CHALMERS

Automated fence surveillance by use of drones

Utilizing the Potential of Drones for Fence Monitoring: Trajectory Planning, Data Collection, and Advanced Analysis Techniques

Bachelor's thesis in Division of Systems and Control

MARCUS BERG
JOAKIM CEWERS BREDBERG
ALBIN FALSEN LINDQVIST
JOHANNES JOHANSSON
CARL KRONQVIST
MALTE OLSSON

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Bachelor's thesis 2023

Automated fence surveillance by use of drones

Utilizing the Potential of Drones for Fence Monitoring: Trajectory Planning, Data Collection, and Advanced Analysis Techniques

MARCUS BERG
JOAKIM CEWERS BREDBERG
ALBIN FALSEN LINDQVIST
JOHANNES JOHANSSON
CARL KRONQVIST
MALTE OLSSON

CHALMERS

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2023

Automated fence surveillance by use of drones
Utilizing the Potential of Drones for Fence Monitoring: Trajectory Planning, Data Collection, and Advanced Analysis Techniques
MARCUS BERG
JOAKIM CEWERS BREDBERG
ALBIN FALSEN LINDQVIST
JOHANNES JOHANSSON
CARL KRONQVIST
MALTE OLSSON

Supervisor: Knut Åkesson, Department of Electrical Engineering
Examiner: Martin Fabian, Department of Electrical Engineering

Bachelor's Thesis 2023
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A drone that is surveilling a perimeter fence at the AstaZero test track.

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

The widespread availability and affordability of UAVs (Unmanned Aerial Vehicles) have opened up new possibilities for their utilization, enabling the development of novel applications. This paper discusses how to use a commercial UAV from the manufacturer DJI to autonomously collect footage of a perimeter fence. The collection phase first involves careful planning of a trajectory. Furthermore, it requires advanced software for controlling the vehicle. This footage is then analyzed using a neural network or a line detection algorithm to detect abnormalities such as fallen trees or holes. This research concludes that data collection using a drone is feasible and is a comparably uncomplicated problem to solve. However, the analysis step is more involved and will require more work to be fully operational. The project shows potential for providing value for facilities that requires a high level of fence integrity.

Sammanfattning

Drönare har med tiden blivit billigare och med ett bredare utbud bidragit till en rad nya användningsområden och tillämpningar. Denna rapport går igenom hur en drönare från tillverkaren DJI använts för att autonomt samla in filmsekvenser av ett staket. Insamlingsfasen innefattar först noggran planering av en flygsträcka runt staketet. Denna rutt matas sedan in i en kontrollprogramvara som autonomt kontrollerar drönaren så att den besöker alla punkter på staketet. Dessa filmsekvenser används sedan i en linjedetekteringsalgoritm samt ett neuralt nätverk för att upptäcka avvikelser såsom nedfallna träd eller hål. Rapporten drar slutsatsen att ruttplanering och datainsamling med drönaren är genomförbart. Däremot är analysen av filmsekvenser desto mer avancerad och kräver mer arbete för att bli helt fungerande. För närvarande befinner sig varken linjedetekteringsalgoritmen eller det neurala nätverket i en tillräckligt färdigutvecklad form för att vara fullt användbara. Trots det har det neurala nätverket visat en betydande potential, vilket gör det till en lovande teknik som kan vidareutvecklas och ge stort värde i framtiden.

Keywords: UAV, drone, line detection, neural network.

Preface

This report is a bachelor thesis at Chalmers University of Technology in the Department of Electrical Engineering in the Division of Systems and Control. It is written by six Swedish students, three studying Automaiton and Mechatronics Engineering, and three studying Mechanical Engineering. During the spring of 2023, from January to May, the Swedish team writing this thesis collaborated with a group of five American students at Penn State University, State College, Pennsylvania.

Together the students worked on the problem statement given by AstaZero AB to help the company automate its fence surveillance. The collaboration between the teams worked through sharing ideas, information, code, and solutions. Weekly meetings were arranged between the teams and supervisors to progress check the work. The students arranged additional meetings as seen as necessary. From the collaboration, we have learned to work across several time zones and the importance of communicating and structuring the work.

The source code for all of the code produced in this project is available in the project's organization at: `https://github.com/EENX16-23-24`. However, some of the code is private and access might be granted by contacting one of the authors of the paper.

Acknowledgements

We would like to thank our supervisor, Knut Åkesson, at the Department of Electrical Engineering, for supervising the project and contributing to discussions that led our work forward. We also want to thank our contact person and supervisor, Alfred Aronsson from AstaZero AB, with whom we have been able to brainstorm ideas, who helped us in the start-up of the project, and who provided drones and premises where we have been able to develop and test. We also want to thank Mikael Enelund for helping with the contact between our team and the American students with whom we have collaborated on the project work. We would also like to thank Ze Zhang, for helping the team get started with work on the Chalmers PC-cluster Vera and answering questions regarding computer vision.

A special thanks to Will Kraus, Matt Denelsbeck, Mason Elliott, Andrew Blickensderfer, and Kabir Bhatia, the American students at Penn State University that we collaborated with during the project, and for their hospitality during our visit to Penn State University.

Marcus Berg, Joakim Cewers Bredberg, Albin Falsen Lindqvist
Johannes Johansson, Carl Kronqvist, Malte Olsson
Gothenburg, May 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ATOS	Automated vehicle Test Operating System
API	Application Programming Interface
CRM	Central Resource Manager
DSS	Drone Safety System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GLONASS	Globalnaja navigatsionnaja sputnikovaja sistema
LLA	Latitude Longitude Altitude
RISE	Research Institutes of Sweden
SDK	Software Development Kit
UAV	Unmanned aerial vehicle

Contents

1

Introduction

Test facilities for autonomous vehicle testing are one component in achieving safety and reliability in future transport systems [1]. Having autonomous vehicles on the road places strict demands on safety to avoid accidents. Developing technology that alerts and avoids safety risks helps prevent fatal accidents [2]. AstaZero is a research company owned by the Research Institutes of Sweden AB (RISE). They provide a testing facility and conduct research to test safety systems in traffic environments [3]. The test track is located in the countryside outside Borås, Sweden, and has a 5.7-kilometer rural road and road network to test different traffic scenarios. The area is enclosed by a chain-link fence to guarantee the safety of the tests, preventing unauthorized individuals and animals from accessing the premises. Any breach in the fence compromises the integrity of the track, rendering the tests inoperable until the area is secured. The presence of animals within the area poses a security risk, and rectifying the situation can be time-consuming, leading to extended shutdowns [4]. Maintaining fence integrity is a necessity.

Today, the perimeter fence is inspected by staff who walk along it to ensure its integrity. However, this task is performed infrequently due to the time-consuming nature of the process. The company aims to automate and streamline this procedure for improved efficiency by implementing flying drones to monitor the fence [4]. Drones, also known as UAVs (Unmanned Aerial Vehicles), have become more affordable and accessible due to advancements in electrical components, resulting in reduced costs and sizes. This has made drones a cost-effective solution for a wide range of tasks [5]. The ability of drones to capture images when equipped with a camera allows them to be used for data gathering and surveillance purposes. The purpose of this research is to develop and automate a system that can utilize a company-provided drone to monitor the perimeter fence around the test track. The goal is to demonstrate the feasibility of using an off-the-shelf drone for autonomous surveillance of a perimeter fence. The collected footage from the drone will be processed in a manner that allows for manual inspection of any abnormalities detected.

Previous research in fence monitoring has predominantly focused on fence surveillance and detecting movement around the fence to prevent intrusions [6]. Additionally, there has been research conducted on utilizing convolutional neural networks for fence detection in images [7]. This project aims to bridge the gap by developing a system that combines surveillance, monitoring, and fence detection using drones.

1.1 Objective

The goal of the project is to investigate whether it is feasible to use a commercial drone to perform autonomous surveillance of a perimeter fence. The solution should, with the use of artificial intelligence, analyze footage of the fence and determine whether or not the fence has been breached.

1.2 Demarcations

The project is limited by the following specifications from the client, AstaZero:

- Choice of drone, *DJI Mavic Enterprise 2 Zoom*. The drone was chosen by the client as it has full support for the DJI MobileSDK [8] which means that all features of the drone can be programmatically controlled.
- The solution should be built only for AstaZero's fence. Mainly because of the need to collect coordinates and camera angles for the entire fence.
- Due to legal reasons a following car will be needed at all times when operating the drone. The following car needs to be in line of sight from the drone at all times.
- The project will not consist of building any hardware, the drone will be used as is. Only software will be focused on.

2
Approach

The project has been approached by splitting the objective into three primary sub-problems: Drone Path Planning, Drone Control, and Fence Detection. Each sub-problem has been responsible for its respective task; generating fence coordinates and flight paths, collecting fence footage, or detecting breaches in the fence autonomously. The three sub-problems and their respective inputs and outputs are illustrated in Figure 2.1.

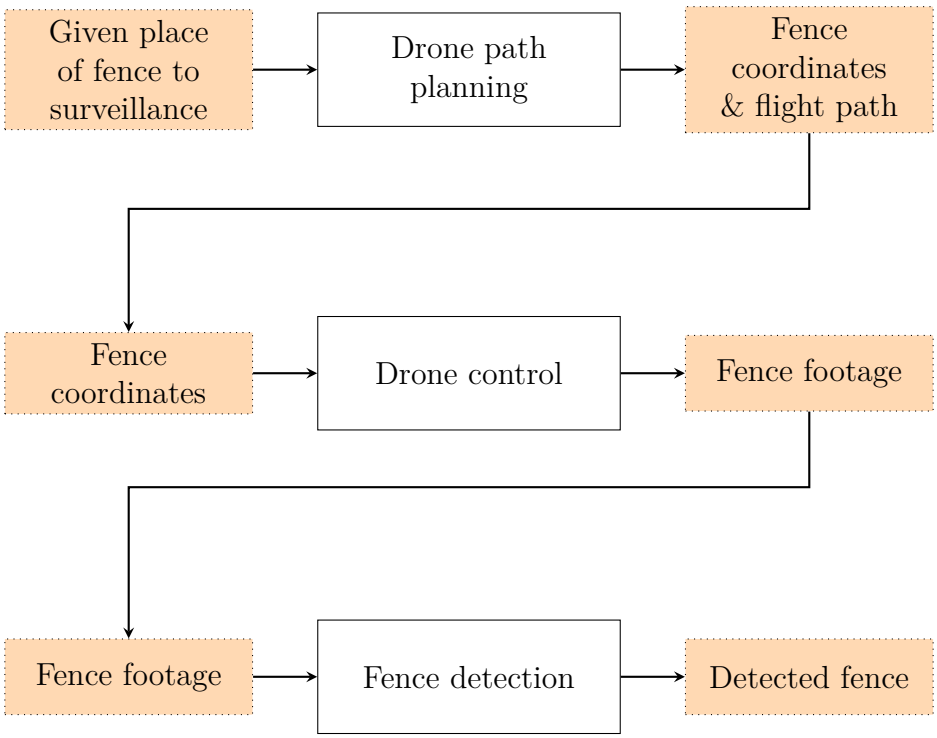

Figure 2.1: Illustrating the division of the problem and how they are connected to each other.

Three sub-teams, each consisting of two Chalmers students, were assigned one sub-problem and were accompanied by students from Penn State University, USA. The advantage of this approach is that it enables parallel workflows. However, the challenge with such a heavy division of work is that the integration, when all developments by each team come together, can be out of sync and require a vast amount of additional work. To address this challenge, regular testing was conducted to continuously verify and validate the interfaces. Despite the high functionality of the interfaces, certain solutions within a sub-problem may have an impact on

the effectiveness of other sub-problems. This necessitates effective communication and general solutions. Striving to create as general yet functional solutions as possible is key to making the whole concept dynamic and robust towards unexpected occurrences and outcomes.

This report is structured in accordance with the approach, with each area having its own sections for methods, results, discussions, and conclusions. Thereafter, joint subjects that affect more than one sub-problem are discussed in Chapter 6 such as interfaces, testing approaches, ethical aspects of the project and future work. Lastly, conclusions are drawn in Chapter 7 based on the outcome of the project.

3
Drone Path planning

Drone Path Planning is the sub-problem that transitions all knowledge about the environment into waypoints that subsequently are utilized by the Drone Control, described in Chapter 4. The knowledge or input parameters can be divided into static and dynamic parameters. Static parameters include the fence's coordinates, the altitude and possible charging stations. It is safe to assume that such data will not change over time. Dynamic data on the other hand constitutes parameters such as surrounding temperature, wind speed and precipitation that have to be considered for every flight. Figure 3.1 illustrates the problem which fits into the larger approach shown in Chapter 2.

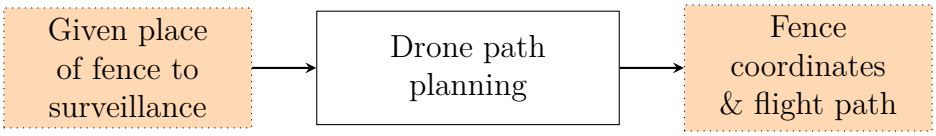

Figure 3.1: The drone path problem is defined as taking the given position of fence to surveil as input and fence coordinates and flight path as output.

Later in Chapter 4 the concept of waypoint missions is introduced which essentially provides the framework of the information needed to fly the drone and its format. The required information in a waypoint mission consisting of one point is:

- Coordinates for the waypoint, including longitude, latitude and altitude
- Speed at which the drone should travel
- Heading of the drone

But again, to determine these parameters several other aspects of the problem need to be reviewed for the route to fulfill the mission safely and efficiently, and in essence, this captures the purpose of Drone Path Planning.

3.1 Method

This section describes the methods for how all parameters are processed and forged together into an algorithm at the end which generates the output; a series of waypoint missions. The section also includes some explanatory information about Global Navigation Satellite Systems which play an important role in this project as well as methods for how to collect coordinates from the internet.

3.1.1 GNSS

An extensive part of this project revolves around positioning and navigation. GNSS, Global Navigation Satellite System, is a general term used to describe satellite-based navigation systems that provide geospatial and timing data. GPS is by far the most utilized GNSS but there are other systems such as GLONASS which is the Russian counterpart [9]. The drone used in this project, Mavic 2 Enterprise Series, support both GPS and GLONASS [8]. However, all implementations have been done with the use of GPS as a reference.

Today the GPS is operated by the U.S. Space Force and is free to use for anyone with a GPS receiver. The accuracy has increased over the years but can still be heavily disturbed. Factors that affect the precision are the satellite's geometries, the potential blockage of signal, atmospheric conditions, space weather and other technical features of the receiving party [10]. When the signal from the satellite travel through the earth's ionosphere towards its receiver, the plasma bends its trajectory. In regular and calm conditions the GPS can calculate this effect and compensate for the disturbance, and deliver an average accuracy of about 1 meter. However, when the ionosphere suffers from heavy space weather, this accuracy can decrease to tens of meters or even more [11].

According to the DJI documentation [8], the drone used in this project has a vertical accuracy of 0.5 meters and 1.5 meters horizontally.

Calculating an approximation of the accuracy of the GPS signal and taking this into account during drone operation is outside the scope of this project. However, it is important to understand that the accuracy varies, which needs to be considered when planing the route and its safety margins.

3.1.2 Collecting coordinates

The collection of coordinates constitutes a crucial element of the project. Given the extensive length of the fence, which spans approximately 10 kilometers, and the vast testing needed to be conducted, a swift and effortless method of acquiring the drone path coordinates is paramount. Following a brief examination of coordinate collection programs, it was determined that Google Earth was the optimal selection due to its broad range of route planning functionalities. Notably, the precision of the collected coordinates with respect to real-world values is of utmost significance. Evaluation of the drone's positioning against the compiled coordinates demonstrated a marginal discrepancy of only 2 meters.

With the "draw route" function in Google Earth [12], it is possible to draw paths by simply clicking on points on the map. These points then form a path, which is showed in Figure 3.2 with a yellow line.

Figure 3.2: Draw function in Google Earth from [12].

After having created the intended route, the file can be downloaded as a KML-file. Subsequently, the data in this KML-file needs to be converted into the desired format of the waypoint mission, as earlier mentioned. To manage this in an efficient way, a parser was programmed in Python which is presented in Chapter 3.2.2.

3.1.3 Dynamic parameters

The dynamic parameters are such that continuously affect the drone's endurance or capability to operate. In this section a few of the affecting parameters are selected for consideration based on their importance:

- Battery capacity
- Surrounding temperature
- Wind speed
- Precipitation

The drone is equipped with a lithium polymer battery, abbreviated as LiPo. These batteries possess a complex structure comprising both chemical and electrical components, making it difficult to gauge the effect of various parameters on the drone's battery. To estimate these values, research and testing have been conducted.

Battery capacity and state of health
The drone's LiPo battery has a capacity of 3850 mAh. According to DJI's website [8], this translates to a flight time of 32 minutes, assuming a speed of 25 kilometres per hour with no wind at sea level. Prior to each flight, the state of the battery's health will be taken into account, as its capacity progressively deteriorates over time. In the DJI application, a function is included that shows the number of times the battery has been charged, which is necessary to calculate the capacity. In [13], a test with eleven different LiPo batteries from mobile phones was conducted. All batteries showed a capacity of 88-94% new. After 250 full cycles, the capacity was down to 73-84%.

Surrounding temperature

The battery's capacity is also affected by the surrounding temperature. From DJIs specification[8] the operating temperature range's from -10°C to 40°C. According to [14], the voltage drop was investigated for different temperatures. The curve in Figure 3.3 shows a preheating phase in the beginning for the lower temperatures. This is because the battery has an optimal temperature for operating. This results in a shorter discharge period in lower temperatures.

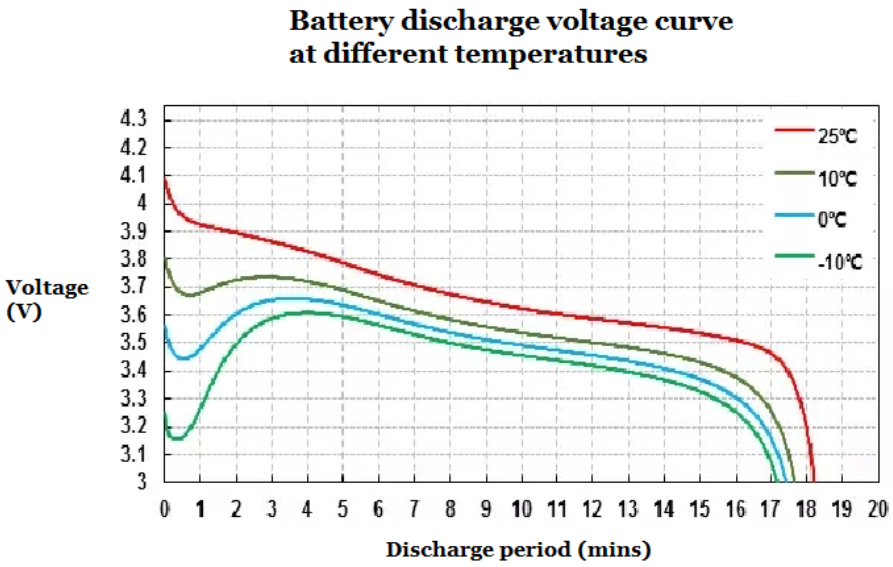

Figure 3.3: Discharge voltage curve at different temperatures. From [14].

In order to evaluate the surrounding temperature's effect on the battery, three tests were completed. For all tests, the drone starts up on the ground and immediately rises up to the reference position, which is two meters above the ground and obviously with a fully charged battery. Then the time is measured for how long it takes to reach 80 % capacity, which is displayed on the controller. The results of these tests are shown in section 3.2.1.

Wind speed

According to the drone's specification[8], the maximal wind speed to operate in is 8-10,5 m/s. The wind speed does not only affect the battery capacity but also the video quality. Therefore, a strict limit of 8 m/s is set to guarantee both good enough endurance and video quality.

Precipitation
According to DJI's guidelines [8], it is highly recommended to refrain from flying the drone during precipitation due to the potential risk of damaging the drone's sensitive electronic components. No operation should therefore be conducted during precipitation.

3.1.4 Static parameters

In addition to the dynamic parameters, static parameters also form part of the final algorithm's input, which must be carefully considered in order to develop safe and efficient waypoint missions. Unlike dynamic parameters, static parameters remain static over time or undergo such a low rate of change that they can be neglected for the purposes of this project. During the development phase, these parameters were refined based on the information obtained through testing in the environment. The static parameters considered in this project are as follows:

- Fence Coordinates including altitude
- Possible charging positions
- Drone Speed

Fence Coordinates including altitude
The fence coordinates were collected with the use of Google Earth as explained in section 3.1.2. Google Earth also provide altitude for each coordinate at ground level. To stay clear of trees and power cables, the waypoints were set to 50 meters above ground level initially. The critical areas of the total route consist of tall pine trees which according to Sydved often grow to 20-30 meters tall [15]. With more testing and evaluation this can be decreased if needed to get a better image.

Possible charging positions
Even though it cannot be stated with absolute certainty that the drone requires a recharge, because of the uncertainty regarding the endurance, at least one charging stop was assumed necessary. Therefore possible charging positions adjacent to the fence were analyzed and reviewed. Along the fence, there is a rural road that constitutes a part of the test track and is situated at varying distances from the fence. This road is depicted as a red line in Figure 3.4, while the fence is represented by the yellow line. Later for implementation purposes, this continuous line was discretized into a set of points.

Figure 3.4: Possible charging positions for the drone marked in blue and fence position marked with yellow.

Drone Speed

It may seem strange to consider drone speed a static parameter but the purpose of this is to determine an operational speed which is used throughout the whole operation and minimizes the overall flight time. The flight speed has by far the greatest impact on the operation time and therefore needs to be explored and evaluated thoroughly. From the tests conducted it was realized that the video recorded was not affected negatively at high speeds when flying at a 50-meter altitude.

The aim for deciding the flight speed is for the operation to be as quick and less time-consuming as possible. However, in order to optimize the speed, there are two types of optimizations of interest.

1. If the number of fully charged batteries is great enough to be considered infinite, picking the highest possible speed within the drone's limitations gives the shortest time, eventuating no need for further optimization of calculations.
2. If the number of fully charged batteries is limited, considering that charging a fully discharged battery takes about one and a half hours, minimizing the number of recharges should give the shortest operational time.

The presented approach above represents a simplification of the problem at hand. In order to determine the actual optimal drone speed, an iterative process would be required, wherein several factors such as the path, charging points, charging time, and other parameters are taken into account.

With regard to the second approach for optimization, which involves minimizing energy consumption and subsequently reducing the frequency of recharges, a free-body diagram were generated, as illustrated in Figure 3.5. The diagram enables static calculations to be conducted on the drone. The aim of the calculations is to find a relation between the generated force of the drone and the horizontal speed.

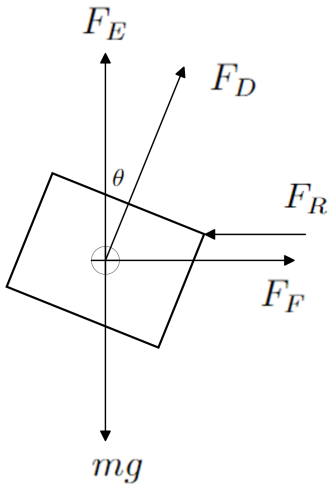

Figure 3.5: Free body diagram of tilted flight.

$$
\begin{aligned}
F_D &= Force\ of\ drone \\
F_F &= Forward\ force \\
F_E &= Elevation\ force \\
F_R &= Resisting\ force \\
mg &= Force\ of\ gravity \\
\theta &= Tilt\text{-}angle
\end{aligned}
$$

Equations for the static state, assuming no acceleration, can be derived from Figure 3.5.

$$F_d cos(\theta) - mg = 0 \tag{3.1}$$
$$F_d sin(\theta) - F_r = 0 \tag{3.2}$$

Further simplification of these equations yields the following expression:

$$\theta = arccos(\frac{mg}{F_D}) \implies \tag{3.3}$$
$$F_R = F_D sin(arccos(\frac{mg}{F_D})) \tag{3.4}$$

Equation 3.4 holds true under the condition that the generated force, F_D, represents the averaged force, without considering fluctuations in the control system. The equation also neglects the partial reduction in resisting force that the propellers on the drone contribute to when flying towards the airflow.

The energy consumption of the drone cannot be determined solely based on Equation 3.4, as it depends on various other parameters that are challenging to measure. For instance, the efficiency of the electrical motors, the energy demand of the control

system, and the energy consumption of other electrical components are significant factors that contribute to the overall energy usage. Despite this, Equation 3.4 allows for a relationship to be established between the force generated by the drone and the resisting force. The following steps involve attempting to establish a relation between the force generated by the drone and the horizontal flight speed with the use of approximating air resistance.

The relationship between force, airspeed and geometry can be approximated with the use of Equation 3.5.

$$C_D = \frac{F_D}{\frac{1}{2}\rho V^2 A_P} \tag{3.5}$$

(Equation 7.66b in [16])

The relation between the resisting force F_D and the velocity V is expressed by Equation 3.5. The equation states that the resisting force is proportional to the square of the velocity with some geometric constants that instead can be represented by an arbitrary coefficient K. Incorporating this relation into Equation 3.4, the following expression can be obtained:

$$K\dot{x}^2 = F_D sin(arccos(\frac{mg}{F_D})) \tag{3.6}$$

Where x is the horizontal distance travelled.

This drone-speed section has described the method for how Equation 3.6 was derived and that there is a relation between the force generated by the drone and the drone's horizontal speed. Again, even though the actual energy consumption cannot be decided only by the generated force, minimizing this force should result in minimal energy consumption.

3.1.5 Algorithm

Finally, both the static and dynamic parameters have to go into some sort of smart model that takes these as input and generate one or several waypoint missions which then are handled by the drone control.

This model is an algorithm depicted in Figure 3.6 which is divided into two modules; EnduranceCalculator and waypointMissionGenerator.

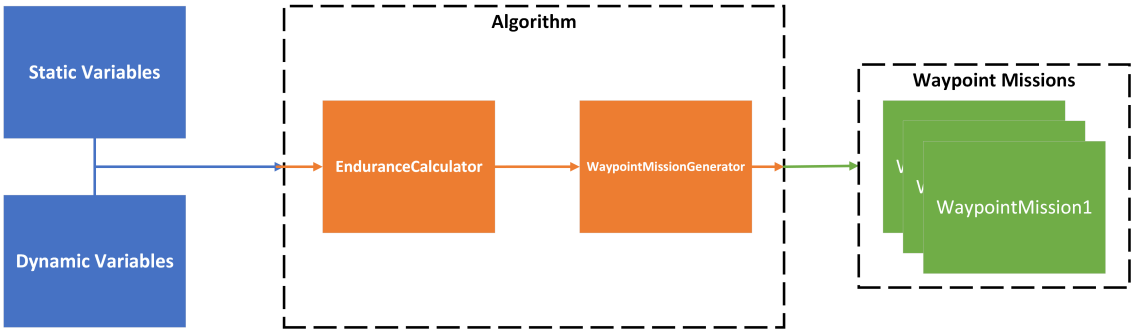

Figure 3.6: System representation of the algorithm.

The upcoming sections provide a detailed explanation of the two modules, including their inputs, algorithms, and outputs. Following this, a brief segment is included that further explains the algorithms, including their simplifications, challenges, and shortcomings.

3.1.5.1 EnduranceCalculator

The EnduranceCalculator task is to calculate the distance the drone can travel given a certain set of parameters.

Input: Battery Capacity, Battery State of Health, Surrounding Temperature, Wind Speed, Precipitation, Safety margin, Drone Speed.

Algorithm:

1. Are the dynamic parameters within the limits of operation? If yes: Proceed. If no: Postpone the operation and break the algorithm
2. Calculate the endurance with regard to the input parameters

Output: Endurance [m]

It is difficult to create, trim and evaluate the algorithm in the EnduranceCalculator without extensive testing in different environments with different batteries. The testing in this project has been more focused on drone control which is the most crucial part to get the drone operational.

The calculations performed in Section 3.1.4 are used to optimising the speed rather than to determine the energy consumption, leaving the EnduranceCalculator rather short-handed.

Given the challenges and uncertainties associated with endurance calculations, the decision was made to deprioritize further development of the EnduranceCalculator module in this project. Instead, the endurance is considered an arbitrary parameter that can be further researched in the future and eventually subject to implementation in an algorithm.

3.1.5.2 WaypointMissionGenerator

The WaypointmissionGenerator creates a series of waypoint missions based on the endurance of the drone.

Input: List of fence coordinates, Set of charging points (coordinates), Initial Position,(In the list of coordinates), Initial Distance, Chosen Charging Point, Endurance

Algorithm:

1. Calculate the i-last point to which the drone can travel with the added initial distance, without exceeding the endurance. With $i = 1$ Initially. If the endurance is not exceeded even with the last point in the list, go to step 5 and set chosen charging position to *none* and break the algorithm.
2. From this last point, calculate the distance to the nearest charging point in the set of charging points.
3. Add the distance from the last point to the nearest charging point to the calculated distance from the starting point to the last point including the initial distance.
4. If the summed distance exceeds the endurance, redo from step 1 with $i = i + 1$
5. Generate waypoint mission with the first point being the initial position, the points in the List of Fence Coordinates which didn't render exceedance. And the last point which is the chosen charging point which was the nearest.
6. Now the Initial Position becomes the last point according to step 1.
7. initial distance becomes the distance between the chosen charging point and the last Initial Position.
8. Redo from step 1.

Output: List of waypoint missions

Within the output files holding each waypoint mission, the drone speed is included as well. The algorithm described is a greedy algorithm that aims to maximize the distance travelled in each iteration. However, this approach does not guarantee that the chosen waypoints together form the shortest possible path. This can be seen in the results presented in Figure 3.11, where the algorithm with an endurance of 1000 meters was used. The algorithm has an important practical role in the project beyond just finding an efficient path. By taking the drone's endurance into account, the algorithm ensures that the generated path does not exceed the drone's operational capabilities. Additionally, the generated path is broken down into individual waypoint missions, which are easy for the drone control to work with and execute. This simplifies the process and helps to ensure that the drone successfully completes its mission.

3.2 Results

The results displayed in this section are divided into two subsections, Endurace Calculation and Waypoint Generation.

3.2.1 Endurance Calculation

The results of the tests described in Section 3.1.3 which evaluate the surrounding temperature's effect on endurance are shown in Table 3.1. The first test was made at the beginning of March when the temperature was below -5°C. And the two other tests were made at the beginning of May, at 10°C. and 20°C.

Table 3.1: Battery test results showing time in minutes for a fully charged battery to reach 80% in different temperatures.

Temperature:	-5°C	10°C	20°C
Time reaching 80%:	6:13	5:23	5:19

According to the table, increasing the temperature does not result in a longer-lasting battery capacity which was the researched theory. On the contrary, the results demonstrate that the battery capacity is contingent upon the number of discharge cycles. The initial test was carried out after the battery had undergone only 12 charging cycles, while in the subsequent two tests, it had undergone 41 respectively 42 discharge cycles.

Moving forward to Section 3.1.4, Equation 3.6 was derived which describes the relationship between the horizontal distance travelled, x, and the force generated by the drone. Describing energy consumption solely based on generated force is not feasible. However, assuming that a lower generated force corresponds to reduced energy consumption allows for the possibility of optimizing energy usage by adjusting the generated force concerning the travelled distance. In Figure 3.7, it can be observed that the contribution of force to horizontal speed gradually decreases over time. The plots show this relation with different geometrical coefficients, k.

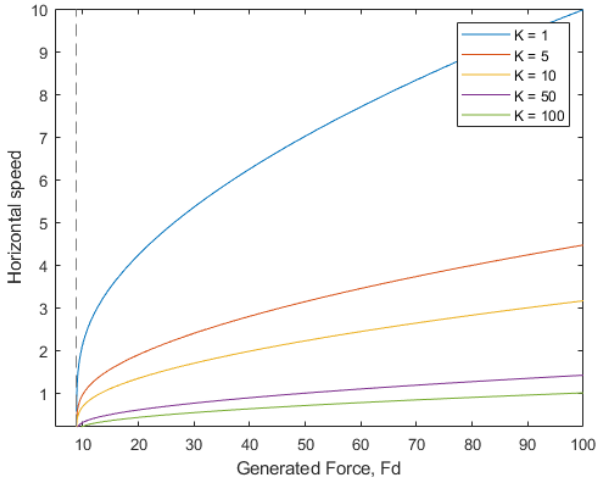

Figure 3.7: Horizontal speed as a function of generated force.

Determining k based on drone specifications and measurements is complicated and requires deep knowledge of fluid mechanics and CFD simulations. Without a rather

accurate approximation of k, the optimization falls short and was therefore not further investigated in this project.

In summary, the effects of temperature, the battery's state of health and the drone's speed were researched. All these parameters can make out some sort of hypothesis for the endurance of the drone, but to really create an accurate model, further testing would be required to evaluate the hypothesis.

Given the current circumstances, it would be most appropriate to follow DJI's recommended endurance. Maximal flight time with no wind and at 25 kilometres per hour is 31 minutes. This converts to approximately 12917 meters. In addition from the tests, it was realized that the drone needs to stop and readjust its course at every waypoint, contributing to added flight time and energy loss. Since the first draft of the path consists of 142 waypoints, this behaviour has a significant impact on the overall operational performance.

3.2.2 Waypoint Generation

In Chapter 3.1.2 the process of collecting coordinates with the use of Google Earth was explained. The exported data is a KML-file including the coordinates and elevation. These coordinates are arranged in sequential order, with the first point listed first and the last point listed last. Every point also has a special order of data which are longitude, latitude, and elevation. In Figure 3.8 an example of a section from a KML-file is shown. The last lines show how the coordinates are stored.

```
1     <LineString>
2         <coordinates>
3             11.98252673283269,57.68147672668603,53.36206484718642
4         </coordinates>
5     </LineString>
6
```

Figure 3.8: An example of a section from a KML-file.

For this type of file, a parsing algorithm was written called KML2CSV, shown in Figure 3.9. Firstly it imports the KML-file, opens it and an empty CSV-file for writing. Secondly, it identifies the line that does not start with '<' since the coordinates line is the only one that does not starts with this symbol. Thirdly it sorts the coordinates in the right order i.e. lat, long, alt. The last column is written "course" because the heading is pointing towards the next coordinate. More about drone heading is discussed in Chapter 4.2.4. Lastly, it writes down all the data in a CSV-file as shown in Table 3.2.

```
def KML2CSV(filename):
    with open(filename, 'r') as fin:
        with open('helavarvet.csv', 'w', newline='') as fout:
            writer = csv.writer(fout)
            l = [['lat', 'long', 'alt', 'heading']]
            for line in fin:
                new_line = line.strip()
                if new_line[0] != '<':
                    coords = line.split(' ')
                    coords[0] = coords[0].strip('\n\t')
                    for set in coords[:-1]:
                        var = set.split(',') + [str(0)]
                        # change order
                        var[0], var[1] = var[1], var[0]
                        # Decide altitude
                        var[2] = float(var[2]) + 20
                        # Decide Heading
                        var[3] = 'course'
                        l.append(var)
                    for index in range(len(l)):
                        writer.writerow(l[index])
                    break

```

Figure 3.9: KML2CSV algorithm in python.

Table 3.2: Output from the KML2CSV code which is a CSV-file with 3 coordinates.

lat	lon	alt	heading	speed
57.77883205	12.779725075	243.17	course	2
57.77916943	12.780320524	241.68	course	2
57.77930978	12.780704307	241.21	course	2

Lastly, Figure 3.10 presents the implementation of the algorithm described in chapter 3.1.5. The main function utilizes a series of functions to derive the desired route. The input parameters involved in this calculation include the endurance, the path file, and the charging point file.

```
def main(endurance,path_file,charge_file):
    ...

# Choose the file for the created path
path_file = 'fullRoute.csv'

# Choose the file for all the charging points
charge_file = 'chargingPoints.csv'
endurance = 1000 #Choose the endurance in meters
main(endurance,path_file,charge_file) #

```

Figure 3.10: Overview of the algorithm which calculate the waypoint mission.

The output of the algorithm consists of several CSV-files that contain the necessary coordinates for each mission. The data comes in the same order as in the example above in Table 3.2. The number of CSV-files varies based on the distance of the endurance. In Figure 3.11 the endurance is set to 1000 meters, which results in ten part waypoint missions. Compared to Figure 3.12 the endurance is set to 3000 meters, which results in three part waypoint missions. As can be seen in the plots the whole path is covered and the algorithm have found the nearest charging point corresponded to the last point of each mission.

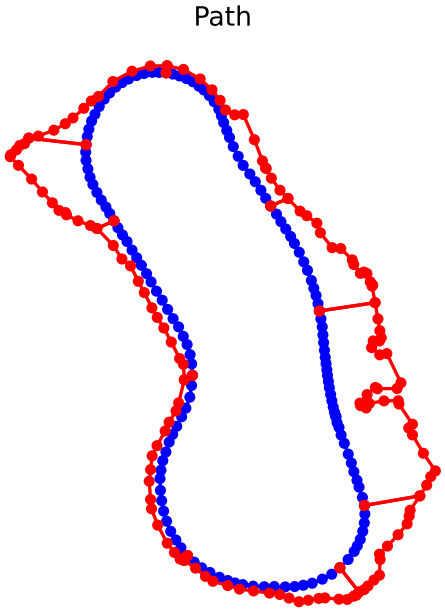

Figure 3.11: A plot showing the generated missions with an endurance of 1000 meters in red, along with the charging points indicated in blue.

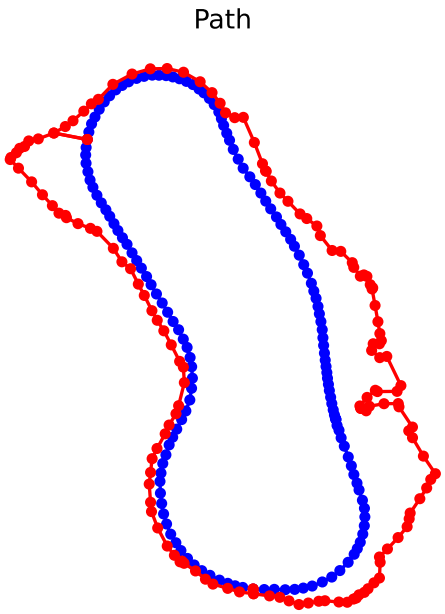

Figure 3.12: A plot showing the generated missions with an endurance of 3000 meters in red, along with the charging points indicated in blue.

3.3 Discussion

The ability to collect coordinates that had high accuracy was a crucial factor for drone control. Luckily the drone's GPS signal was strong and Google Earth had a precision of 2 meters in their coordinates which help a lot in this project. With plenty of coordinates to collect, the built-in function that Google Earth provides made this part work-friendly and reliable. Not only is the collection of coordinates important for creating the final path along the fence, but it also played a big role in making testing efficient.

The endurance calculations did not yield a result that could be effectively utilized in this project, as initially anticipated. It was realized that complex components such as the LiPo battery was difficult to analytically quantify into endurance, and too few tests were conducted to give good enough approximations. Furthermore, it became evident during the calculation of optimal speed that the process heavily relied on geometric coefficients, which necessitated simulations and tests which reached beyond the available knowledge, time resources, and project scope.

The two Python implementations, KML2CSV and WaypointMissionGenerator streamline the Drone Path Planning and make the solution to the sub-problem more general and dynamic. If needing to change the route due to some unexpected occurrence, generating easy-to-use CSV-files for Drone Control is fairly simple. The KML2CSV

implementation is itself hardcoded to some extent but if the KML-file-format does not change its essence, the parser should work. The WaypointMissionGenerator's main function is to generate routes in the correct format that does not exceed the endurance of the drone in a rather efficient way, which it fulfills. But as described in Section 3.1.5, the algorithm is greedy and to be fully optimal would require more knowledge about the number of waypoints effect on the efficiency.

3.3.1 Future work

For future work and to continue this project, the method of calculating and approximating the endurance can be further developed. As mentioned before, conducting extensive testing would be the next step in this phase of the project. To examine the impact of temperature in greater detail and with increased precision, measuring the time taken to complete a lap would be advisable. It is important to conduct these tests on the same day for optimal results and avoid changing the battery between runs. This ensures minimal influence from the degradation of the battery's state of health. Ideally, starting the tests on a cool morning would be preferable, complete one lap, then recharge the battery for the next lap. Repeat the process as the temperature increases.

There are also some areas of improvement in the optimization of drone speed. To achieve a complete optimization the next step would be to commence analyzing the drone's aerodynamics by wind-tunnel-tests and CFD simulations. Furthermore, it would be prudent to take into account the mandatory stops for each coordinate in a waypoint mission. Estimating the precise impact of these stops on the drone's endurance is challenging, but it is likely that they significantly contribute to energy consumption.

3.4 Conclusion

As a part of the larger approach in this project, this particular sub-problem was focused on extracting the coordinates of the fence and generating a suitable flight path for the drone. In the end, the following objectives were accomplished:

- A streamlined method of drawing paths in Google Earth, then through a Python-programmed parser, generating waypoint missions for the drone control with the desired format.
- With this streamlined method, collect the coordinates of the fence.
- Researched the effects on endurance from surrounding temperature, state of health and drone speed.
- Creating an algorithm that ensures that the generated path does not exceed the drone's operational capabilities. And breaking the path down to individual waypoint missions, which are easy for the drone control to work with and execute.

4

Drone control

The drone control sub-problem concerns the problem of controlling the drone to autonomously collect video and/or photographic footage of the fence. Controlling the camera gimbal includes collecting footage of the fence suitable for later analysis. This chapter will discuss how solving the problems can be achieved. The Drone control problem is illustrated in Figure 4.1.

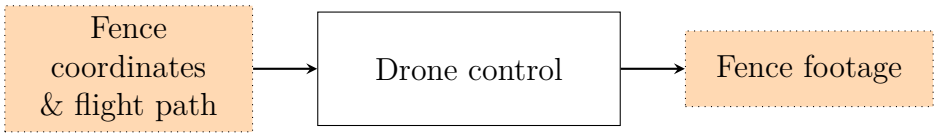

Figure 4.1: The drone control problem is defined as having fence coordinates and a flight path as inputs while requiring fence footage to be outputted.

The problem seen in Figure 4.1 can be further divided into the problems depicted in Figure 4.2. To achieve automatic control of the drone, suitable software needs to be selected.

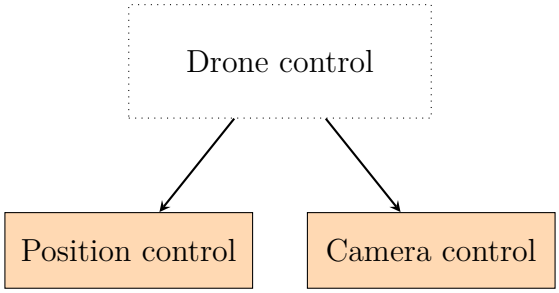

Figure 4.2: The figure shows the drone control and its subproblems. Position control and camera control. The first issue concerns the problem of controlling the drone's physical position and the second one concerns controlling the camera to capture footage in a satisfactory manner.

The high-level requirements for the controlling software are the following:

- Ability to control the drone's position
- Ability to control the drone's gimbal angle(s)
- Ability to take photos/record
- Ability to read the drone's status such as position, height, and battery level in real-time.

The drone used in the project, DJI Mavic Enterprise 2, was chosen due to its support of the DJI SDK [17]. Due to the time-consuming and complex nature of directly integrating the SDK, alternative bridging software solutions were explored. These

make it possible to interact with SDK without directly integrating the SDK. The bridge exposes a simpler API interface. The next step is to choose such software with the required functionality.

The fence coordinates are obtained from the path planning described in Chapter 3 in the form of a CSV-file which can be seen in Table 4.1. With the CSV-file as input, the drone control should output fence footage that is processed by the computer vision sub-module described in Chapter 5.

Table 4.1: Example of the expected fence input coordinate file.

lat	**lon**	**alt**	**heading**	**speed**
57.77883205	12.779725075	243.17	course	2
57.77916943	12.780320524	241.68	course	2
57.77930978	12.780704307	241.21	course	2

During the project's duration, two alternative computer vision implementations have been worked on in parallel. The first is a neural network, which finds the fence by training on a large amount of data. The other implementation is a simple line detection algorithm. These require somewhat different input data. The neural network has been trained on data primarily from the side, while the line detection expects input data from directly above. Because of these two different requirements, the implementation chosen should include the ability to collect data suitable for both methods of image analysis.

Drone control consists of two sub-problems waypoint control and camera control. The first problem, waypoint control consists of solving the issue of making the drone move to the correct coordinates, while the second one, camera control, will consist of controlling the gimbal. Camera control consists of deciding how to make the camera move and creating algorithms that calculate what angles the camera should have during different parts of the mission.

4.1 Method

Two different drone-controlling software were evaluated during the initial phase of the project. The first was ATOS (Autonomous Testing Operating System) [18], which was developed by AstaZero, the sponsor of the project. ATOS was developed to be a testing orchestrator for autonomous vehicles. Another software evaluated was called DSS (Drone Safety System). DSS was developed by another RISE branch and is a software dedicated to controlling drones. After studying both software and the requirements for this project, the DSS software was selected due to multiple reasons. The first was the simplicity of using the software. Since DSS was made to only control drones it was not as complex as ATOS, which could control any vehicle. Secondly, the drone software DSS was written in Python compared to ATOS, which was written in C/C++ which is more complex and would take more time to learn. How this software works will be discussed more in Chapter 4.2.

4.1.1 Equipment

The following equipment was used to achieve autonomous drone control:

- Drone: DJI Mavic Enterprise 2 (Zoom)
- DSS & DJI application: iPad Air 2020
- Computer: Macbook Air 2016 or a computer running Ubuntu 20
- Internet connection: iPhone 12 tethering

The *DJI Mavic Enterprise 2 (Zoom)* wirelessly connects with the controller, which is connected via a USB cable to the iPad. This is further described in Chapter 4.1.4. It should be noted that the software might work on other drones apart from the DJI Mavic Enterprise 2, however, that has not been tested. This also applies to the computer there are no limitations to what operating system could be used, apart from potentially missing dependencies.

4.1.2 DSS (Drone Safety System)

DSS [19] works as a safety bridge for communication with a drone's autopilot. The DSS is an umbrella term for three different pieces of software: the CRM, the Mission Manager, and the iOS application. These are explained in depth in the following chapters. Usage of the DSS API enables safer control of a drone. While using DSS the drone can be controlled in three ways; through a pilot with the remote control, a safety system (DSS), or a Mission Manager. The pilot can give control to the application. If the Mission Manager loses connection to the drone, the safety system takes over the control. At any point, the pilot can take over the control from either the safety system or the application.

It should be noted that DSS can be used to control many drones simultaneously however this functionality will not be used in this project.
Figure 4.3 illustrates the high-level interaction between sub-modules while using DSS for controlling the drone. The interactions are complex and described in more detail below.

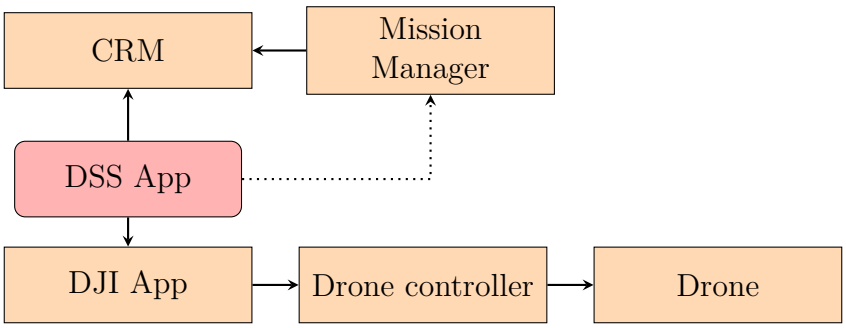

Figure 4.3: Diagram of all components in the control system. The arrows shows only how the components are related. What each arrow represents specifically will be discussed in the following chapters.

The Mission Manager is the Python program responsible for the control of the drone. This means deciding what commands should be sent to the drone at which times. It interacts with the drone using the DSS API later discussed in Chapter 4.6.

4.1.3 CRM (Central Resource Manager)

CRM stands for Central Resource Manager and is a Python software included with the DSS software, created for managing drones and applications that need drones. The CRM waits for a connection from both a drone and a Mission Manager when both are available it sends the connection details of the drone to the Mission Manager. After this stage, the CRM is no longer involved in any connection between the two. The CRM is therefore only necessary during the initial connection phase. More about the connection details can be read in Chapter 4.6. The CRM (Central Resource Manager) name comes from the fact that it can be used with many drones and Mission Managers at the same time. Launching the CRM is done with the following command:

```
$ python3 crm.py --stdout \
--ip 0.0.0.0 \
--port 1000
```

Due to how the DSS app works, the port is fixed and cannot be decided freely. How the port is chosen is described in Section 4.1.4.

4.1.4 iOS applications

Controlling the drone from the Mission Manager requires two different iOS applications. The first one is the official DJI application which is connected to the control via a USB cable. From the DJI app, most aspects of the drone can be controlled. Additionally, it exposes an SDK which allows other apps to interact with the drone and control it programmatically. This SDK exposes different APIs which can be used by any application on the same devices. The DSS application uses these APIs and effectively creates a bridge between the DSS commands and DJI commands. More details about how the DSS and the DJI SDK interact are discussed in Chapter 4.1.6. Changes can be made to the DSS API, but as it is written in the Swift programming, it would require both knowledge of the programming language and the proper equipment. Therefore the project might need to work around any limitations imposed by the DSS application.

Limitations with how the app was developed mean that there is no way to input a specific port for the app to connect to the port is decided by multiplying the third octet by 100 [20]. The following IP-address: $X1.X2.X3.X4$ means the drone would connect to $Port = X3 \cdot 100$.

4.1.5 Connection sequence

When all modules are available, the pre-launch sequence outlined in Figure 4.4 can be executed. The two branches can be prepared separately, but the *Connect CRM* step requires that the drone is connected to the CRM and the drone. If all steps have been executed properly the mission will start, and the drone will visit all waypoints in order.

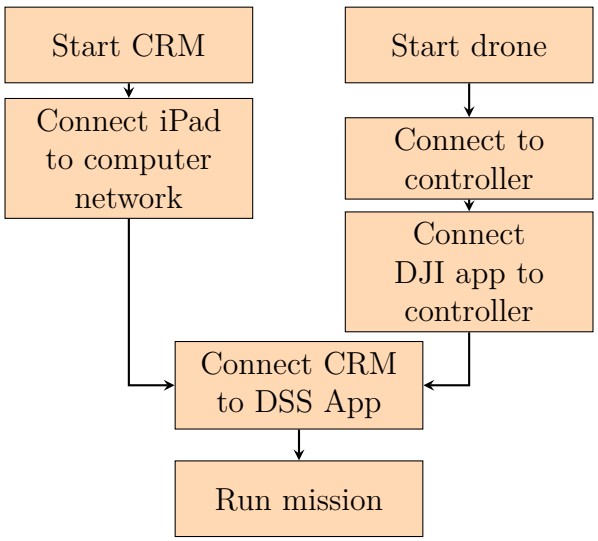

Figure 4.4: Steps needed before running the drone mission. All steps need to be taken in the right order in order for the mission to be successful.

4.1.6 Mission Manager and DSS application communication

In this section, the communication between the Mission Manager and the DSS iOS application will be discussed as seen in Figure 4.5.

Figure 4.5: High-level view of the communication between the Mission Manager and DSS application and subsequently the drone.

A more complete view of how the communication works are shown in Figure 4.6. This is by far the most important part of the communication to understand as it is where most if not all of the development will happen. The communication is carried over the network using TCP sockets.

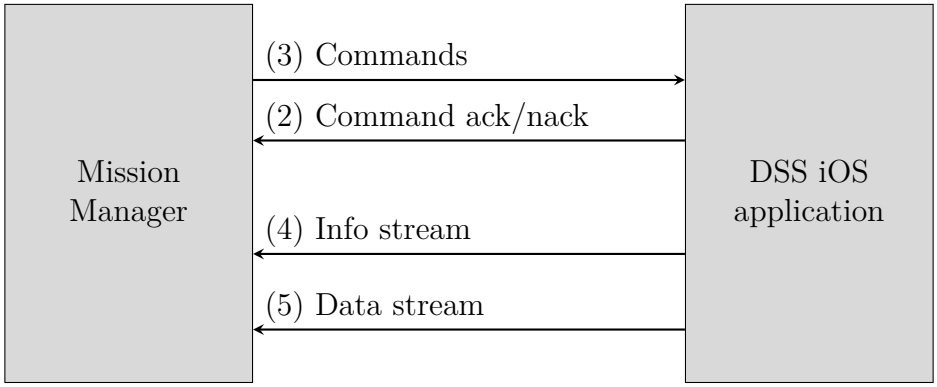

Figure 4.6: Communication between Mission Manager and DSS iOS application and subsequently the drone. The number in the parentheses indicates which socket is responsible for data transfer as seen in Table 4.2. The arrow represents in which direction commands are sent.

Once the Mission Manager has found an appropriate drone for the mission, it starts to listen on two ZeroMQ sockets. ZeroMQ is a protocol for sending and receiving event data between different systems in real-time. The DSS iOS application then connects to these sockets and the connection is established. The first socket is used for incoming messages to the Mission Manager while the latter is used for replies from the script to the DSS application. Additionally, the DSS iOS application opens two more ports. Of which the first is streaming data about the drone's state to the Mission Manager in real-time, and the second socket is responsible for transmitting requested photo-data to the application. In Table 4.2 a detailed list of all mentioned ports are shown.

#	**Port**	**Listener**	**Program**	**Use-case**
1	1000	Computer	CRM	Meeting point for drone & Mission Manager.
2	1001	Computer(2)	Mission Manager	DSS iOS application → Mission Manager replies.
3	1002	Computer(2)	Mission Manager	Mission Manager → DSS iOS application communication.
4	5000	iPad	DSS iOS Application	DSS iOS application → Mission Manager streaming info.
5	5001	iPad	DSS iOS Application	DSS application → Mission Manager streaming data.

Table 4.2: A list of all used TCP ports and their respective task. Note that the exact port numbers will wary between each setup, as mentioned in 4.1.4. It should also be noted that it is not strictly necessary to use two different computers for the setup, it merely shows that it is possibly if desired.

The use of ZeroMQ means that the communication is done using JSON messages. Commands are sent between the Mission Manager and the DSS application in the format seen in Figure 4.7, the reply can be seen in Figure 4.8.

```
{
    "fcn": "<command>",
    "id": "<requestor id>",
    <additional parameters>
}
```

Figure 4.7: Message sent from Mission Manager to the DSS iOS application. Sent over socket #3. The requestor id is a uniquely identifying identifier that is assigned to each Mission Manager connected to the CRM.

```
{
    "fcn": "<command>",
    <additional parameters>
}
```

Figure 4.8: Reply from DSS iOS application to Mission Manager. Sent over socket #2.

However, the DSS software [19] provides an abstraction layer so that these messages are hidden from normal mission development. The info stream allows for real-time streaming of the drone's status. They can be helpful when data such as position, altitude, heading, or battery level is needed rapidly. Some of the most used API commands are listed in Table 4.3. Some of these commands are sent to the drone, while others are used purely within the DSS system itself.

Command	Description
set_init_point	Sets current point as init point. Must be sent before being able to take off.
arm_take_off	Makes the drone ready for take off and takes off.
upload_mission_LLA	Uploads a waypoint mission to the drone. Takes in a list of lat, lon, alt and heading. See Figure 4.9 for exact formatting.
set_gimbal	Sets the camera's gimbal angles. In this project only the pitch as the drone use only allows for that kind of movement.
photo	Supports different arguments. Used to take photo, video, download photos from the drone.
land	Lands the drone.
gogo	Activates a pending drone mission.
get_state	Gets the current state of the drone. Returns latitude, longitude, altitude, battery level, aswell as some other parameters that are irrelevant for this project.
follow_stream	Configures an additional socket used for real-time waypoints for the drone to follow.

Table 4.3: A list of the used DSS API commands in the project. A complete command list as well as more information about each command can be found in the DSS [19] documentation.

Two of the DSS commands enable drone movement. The *upload_mission_LLA* and *follow_stream*. The first command expects data as shown in Figure 4.9.

```
{
    "id0": {
        "lat": "<latitude>",
        "lon": "<longitude>",
        "alt": "<altitude above sea level>",
        "heading": "<heading 0-359 degrees>",
        "speed": "<speed in m/s>"
    },
    ...
}
```

Figure 4.9: Example on how the upload_mission_LLA command expects input. Points are numbered id0 -> idX.

In addition to the upload_mission_LLA command, the follow_stream command would also allow for drone flight. The command takes in two parameters, an

IP-address, and a port. The DSS iOS application will then open a socket and await LLA (latitude, longitude, altitude, and heading) messages, instructing the drone where it should move. This would mean that a mission implementation using follow_stream would require advanced logic compared to the upload_mission_LLA, which takes full control of the drone until all points have been visited. The benefit of the follow_stream command would be that it offers a more dynamic flight of the drone, as mission waypoints could be altered at any time. This is something not possible with the upload_mission_LLA, as it requires a complete list of commands before executing the mission.

4.1.7 Data transfer

Data transfer of the collected footage from the drone to the data processing system can be done in a number of ways. The DSS API allows for transferring of photos in real-time from the drone to the Mission Manager which is useful if one wants to process images in real-time. Images can also be retrieved by manually ejecting the SD-card after a mission and then subsequently entering it into a computer for transfer and analysis. For video footage, it is not possible to transfer in real-time with the current setup. This is due to limitations with the DSS software.

4.2 Results

Because of the system's complexity a great amount of time was spent researching and testing the behaviour of the drone and the DSS software. AstaZero provided a template code that could run a simple waypoint mission, but it was unable to take photos or adjust the gimbal position. The focus of the development has been on automating the camera gimbal and camera actions and establishing an interface for a more robust drone operation.

4.2.1 Waypoint mission

A waypoint mission is a list of ordered coordinates a drone should visit. It is one of the building blocks required for autonomous drone flight. Figure 4.9 shows an example of a waypoint mission. The software came with an example code of a waypoint mission and was further developed to allow any CSV-file as input instead of having a few hard-coded coordinates.

Further, the code improved to monitor a running mission, which allows one to query the ongoing mission and make decisions on the fly. Making decisions during the flight is a prerequisite for controlling the camera's gimbal which will be further discussed in Section 4.2.3.

The example code ran an internal fly_waypoints function which starts the mission and waits until it is complete, however our requirement. See Figure 4.10.

```
# initialize the drone
...
drone.fly_waypoints(start_wp) #blocks until finished
```

Figure 4.10: Figure shows the example code initially provided in the DSS repository. The code starts the mission and waits until it is finished before continuing the code execution.

The code was updated to monitor the state of the drone at all times and make relevant decisions about the mission as seen in Figure 4.11. The get_state command is used to get the current coordinates and altitude of the drone. This loop is further illustrated in the Figure 4.17.

```
drone._dss.gogo(start_wp)

while drone._dss.get_armed(): # while Mission Manager is in control
    time.sleep(0.5)

    # Coordinates of drone
    state = drone.get_state() #{lat: x, lon: y, alt: z, ...}

    # use state to decisions about the mission
```

Figure 4.11: Figure shows how the code has been altered to now provide monitoring of the running mission.

During testing and evaluating of the waypoint mission code, it was observed that the drone slows down and stops at each waypoint. Something that isn't a problem per say but increases total mission time.

4.2.2 Mapping drone coordinates to fence coordinates

Mapping between the fence and the drone coordinates is required if the drone collects data from the side of the fence. The input is a waypoint route illustrated with a yellow line and a fence file portrayed as the red line on the outfield lines on the soccer pitch in Figure 4.12.

Figure 4.12: Illustrated path and camera target from Google Earth [12].

A Python function was written to map the drone coordinates to the fence coordinates. Then, it calculates the distance to each fence coordinate and maps to the coordinate with the least distance.

With the coordinates of the fence and the coordinates of the drone, the heading of the drone could be calculated as described in Section 4.2.4. The problem with too great of a distance between the points on the fence is illustrated in Figure 4.13. The green dotted line is the heading of the drone, and the blue line is the fence. The stars illustrate the drone and show the difference in heading depending on a modest difference in position. If the drone is too close to the fence or the points are too sparse, it can result in a significant error in heading.

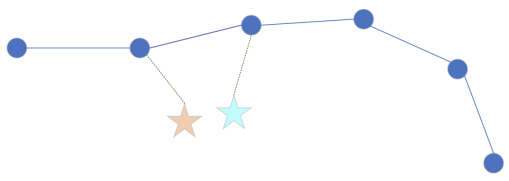

Figure 4.13: Mapping with sparse interval leads to the drone not facing the fence perpendicular.

With more points, the heading difference becomes less, and with enough points, the error becomes insignificant, which is illustrated in Figure 4.14

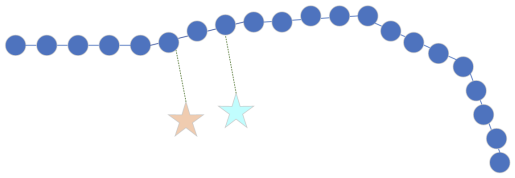

Figure 4.14: Mapping with dense interval. The drone will always face the fence perpendicularly since the closest point selected always will be the fence's closest point.

In order to solve this problem while not changing the coordinate collection step, a method for augmenting the already collected coordinates was developed. The algorithm adds midpoints between all the points in the dataset until no two points have a larger distance than defined. A suitable value for this variable was found to be 0.50 meters, which was found experimentally.

The mapping method presented above works well in most cases, but sometimes the mapping might not be sufficient, for instance, if the desired coordinate to record is not the closest coordinate to the drone. Then a one-to-one mapping is required where it is possible to put in a coordinate the gimbal will point at. Both mappings are able to mix within a waypoint mission. In Figure 4.15, each pair is illustrated as a color, and the green dotted line resembles the heading angle. The problem with one-to-one mapping is that each waypoint manually has to be mapped to the fence coordinate, which is rigorous. Therefore one-to-one mapping is only used in corner cases.

Figure 4.15: One-to-one mapping, maps a waypoint coordinate to a chosen fence coordinate. This is useful in cases where the desired fence coordinate isn't the one closest to the drone. Since it is time-consuming to generate a list of one-to-one mapped coordinates, the Mission Manager will default to using the closest fence coordinate. However, providing one-to-one coordinates for a point will override this behaviour for the given coordinate.

4.2.3 Gimbal control

The DSS API's set_gimbal command lets the Mission Manager set the gimbal. For the drone used in this project, DJI Mavic Enterprise 2 the available pitch angle is between 20 and -90 degrees. Which translates to an angle of 20 degrees above the horizontal plane to directly vertical. In order to point the camera at the fence, the drone's coordinates need to be mapped to a fence coordinate, described in more

detail in Chapter 4.2.2. DSS also has a function to get the current location of the drone. The coordinates of the fence need to be distributed before the flight. Figure 4.16 illustrates the pitch angle.

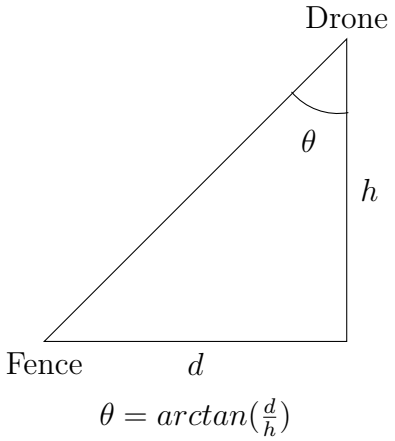

Figure 4.16: How the camera's pitch angle is calculated.

With the coordinates of the drone and of the fence, the distance between the drone and fence could be calculated with the following equations [21]:

$$d = \arccos(sin(lat_1) \cdot \sin(lat_2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \cos(lon_2 - lon_1)) \cdot r_{earth} \quad (4.1)$$

$$\theta = \arctan(\frac{d}{h}) \quad (4.2)$$

The pitch angle is updated dynamically during the flight depending on the GPS location of the drone and the mapped fence coordinates. Figure 4.17 illustrates how the pitch angle dynamically is updated during the flight.

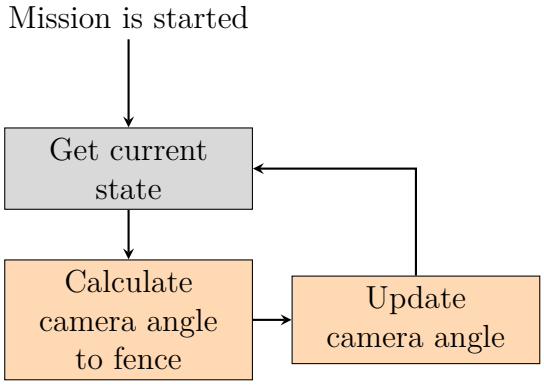

Figure 4.17: Gimbal control loop. The loop is executed until the mission is finished.

4.2.4 Drone heading control

To automatically point the camera towards the fence, the yaw or roll direction of the gimbal needs to be controlled. However, the drone provided by AstaZero only

has the availability to control the gimbal in the pitch direction. There are similar drones produced by DJI that can control roll and yaw. For the drone distributed by AstaZero, the heading of the drone needs to be changed to direct the camera toward the fence. Changing the heading is not possible to perform dynamically during a waypoint mission. Hence, the heading is calculated in the Mission Manager before the waypoint mission starts based on the fence and path coordinates.
The following equations are used to calculate the heading of the drone [22]:

$$d_{Lon} = lon_2 - lon_1 \tag{4.3}$$
$$x = \cos(lat_1) \cdot \sin(lat_2) - \sin(lat_1) \cdot \cos(lat_2) \cdot \cos(d_{Lon}) \tag{4.4}$$
$$y = \sin(d_{Lon}) \cdot \cos(lat_2) \tag{4.5}$$
$$\phi = \arctan(\frac{y}{x}) \tag{4.6}$$

Where lat_1, lon_1 refers to the fence coordinates, lat_2, lon_2 to the fence coordinates, and ϕ refers to the heading of the drone.

4.2.5 Footage collection and transfer

This chapter discusses how the drone was programmed to collect imagery data of the fence. The DSS API has functions that allow for taking photos and videos. It also can download individual images over the air. Taking photos would be the best choice for fence detection. However, images can easily be extracted from a video recording even though it would result in lower quality, hence somewhat worse fence detection. Therefore the drone's camera module was investigated rigorously.

Testing the DSS app's source code concluded that the DJI Drone camera module and the DSS app have four different operating modes. *Taking a photo*, *recording video*, *storing photo*, and *downloading photo* the four actions cannot be performed in parallel. A switch needs to be performed if a change of operation is required. Trying to perform an action when the module is not in the correct state will result in a switch operation. Sometimes the switch operation fails, which results in the error message *Camera resource busy*. In some scenarios, the error is expected, for instance, when the drone is already recording a video. However, in other cases, the error could appear seemingly randomly.

Research and experimentation have led to the discovery that the AF (auto-focus) feature could interfere with changing of operation. The reason for this makes the camera occupied for a while. After disabling AF the camera operated better in many cases but still with problems. Taking photos and changing operations was time-consuming, which resulted in the drone taking photos too slowly. Therefore the speed has to be declined to catch the fence breaches. Taking photos and downloading them was concluded to be too unreliable and time-consuming. Another option would be to take photos and store them on the SD card for processing. During testing, taking photos was discarded because of its unreliability previously concluded. Therefore video recording of the fence is the most robust since it rules

out all potential issues. Hence, the drone's SD card has to be manually extracted from the drone and inserted into a computer for processing.

4.2.6 Operator interface

During testing and operation of the drone, there were many moving parts needed to work for a successful drone mission as illustrated in Figure 4.4. Therefore a user-friendly interface has been developed. The interface allows for automatic start of the CRM, displaying missions on a map, as well as launching missions. This interface is far from completed at this stage but can be useful in certain situations. Figure 4.18 shows a screenshot from the control panel.

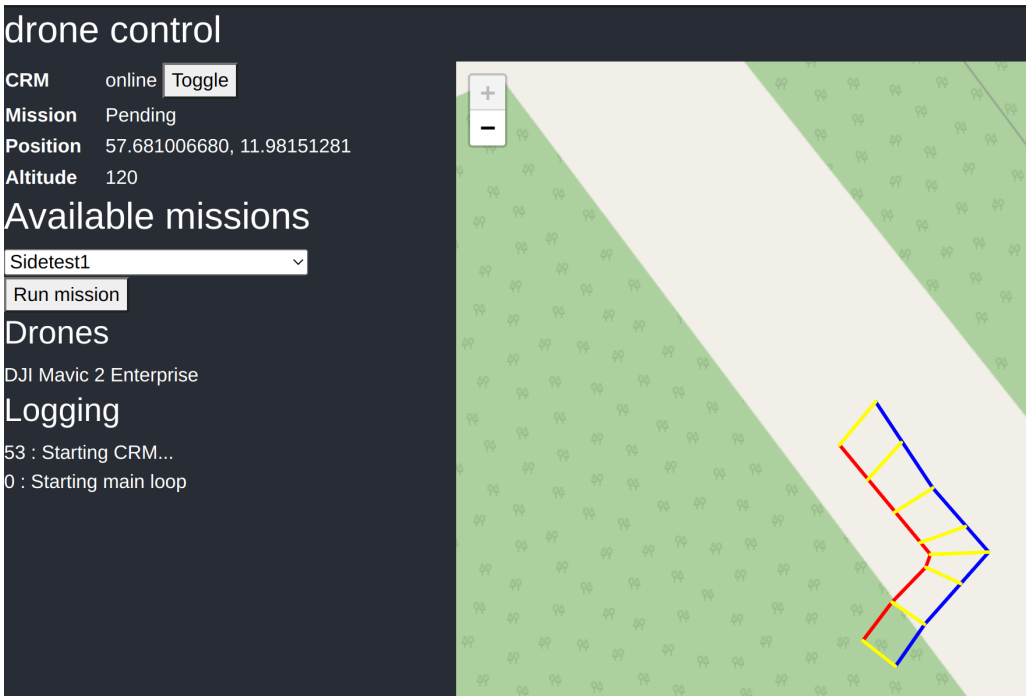

Figure 4.18: A screenshot of the operator interface. The left side of the interface shows the drone's current status as well as controls to starting a mission. The right side shows a preview a mission. The red line represents the drone's trajectory, the blue line represents the fence and the yellow lines show how the gimbal will be faced. Copyright: OpenStreetMap contributors.

The interface consists of a Python backend server and a frontend written in ReactJS, which is a framework for building interactive javascript websites. Clients connect from the frontend to the backend via an WebSocket connection and then received real-time updates from the server which is then displayed to the client.

4.2.7 Running a mission

There are two computer vision implementations, line detection, and neural network, which will be discussed in detail in Chapter 5. The two implementations require different fence footage; directly from above contra the side. Hence, the input format to the Mission Manager differs. The drone's path is described by the required input argument *coordfile*, and the fence coordinates are described by the optional input

argument *fencefile*.

If no *fencefile* is provided, the fence is assumed to be directly under the drone path. Hence, the gimbal is set to -90 degrees and follows the waypoint mission from *coordfile*. Otherwise, the pitch angle and heading of the drone are calculated as described in Chapter 4.2.3 and Chapter 4.2.4. The input format for *coordfile* are illustrated in Table 4.4.

Table 4.4: Example of the input arguments in the *coordfile*. Some of these arguments are optional.

lat	**lon**	**alt**	**lat_fence**	**lon_fence**	**alt_fence**	**heading**	**speed**
57.7762	12.7845	67.57	0	0	0	calc	2
57.7761	12.7846	67.57	57.7763	12.7846	63.6217	calc1	5
57.7760	12.7847	67.57	0	0	0	127	3
57.7765	12.7848	67.57	0	0	0	course	4

The input rows lat_fence, lon_fence, and alt_fence in *coordfile* are not required for the Mission Manager. Depending which mode is selected, different input data is required. The modes are defined in Figure 4.19.

Figure 4.19: The four different operating modes. These can be freely mixed within one fencefile, however some modes require additional data.

'heading' value	**Explaination**
calc	Mapping to the closest fence coordinate. Requires that the Mission Manager is provided with a *fencefile*.
calc1	One-to-one mapping. The drone coordinate is mapped to the coordinate defined by the lat_fence, lon_fence, and alt_fence fields.
course	heading towards the next waypoint.
(integer)	Heading in degrees between 0-359, relative to true north.

An example of *fencefile* input is illustrated in Table 4.5.

Table 4.5: Example of the expected fence input coordinate file. Note that a fence file usually contains more coordinates than listed here.

lat	**lon**	**alt**
57.77883205	12.779725075	243.17
57.77916943	12.780320524	241.68
57.77930978	12.780704307	241.21

The previous section mentioned the possibility of running the missions from the interface. If one wants to run a mission manually, it is possible to use the terminal to enter the command instead, illustrated in Figure 4.20.

```
$ python3 app_fence_mission.py --stdout \
    --crm 127.0.0.1:1000 \
    --app_ip 192.168.0.2 \
    --fencefile ./fence_coordinates.csv \
    --coordfile ./mission_coordinates.csv
```

Figure 4.20: How to run the mission manually. Note that the the mission requires the CRM to be running on the specified port before running the mission.

Figure 4.21 shows the command sequence after executing a mission.

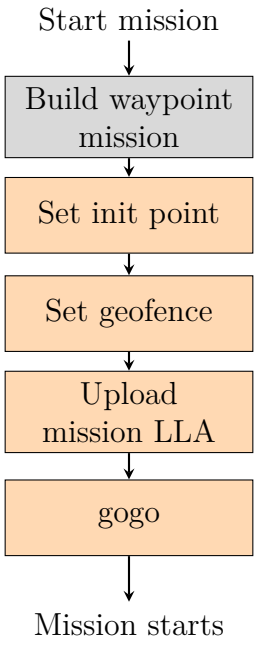

Figure 4.21: Mission Manager command sequence.

The set init point command and set geofence are required to be sent by the DSS before takeoff.

4.2.8 Testing results

A test was performed at AstaZero's test track to investigate whether the drone could surveil their fence and be used daily by AstaZero. The test had two parts that examined the two different operating modes. The first test had a path straight over the fence, and the second test had coordinates from the side. A test with 30 waypoints located directly above the fence was performed, which resulted in the drone flying the whole mission and recording the fence. Because of restricted areas to fly over at AstaZero, it was not possible to test the whole path around the track. Based on the outcome of the first test, it was concluded that executing a waypoint mission with a straight flight path over the fence is utilized for AstaZero to use. The second test recorded that the drone had an accurate position, heading, and pitch angle, however at some parts of the testing track the forest is dense, and it is not

possible to fly from the side.

Testing of the drone concluded that the drone slowed down during each waypoint visit, which increased the total time needed to visit all points. It also increased the number of charging stops needed to surveil the fence. Hence, as few waypoints as possible are beneficial during path planning. However, fewer waypoints results in a decrease in accuracy as it follows the fence less precisely.

4.3 Discussion

Automated autonomous systems all have high demands for accuracy and reliability. Drone projects require high accuracy due to safety risks and the risk of colliding with objects. During the project, much thought has been given to improving reliability in different ways. Improving the reliability started with looking at the Python code provided by Asta Zero and investigating how errors were handled. It is vital to have a stable LAN connection between the iPad and the computer running the Mission Manager. Note that an internet connection is not required to run the Mission Manager.

Apart from the Mission Manager, other applications were required, the DSS application and the DJI application. The DJI application is stable and rarely has problems. However, the DSS App is under construction, and occasionally the application crashes. Most of these were not critical since they often happened when other things stopped working. See the Table 4.6 for subsystems and possible errors that may occur. Limitations in the DSS App meant that it often had to be restarted between missions and had difficulty reconnecting to the CRM when the connection was lost. Hence, the system is still quite fragile in some cases and requires hands-on work to be operable.

Another limitation with the DSS app was that the drone would not take off unless the pilot manually had taken off the drone and landed it before running the mission. To manually start the drone needed to be done each time after reconnecting to the CRM. The exact cause of the error originates from within the DSS app but has not been investigated. The error is a minor detail when looking at the project as a whole, but something important to look at if the project were to be used in daily used operations.

Furthermore, due to the specific implementation of the waypoints in the DSS iOS application, the drone had to slow down and stop at each waypoint during a mission, which potentially could be avoided by using the follow_stream command available in the DSS application. However, the follow_stream command was not explored due to time constraints.

Multiple of the problems mentioned is out of the control of this project due to the choice of drone-controlling software. With a different software, some of the limitations could be avoided. However, for this project, the choice of software was

made primarily with the simplicity of implementation due to the time restrictions. See Table 4.6 for details regarding possible errors.

Table 4.6: Possible errors and their respective solutions. The DJI App and controller have been combined into one subsystem as they are not distinguishable from the API's point of view.

Subsystem	Possible errors	Solution
Mission Manager	Python errors, lost WiFi connection	Code reviews, and testing. A robust internet connection to the application
DSS App	Application crash	None (Out of scope for this project)
DJI App/Controller	Lost drone connection	Flying drone well within range recommendations
DJI Drone	Drone crash	Flying the drone with large marginals concerning nearby objects

4.3.1 Future work

As previously mentioned, it has been a demarcation of the project to not working with software other than Python code for the Mission Manager. For future development within the field, the DSS app needs to be improved or switched to another software. As mentioned in Chapter 4.3, improvements to the system's robustness have been challenging to implement due to the software used. Improvements to the DSS app could improve its robustness.

Another area of improvement is the use of waypoints. Testing has shown that the drone deaccelerates at each waypoint and stops completely before traveling to the next waypoint, which significantly increases the total mission time. (The use of the follow_stream command which is mentioned in Figure 4.3) was explored somewhat. The benefits are that the drone can be controlled more dynamically. The follow stream command activates a mode where the drone connects to an LLA stream and then travels to the coordinates specified by the streamed data. The follow stream command would allow for implementing a completely new waypoint mission controlled in Python rather than using the waypoint mission functionality provided by DSS, which is core using the SDK's internal waypoint mission. Implementing the mission code in Python would allow for much greater control over the drone. It would also mean that the drone could be faster by never hovering still at waypoints.

4.4 Conclusion

In conclusion, the high-level requirements for controlling the drone were:

- Ability to control the drone's position.
- Ability to control the drone's gimbal angle(s).

- Ability to take photos/record.
- Ability to read the drone's status, such as position, height, and battery level, in real-time.

These abilities were met, as shown by our tests at AstaZero. The drone's position is effectively controlled using waypoint missions, while the camera angle is controlled by sending consecutive set_gimbal commands. These enabled the drone to record the fence directly above and from the side, which satisfies the requirements set. The system effectively serves as a proof of concept at AstaZero. Where the system currently has its weaknesses is in the area of user-friendlyness. It is complex to operate and could be developed to be even more robust. These arose from the choice of software, DSS. Any bridging software will inevitably incur some limitations on possibilities. The reason for using a bridging software was the time constraints of the project. Using the DJI SDK directly would have offered more possibilities and fewer constraints while implementing the API.

5
Fence detection

Analyzing and classifying images can be both time-consuming and challenging. Taking help from computers can save time and help us find details and information more efficiently. Computer vision is the broad term in artificial intelligence for computers to understand the world and the ability to see as humans visually can perceive the environment [23]. Computer vision applies to several areas, and this chapter describes two implementations, line detection and neural networks.

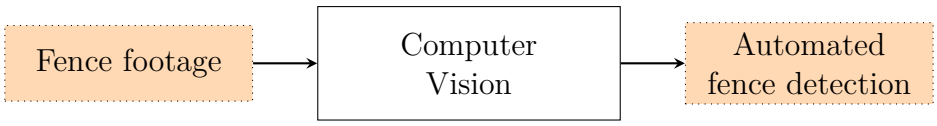

Figure 5.1: The Computer Vision problem is defined as having fence footage as input and to output fence detection.

The primary aim is to develop a novel model for detecting fences and breaches in fences from video footage. To accomplish this goal, a comprehensive dataset was created to enable the model's training. Given the specialized nature of the fence detection task, it was necessary to construct a custom dataset from the ground up. This entailed capturing video footage of multiple fences from diverse vantage points, followed by splitting these videos into individual images, and augmenting them to increase dataset variability. The resulting dataset was then utilized to train a U-Net model, which was executed on the Chalmers Vera cluster to expedite computation.

5.1 Methods

There are many different approaches to computer vision. One approach was to use the Python library OpenCV and its tools for line detection [24]. The line detection method attempts to detect the lines in the fence from aerial footage. Another approach that the project investigated was semantic segmentation. Semantic segmentation is a method that sections an image into regions and classifies them with labels [25] and can be trained to classify different objects.

From an aerial point of view directly above the fence, the problem can be seen as a line detection problem. The angle makes the fence appear as a continuous line across the image, see Figure 5.2. Using the OpenCV library [24], the images can be processed with a Canny edge detector to detect the lines from the contours in the image. Using the Hough Line Transform and tuning the parameters, the desired lines

can be detected [26]. The input image is converted to grayscale, which is done to time-optimize the calculations that are later made in the Canny edge detector. The detector processes the imagery only to show the locally minimal values, by setting upper and lower thresholds, the contours can be filtered to the lines searched, as illustrated in Figure 5.2. Then the Hough Line Transform [27] is calculating a curve representing every line going through a point in the picture. This is done for every pixel and by finding the intersections between these calculated curves, a line can be detected. The transform is also thresholded with upper and lower values to remove unwanted lines.

Figure 5.2: On the right, an overview directly above the fence transformed into greyscale. Middle image showing the contours found with Canny edge detector, thresholded to minimize unwanted contours. On the right showing the detected lines from Hough Line Transform, drawn as green lines over the image.

One advantage of using this method is that it is easy to apply, as there are many pre-existing code libraries that already have these functions implemented. However, as shown in the Figure 5.2, there are still many incorrect detections. This indicates that further optimization of the parameters is necessary. Due to the terrain and surrounding trees, it is challenging to isolate the fence without cropping the footage. Unlike related work using OpenCV to detection of power lines [28], the fence does not always provide a clear, straight stretch across the image. This makes it more difficult for line detection algorithms to identify it. As a result, detecting fences requires more complex algorithms and techniques than detecting power lines.

The second method investigated is the method of semantic segmentation as described in [25]. The problem can be seen as a pixel-wise labeling problem, where each pixel in an image is classified into different classes. Illustrated in Figure 5.3, the fence image and its mask is seen, where the mask would be the ideal classification of the image. To achieve this, a model is trained using supervised machine learning. A neural network is trained on a annotated dataset to learn the segmentation. The projects dataset and its preparation is described in Section 5.1.1.

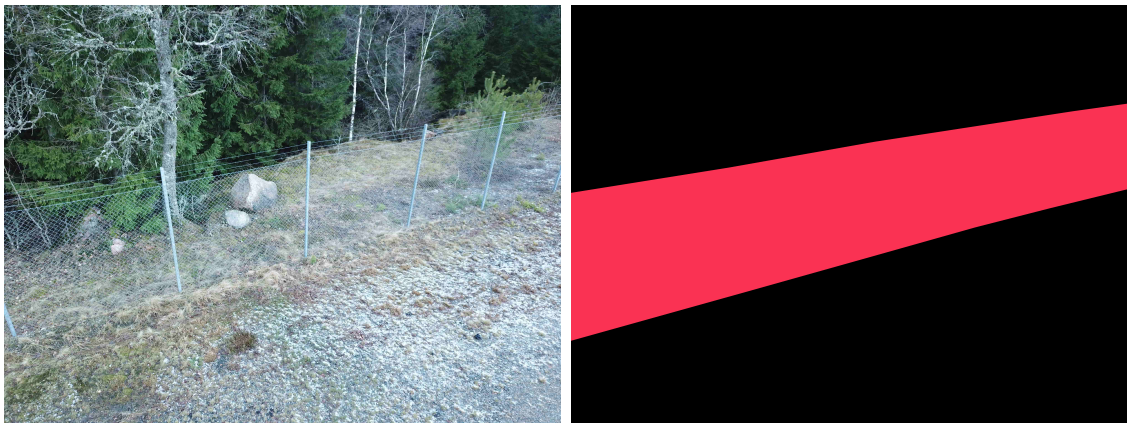

Figure 5.3: Side view image of a fence and its mask. The mask is the ground truth and shows the neural network what to classify. The mask has two classes, the background in black and the fence classified in red.

Training the neural network on images of fence from different angles makes it possible to detect the fence from more angles than a top-down view from the drone. Having a side view or an angled view of the fence gives more visual data that can be examined and potential breaches to the fence could potentially be more easily identified. Compared to line detection, semantic segmentation can potentially have a better robustness when detecting fence. This is because a well trained semantic segmentation model can detect fence from many different angles, if the dataset used is comprehensive enough. Using a neural network also creates the ability to classify more than just the fence, with enough data one could classify cut holes or fallen trees that affect the integrity of the fence.

The disadvantage of this method, however, is that it requires a lot of computer power to be able to train efficiently. A large dataset of varied images is needed for the model to be generally applicable. Further, collecting enough images and processing them to be used for image segmentation can be time-consuming.
In related work [29], a successful semantic segmentation on footage from a UAV (Unmanned Aerial Vehicle) has been achieved. This does give credibility to the choice of using semantic segmentation for detecting fence from images taken by a drone. However, the purpose and dataset are different, where the rice fields can be distinguished more distinctly against the background class. The fact that the fence is transparent can create problems when it has to be distinguished from the background.

5.1.1 Data preparation

This Section tackles the process of preparing the data into the correct format required by the model in Chapter 5.1.2. The input data will consist of a video of a fence captured by the drone in Section 4. The model requires individual image inputs, not a full video. The first step to solve this problem is to split the video into individual frames.

In order to achieve this, a Python script was utilized to separate videos into individual images with a specific frame interval. Typically, a frame separation of 30-50 was used for the majority of the data. The decision regarding the frame separation was based on two primary reasons. Firstly, a small frame separation would produce images that are quite similar to one another, leading to numerous duplicate images in the dataset. On the other hand, a large separation would result in fewer images being extracted from the video, ultimately reducing the number of images in the final dataset. The final choice was a good middle ground between the two factors.

5.1.1.1 Data annotation

To train a neural network model for semantic segmentation, it is essential to have quality training data that the model can learn from. The dataset should include masks that label the ground-truth for the different classes of interest in the image. In our case, the labeling classes were limited to only two classes: background and fence. The quality of the results depends on both the quality of the training dataset and the quality of the annotations on the dataset [30]. There are many techniques and tools for annotating digital images, and this project used the open-source Computer vision annotation tool (CVAT) [31][32]. The images were manually annotated using polygons. Using polygons allows for annotating the fence that might have difficult shapes depending on the camera angle from the drone. As shown in the Figure 5.4, polygon annotation is used to highlight where in the image the fence is. The annotation follows the outer edge of the fence, creating an area marked in red.

Figure 5.4: Illustrating the CVAT online tool used to annotate the ground-truth on the images.

The data collected for the dataset contains images with different angles from a section of the fence at the AstaZero test track. This data was gathered by manually flying a drone and capturing both pictures and videos. The videos were then

processed using the video splitting script. To expand the dataset, more data were collected at a soccer field with a similar fence to get more data to train the model. Altogether, the dataset for annotation contained 324 images, see Table 5.1, where 37 of them were from the football field, and the rest from the test track. The images in the dataset vary in the angle at which they are taken. The annotated dataset is not very large, and there are various options to consider for increasing its size.

5.1.1.2 Data augmentation

A method to increase the size of the dataset is using data augmentation. This project has investigated splitting images into smaller images and augmenting by warping and rotating images randomly.

To split the images a splitting algorithm was used. This algorithm works by taking an image and its mask and splitting it into a chosen amount of smaller images from the original image. The image is first split horizontally into the desired amount of smaller images, then cropped to include a desired amount of fence. The function then outputs the image and mask of the cropped image. Figure 5.5 contains the image in Figure 5.3 split with the image splitting algorithm. The output images have been reshaped into the wanted size of 256 x 256 pixels.

Figure 5.5: Images output by using the image splitting function with 4 splits and 40 percent fence in each image. The image and mask in Figure 5.3 is used as input.

The cropping of the images works by finding a bounding box around the fence from the mask. The mask contains two values, 0 for background and 1 for the fence. The mask is then changed into an array with all the non zero elements. Then the coordinates of the corners of the mask is found using Numpys min and max functions.

```
def find_corners(mask):
    mask = np.array(mask)
    non_zero = np.nonzero(mask)
    min_x, max_x = np.min(non_zero[1]), np.max(non_zero[1])
    min_y, max_y = np.min(non_zero[0]), np.max(non_zero[0])
    return min_x, min_y, max_x, max_y
```

After finding the corners the image is resized from these coordinates. This creates an image with the maximum amount of fence in the image as shown in Figure 5.6.

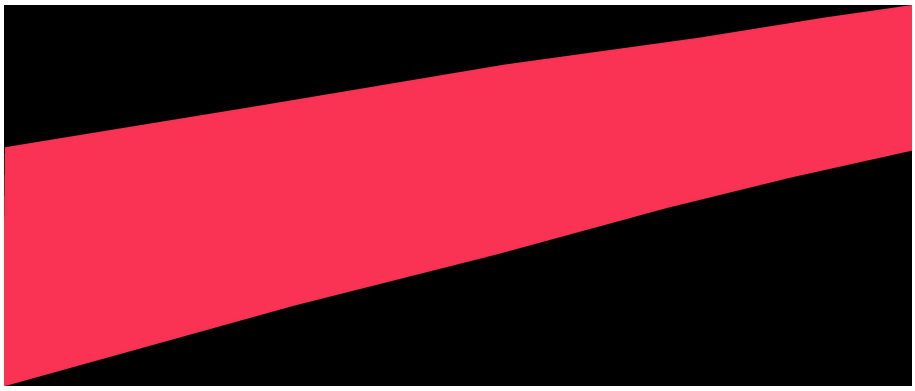

Figure 5.6: Cropped image from corners of mask.

To check the percentage of fence in the image, each pixel of the fence mask is added up and divided by all the pixels in the image. If the percentage higher than the desired percentage, the y-values of the bounding box will be increased to include more background in the image until the desired amount of fence is achieved.
Splitting the images into smaller pictures increased the amount of training data which resulted in a larger dataset, called the Split Dataset, see Table 5.1.

Table 5.1: Contains the number of images in the datasets and how many of the images are divided into training- and validationsets.

Dataset	Trainingset	Validationset	Total images
Annotated Dataset	260	64	324
Split Dataset	1788	64	1852
Split & Augmented Dataset	6963	64	7027

The second method to augment the dataset was to create new images from existing ones. This was done by transforming the existing image using functions in the Torchvision library in Python. The images were transformed randomly using functions that flipped and rotated the image, creating a similar-looking new image. First, the image and mask were merged and then transformed, to perform the same random transform on both mask and image, assuring not to mismatch the corresponding pair. Due to the random transformation of the new image, multiple new images could be created from the existing in the dataset. Combining the two methods of augmenting the dataset, resulted in a new dataset that was called Split & Augmented Dataset , see Table 5.1.

5.1.2 Model

There are many different neural network models for image segmentation, one of them is the U-Net model which was used for this project. The choice of model was due to the fact that the model is well documented, has good performance and is not too complex. The model consists of two parts, an encoder and a decoder [33][34] see Figure 5.7. The encoder is made of a series of high-to-low convolutional layers that reduce the spatial dimensions while increasing the number of image channels. The process captures high-level features in the image and the context. The second part

of the model is the decoder which recovers the initial dimensions of the input image, by having a series of upsampling layers. It also decreases the number of feature channels and preserves low-level features. Between the encoder and the decoder, there are connections that allow both high and low details to be taken into account in the model's predictions.

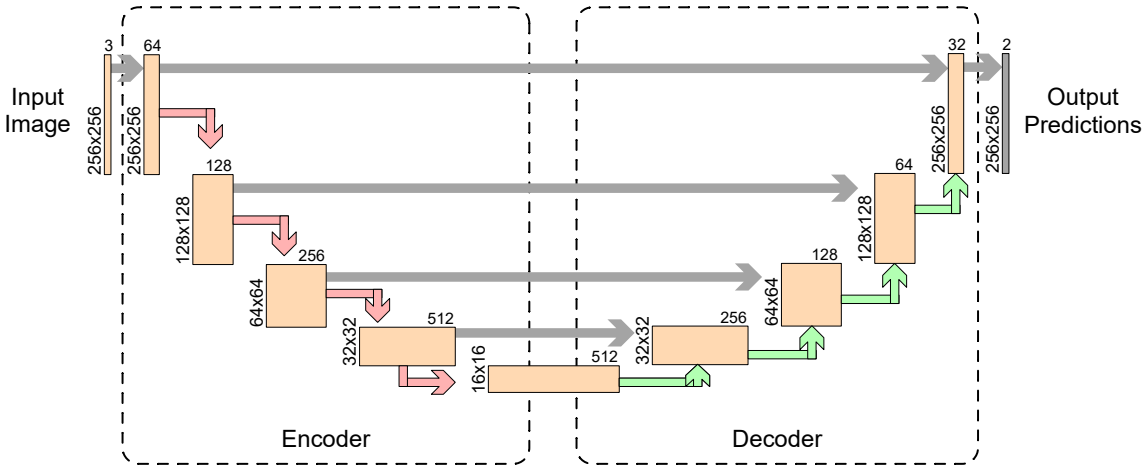

Figure 5.7: Illustrates how the layers of the U-Net model are interconnected and how the input image is processed through the network. The image dimensions is shown to the left of each layer as (height × width), the number of channels is shown above each layer.

The project model captures images with dimensions of 256 × 256 pixels (height × width), with 3 channels as input. The first layer of the model is a convolution layer that does not change the image's dimensions but increases the number of channels to 64 channels, see figure 5.7. The encoder then consists of four down layers illustrated as the red arrows. Each down layer consists of a Max Pooling function that halves the image size and of a convolution layer that doubles the number of channels for the image, with the exception of the last layer that keeps the number of channels and only reduces the image size. The decoder also has four layers but instead uses Up-layers, illustrated as the green arrows, which use a function to resample the image from the layer below to double its size.

Each Up-layer also has a skip connection which forwards an image from a Down-layer, these are illustrated as the grey arrows in between the layers. The layer now has two images of the same size which it merges, the Up-layer then uses a convolutional layer that combines the two images' features and reduces the channels. Each Up-layer, therefore, doubles the image dimensions and halves the number of channels. The last layer in the model consists of a convolution layer that transforms the result into a probability vector. In our case, the model classifies two classes, background and fence, thus the model results in a tensor for each image with the same dimensions 256 × 256 pixels and 2 channels that show the probability for each pixel to belong to background or fence.

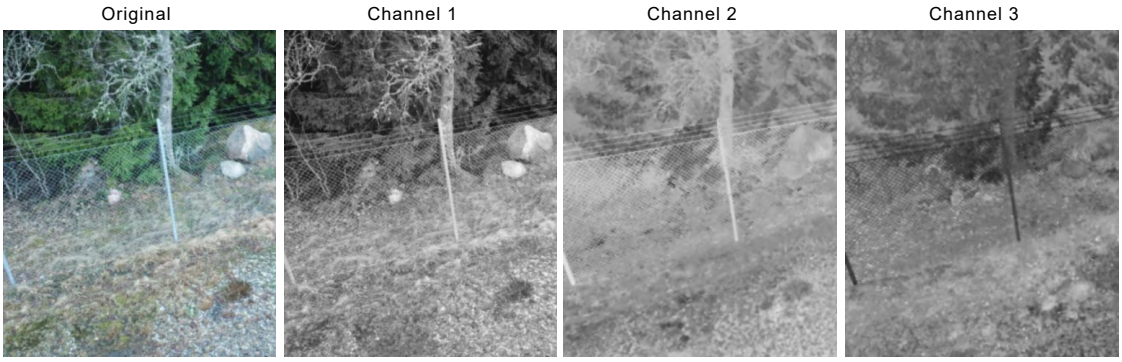

Figure 5.8: Showing the original image split up into different channels, which shows how different channels can highlight different features in the image.

By dividing the image into more and more channels, the image is broken up into smaller and smaller details, where each channel describes different details in the image. This allows the model to train on more specific features in the image and has the possibility to learn more. As illustrated in Figure 5.8, the original image with 3 channels can be represented with black and white images that represent different aspects in the image.

5.1.2.1 Training of the model

Due to the large size of the model used training the model it takes a large amount of computing power to train, therefore the training was done on the C3SE Vera cluster[35]. The cluster itself consists of one main scheduling CPU and multiple other CPU's and GPU's for running jobs on. The scheduling CPU is the one that all users of the cluster communicates with and submits jobs. A job consists of a bash script that defines the parameters of the job.

```
#!/usr/bin/env bash
#SBATCH -A C3SE2023-2-4 -p vera
#SBATCH -N 1 --gpus-per-node=A40:1
#SBATCH -t 0-06:00:00
```

The first part of a jobscript defines what project the project is for, the CPU/GPU used and the run time. In this case one node with one A40 GPU is chosen. The code is set to run for a maximum of 6 hours.

```
echo "Loading packages"
ml load PyTorch/1.12.1-foss-2021a-CUDA-11.3.1
ml load matplotlib/3.4.2-foss-2021a
```

This next part of the script loads the python packages needed to run the code. The PyTorch library is loaded because the model itself is created with PyTorch and the matplotlib library is loaded for visually showing guesses from the trained model.

```
cp -r Dataset $TMPDIR
echo "Dataset copied to temp..."
cd $TMPDIR
echo "Script starting..."
python3 ~/image_Recognition/main.py

```

```
echo training done, saving model
time=$(date +%Y%m%d_%H%M%S)
mv best_model.pth model_${time}.pth
mv history.csv history_${time}.csv
cp history_${time}.csv $SLURM_SUBMIT_DIR
cp model_${time}.pth $SLURM_SUBMIT_DIR
sleep 1
```

This last part of the script copies the Python files and the dataset to a temporary directory on the GPU node. This is done to speed up the read-write times when training the model. The Python file is then run to train the model. This training continues until a set amount of epochs are reached, or if the time allocated runs out. When the training is done the best model is saved and all the history of the train and validation data for each epoch.

The model and the training scripts were written in Python using the Pytorch library. The variables chosen for the training are listed below.

```
    #Training parameters
    num_epochs = 100
    batch_size = 64
    learning_rate = 0.001
    num_workers = 16
```

The number of epochs controls for how many epochs the training will continue. Each epoch consists of all images being passed through the model and evaluated. From this evaluation the weights and biases of the neural network is tweaked to lower the loss function.

The batch size determines how many images are processed at once. The dataset will be split into chunks of the batch size and processed one chunk at a time. The magnitude of the batch size is often chosen as the largest the computing unit used can handle. In general, the larger the batch size the faster will the network train.

The learning rate is the step size used for minimising the loss function. This can be seen as how fast the network learns from the data. If this value is too high the network may start to diverge. However if the learning rate is too small the network will converge very slowly, and may get stuck on a suboptimal solution.

$$Accuracy = \frac{True\ positive\ pixels + True\ negative\ pixels}{All\ pixels} \tag{5.1}$$

To be able to evaluate the training of the model, both loss and accuracy are printed when running the software. When calculating the accuracy, all pixels are taken into account using the Equation 5.1.

5.2 Results

In this section the results gathered from the trained U-Net models are presented. The results are split into two sections depending the dataset used. The datasets are from Section 5.1.1. The two used are the Split dataset and the Split & Augmented dataset. For all the training the hyperperameters in Table 5.2 were used.

Table 5.2: The hyperparameters used during training of the models.

Batch size	64
Learning rate	0.001
Optimizer	Adam

5.2.1 Model trained on the split dataset

In this section the results from the training of the model on the split dataset are presented. The accuracy of the model during the 60 epochs trained can be seen in Figure 5.9. In this graph the training and validation accuracy both start at around 90% accuracy and increases to above 98% accuracy during the 60 epochs. The final validation accuracy reaches 0.9907 and the final training accuracy reaches 0.9908.

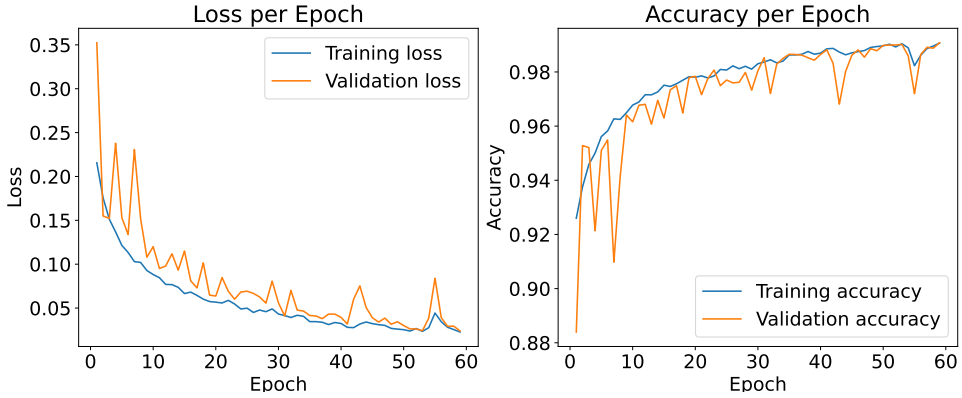

Figure 5.9: Left plot illustrating validation and training accuracy for 60 epochs on the Split Datset. Right illustrates training and validation loss for 60 epochs of training on the Split Datset.

The loss follows a similar pattern, as the accuracy can be seen in Figure 5.9. The validation and training loss starts above 0.2 and during the training decreases below 0.05. However the training loss does decrease more rapidly in the beginning of the training compared to the validation loss. The final validation loss was 0.0240, and the final training loss was 0.0227.

Figure 5.10 shows some predictions made by the model on validation images from a top view perspective. In the images the model results vary quite largely depending on the image analysed. In both images much of the fence is detected, however a lot of the background is incorrectly classified as fence.

Figure 5.10: Predictions made by the model trained on the Split dataset for 60 epochs, on top view images of fences.

Figure 5.11 shows predictions made by the model on side view images from the validation data. In these images most of the fence is detected but similarly to the top view images a large amount of background is classified as fence.

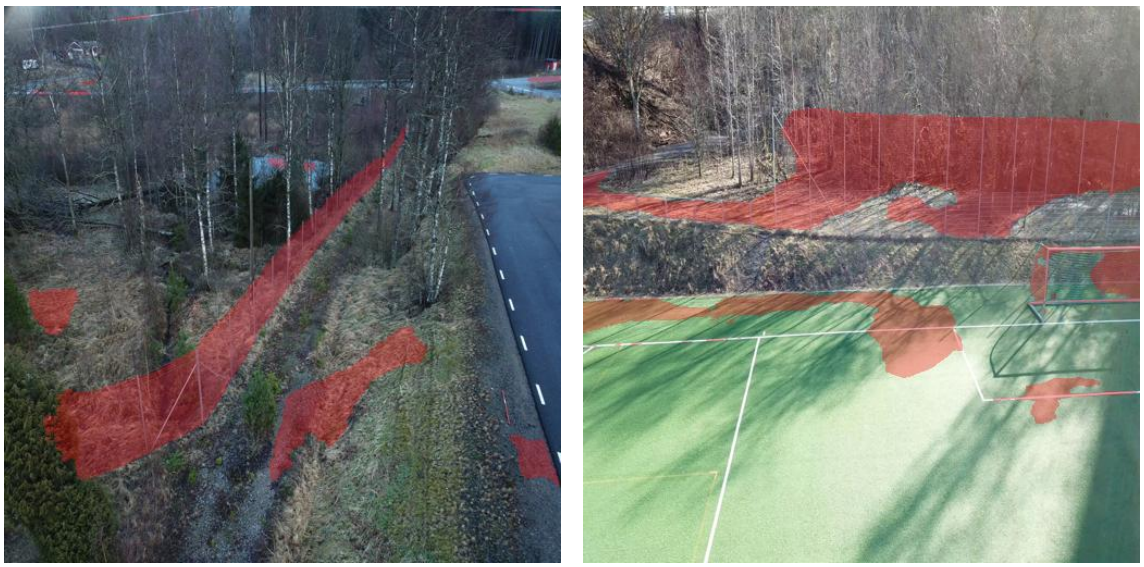

Figure 5.11: Predictions made by the model trained on the Split dataset for 60 epochs, on side view images of fences.

5.2.2 Model trained on the Split & Augmented Dataset

In this section the results from the training on the Split & Augmented dataset is presented. In Figure 5.12 the accuracy per epoch is plotted. The accuracy for both the training and validation data start above 92% and increases gradually to above 98%. The validation accuracy does fluctuate a lot more than the training accuracy which can be seen in epoch 4 where the accuracy decreases to 82% for one epoch. The final values for the validation accuracy is 0.9918 and the final training accuracy is 0.9932.

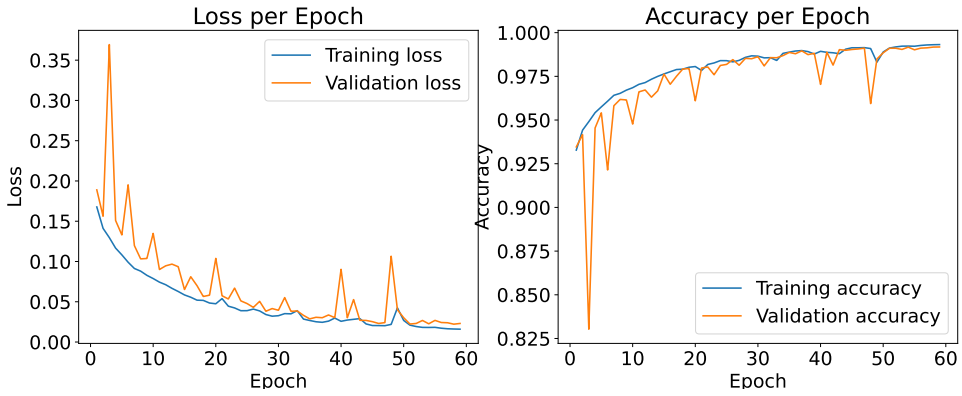

Figure 5.12: Right plot illustrates training and validation accuracy for 60 epochs on the Split & Augmented Dataset. Left illustrates training and validation loss for 60 epochs on the Split & Augmented Dataset.

In Figure 5.12 the training and validation loss are plotted. Both the train and validation loss start below 0.2 and gradually decreases to below 0.03. The training loss does decrease slightly faster than the validation loss, however both start to even out after about 35 epochs. Similarly to the validation accuracy the validation loss does fluctuate quite a lot between epochs. The final values for the validation loss is 0.0231 and the final value for the training loss is 0.0160.

Figure 5.13 shows some predictions made by the model on validation images from a top view perspective. In both images much of the fence is detected but similarly to the model in Section 5.2.1 a lot of background is incorrectly classified as fence.

Figure 5.13: Predictions made by the model trained on the Split & Augmented dataset for 60 epochs, on top view images of fences.

Figure 5.14 shows predictions made by the model on side view images from the validation data. In the left image the fence is almost perfectly classified, but the background is in some areas classified as fence. In the right image almost none of the fence is found.

Figure 5.14: Predictions made by the model trained on the Split & Augmented dataset for 60 epochs, on side view images of fences.

5.3 Discussion

When comparing the two models, the Split & Augmented model did reach a higher accuracy and lower loss than the Split model. This does suggest that using augmented data is a way to increase the accuracy of a model. Another interesting finding when looking at the graphs in Figure 5.9 and Figure 5.12 is that the Split & Augmented model reaches a higher accuracy in less epochs than the Split model. One thing to note however is that due to the increased size of the Split & Augmented dataset, the training times were increased.

Due to secrecy at the test track filming and photographing were only allowed at a certain stretch of the fence. This limited the amount of image material that was collected. This has been a limiting factor throughout the project and has affected both methods and results. The images collected are very similar looking and therefore the neural network model only gets trained on this specific area and has difficulties with generalized data.

Another factor that could potentially limit the model's performance is the compression of the input images. When aerial images captured by the drone are reshaped for efficient processing, they are downsized to a resolution of 256 x 256 pixels (width x height). The purpose of this compression is to reduce the computational burden during training and evaluation. It is important to note that without compression, the model would require more time for training and evaluation due to the larger image size. On the other hand, larger images contain more features and details that the model could potentially learn from. The input size was chosen to accommodate the U-Net model, but in future work, comparing other models that do not compress the images as much could be beneficial.

When evaluating the performance of the model the accuracy is calculated. The accuracy calculation describes how accurate the model is in classifying each pixel, see Section 5.1.2.1. Therefore this score does not solely represent how well the model performs in classifying the fence. With an accuracy of over 99% one could expect the final predictions to be almost perfect. But when looking at the image predictions in Figure 5.14 one can see that the model does suffer from a large amount of false positives. Due to the amount of pixels in the image, the accuracy remains high. For a final model this does pose quite a problem.

For this reason other metrics such as precision, recall, and F1-score should have been considered and calculated. Taking them all into account would give a better representation of the model's overall performance. Precision describes how accurate the model is in its true predictions and would give a perspective on the ratio between true positives and false positives. Recall would proportion the correctly predicted positive classifications out of all predicted positives and F1-score combines precision and recall into a single balanced measure, describing the overall effectiveness of the model in the classification task. In this project, only accuracy was measured because of how the training was implemented in code. This error was not fixed

due to time limitations and therefore no other metrics than accuracy was recorded. These metrics should however be considered in future work.

A choice made in the project was to not augment the validation data. The reasoning for this choice was to keep the validation data as true to the drone footage captured as possible. To build on this, augmenting the data would not have resulted in new validation data and therefore did not seem necessary. Even though augmenting the data does not create new validation images, it may have evaluated how the model works at fences at different rotations and angles. Further, the validation dataset consisted of only 64 images, which can be argued is a very small amount of images to validate a model trained on 1788 or 6963 images. For future work more data should be collected to train and validate the model so that less data augmentation is needed.

5.3.1 Future work

In future work, more data gathering is necessary to improve the semantic segmentation performed on general data. Even though the model reaches a high validation accuracy see Figure 5.12, the small size of the validation dataset and the lack of general data has to be taken into account. Further, the data is from only a specific stretch of the AstaZero test track and a football field. Therefore the model will may not be a generalizable solution for the entire track, or other fences.

Another factor that may be necessary to investigate is how different seasons will impact the performance in of the fence classification. The datasets contain images taken in the early spring in Sweden. Nature variation in Sweden can have a considerable effect depending on the season, where snow, greenery, and autumn can impact the performance of the models. Due to the time limit and the season, gathering images from other seasons has not been possible, limiting the testing of the seasonal effect on the models classifying the fence.

Proven that it is possible to detect fences using semantic segmentation, future development of this fence detection software would be to try to classify breaches and automatically notify the personnel at the test track. Investigating the possibility of acquiring closer footage would be applicable to get a better chance of detecting breaks. Examining this needs to be revised in cooperation with drone path planning and drone control to maintain safety when flying the drone autonomously.

5.4 Conclusion

In conclusion this part of the project has shown that it is possible to detect fence from aerial images taken by a drone. A U-Net model has been created that can detect fences at a 99% accuracy on the validation data used. However, the accuracy can be misleading as no other metric was recorded for the model. When analysing the predictions made by the model, a considerable amount of false positives can be seen.

Another finding was that using augmented data did improve the accuracy of the final model. This improvement however was not very large, while the training times were increased significantly. Lastly, it is important to mention that no effort has been made to implement fence breach detection, which was one of the project's objectives. Furthermore, no interface currently exists for easy analysis of video footage to detect fence breaches.

6
Discussion

This chapter discusses issues and topics that transcend the specific sub-problems discussed in Chapters 3, 4, and 5. As introduced in Chapter 2, having clear interfaces, effective communication, and developing general solutions are crucial for the final solution to be successful.

Throughout the project, the interfaces have been continuously tested. At an early stage, Drone Path Planning and Drone Control decided on the common data format, the CSV-file, which has continued to be used during all development. Even though there have been situations where another type of format for data transfer between the two parties might have been more efficient, it was determined that changing the format could lead to unexpected issues. By testing an interface between two sub-problems, value and resources are invested in its development. Changing this interface devalues the previous testing, making the changes costly. The interface between Drone Control and Fence Detection has not required the same amount of determination as Drone Path Planning and Drone Control. There has been flexibility on Fence Detection's part regarding the file format of the recorded video.

Figure 6.1: Showing possible dangers flying near powerlines and tree tops.

During the project two different solutions were proposed and tested; gathering video footage from above the fence and gathering video footage from the side of the fence. From the results found in Section 4.2.8, both filming above or side on the fence performed equally as good. Drone Path Planning have created algorithms and tools to seamlessly generate waypoint missions. In the end however, the coordinates always represent a geographical point in the terrain. When collecting coordinates, this must be considered, so that the drone will not hit any object such as a tree or a power grid see Figure 6.1. Objects do not only infer the risk of collision but can also disturb the footage. It is much easier to ensure a clear path above the fence

than beside it, mainly because of the absence of vegetation. The current path with coordinates collected by Drone Path Planning only consists of coordinates above the fence. Creating a path for collecting footage of the side of the fence is possible but requires a very precise GPS-signal and extensive manual supervision at the actual fence location.

On the other hand, in Chapter 5 the U-Net model does produce better results on side view images compared to top view images. It is important to note however, that the model demonstrates good performance on images captured from a top view perspective and has the potential to further improve with additional training data.

6.1 Ethics/Integrity

During the project development, consideration of the ethical aspect influenced the work. Being conscientious was significant to maintain safety around the flight, consider the sensitivity of collecting data, and reduce the possibility of misusing the surveillance drone.

When using a drone to collect data in terms of imagery, there is a possibility to capture more than intended due to a oversight depending on camera width, angle and altitude of the drone. During the data collection phase, precautions were made to not collect data that could have been intrusive to an individual's integrity. The photographs and videos collected were only for developing the neural network model.

The purpose of the surveillance drone is to simplify the safety checks around the fencing at AstaZero's test track. But there is a possibility of misusing the final product. In the wrong hands, surveillance drones could collect sensitive data on individual or a societal scale. The project only intended to use image recognition for the detection of fence and fence breaches, which limits the possibilities of malicious use. Further developing the image recognition to detect more objects was possible, but was not explored to diminish the prospect of further building on this project with intentions of misuse.

For bystanders to this project, it may be difficult to determine the purpose of an unknown flying drone. The challenge of locating the drone operator and its intentions can cause an invasive feeling to bystanders. When operating the drone, being seen is needed to convey intentions and could have been eased by using a high-visibility jacket and explaining our work to bypassers. These simple steps could have helped reduce the anxious feelings outsiders could have received from the flying drone.

7
Conclusion

The objective of this project was to explore the feasibility of using an autonomous drone to surveil a perimeter fence and utilize artificial intelligence to determine if the fence has been breached. The project concluded that it is feasible to automate a commercial drone to follow a planned path and gather footage of the fence that can be post-processed and analyzed.

In summary, a method was developed to collect arbitrary fence and drone coordinates. A planning algorithm was created that can generate paths within the drone's endurance. The path can be fed into a drone control software used for controlling the drone. It can control the movement, heading, and camera angle of the drone. Further, the product holds software that can split video footage into a set of images and with the use of a U-Net model recognize the appearance of a fence. In its current state, the product can not yet detect anomalies in the fence; it merely detects the existence of the fence. Further development of the neural network will be needed to fully automate the process of detecting anomalies.

The subsystems have effective interfaces which allow for good integration, however, the individual subsystems will need more fine tuning before the concept works as a whole. To access more detailed conclusions, see the respective conclusion to each sub-problem.

To conclude, even though the solution requires more integrated testing to validate robustness. The development asserts the possibility of incorporating a fully functional product, thus succeeding in fulfilling the objective.

Bibliography

[1] AstaZero, *Tracks & facilities*, Nov. 2021. [Online]. Available: https://www.astazero.com/en/tracks-facilities/.

[2] J. Wang, L. Zhang, Y. Huang, and J. Zhao, "Safety of autonomous vehicles," *Journal of Advanced Transportation*, vol. 2020, p. 8 867 757, Oct. 2020, ISSN: 0197-6729. DOI: 10.1155/2020/8867757. [Online]. Available: https://doi.org/10.1155/2020/8867757.

[3] AstaZero, *About us*, Dec. 2021. [Online]. Available: https://www.astazero.com/en/about-us/.

[4] A. Aronsson, private communication, Jan. 2023.

[5] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, Apr. 2015, ISSN: 1476-4687. DOI: 10.1038/nature14542. [Online]. Available: https://doi.org/10.1038/nature14542.

[6] Y. Kim, J. Kang, D. Kim, E. Kim, P. K. Chong, and S. Seo, "Design of a fence surveillance system based on wireless sensor networks," in *Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems*, ser. Autonomics '08, Turin, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, ISBN: 9789639799349. [Online]. Available: https://dl.acm.org/doi/pdf/10.5555/1487652.1487656.

[7] J. Jönsson and F. Stenbäck, "Fence surveillance with convolutional neural networks," M.S. thesis, Halmstad University, Halmstad, Jun. 2018. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1219617&dswid=3355.

[8] DJI, *Dji mavic enterprise 2 users manual v.1.8*, DJI, Shenzhen, Guangdong, China, Apr. 2021. [Online]. Available: https://dl.djicdn.com/downloads/Mavic_2_Enterprise/20210413/Mavic_2_Enterprise_Series_User_Manual-EN.pdf.

[9] GPS.GOV, *Other global navigation satellite systems (gnss)*, Oct. 2021. [Online]. Available: https://www.gps.gov/systems/gnss/.

[10] GPS.GOV, *Gps accuracy*, Mar. 2022. [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/.

[11] *Space weather and gps systems*, en, https://www.swpc.noaa.gov/impacts/space-weather-and-gps-systems/, Accessed: 2023-5-21.

[12] *Google earth*. [Online]. Available: https://earth.google.com/web/@57.77888501,12.77529037,196.26801636a,6502.6931844d,30y,-0h,0t,0r.

[13] *BU-808: How to prolong lithium-based batteries*, en, https://batteryuniversity.com/article/bu-808-how-to-prolong-lithium-based-batteries, Accessed: 2023-4-21, Sep. 2010.

[14] M. Lam, *How lipo battery's performance affected by temperature?* en, https://www.gensace.de/blog/temperature-affect-lipo-battery-performance/, Accessed: 2023-4-21, Jul. 2019.

[15] A. Öhman, *Tall – näst vanligast*, se, https://www.sydved.se/aktuellt/inspiration/djur-och-natur/tall-nast-vanligast, Accessed: 2023-5-21.

[16] *Formulas, Tables & Graphs*, Equation 7.66b, Division of Fluid Dynamics, Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Sep. 2022. [Online]. Available: https://courses.onlineflowcalculator.com/fluidmech/docs/MTF053_Formulas-Tables-and-Graphs.pdf.

[17] DJI, *Dji ios mobile sdk*, https://developer.dji.com/api-reference/ios-api/Components/SDKManager/DJISDKManager.html, 2023.

[18] RI-SE, *Atos*, https://github.com/RI-SE/ATOS, 2023.

[19] RI-SE, *Rise-drones (dss)*, https://github.com/RISE-drones/rise_drones, 2023.

[20] RI-SE, *Dss-dji-ios*, https://github.com/RISE-drones/DSS-DJI-iOS, 2023.

[21] T. Bajaj, *Program for distance between two points on earth*, Feb. 2023. [Online]. Available: https://www.geeksforgeeks.org/program-distance-two-points-earth/.

[22] A. Upadhyay, *Formula to find bearing or heading angle between two points: Latitude longitude*, Nov. 2022. [Online]. Available: https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/.

[23] K. Salman, R. Hossein, S. Syed Afaq Ali, B. Mohammed, M. Gerard, and D. Sven, *A Guide to Convolutional Neural Networks for Computer Vision.* (Synthesis Lectures on Computer Vision). Morgan & Claypool Publishers, 2018, ISBN: 978-1-68173-022-6. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8295029&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[24] OpenCV, *Opencv modules*, Accessed: 2023-4-14. [Online]. Available: https://docs.opencv.org/3.4/index.html.

[25] J. Shotton and P. Kohli, "Semantic image segmentation: Traditional approach," in *Computer Vision: A Reference Guide*. Cham: Springer International Publishing, 2019, pp. 1–4, ISBN: 978-3-030-03243-2. DOI: 10.1007/978-3-030-03243-2_251-1. [Online]. Available: https://doi.org/10.1007/978-3-030-03243-2_251-1.

[26] Y. Liu, R. Nan, and W. Feng, "Lane line detection based on opencv," in *2022 7th International Conference on Intelligent Informatics and Biomedical Science (ICIIBMS)*, vol. 7, 2022, pp. 301–304. DOI: 10.1109/ICIIBMS55689.2022.9971627.

[27] OpenCV, *Hough line transform*, Accessed: 2023-4-5. [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html.

[28] M. Solilo, W. Doorsamy, and B. S. Paul, "Uav power line detection and tracking using a color transformation," in *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2021, pp. 1–5. DOI: 10.1109/ICECET52533.2021.9698499.

[29] I. B. M. Y. Wirawan, I. M. G. Sunarya, and I. M. D. Maysanjaya, "Semantic segmentation of rice field bund on unmanned aerial vehicle image using unet," in *2022 14th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2022, pp. 211–216. DOI: 10.1109/ICITEE56407.2022.9954091.

[30] D. J. Pangal, G. Kugener, S. Shahrestani, F. Attenello, G. Zada, and D. A. Donoho, "A guide to annotation of neurosurgical intraoperative video for machine learning analysis and computer vision," *World Neurosurgery*, vol. 150, pp. 26–30, 2021, ISSN: 1878-8750. DOI: https://doi.org/10.1016/j.wneu.2021.03.022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1878875021003909.

[31] CVAT, *Computer vision annotation tool*, https://www.cvat.ai/. [Online]. Available: https://www.cvat.ai/.

[32] M. Aljabri, M. AlAmir, M. AlGhamdi, M. Abdel-Mottaleb, and F. Collado-Mesa, "Towards a better understanding of annotation tools for medical imaging: A survey," *Multimedia Tools and Applications*, vol. 81, no. 18, pp. 25 877–25 911, 2022. DOI: https://doi.org/10.1007/s11042-022-12100-1.

[33] J. Wang, "High-resolution network," in *Computer Vision: A Reference Guide*. Cham: Springer International Publishing, 2020, pp. 1–5, ISBN: 978-3-030-03243-2. DOI: 10.1007/978-3-030-03243-2_885-1. [Online]. Available: https://doi.org/10.1007/978-3-030-03243-2_885-1.

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28.

[35] *Vera - C3SE*, en, https://www.c3se.chalmers.se/about/Vera/, Accessed: 2023-5-4.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS