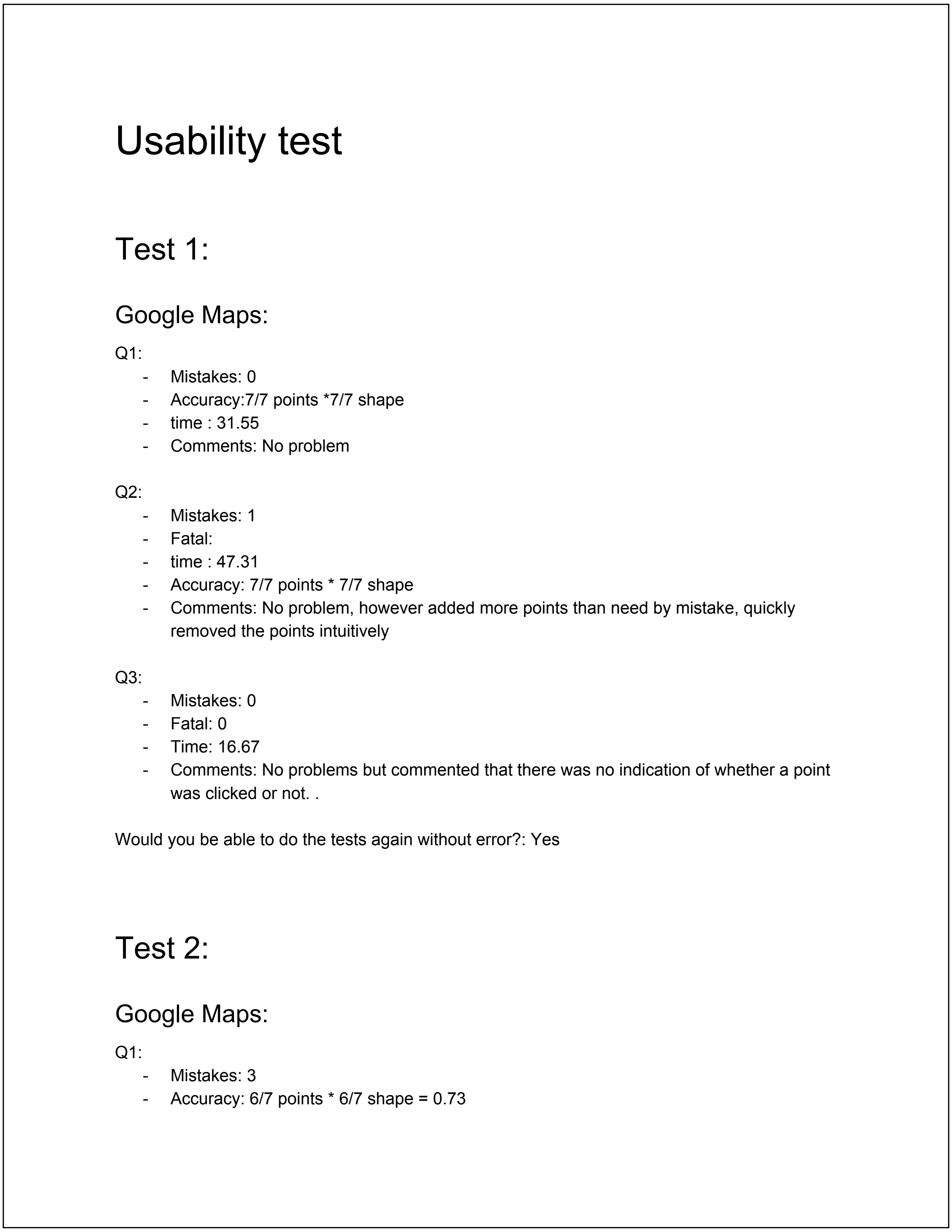
[58] J. Linnestad and O. H. Ødegaard. (2020). Sheepfinder-backend, [Online]. Available at: `https://github.com/jowies/SheepFinder-backend`.

[59] Nortura. (20th May 2020). Prisløyper 2. halvår 2020, [Online]. Available at: `https://medlem.nortura.no/smafe/priser-vilkar/prisloyper-2-halvar-2020-article43964-11787.html` , visited on: 29/05/2020.

Appendices

Appendix A

Raw Test Results from Usability Test



Usability test

Test 1:

Google Maps:

Q1:
- Mistakes: 0
- Accuracy:7/7 points *7/7 shape
- time : 31.55
- Comments: No problem

Q2:
- Mistakes: 1
- Fatal:
- time : 47.31
- Accuracy: 7/7 points * 7/7 shape
- Comments: No problem, however added more points than need by mistake, quickly removed the points intuitively

Q3:
- Mistakes: 0
- Fatal: 0
- Time: 16.67
- Comments: No problems but commented that there was no indication of whether a point was clicked or not. .

Would you be able to do the tests again without error?: Yes

Test 2:

Google Maps:

Q1:
- Mistakes: 3
- Accuracy: 6/7 points * 6/7 shape = 0.73

- time : 1:00.60
- Comments: No problem

Q2:
- Mistakes: 6
- Fatal: 1
- time : 2:13.16
- Accuracy: 6/7 points * 4/7 shape = 0.47
- Comments: Did not understand that you need to hold and drag to move points. Started making several points.

Q3:
- Mistakes: 3
- Fatal: 0
- Time: 1:45.13
- Comments: Did not understand that you need to click and then click delete button. Said that there was no indication that a point was clicked.

Would you be able to do the tests again without error?: Yes

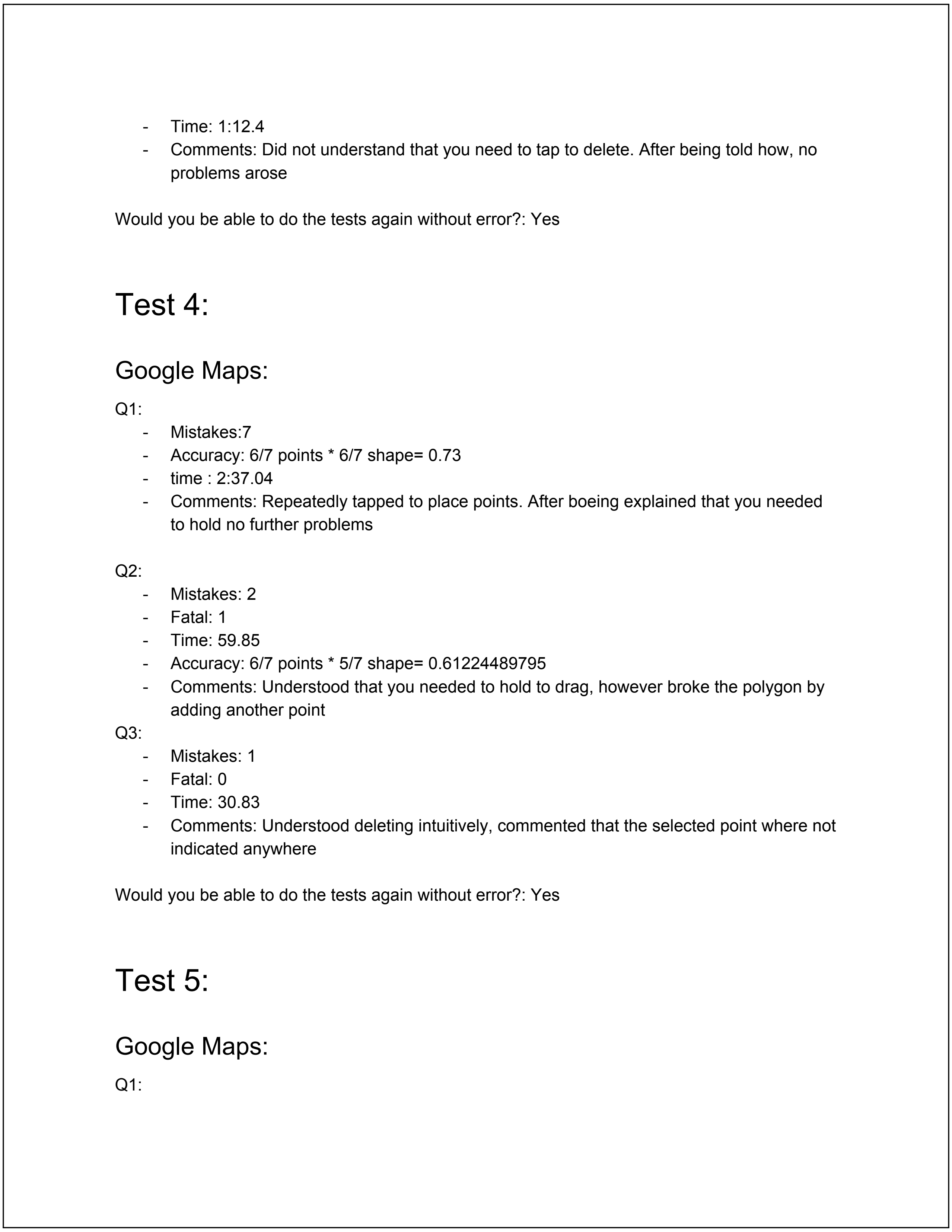
Test 3:

Google Maps:

Q1:
- Mistakes: 4
- Accuracy: 6/7 points * 6/7 shape= 0.73
- time : 1:55.37
- Comments: Lots of problems when trying to place points, dragging and and tapping.

Q2:
- Mistakes: 3
- Fatal: 1
- Time: 1:59.06
- Accuracy: 7/7 points * 6/7 shape= 0.86
- Comments:Understood that there missed a point and added this, added ekstra point by mistake and created a broken polygon. Understood that you need to hold and drag after a while without help.

Q3:
- Mistakes: 3
- Fatal: 0



- Time: 1:12.4
- Comments: Did not understand that you need to tap to delete. After being told how, no problems arose

Would you be able to do the tests again without error?: Yes

Test 4:

Google Maps:

Q1:
- Mistakes:7
- Accuracy: 6/7 points * 6/7 shape= 0.73
- time : 2:37.04
- Comments: Repeatedly tapped to place points. After boeing explained that you needed to hold no further problems

Q2:
- Mistakes: 2
- Fatal: 1
- Time: 59.85
- Accuracy: 6/7 points * 5/7 shape= 0.61224489795
- Comments: Understood that you needed to hold to drag, however broke the polygon by adding another point

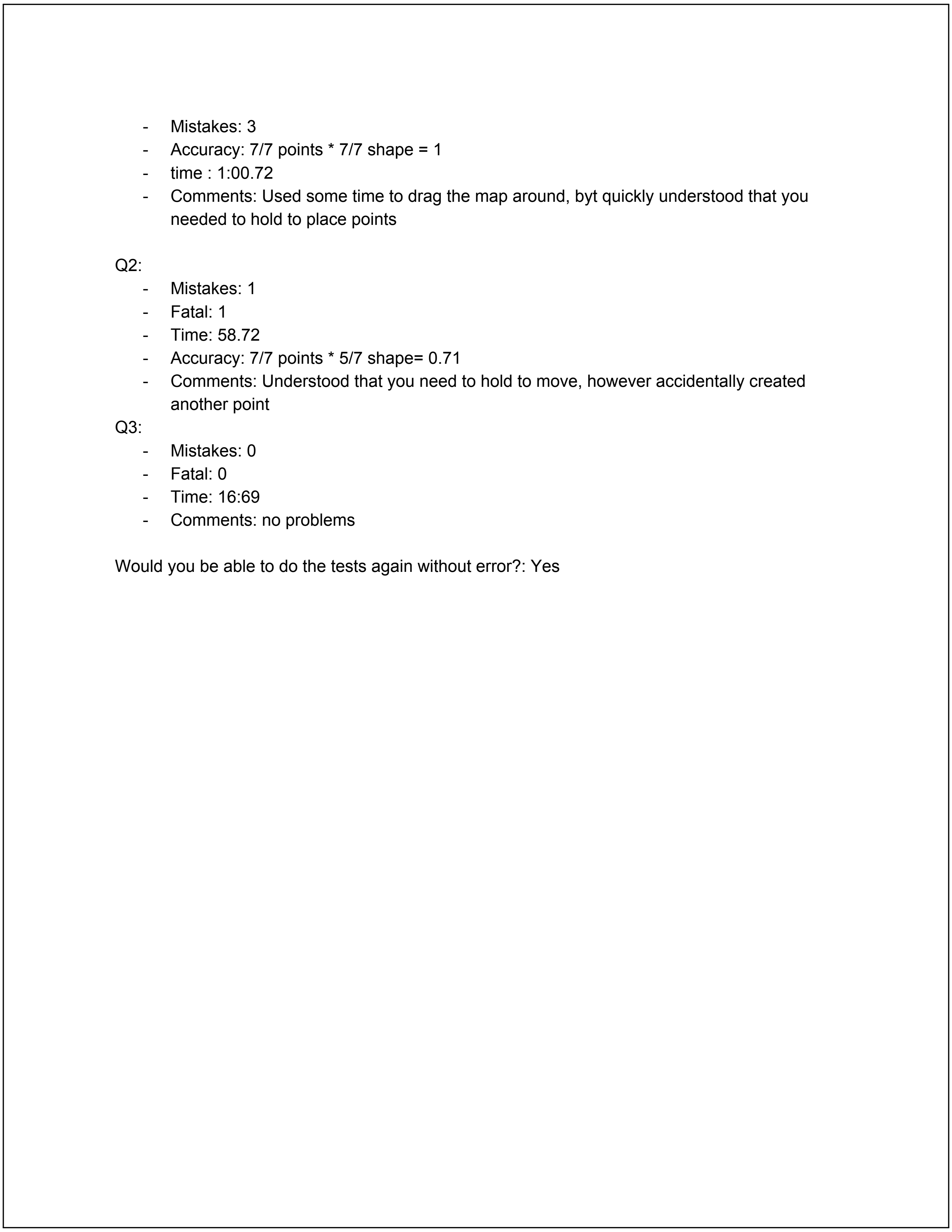
Q3:
- Mistakes: 1
- Fatal: 0
- Time: 30.83
- Comments: Understood deleting intuitively, commented that the selected point where not indicated anywhere

Would you be able to do the tests again without error?: Yes

Test 5:

Google Maps:

Q1:



- Mistakes: 3
- Accuracy: 7/7 points * 7/7 shape = 1
- time : 1:00.72
- Comments: Used some time to drag the map around, byt quickly understood that you needed to hold to place points

Q2:
- Mistakes: 1
- Fatal: 1
- Time: 58.72
- Accuracy: 7/7 points * 5/7 shape= 0.71
- Comments: Understood that you need to hold to move, however accidentally created another point

Q3:
- Mistakes: 0
- Fatal: 0
- Time: 16:69
- Comments: no problems

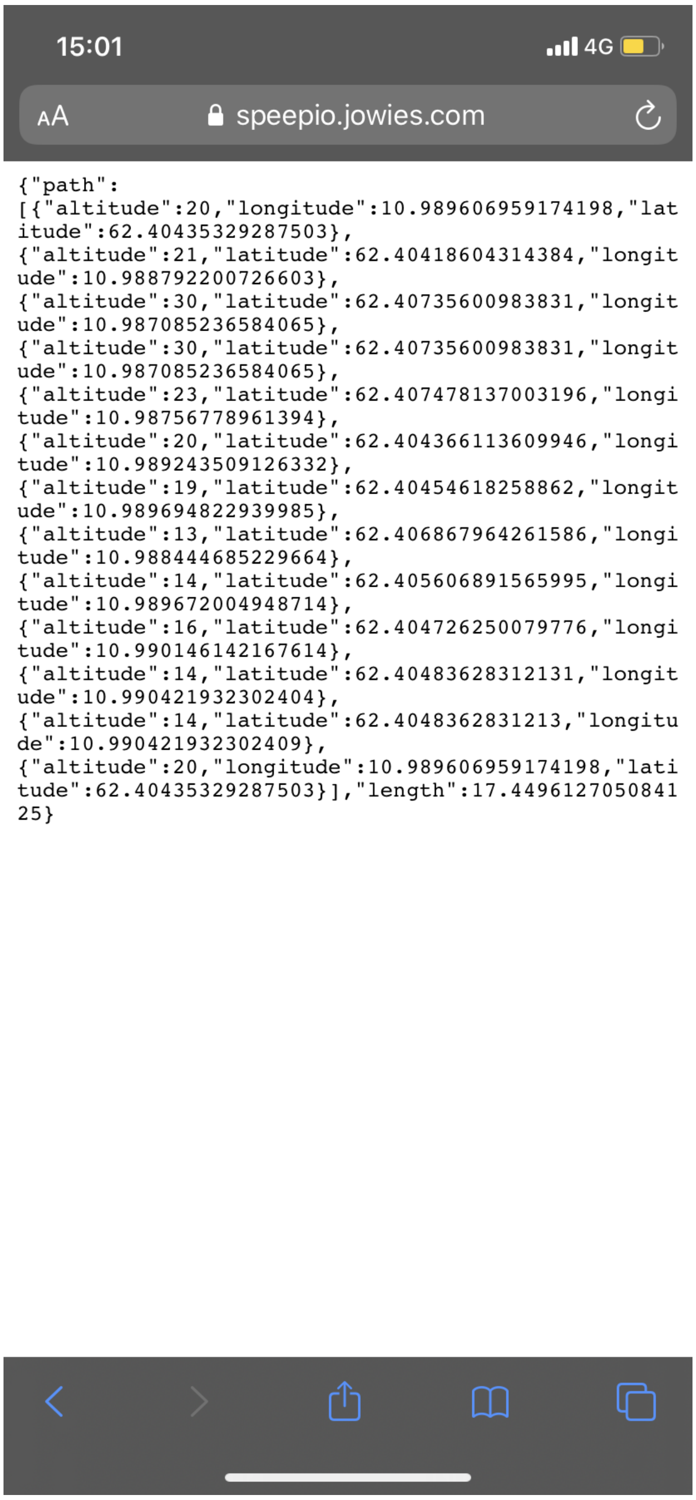
Would you be able to do the tests again without error?: Yes

Appendix B

Full-Scale Flight Tests

```
{"path":[{"altitude":40,"longitude":10.989615900418302,"latitude":62.40432879916252},
{"altitude":51,"latitude":62.403463620703405,"longitude":10.987133887660283},
{"altitude":58,"latitude":62.40274595652359,"longitude":10.98507526683359},
{"altitude":58,"latitude":62.40274595652359,"longitude":10.98507526683359},
{"altitude":68,"latitude":62.404526715424154,"longitude":10.984529087203653},
{"altitude":79,"latitude":62.406307472187045,"longitude":10.983982842615804},
{"altitude":82,"latitude":62.40735961669884,"longitude":10.983660067951503},
{"altitude":82,"latitude":62.40735961669884,"longitude":10.983660067951503},
{"altitude":82,"latitude":62.40736288303237,"longitude":10.983767513731216},
{"altitude":71,"latitude":62.40504799648664,"longitude":10.98447912636995},
{"altitude":60,"latitude":62.40308924349902,"longitude":10.985081174218383},
{"altitude":58,"latitude":62.40274527791174,"longitude":10.985186888241467},
{"altitude":55,"latitude":62.40273872830346,"longitude":10.98626344468892},
{"altitude":66,"latitude":62.40469748638678,"longitude":10.9856614811505},
{"altitude":77,"latitude":62.40594396745148,"longitude":10.985278372443457},
{"altitude":78,"latitude":62.40739524716124,"longitude":10.984832277827469},
{"altitude":66,"latitude":62.40741879800219,"longitude":10.985607258960915},
{"altitude":64,"latitude":62.40742760314449,"longitude":10.985897044075784},
{"altitude":53,"latitude":62.40440042890305,"longitude":10.986827379270458},
{"altitude":46,"latitude":62.40273217037833,"longitude":10.987340000515186},
{"altitude":44,"latitude":62.4027256041364,"longitude":10.988416555719589},
{"altitude":50,"latitude":62.4074599509821,"longitude":10.986961812475517},
{"altitude":40,"latitude":62.40748348997224,"longitude":10.987736796961642},
{"altitude":36,"latitude":62.407492290674014,"longitude":10.988026583026013},
{"altitude":36,"latitude":62.4027190295776,"longitude":10.989493110301447},
{"altitude":40,"longitude":10.989615900418302,"latitude":62.40432879916252}],
"length":34.89922541016825}
```

Code listing B.1: The path in raw JSON-format of the 40 meter altitude test.



```
{"path":
[{"altitude":20,"longitude":10.989606959174198,"lat
itude":62.4043532928750З},
{"altitude":21,"latitude":62.40418604314384,"longit
ude":10.988792200726603},
{"altitude":30,"latitude":62.40735600983831,"longit
ude":10.987085236584065},
{"altitude":30,"latitude":62.40735600983831,"longit
ude":10.987085236584065},
{"altitude":23,"latitude":62.407478137003196,"longi
tude":10.98756778961394},
{"altitude":20,"latitude":62.404366113609946,"longi
tude":10.989243509126332},
{"altitude":19,"latitude":62.40454618258862,"longit
ude":10.989694822939985},
{"altitude":13,"latitude":62.406867964261586,"longi
tude":10.988444685229664},
{"altitude":14,"latitude":62.405606891565995,"longi
tude":10.989672004948714},
{"altitude":16,"latitude":62.404726250079776,"longi
tude":10.990146142167614},
{"altitude":14,"latitude":62.40483628312131,"longit
ude":10.990421932302404},
{"altitude":14,"latitude":62.4048362831213,"longitu
de":10.990421932302409},
{"altitude":20,"longitude":10.989606959174198,"lati
tude":62.40435329287503}],"length":17.4496127050841
25}
```

Figure B.1: The path in raw JSON-format of the 20 meter altitude test

J. S. Linnestad & O. H. K. Ødegaard

Utilizing Drones to Automatically Cover Outfield Pastures

NTNU
Norwegian University of
Science and Technology