University of Manitoba
Department of Electrical and Computer Engineering

Ground Penetrating Radar Data Collection From an Autonomous Drone

ECE 4600 Group Design Final Project Report

Prepared by Group 10:

Rafid Javid Scott Murray Vrushil Patel
Noah Wiens Darren Xie

Academic Advisors:

Dr. Philip Ferguson
Dr. Ian Jeffery

Final report submitted as per the requirements for the degree of Bachelor of Science in Electrical and Computer Engineering

Submission Date:
March 18, 2022

Abstract

Ground penetrating radars (GPRs) are devices capable of imaging material under solid surfaces, such as the ground or ice. These devices are typically moved by a human operator. However, in environments that are unsafe for humans, GPRs are not able to be used. In such circumstances, it would be ideal if a drone could propel the GPR instead. This project is to serve as a proof of concept for such a system. We have developed an autonomous drone system capable of automatically plotting a path around an imaged area, avoiding obstacles, and then flying the planned path. Additionally, we have developed an interpretation module that can identify the presence of objects during the drone's flight.

We developed two separate systems: a down-scaled drone subsystem and an at-scale GPR subsystem. The drone subsystem contains four modules: Obstacle Detection, Path Planning, Object Tracking, and Drone Control. These are responsible for detecting obstacles to avoid, plotting a route for the drone, tracking the drone's payload, and piloting the drone itself. Testing for the drone subsystem took place in STARLab (Space Technology and Advanced Research Laboratory). We validated that the system performs to our expectations and that the drone is capable of flying autonomously in such a way as to have its payload cover the area of interest.

The GPR subsystem contains the Data Collection module and the Data Interpretation module. These are responsible for collecting and interpreting the data obtained from using the GPR. Testing for this subsystem took place at the SERF (Sea-ice Environment Research Facility) testing location. A GPR was pulled across ice where pig carcasses lay underneath. An algorithm was developed to automatically determine whether certain scans of the GPR contained any objects of interest. Testing revealed that this subsystem met our expectations and is capable of recognizing objects of interest underneath the layer of ice.

Overall, our system was successful in demonstrating a proof of concept for an autonomous system capable of controlling a drone to obtain and interpret GPR information. We believe that such a project can serve as an example of future developments for similar systems.

Acknowledgements

We want to take the time to thank the following individuals for their mentorship and guidance throughout all aspects of our project.

First and foremost, we would like to express our sincere gratitude to our advisors, Dr. Philip Ferguson and Dr. Ian Jeffrey, for their constant support, guidance and encouragement. We would not have been able to achieve our goal without their extensive technical knowledge. We would also like to thank Dr. Ferguson for providing the vast majority of components used in this project.

Additionally, we would like to thank Mitesh Patel and Ali Barari from STARLab for their technical assistance with the Quanser QDrone and Vicon Motion Capture System. We would also like to thank Dr. Gordon Giesbrecht for helping us with the testing along with allowing us to use the data that had been recorded for his research project.

Finally, we would also like to thank James Dietrich, Dr. Derek Oliver, and Aidan Topping for their feedback and support throughout the year.

Contributions

Task	Responsibilities				
	Rafid Javid	Scott Murray	Vrushil Patel	Noah Wiens	Darren Xie
GPR Data Collection Module			●		
GPR Data Interpretation Module		○	●		
Obstacle Detection Module		●			
Path Planning Module					●
Object Tracking Module	●				
Drone Control Module				●	
GPR System Testing and Validation	○	○	●	○	
Drone System Integration	●	●		●	●
Payload Creation	○			●	
Drone System Testing and Validation	○	○		●	○
Report Writing	●	●	●	●	●

Legend

Lead: ● Contributor: ○

Contents

List of Figures

List of Tables

Nomenclature

CNN Convolutional Neural Network

CSV Comma-Separated Values

DGPS Differential Global Positioning System

GPR Ground Penetrating Radar

GPS Global Positioning System

IP Internet Protocol

KLT Kanade-Lucus-Tomasi

NIC Network Interfacing Controller

S&S Sensors and Software

SERF Sea-Ice Environment Research Facility

STARLab Space Technology and Advanced Research Laboratory

YOLO You Only Look Once

Chapter 1

Introduction

1.1 Motivation

A ground penetrating radar (GPR) is a device used for collecting data about material and objects that lie beneath a solid surface. Radar pulses sent out from a GPR are collected after they rebound from materials beneath the surface the GPR is surveying. These pulses are then processed into an image representing the subsurface geometry [1]. Depending on the permittivity and conductivity of the subsurface materials, the image can provide useful information about the location of these materials.

The application we are interested in is collecting GPR data in circumstances where it cannot be collected safely by a human operator. Typically, GPR data is collected by a person pushing or pulling the device over the surface of interest, but in some cases where GPR data collection is desired, it is not safe for a human operator. One example is the detection for body recovery below the ice of a frozen river. In the unfortunate case where someone falls through thin ice on a river, it can be difficult to track the body's location, as it may float due to residual air in the lungs and move some distance before coming to a rest. Sending a GPR operator to walk over the thin ice would be far too dangerous. In this case, an automated drone system could drag the GPR over the ice to collect the data needed to determine the location of the body. Figure 1.1 shows a sketch of a system such as this.

The goal of our project is to create a proof of concept for a system capable of GPR data collection using an autonomous drone. We provide a scaled-down drone system capable of dragging a payload along a path that covers a given area of interest while avoiding obstacles. This, in principle, proves the concept for a drone system that could survey a frozen river without damaging the GPR or requiring a human operator. We also provide data analysis on real GPR data collected from a frozen ice tank that contains pig carcasses beneath the ice. This demonstrates a method that could detect a body under the ice using GPR data.

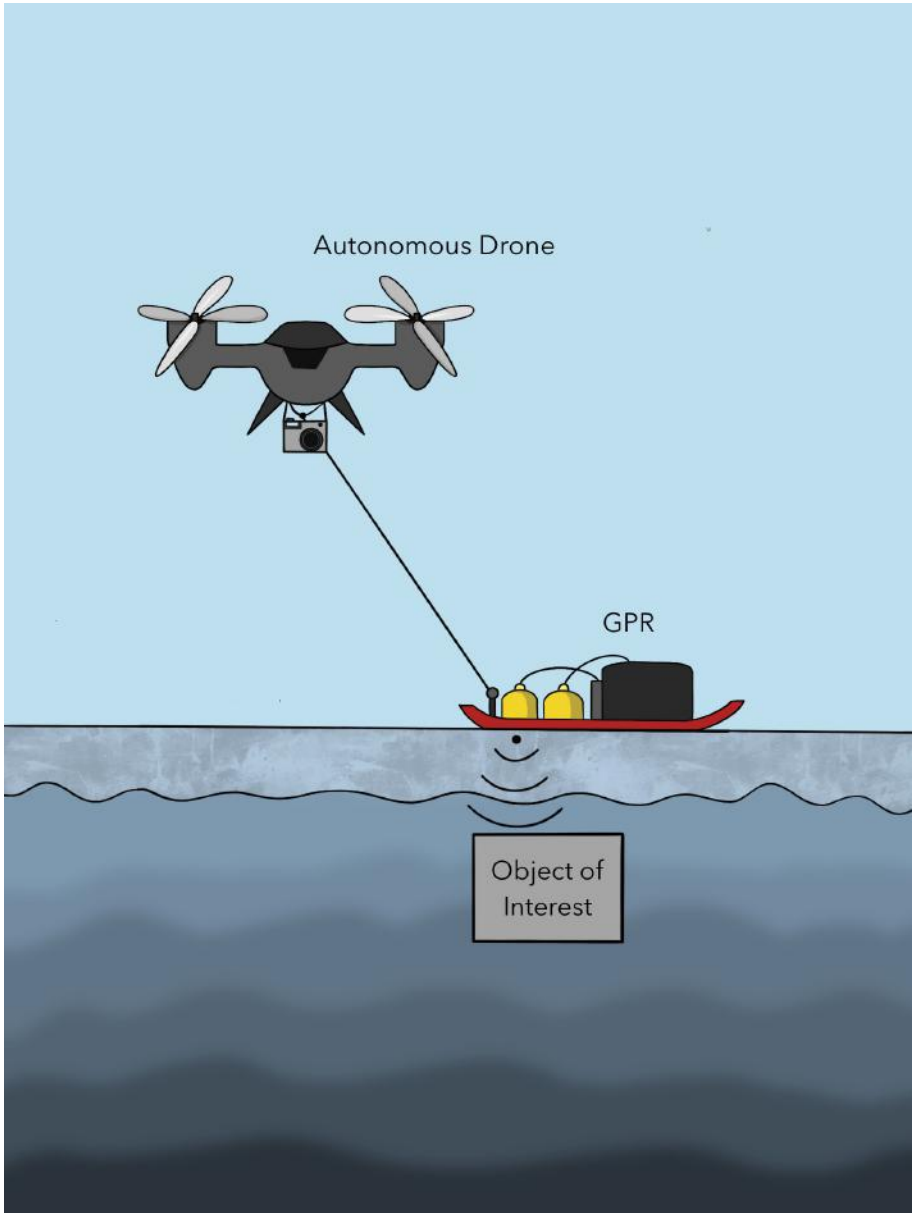


Figure 1.1: Sketch of the autonomous drone-controlled GPR system

1.2 Overview

Our proof of concept consists of an autonomous drone system and a separate GPR system. Our proposed project can be decomposed into six modules split between the two systems. The system flowchart in Figure 1.2 gives a general overview of the project. The GPR system is split into two modules: the Data Collection module and the Data Interpretation module. These modules are responsible for collecting GPR data and interpreting the collected data respectively. The drone system consists of four modules: Obstacle Detection, Path Planning, Object Tracking, and Drone Control. The Obstacle Detection module takes an image of the area of interest to detect obstacles. These obstacles are passed to the Path Planning module, which constructs a path around the detected obstacles. This path is then given to the Drone Control module which is responsible for flying the drone while dragging the payload. During the drone's flight, the Object Tracking module works with the Drone Control module to ensure the GPR is following the given path.

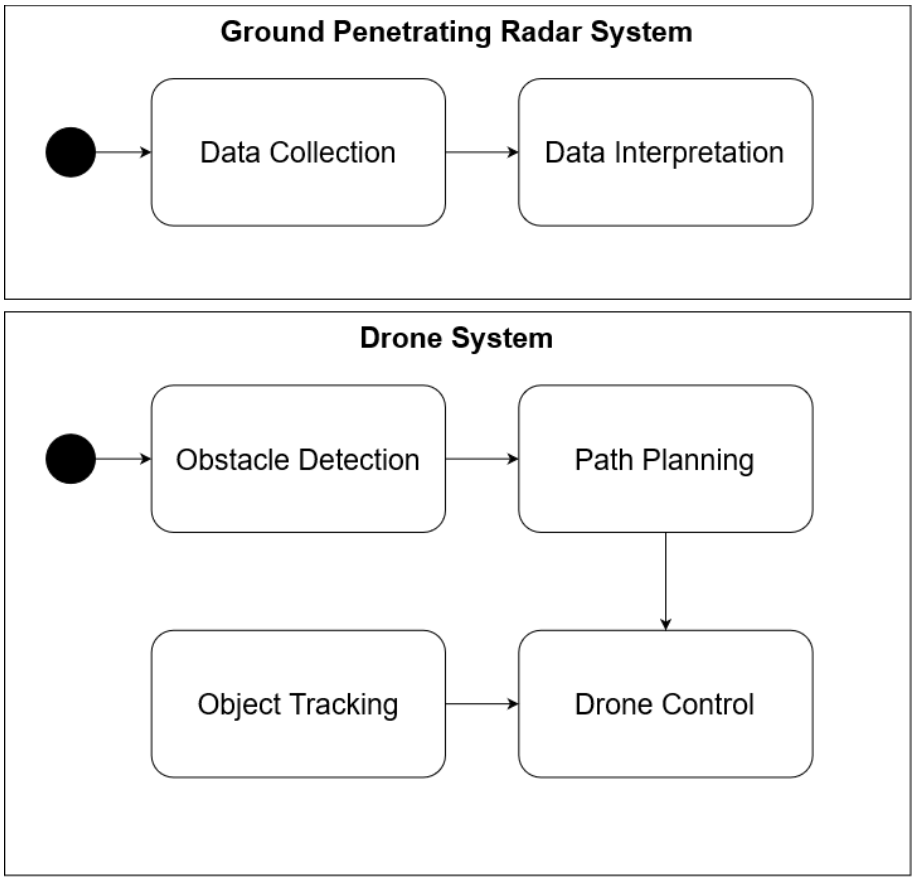


Figure 1.2: System flowchart for the proof of concept provided

This report provides a more detailed description of each module in its respective chapter.

1.3 Project Scope

This project is divided into two major components, which are the autonomous drone system and the GPR system. The scope of the autonomous drone system is limited to cover its scaled-down operation and testing in a lab environment rather than a real-world application. The scope of the GPR system does not have this limitation, as its operation and testing take place at full-scale in the real world.

1.3.1 GPR System

The GPR system has been developed at full-scale and has been used at the Sea Ice Environmental Research Facility (SERF) to gather data from frozen pig carcasses underneath a layer of ice. This data has been processed to test and validate the GPR data interpretation module, as discussed in detail in Chapter 3.

1.3.2 Drone System

The drone system has been developed at a smaller scale than the real-world system would require. This is primarily due to the fact that the drone we had access to is not large enough

to feasibly pull the GPR system along an icy surface. The available drone was a Quanser QDrone, which has a maximum lift capacity of 300g. This drone has an associated drone cage with supporting hardware and software within the STARLab, which was the environment testing was completed in.

1.4 Metrics

Because this project is split into two systems, each with its own scope, the performance metrics for the project are unique for each system. The following subsections will outline the metrics for each system and provide a rationale for the chosen metrics.

1.4.1 GPR Metrics

The GPR system was tested at full scale in a lab scenario that is similar to how it would be used in the field. Table 1.1 outlines the metrics the GPR system aimed to meet.

The GPR system had its tests conducted in a controlled live environment. The metrics shown in Table 1.1 are there to verify our success or failure. The false detection and detection rate ensured system reliability when finding the correct objects of interest. Metrics were also determined for the locating accuracy and GPR frequency of the 500 MHz and 1000 MHz antennas. These metrics guaranteed consistency in the data recorded by the antennas from one set of tests to the next. The metric for the data acquisition speed was set to mimic the measurements taken as if a user were physically walking over the ice. Thus, we set a target speed at which we pulled the GPR over the ice to meet this metric. The data interpretation metric was established to test the location accuracy and resolution of the Data Interpretation module. The goal of this metric is to compare the output visual of our Data Interpretation module with the real location of the pigs placed in the tank. The target length is the threshold we wanted to be below when comparing the relative positions of the pigs, to the positions received from our visual outputs.

Table 1.1: Metrics for GPR System

Feature	**Metric**	**Target**
Data interpretation	Location accuracy/resolution	<60 cm
Detection	Object of interest detection rate	>80%
False detection	False detection of object of interest	<10%
500MHz antenna locating accuracy	Accuracy of marked location of object of interest	<1 m
1GHz antenna locating accuracy	Accuracy of marked location of object of interest	<0.5 m
Data acquisition speed	Speed at which the GPR travels	1m/s - 1.42 m/s
GPR frequency	Center frequency of GPR	500 MHz & 1 GHz

1.4.2 Drone System Metrics

The drone system is a scaled-down proof of concept for the real-world system capable of dragging a GPR. Tests for the drone system were carried out in an indoor lab environment with a Vicon system. This testing environment will allow for more precise margins in our metrics.

Table 1.2 outlines the metrics for the drone system. In lab tests, the area of interest was a subsection of the drone cage, and thus was expected to be trivial to map fully. Path planning was also simple within this area because of the simplicity of the obstacle layouts in a lab test. Hazards were modelled with pieces of white paper which have a large amount of contrast with the black floor, so we aimed for high detection rates, and less than one hazard encountered out of 100 tests. The camera of the drone can record at 30 frames per second, so a target per-frame processing time for the Object Tracking module of 33ms allows for real-time analysis of the drone video feed. The proximity to each waypoint metric was decided to be 5cm to allow for the GPR payload to follow the path closely without the drone requiring to slow down significantly to get very close to each waypoint. The time to begin correcting the drone's path to compensate for box drift was chosen to be 40ms so that the GPR will not drift far off the desired path before corrections are made. The drone battery life metric of 8 minutes was decided based on the specifications of the Quanser QDrone; the aim is to reduce any unnecessary load on the drone that may cause the battery to drain quickly.

Table 1.2: Metrics for Drone System

Feature	**Metric**	**Target**
Path detection	Percentage of area of interest mapped by algorithm	>99%
Path planning	Percentage of area of interest covered by GPR	>99%
Obstacle detection	Percentage of obstacles detected in lab tests*	>99%
Hazard avoidance	Rate of hazard encounter	<1/100 hazards
Object tracking	Per-frame processing time	<33ms
Drone control	Maximum drone proximity to each waypoint	5cm
Drone control	Time to begin correcting drone path to compensate for box drift	<40ms
Battery life	Time between required battery change	≥8 minutes

* Metric added after design review 2

Chapter 2

Ground Penetrating Radar – Data Collection

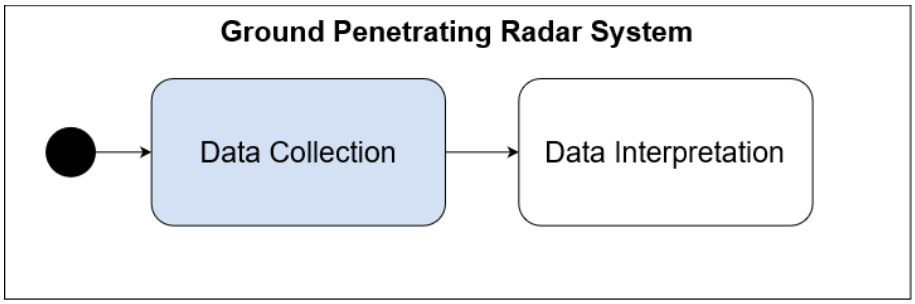


Figure 2.1: Data Collection module in the overall system

The GPR Data Collection module is responsible for collecting data using the PulseEKKO GPR. The device takes measurements of the reflected electromagnetic pulses from the surface below it. It then takes this input and produces an image for us to interpret in an effort to determine what lies underneath. In this section we will further discuss the objective, theory, assembly, and testing of the Data Collection module.

2.1 Overview

The objective of the GPR Data Collection module is to use the GPR system to collect data through controlled live environment tests. The GPR system was provided to us by STARLab; it did not require us to order any additional parts. The controlled live environment tests involved placing pig carcasses in a frozen ice tank. To conduct the tests, the GPR system was dragged across the ice to collect data. A variety of different tests were conducted between the months of November 2021 to February 2022. The data sets collected from these tests gave us working data to use as input for our Data Interpretation module. This data also allowed us to determine some of our main metrics early on such as the detection and false detection rates at 85.71% and 15.38% respectively.

2.2 Ground Penetrating Radar Theory

2.2.1 GPR Fundamentals

The GPR system sends a pulse of high-frequency electromagnetic waves, specifically radio waves, into the earth via the transmitting antenna [2]. The pulse consists of electromagnetic waves oscillating around a centre frequency determined by the antennas that are attached to the GPR system. Based on the electromagnetic properties present in the material below the GPR, the waves can be reflected, refracted, or transmitted [2]. The signals that are reflected and measured by the receiving antenna are then used by the GPR system to generate an image known as a radargram. In general, GPR data is usually recorded from several spatial positions by dragging the antenna along the ground [3].

2.2.2 Detection

To detect an object of interest below the surface of the GPR, a radar scan is needed. A radar scan encompasses an operator dragging the GPR between two points over the object of interest. After taking the radar scan, the image can be visualized using the software that comes with the GPR system. This visualization is called a radargram.

The theory behind how a radargram is generated is as follows. From a fixed position, a radar scan consists of emitting a bandwidth of frequencies around the operating frequency, as shown in Figure 2.2. The operating frequency is provided by the type of antenna that we select. For our project, this would be either the 500 or 1000MHz antennas that are available for the Sensors and Software GPR system. This scan, referred to as an "A-Scan", will provide information about the subsurface at a single point. However, the beam of the antenna has a finite width, so reflections from objects that are not directly below the GPR are also received.

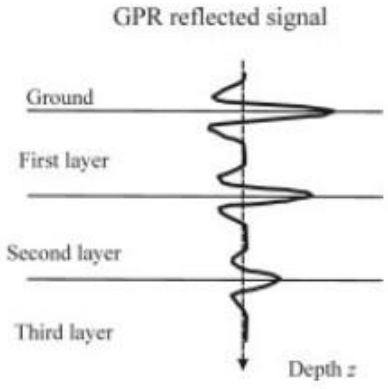


Figure 2.2: A-Scan [4]

To generate more information about the subsurface, an operator can walk the GPR system along a line as shown in Figure 2.3 . This effectively results in stacking A-Scans together to form a "B-Scan" or a radargram as shown in Figure 2.4.

Figure 2.3: Procedure to get hyperbolic reflection

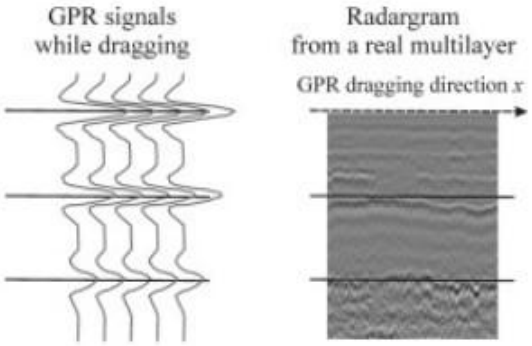


Figure 2.4: Multi-layer radar scan [4]

2.2.3 Range

The GPR system's range, a capability that refers to its penetration depth, is dependent on the frequency of the interrogation energy. We used the 500MHz and 1000MHz antennas in our system which corresponds to a depth range of 0.5m and 1m respectively [5]. Thus our metrics for the locating accuracy of our antennas were established based on these figures. A table referencing the centre frequencies compared to depth is provided by the manual included with the GPR as shown in Figure 2.5. It is important to note that choosing lower frequencies reduces the resolution of the radargram produced [5].

Center Frequency (MHz)	Depth (m)
1000	0.5
500	1.0
250	2.0
100	5.0
50	10
25	30
12.5	50

Figure 2.5: Center frequency vs. depth table

2.3 Ground Penetrating Radar Assembly

The GPR system used for this project consists of three primary components: the transmitting and receiving antennas, the SPIDAR Network Interfacing Controller (NIC), and the battery pack. All components were manufactured by Sensors and Software, and were provided by STARLab.

We used two different sets of antennas shown in Figure 2.6. One pair corresponds to a center frequency of 500MHz and the other to 1000MHz. Since this is a bi-static radar system, each set contains a transmitting and receiving antenna. The use of these two different antennas helped in determining multiple metrics such as detection, false detection and locating accuracy.

Figure 2.6: Top: 500 MHz antennas —— Bottom: 1000 MHz antennas

In the GPR system, both the antennas and the battery pack are connected to the SPIDAR NIC. The NIC has two main functions. The first function is to provide an online interface for an operator to remotely calibrate and trigger the GPR system to conduct radar scans. This function was predominately used by our team throughout our testing phase. The second function is to allow the GPR user to connect either one or two pair of antennas to get a wider range of coverage for the radargram [6]. We did not utilize this function for our project due to our metric of only having one centre frequency at a time for testing. The initial build of the GPR system with all parts connected is represented below in Figure 2.7.

Figure 2.7: First Iteration of the GPR build

To safeguard the expensive machinery from water and snow on the ice, the entire GPR system was placed in a vessel such as the one shown in Figure 2.8. As more tests were conducted, a more sophisticated vessel was constructed to hold the GPR system and allow more mobility on the ice. This final version is shown in Figure 2.9.

Figure 2.8: Second iteration of the GPR build

Figure 2.9: Final iteration of the GPR build

2.4 Ground Penetrating Radar Testing

2.4.1 Purpose

The assembled GPR system was used to conduct initial testing for detecting bodies trapped beneath the ice. This testing was performed in a controlled environment at the Sea-Ice Environment Research Facility (SERF) from November 2021 to February 2022. The purpose of this testing was to collect enough data to prove the success or failure of the subsystem, to meet our metrics, and to acquire sufficient data to use as input for the Data Interpretation module.

2.4.2 Approach

To verify the functionality of the GPR module for this proof of concept, it was vital to conduct a test that mimics a real-life situation and environment as close as possible. As a result, we used pig carcasses and a large tank filled with water. Pigs share similar physiological traits to humans, which allows our tests to accurately replicate a real-life scenario [7]. The large tank facilitated a controlled area where we are allowed to conduct a multitude of tests without any discrepancies in the data due to external factors. The tank remained outdoors to keep the water frozen.

Prior to submerging the pigs below the top layer of ice, it was required for them to be intubated so that their lungs could be inflated. This was necessary as human bodies float when air is in the lungs, and the change in the electromagnetic properties between tissue and air would help the GPR better locate the pigs under the ice. One of the modifications made was to have one pig submerged with a jacket while the other is placed in bare skin. This modification was used to further mimic a real-life situation and to see what difference there would be in terms of identifying the pigs with and without a jacket.

A single radar scan was conducted in the following manner. Segments were marked on the short side of the tank. Each segment was approximately 6 inches wide. This was the width of the 1000MHz antenna. For the 500MHz antenna, we used segments that were 9 inches wide. The GPR system was placed in the corner of the tank and pulled across, going from one end of the tank to the other. This procedure was repeated until all segments of the tank have been scanned. A diagram of this procedure is shown in Figure 2.10.

The procedure to get one radar scan of a 6-inch-wide segment involved three operators. The first operator was responsible for triggering the GPR system to start and stop a radar scan via a laptop. The second operator was on one end of the tank, and was responsible for pulling on the harness attached to the GPR system and dragging it across at a steady rate. The third operator's role was to ensure the GPR stayed in a straight path while it is conducting the radar scan. This helps by allowing us to collect accurate data without too many discrepancies. By changing the order of the transmit and receive antennas, we determined that radar scans are not affected by their orientation, and so it does not matter in which direction the GPR is pulled by the second operator. It is important however for the second operator pull the GPR at a steady pace. This is to help us achieve our metric of

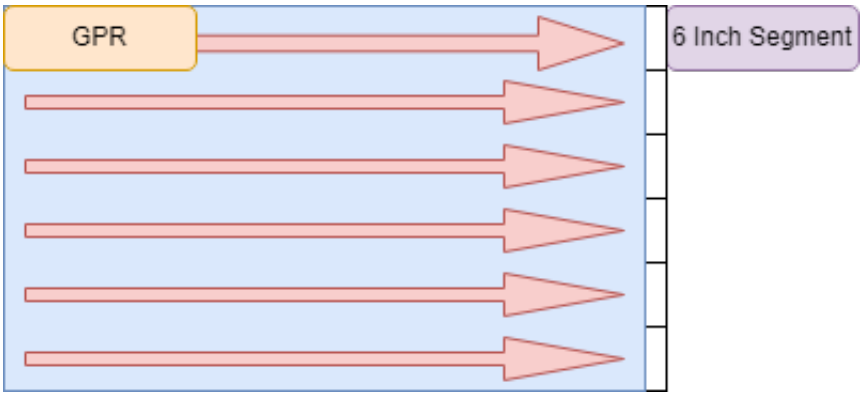


Figure 2.10: Radar scan diagram

Figure 2.11: Snapshot of radar scan

being able to drag the GPR at a steady walking speed of 1 meter per second. If the path of the GPR is clear as seen in Figure 2.11, then there is no need for a third operator.

2.4.3 Testing Results

From November 2021 to February 2022, multiple tests were conducted in order to test for a variety of the metrics outlined in Table 1.1. Variations of the same test were conducted to test different metrics.

The first metric that the team was able to achieve was the object of interest detection rate. This metric measures the amount of times we can detect the pigs under the ice over the course of all tests. The way this was measured was by examining the radargrams of all

the locations we knew the pigs were supposed to be in and comparing them to see if they were actually present.

An example of a radargram with proper locations of the pigs is shown in Figure 2.12. The target that we set for this metric was 80% or greater and the team was able to achieve a result of 85.71%. The results were obtained by viewing the radargrams that were taken over the actual location of the pigs. With 21 total radargrams which were taken right above the pigs, we had 18 showing us a clear presence of them, thus resulting in a 85.71% detection rate. With the completion of this metric, we were also able to prove the success of the antenna locating accuracy metric by re-observing the same radargrams used with their respective antennas. To satisfy this metric, we noted the depth obtained from all the detected locations of the pigs and confirmed that each depth was within the threshold.

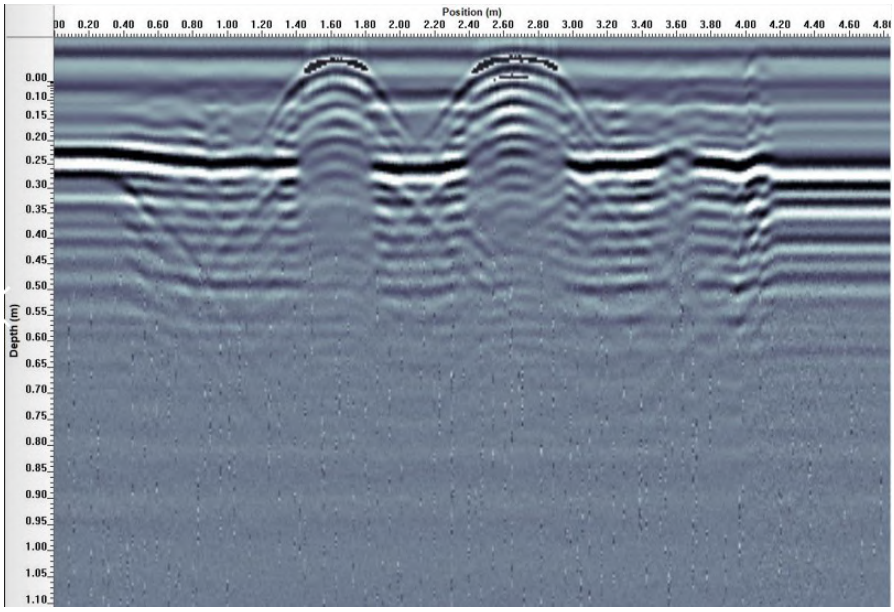


Figure 2.12: Radargram with Pigs Present

The second major metric set out to achieve was the false detection rate. This metric was tested by placing a third pig along with a medium piece of plywood nearby. From viewing the radargrams of all the tests conducted following this procedure, we have concluded that the false detection rate did not pass our original target of less than 10%. The actual result that the team achieved was that of 15.38%, failing to reach the target by a small amount. A radargram showing the false detection of the plywood can be seen in Figure 2.13. The plywood piece can be seen between points 2.55 and 2.75 on the x-axis. Due to a lack of time and resources, this metric could not be tested further with different objects to acquire varied data in false detection.

To summarize, we were able to test the GPR system in a wide variety of tests to see its full capabilities. The tests conducted provided us with sufficient data to use for the Data Interpretation module. Having successfully produced radargrams that appear to provide indications of the locations of the submerged pigs, we can move on to additional processing techniques used to improve interpretation and provide better visualization of the original radargram data.

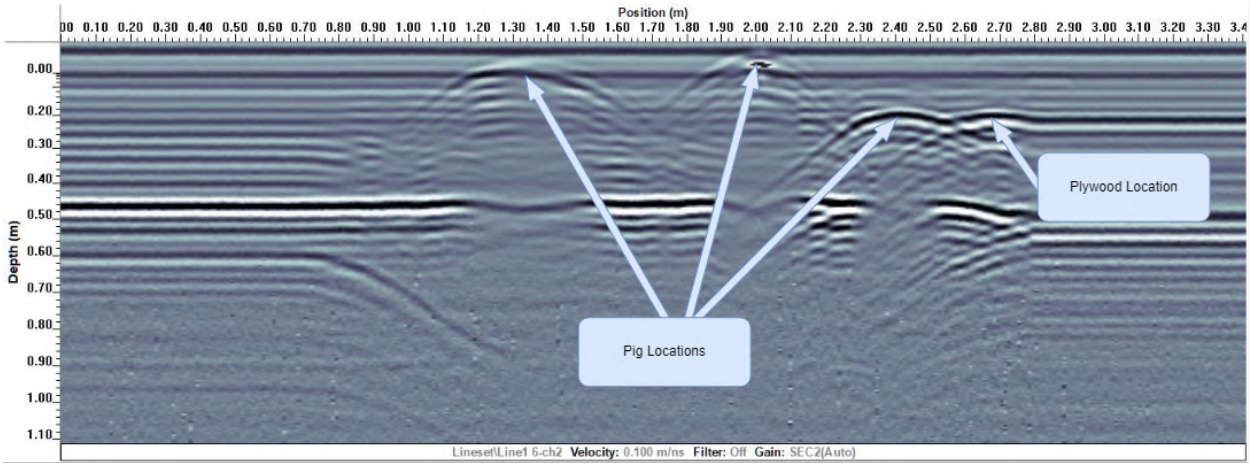


Figure 2.13: Radargram with False Detection of Plywood

Chapter 3

Ground Penetrating Radar – Data Interpretation

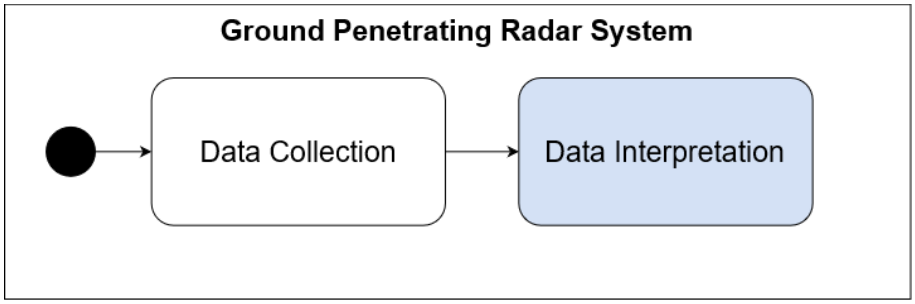


Figure 3.1: Data Collection module in the overall system

The GPR Data Interpretation module takes input in the form of comma-separated value (CSV) files of radargram data and converts them to a 2D image over the x-y surface of the ice for better visualization. All the data for this module is collected from the tests that were conducted in the previous module. This chapter will further discuss data processing, methods of approach for visualization, and the final results of our Data Interpretation module. Figure 3.1 illustrates where the Data Interpretation module falls within the GPR system.

3.1 Overview

The objective of the GPR Data Interpretation module is to take the data from a radargram and process the information using our custom-built algorithm. From there, it outputs the location of the pigs in the form of a 2D visual over the x-y surface of the tank. After multiple trial and error tests with different methods of visualization, we were able to create a data parser that takes input from a CSV file and produce a heat map locating the objects of interest from our SERF facility experiments.

3.2 Data Processing

We used Python to process the data that was received from the radargram. Numerical representations of the radargrams measured during testing were saved in CSV files. To handle the large amounts of data, we loaded the CSV files into Python using Pandas data-frames. Pandas is an open-source Python package that is most widely used for data science, data analysis, and machine learning tasks [8].

To translate the numerical representation of the radargram into a single line plot that shows the locations of objects, we used correlation. Correlation is a statistical term describing the degree to which two signals move in coordination with one another. If the two signals move in the same direction, then those signals are said to have a positive correlation. If they move in opposite directions, then they have a negative correlation [9].

When we calculate the correlation between two columns in the CSV file, we are effectively computing a measure of similarity (or dissimilarity) between two neighbouring A-scans. The result is a single coefficient that falls between the range of -1 and 1. Coefficients closer to 1 represent similarity in the columns, while coefficients closer to -1 represent an inverse relation. This data processing technique is used in a single radargram to see which areas of the scan are similar and which are dissimilar. Areas of dissimilarity suggest anomalies under the ice and may indicate the position of the pigs located inside the tank. The stronger the dissimilarity, the stronger the indicator is for a potential object of interest.

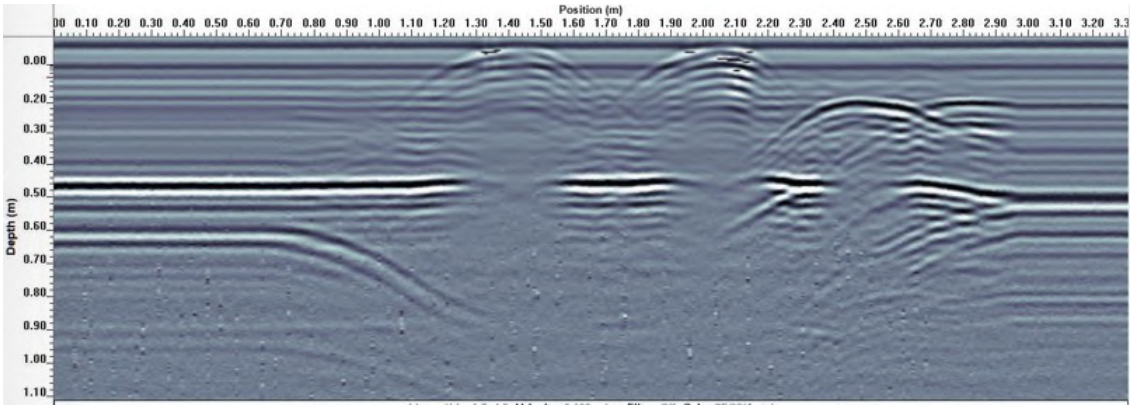


Figure 3.2: January 13th's line scan 7

The first version of our algorithm was implemented using this processing method. Using a single radargram CSV file, the algorithm calculates the correlation between the 1st column and the 10th column to produce a coefficient. Following this calculation, the algorithm places the coefficient in a list. The algorithm repeats this procedure with the next pair of columns (2nd and 11th column) until it has reached the end of the radargram. From here, the algorithm subtracts all the values in the list by 1 and plots them on a line plot as shown in Figure 3.3. The peaks on the plot close to 1 or above would represent the locations of the pigs.

An improved method of the correlation approach compares multiple columns at a time rather than comparing each column individually. Rather than finding the correlation between a single column and another column a set separation away, this modified approach finds the

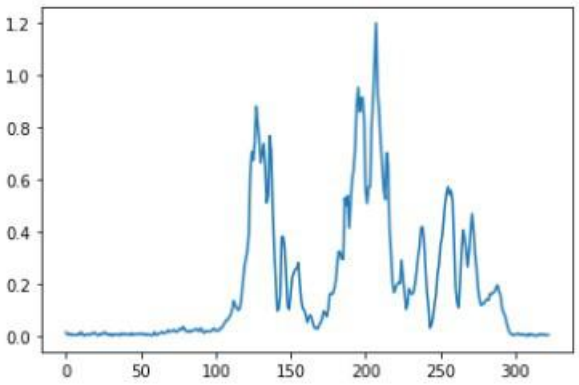


Figure 3.3: Data correlation line plot of January 13th's line scan 7

correlation between the mean of a specific width of columns and the mean of the same width of columns a set separation away. This method allows us to tune two variables, width and separation to determine the best option for interpreting the data.

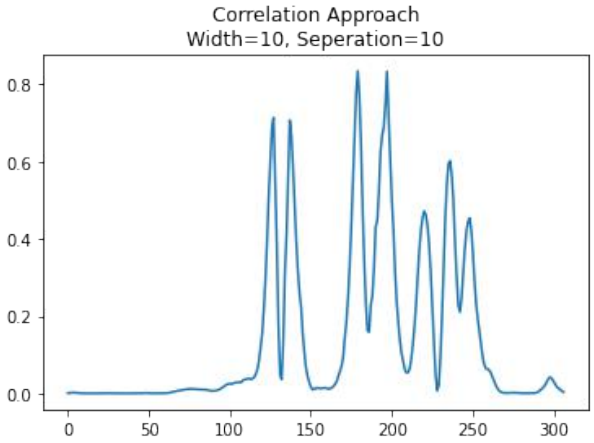


Figure 3.4: Modified correlation approach of January 13th's line scan 7

Preliminary but non-exhaustive testing suggested that a width of 10 and a separation of 10 was a reasonable choice for accurate results, but more testing is required. Using a width of 10 and separation of 10, the first output point is the correlation between the mean of columns 1 to 11, and the mean of columns 11 to 21, the second point is the correlation between the mean of columns 2 to 12 and the mean of columns 12 to 22, and so on until every column has a value.

Figure 3.4 shows the output of the modified correlation approach on the data from January 13th's line scan 7. The modified approach produces sharper peaks at objects, and has a larger contrast between locations with and without objects.

3.3 Visualization

The goal of visualizing the data is to clearly display the location of objects within a surveyed area in a 2D plot. Section 2.4.2 describes the method of data collection, where multiple line scans are taken to cover the full area of the tank. Each of these line scans

produces a radargram.

In the previous section, we showed how a single line plot that represents object locations can be generated from a radargram. To display the findings of a completed test, the data of these plots must be collected and displayed in a single plot. Multiple methods were considered but the best results were found using a heat map to display the correlation data.

3.3.1 Heat Map

The heat map approach collects the line plot data for every scan of a given day and resizes each line scan so they are all uniformly sized. The GPR takes measurements at a constant rate, so this resizing is required to account for any differences in the speed that the GPR was pulled at which would result in a different number of data points in a radargram. Because the tank is a uniform size, this resizing allows for the locations of objects within each line scan to be scaled to the size of the tank.

This resized collection of line plots is then displayed as an image, with the larger values of the plots displaying as a brighter colour, and the lower values displaying as a darker colour. Larger values of the line plots means a higher likelihood of an object at that location, so the heat map has bright spots where objects are most likely to be found, as described by the colour bar in Figure 3.6. The axes of the heat map represent the location in real space of the objects. For example, in the heat maps in Figure 3.5, you can see that there is likely to be an object 130cm from the north end of the tank, and 80cm from the west end of the tank, as well as other objects shown by the bright areas on the heat map.

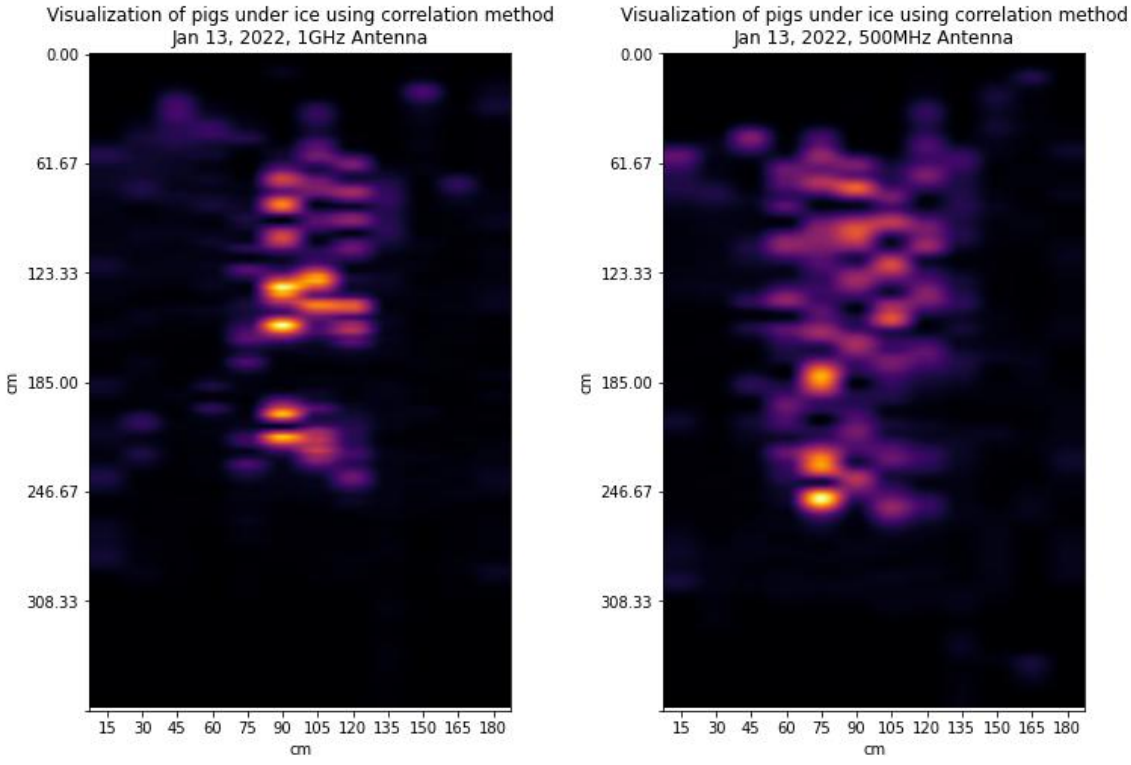


Figure 3.5: Heat map visualizations of data collected on January 13th

Figure 3.6: Colour bar describing the likelihood of objects in the heat maps

By looking at the quality of the heat maps, we were able to better tune the method of data processing described in Section 3.2. Multiple methods were tried but the best results were determined using the modified correlation method with a width of 10 and a separation of 10. The results found are discussed further in Section 3.4.

3.4 Final Results

The objective of the GPR Data Interpretation module is to translate the data from the GPR images taken from line scans to a location in space in order to locate any objects that were detected in the line scans. The metric we were aiming to meet with this module was to have a location resolution of less than 60 cm. Various approaches were considered, but ultimately, we decided that the modified correlation method yields the best results. To display the location of objects, we create a heat map that is the same dimensions as the area surveyed, with the brighter areas of the heat map corresponding to where objects are likely located.

In Figure 3.5, the data collected on January 13, 2022 is visualized using a heat map. The data shown were obtained using the correlation method described in Section 3.2 with a width of 10, and a separation of 10. In the 1 GHz plot on the left, the three pigs appear quite clearly along the center of the figure, while on the 500 MHz plot it is less clear where each individual pig is, but the area where objects are is still highlighted. We consider this a successful translation from a collection of GPR data to a map of object locations in real space.

To test the metric of location resolution, we selected the heat map shown in Figure 3.7. This heat map was generated from the data collected using the 1 GHz antenna on January 13th, 2022, and the correlation method described in the previous paragraph. The highlighted regions, each corresponding to the detected location of a pig carcass, narrow down the location of each pig to an area of width 50 cm, and a maximum height of 57 cm. This preliminary test shows that the location of a detected object can be localized within 60 cm, but because we cannot confirm the exact location of the pigs at the time of testing, the accuracy cannot be determined. Thus, the metric can not be conclusively determined, although preliminary tests show this approach to be promising.

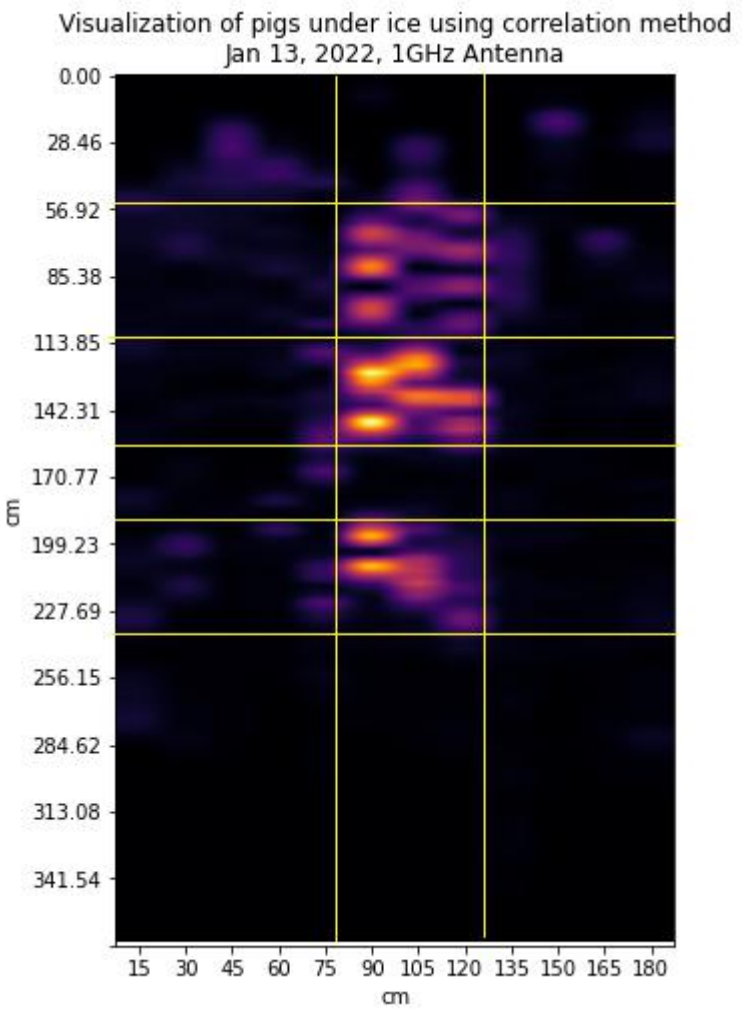


Figure 3.7: Measuring the resolution of the best heat map

Chapter 4

Computer Vision – Obstacle Detection

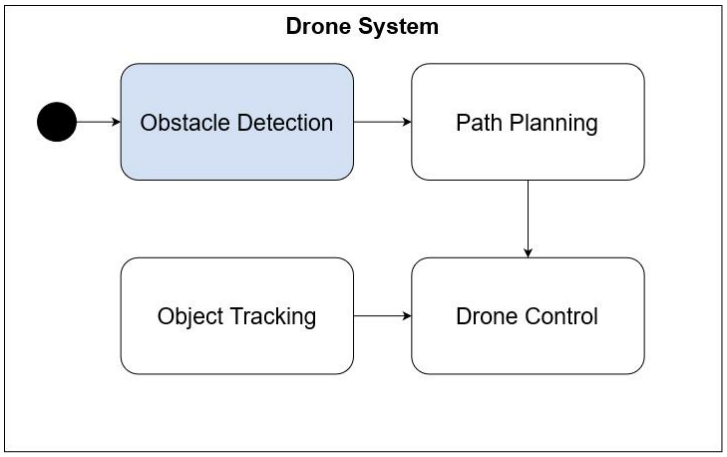


Figure 4.1: Obstacle Detection module in the overall system

In the case that a GPR is being dragged across a frozen body of water, obstacles such as holes in the ice or debris on the surface of the ice could cause serious damage to the GPR. The Obstacle Detection module is responsible for finding and marking such obstacles. To do so, input is taken from the Omnivision OV7251 camera of the Quanser QDrone and computer vision algorithms are used to determine the location of obstacles, then a digital map of the obstacle's location is returned to the Path Planning module. Figure 4.1 illustrates where the Obstacle Detection module falls within the drone system, and Figure 4.2 describes the operation of the module.

4.1 Overview

The Obstacle Detection module takes a greyscale image from the Omnivision OV7251 camera on the Quanser QDrone as input. Region growing image segmentation is used to find any obstacles within the image and then a CSV file of integer values that describes the locations of obstacles within the image is written. The resulting CSV is sent to the Path Planning module.

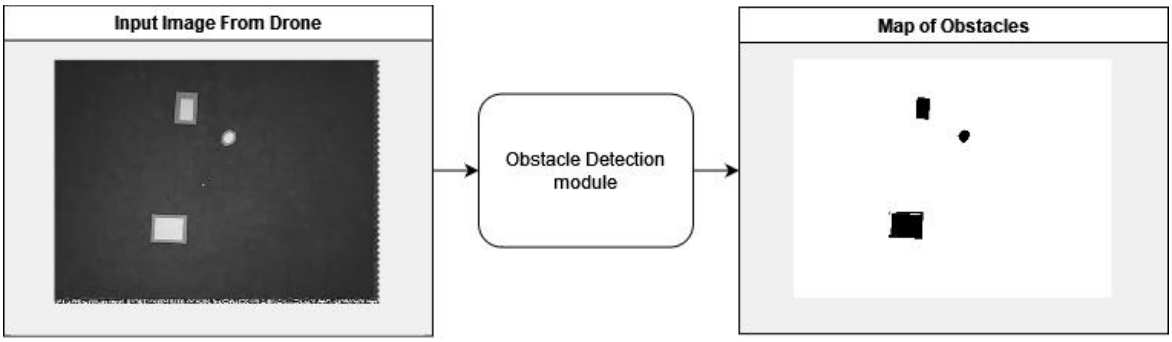


Figure 4.2: Obstacle Detection module operation

The algorithm used in our design was region growing image segmentation. The obstacle detection algorithm was implemented in MATLAB using the Computer Vision Toolbox. The Obstacle Detection module was designed to meet the metric of detecting >99% of obstacles in lab tests. In unit testing this metric was met for all tests conducted.

4.2 Approach

Initially our idea for this module was to detect and classify obstacles in real time as the drone is in flight. However, the problem can be simplified by detecting obstacles before the flight and then planning a path around them. Region growing image segmentation was chosen to detect the obstacles because it was fast to implement and did not require extensive training data.

Image segmentation is a technique that is used to partition an image into multiple regions. For our design, we were concerned about a "safe" region and an "obstacle" region.

A simple image segmentation technique called region growing proved useful for our purposes. Region growing analyses the pixels of an image from specific points and groups together similar pixels [10]. In a greyscale image, each pixel can be represented by a decimal value between 0 and 1, with 1 representing white and 0 black. In this method, a pixel called the seed point is selected and starting from that point the value of neighbouring pixels is compared to the starting pixel. Neighbouring pixels that meet some criteria are included in the region. The process repeats, examining the pixels that neighbour the region. This repeats recursively until the entire image has been segmented. This process is described in Figure 4.3.

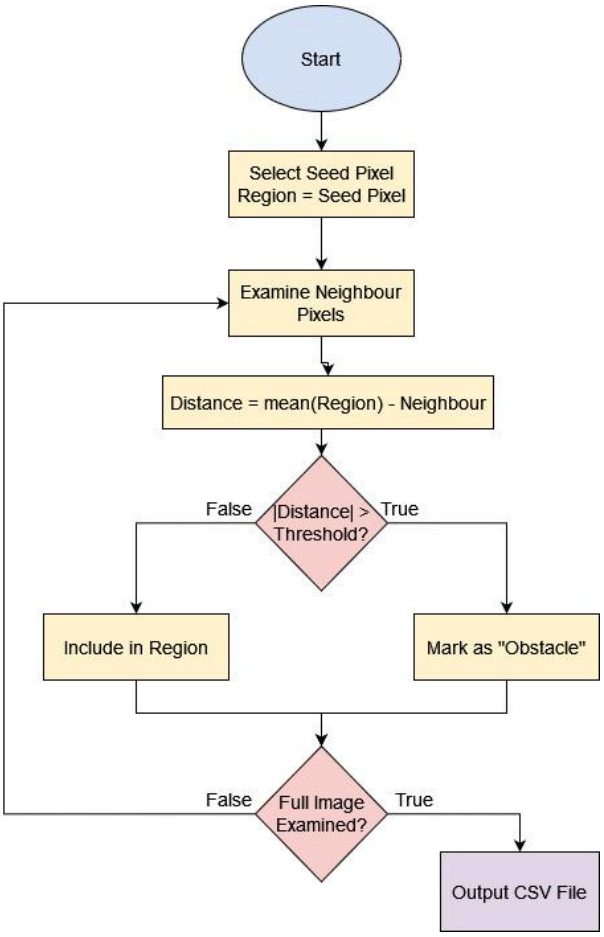


Figure 4.3: Flowchart describing the region growing algorithm

4.3 Implementation

The implementation of the obstacle detection module was done in MATLAB with the help of a function from the MATLAB central file exchange [11]. Figure 4.4 shows the overall flow for the MATLAB script. The script takes an input image, ensures that it is a greyscale image represented using float values between 0 and 1, then it runs the region growing function [11] with parameters that we determined experimentally. The output region is displayed to inspect for quality and written to a CSV file for the Path Planning module.

The function from [11] implements region growing on a given image, using the distance between the average pixel value in the region and the values of neighbour pixels as the criterion to determine whether or not to include the neighbours in the region. The threshold distance to include neighbouring pixels is an input to the function, and so is the location of the seed pixel.

To detect obstacles within an image, the center pixel is used as the seed point for the region. This is because the image that this script uses as input is taken after the drone flies straight up from a safe takeoff position, meaning that directly underneath the drone will be a safe spot for the GPR. The threshold value of 0.4 was determined experimentally for the lab tests, by taking pictures of the lab environment and trying different values. See Section 4.4 for more information on how the threshold value was tuned.

Initially, the obstacle detection algorithm was going to be run on the drone while the drone was in flight. Due to issues with drone system integration, which are discussed in Chapter 8, the Obstacle Detection module was redesigned to run on the base station after the drone lands.

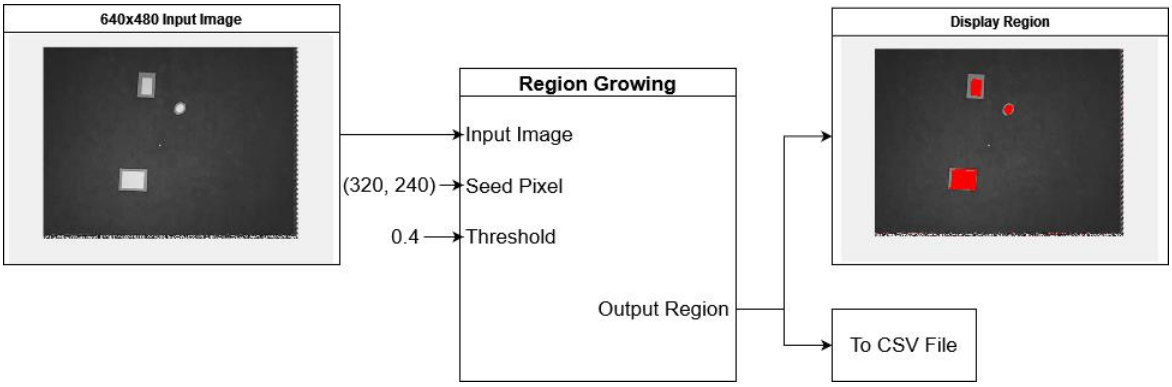
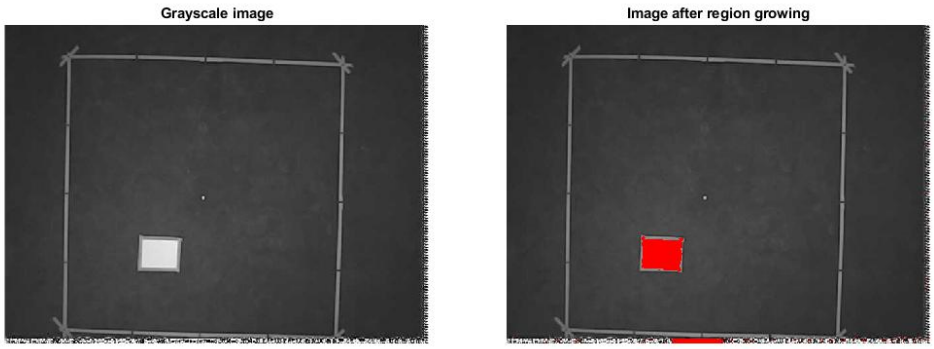


Figure 4.4: Functional diagram for MATLAB script

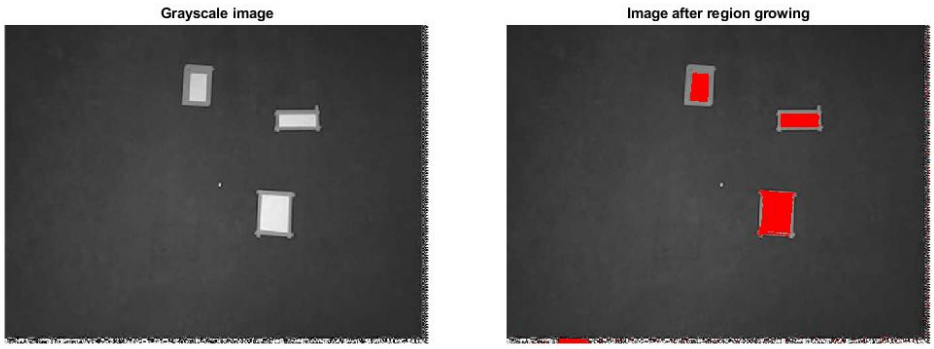
4.4 Obstacle Detection Testing

To verify that the obstacle detection algorithm will work in integration testing, and to tune the seed pixel and threshold variables, we took pictures with the drone camera in the test environment with different arrangements of pieces of paper on the ground to serve as obstacles. Using the pictures taken from the drone's camera, we tuned the threshold value so that the white papers on the ground were considered obstacles while the green tape was not. Once tuned, the algorithm was able to successfully classify obstacles as intended. The results of these tests are shown in Figure 4.5. Note that the noise around the right and bottom sides of the image comes from the Omnivision OV7251 camera and is dealt with during path planning (see Chapter 5).

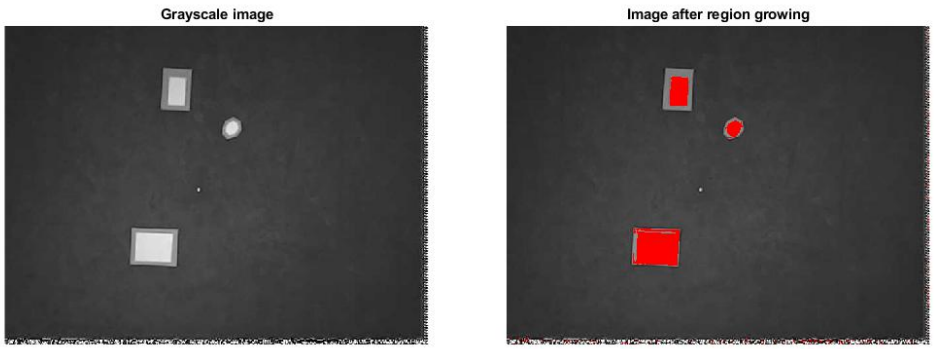
In all tests run at the lab, the obstacles were detected successfully. This satisfies the metric set for this module. For the proof-of-concept system that this project provides, obstacle detection is successfully implemented.



((a))



((b))



((c))

Figure 4.5: Output of several unit tests

Chapter 5

Path Planning

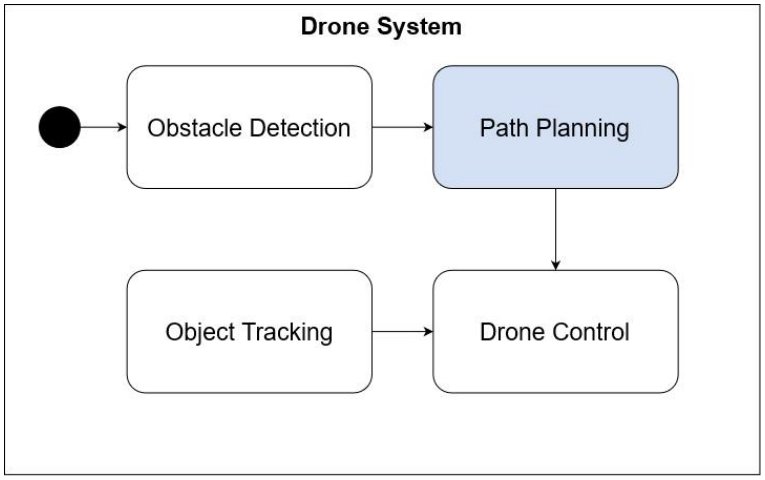


Figure 5.1: Path Planning module in overall system

The Path Planning module takes input from the Obstacle Detection module in the form of a CSV of integer values describing an area of interest. Each value in the CSV corresponds to a pixel in the image containing the area the the drone will pilot within. The values in the CSV are equal to 0 if no obstacle is present, or greater than 0 if an obstacle is present. The results of this module are another CSV of coordinates in real-space to be sent to the Drone Control module to process.

5.1 Overview

Once the area that the drone needs to map using the GPR is determined and obstacles have been identified, the Path Planning module devises a path for the drone to follow. The final path that is produced by this module is a series of waypoints in 3D-space that the drone will visit in order. This path needs to have the drone cover the area sufficiently such that the GPR could be able to find an object of interest regardless of its position. Additionally, the drone also must be able to avoid obstacles, such as holes in the ice, as these represent serious hazards for the GPR. Because the GPR is attached to the drone via a tether, directing just the drone to the series of waypoints will cause the GPR to drift behind it due to the

length of the tether. Therefore, the drone must "overshoot" the waypoints it comes across, where the drone flies past the waypoint far enough for the GPR to reach the waypoint rather than the drone. This introduces additional drone movement at every turn; however, the implementation of overshooting is placed on the Drone Control module rather than the Path Planning module, as discussed in Section 7.4.1.

5.2 Design and Implementation

The Path Planning module was written in Python as a standalone unit. It is not intended to be run in real-time. Rather, the module is a part of the system's pre-processing prior to the drone's flight.

5.2.1 Requirements

The Path Planning module can treat the input image as a simple 2D array of cells. These cells can be viewed as containing a binary obstacle indicator: either a cell has an obstacle, or it does not. For each cell that has an obstacle, the Path Planning module cannot plan a path that travels through that cell. For each cell that does not have an obstacle, the Path Planning module must plan a path that travels through that cell to ensure complete coverage. However, because the GPR carried by the drone has a certain thickness, we can safely rasterise the image of the area of interest such that the final size of each individual pixel is roughly the size of the GPR.

Additionally, it is preferable if the final path has few turns, as the more turns there exist in the path, the more overshooting the Drone Control module will need to perform, increasing the complexity of the actual flight of the drone. Finally, the path should minimize backtracking and re-treading ground that has already been scanned.

5.2.2 Approach

The approach of the Path Planning module is simple and straightforward. Beginning from a safe position, the drone moves left-to-right and right-to-left, slowly progressing downwards through the area of interest. When an obstacle is encountered, it is avoided by moving above it, across it, and then down to the location the drone was. This is similar to the way one would mow their lawn, hence the naming of this technique the "lawnmower" path planning method. Other area coverage algorithms were considered, but were ultimately not used, as discussed in Appendix A.

5.2.3 "Lawnmower" Path Planning

The Path Planning module's methodology is the "lawnmower" method. There are three general steps: initial processing, path planning, and real-world translation. A software flowchart of this module is provided in Figure 5.2.

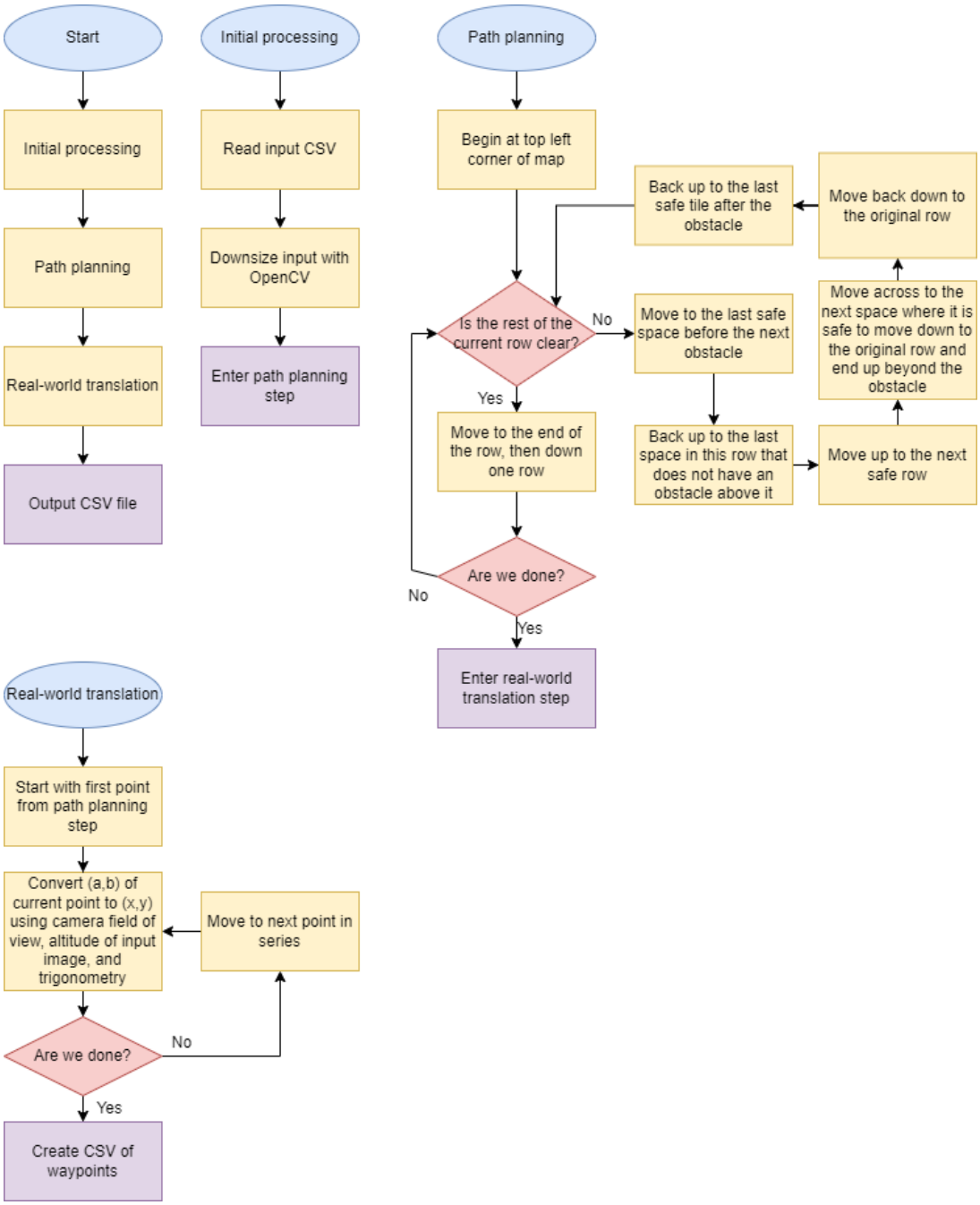


Figure 5.2: Path Planning module software flowchart

Beginning with the initial processing, the module reads in a CSV. This CSV is passed from the Obstacle Detection module, and therefore is a 480 x 640 image translated into a CSV, with the image's brightness values replaced with a binary indicator of whether a pixel contains an obstacle or not. Values in the CSV that are equal to 0 indicate no obstacle, whereas values that are greater than 0 indicate the presence of an obstacle. This format of CSV is read, then resized such that the final area covered by each pixel is roughly the size of the GPR. This was done using OpenCV, as OpenCV contains a useful resizing function that both preserves the aspect ratio and is able to represent how much of a down-sized pixel has an obstacle in it. Pixels in the resized image with lower values (closer to 0) represent very little of an obstacle within that pixel. Pixels in the resized image with greater values (closer to 255) represent a region of high obstacle density. We have experimentally determined a

specific threshold for the Path Planning module to treat a cell as an obstacle worth avoiding, which is a threshold equal to 15. In testing, we concluded that moving from a 480 x 640 image to a 12 x 16 image was sufficient for our purposes.

Next is the path planning step. Once the area has been properly initialized, we begin plotting our route. The Drone Control module is expecting a series of waypoints in real-space, and therefore we need only generate the points we will visit, and not the line between those points. Beginning with the top left corner, we move across the row to the right, down one row, and then across to the left. We continue like this until we move off the bottom of the image. If we come across an obstacle in the map, we move up to the next safe row that allows us to move across the obstacle, then move laterally until we are safe to come back down, and then we move down to the row we started with. However, it is possible that during this movement, we run into an obstacle on the way up or the way down. Therefore, if an obstacle is in the way for either of these movements, we move backwards or forwards accordingly to accommodate.

Finally is the real-world translation step. The final sequence of points is iterated through and converted into a series of waypoints in real-space, using the field of view of the camera that took the original image, its height when the image was taken, and trigonometry to calculate the location of each waypoint. This final series of waypoints is formatted into a CSV, which will be passed into the Drone Control module.

A sample of this path planning strategy is shown in Figure 5.3. The following path was generated from an obstacle map using an image taken in the lab we used for integration testing. The thin blue-purple line along the far-right column and bottom row are due to noise from the drone's camera, and are treated as obstacles.

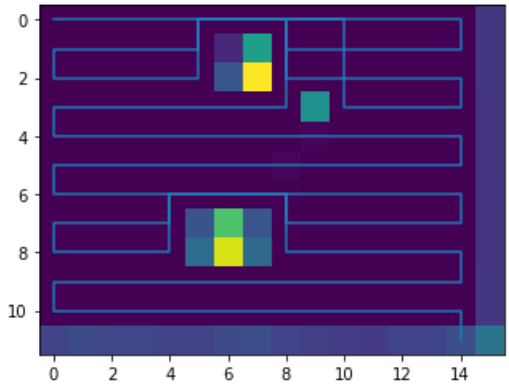


Figure 5.3: Example of "lawnmower" path planning

5.3 Path Planning Testing

5.3.1 Metrics

Our original metrics for path planning were a greater than 99% area of interest to be covered by the path planning algorithm, and a greater than 99% area of interest to be covered by the drone in-flight. These were motivated by the original application and scope of the

project; at the point in time these metrics were decided upon, it was thought that our final tests would be on river ice with a full-sized GPR and drone system. We have since reduced the scope of the project, but our metrics were retained.

5.3.2 Tests and Results

We tested on a variety of samples, including fabricated samples, where the field of obstacles was artificially generated using software, and lab samples, where an image of an actual obstacle field was taken, converted into a CSV, and read. These samples were fed as input to the module, and the final results were examined manually. Regarding the greater than 99% area of interest metric, we determined that the software was never incorrectly planning a path that skipped regions of the area. Though it is impossible to say that it will work in every possible layout of obstacles, we are confident in saying that this metric has been met and that the module is successful in plotting a path through an area of interest.

In regards to the greater than 99% area of interest covered by the GPR, we will discuss this metric more in the integration and validation section.

Chapter 6

Computer Vision – Object Tracking

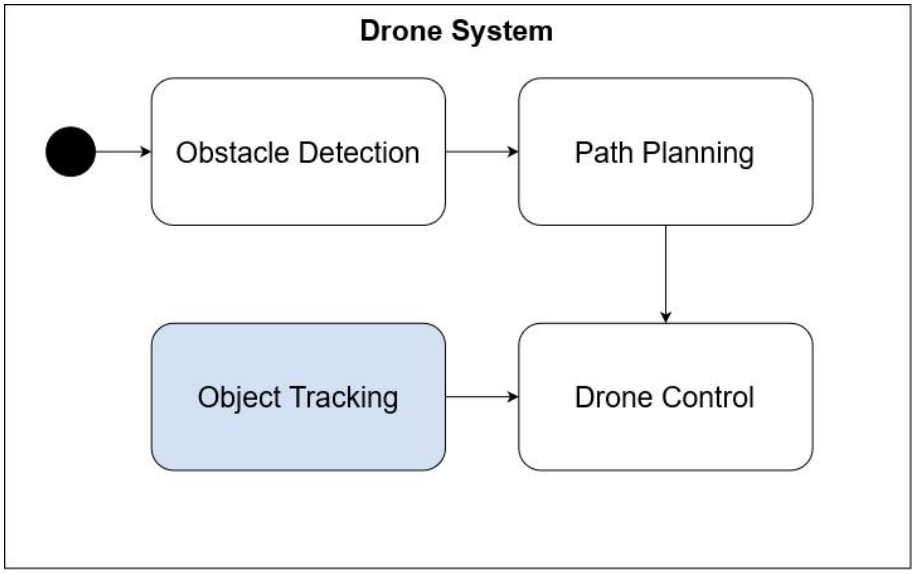


Figure 6.1: Obstacle Tracking module in overall system

The Object Tracking module takes input from the bottom facing camera on the Quanser QDrone in real time and tracks the object of interest (GPR or payload) in the given video frame. Once the object is detected, the module relays the location of the object of interest to the Drone Control module for further processing. This chapter will further discuss the objective, design, implementation, and module testing of the Object Tracking module. A diagram of the Object Tracking module in relation to the other drone system module can be seen in Figure 6.1.

6.1 Overview

The objective of the Object Tracking module is to track the GPR/payload with respect to the position of the drone during the given flight. In collaboration with the Drone Control module, it is responsible for ensuring the payload is following the path that is given from the Path Planning module. The module uses computer vision to detect and track the payload as it is dragged by the drone. This is accomplished by using the downward-facing camera

on the drone to get live video frames. The payload of the drone is detected in that specific frame. The location of the payload in the frame is then passed to the Drone Control module to ensure the payload is on the correct path. Figure 6.2 shows the Object Tracking module operations in relation to the Drone Control module. If the object is on the correct path, it continues the same process for the duration of the drone flight. If the object is not on the path, the path is corrected as described in Section 7.4.4.

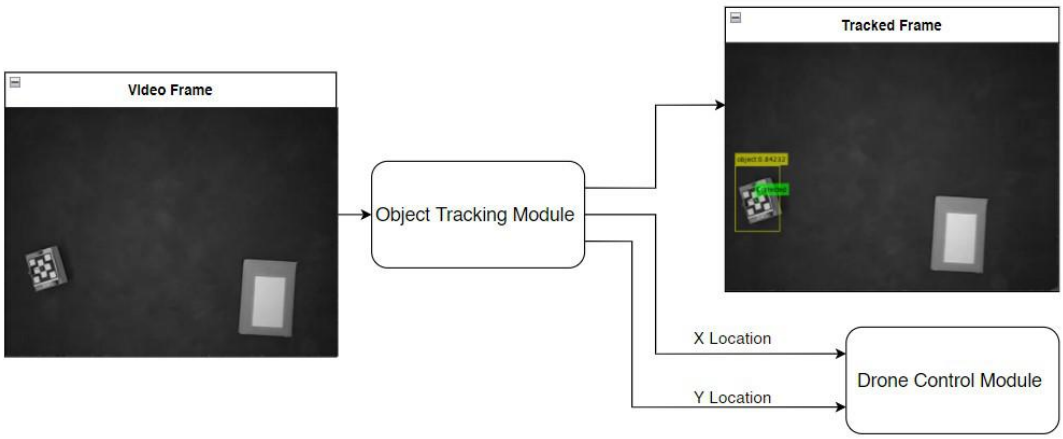


Figure 6.2: Object Tracking module operation

6.2 Design and Implementation

In the early stages of development, various tracking and object detection algorithms and frameworks were considered. For the purpose of this module, we had to consider the per frame processing time, accuracy, and ease of implementation in MATLAB. Since the module would be used in real time, we required fast object detection with a short per frame processing time. Additionally, the module needs to be capable of detecting the object with high accuracy, as the drone system is required to track the specific payload. Based on these requirements, and experimentation with some preliminary algorithms, a YOLO-based object tracking algorithm was decided on which is described below.

The Object Tracking module was initially developed in MATLAB and later converted into a Simulink block to ease integration to the Drone Control module. The module was designed to be run in real-time with input from a camera. Due to roadblocks faced during the drone system integration (discussed in Section 8.1), we were unable to completely integrate with the Drone Control module. As a result, an alternative system testing with video saved from the drone's flight was done to demonstrate proof-of-concept viability, as discussed in Section 8.2. The module was therefore developed to be capable of taking in live video input for module testing and saved video input for system testing. Since the module needs to detect and track a single object, we decided to design the module to track a specific unique fiducial that is placed on the object of interest. The fiducial used for this project can be seen in Figure 6.3. This module is considered successful if the fiducial is detected and tracked in real-time.

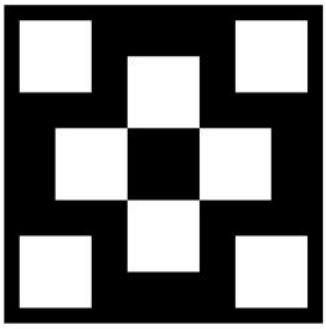


Figure 6.3: Fiducial used to track payload

The design of the Object Tracking module has two major components: object detection and detection correction. The object detection component was implemented using YOLO version 3 (YOLOv3) to detect and track our object of interest across video frames. To further improve the tracking of the object of interest, detection correction was added to the module. This was accomplished by using a Kalman filter to reduce noise introduced by inaccurate and failed detections. Both components were implemented with the help of MATLAB's Computer Vision Toolbox.

6.2.1 YOLO Object Detector

YOLO or "you only look once" is an object detection algorithm that is capable of real-time object detection with relatively high accuracy. YOLO makes use of deep learning techniques to train an object detector [12]. With YOLO, the input image or frame is looked at only once and requires one forward propagation through the neural network to detect the object in the given image. Object detection is cast as a regression problem, as the algorithm predicts classes and bounding boxes for the whole image with a single convolutional neural network [13].

For a given input image, YOLO works by splitting the image into an $S \times S$ grid, where if the center of an object is in a grid cell, that cell becomes responsible for detecting that object [12]. The cells in the $S \times S$ grid predict a number of bounding boxes with corresponding confidence scores. This confidence score represents the level of confidence that a certain object is in the bounding box, as well as how accurate this prediction is. During its forward propagation through the neural network, YOLO predicts the probability that certain objects are in the cell, and the class with the highest confidence score is chosen and assigned to the specific cell. To ensure the object detection algorithm detects an object once, YOLO eliminates overlapping bounding boxes through non-max suppression [12]. The steps described above are accomplished through a convolutional neural network which allows for fast and accurate object detection. YOLO is considered a viable solution and was therefore chosen for the implementation of this module. The implementation and training of the YOLO object detector used for this module will be discussed in Section 6.2.3.

6.2.2 Kalman Filter

A Kalman filter can be used to reduce noise introduced by inaccurate detection and failed detections. A Kalman filter is an an estimation algorithm that estimates the state of the system based off measurements and previous observations [14]. This filter was added to the design to ensure the module can still track the fiducial when the fiducial goes undetected. The algorithm is composed of two steps: prediction and correction. The first step predicts the state of a system based the previous estimated states and current measurements, and the second step refines or corrects the measurement noise of system state based on the prediction [14]. The configuration and implementation of the Kalman Filter used for this module will be discussed in Section 6.2.3.

6.2.3 Object Tracking Implementation

The Object Tracking module was completed in MATLAB and Simulink. Object detection was implemented with a YOLOv3 object detector. This object detector was trained to detect the fiducial shown in Figure 6.3. Object tracking was improved by incorporating detection correction. This was done by using a Kalman Filter to reduce noise in the object's detected location and account for failed detections. The implementation of each component of the tracking module is described as follows. The flowchart shown in Figure 6.4 shows a detailed overview of the module.

Training Object Detector

Before tracking, we had to train the YOLOv3 object detector [15]. For the purposes of this project we were required to train the object detector for a single class of objects: the fiducial. To train the object detector we created a training data set consisting of 152 images. These images were taken in a variety of settings with the fiducial in varying positions and locations. As the images were taken with multiple cameras with different aspect ratios and resolutions, all images were standardized to to be the same size. The images were cropped to a 4×3 aspect ratio and then scaled down to have a dimension of 640 pixels by 480 pixels. Each image was labelled with bounding boxes around the fiducial with the class name 'fiducial'. These labelled images were trained in MATLAB, producing a YOLOv3 detector object. The object detector was saved in a MAT file (file that stores MATLAB formatted data and variables) to be used in the object tracking algorithm. Eighty percent of the image data set was used for training while twenty percent were used for testing. The code implementation can be found in the Appendix.

The object detector was validated using the allocated test set. The test data consisted of 31 images, for which the object detector was able to accurately detect the fiducial. Samples of these test images can be seen in Figure 6.5. Further validation was done by finding the average precision. The average precision measures the performance of an object detector by the average precision scores for the object class. Precision refers to the ratio of true positive instances to all positive instances of objects in the detector. This value can be obtained using

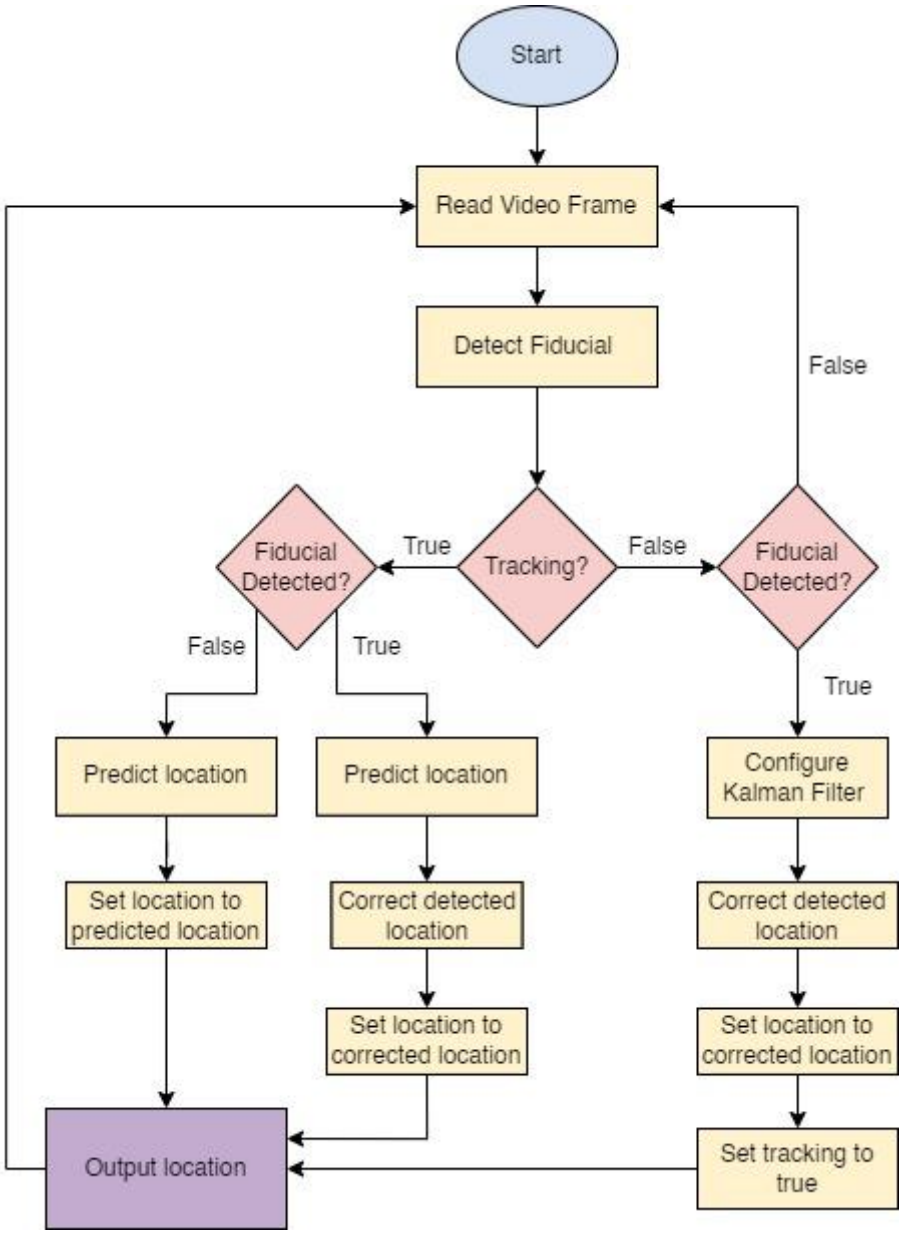


Figure 6.4: Tracking algorithm

the average precision function in the Computer Vision Toolbox [16]. The average precision of the allocated test set was found to be 0.9.

Object Detection

The Object Tracking module initially accepts a single video frame. This video frame is used as input into the object detector, which attempts to detect a single fiducial object. The threshold for the confidence score of the detector was set to 0.3. The module will not begin its tracking state until the fidcucial is detected. Once the fiducial is detected, the center location is then passed to the Kalman filter for detection correction. For proceeding frames, the input image will be entered into the object detector before it is passed to the Kalman filter for detection correction.

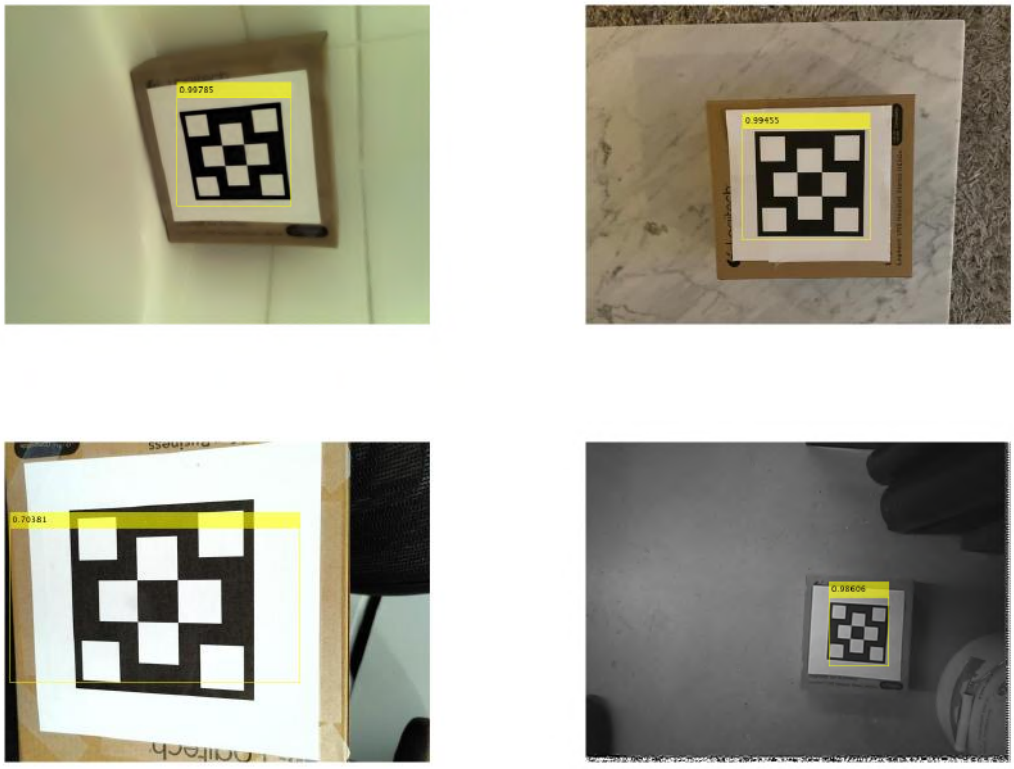


Figure 6.5: Test images results for object detector

Detection Correction

The `vision.KalmanFilter` object from the Computer Vision Toolbox [17] is implemented as follows. Upon the first detection of the fiducial, the Kalman filter is configured to follow a constant velocity motion model using the `configureKalmanFilter`. A constant velocity motion model was chosen as the drone moves near constant velocity, and since the payload moves slowly from the perspective of the drone camera. The main reason for implementing the Kalman filter is so the fiducial could be tracked when it is not detected. When the object is detected, the filter predicts its state at the current video frame and then uses the newly detected object location to correct its state. When the object is not detected, the Kalman filter relies on its previous state to predict the object's current location. The location is then outputted to be used by the Drone Control module.

Simulink Block

The algorithm was converted from MATLAB code to a Simulink MATLAB function block. Since our algorithm was dependent on some functions from the Computer Vision Toolbox that cannot be compiled in C/C++, we were required to use MATLAB's "extrinsic function" which would run certain functions in MATLAB. The Simulink block takes an image as an input and outputs X and Y values.

6.3 Object Tracking Testing

Independent module testing was done with both live video and saved video input. Screenshots from tests can be seen in Figure 6.6. Testing with live video was completed using input from a webcam. The module was given a variety of sample videos and live video inputs, while the output was observed through a video player while also printing the tracked location of the fiducial.

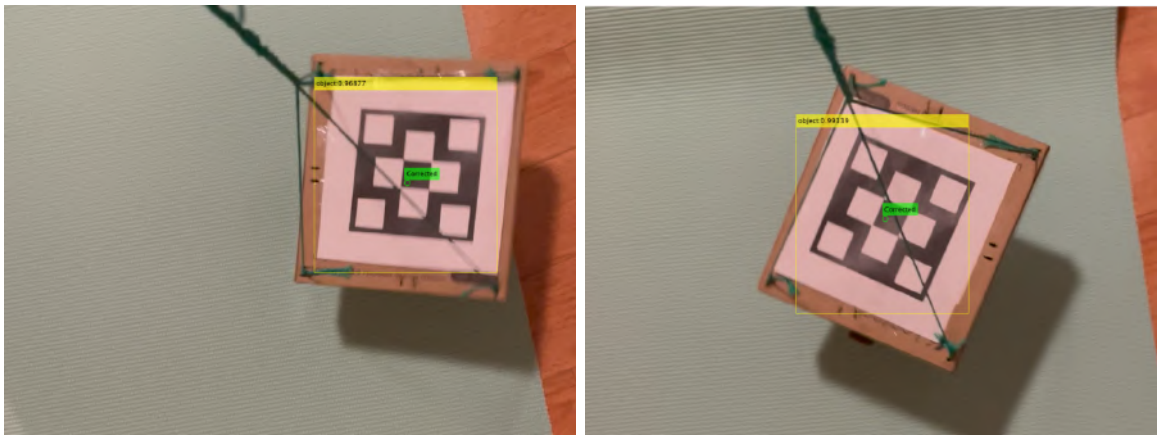


Figure 6.6: Object Detector Tests

Based on visual observation, we are confident in saying the algorithm was able to sufficiently track the payload for both live video and saved video. We noticed that the module struggled when the object was at or near the edges of the camera frame, as shown in Figure 6.7. This could be improved by training with a larger and more diverse image data set. The target for the per-frame processing time was 33 ms. Upon actual testing, this metric was found to be out of scope given our current design and hardware capabilities. In our tests, we were able to track the payload with a per-frame processing time of around 100ms to 400ms. This value varies depending on the input video and hardware used to run the test. Even though we were unable to meet our initial targets, we are still able to track the object of interest with both live video feed and saved video as the object does not move a significant distance within 400ms in our application.

Figure 6.7: Tracking algorithm

Chapter 7

Drone Control

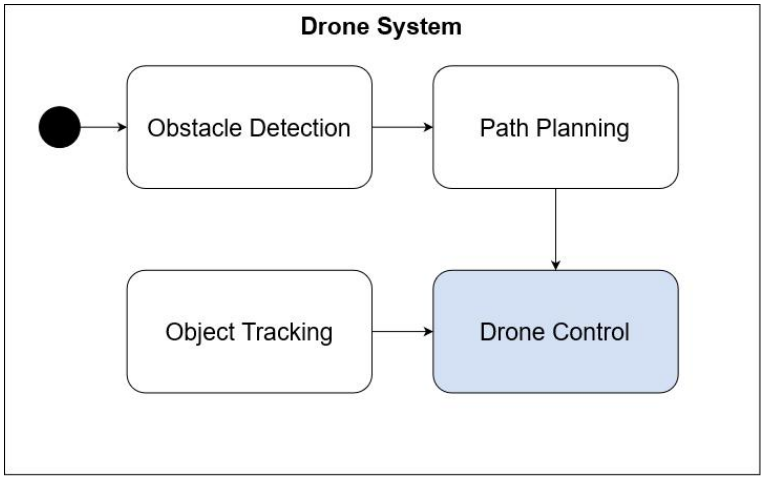


Figure 7.1: Drone Control module in the overall system

The Drone Control module is responsible for maneuvering the Quanser QDrone such that a dragged payload follows the path planned by the Path Planning module, while making corrections based on positional data from the Object Tracking module. The Drone Control module takes a CSV file containing real-space coordinates from the Path Planning module as input, as well as positional data for the drone from a Vicon Motion Capture System. The Drone Control module also takes the payload's positional as input from the Object Tracking module as it tracks the payload dragged by the drone. The output of this module consists of four continuous signals that control the X, Y, Z, and yaw positions of the drone. An overview of how the Drone Control module fits into the drone system is shown in Figure 7.1.

7.1 Overview

The Drone Control module is responsible for autonomously controlling the Quanser QDrone in order for a dragged payload to follow a predetermined path. It is important to reiterate that all drone-related development was in preparation for scaled-down testing in a lab environment, rather than real-world testing with a full size drone. This is most relevant when considering the positional data of the drone, which is gathered using an array of Vicon

Motion Capture cameras rather than a Global Positioning System (GPS). The transition from Vicon to a full-scale positioning solution such as GPS or a similar technology would be simple, as a 3-dimensional position can be obtained easily, and a digital compass can be used to determine rotation. The Quanser QDrone is controlled using Simulink models that work in tandem with the Vicon Motion Capture System [18], and Quanser has developed several models to enable simple control of the drone out of the box [19].

The Vicon system and the Simulink models are the foundation of the Drone Control module, and have been built upon in order for the drone to follow the coordinates provided by the Path Planning module. However, the goal of this module is for the payload dragged by the drone to follow the path rather than the drone itself. Using a simple tether, the payload can be assumed to be following the drone along its axis of movement until the drone performs a turn. This is, however, only an assumption, and the drone must compensate for any unexpected movement perpendicular to the axis of movement. To accomplish this, the Object Tracking module is used to track the position of the payload using the Quanser QDrone's on-board camera, and corrections are made to the drone's trajectory to compensate for the payloads movement. Figure 7.2 demonstrates how these various systems come together for the operation of the module.

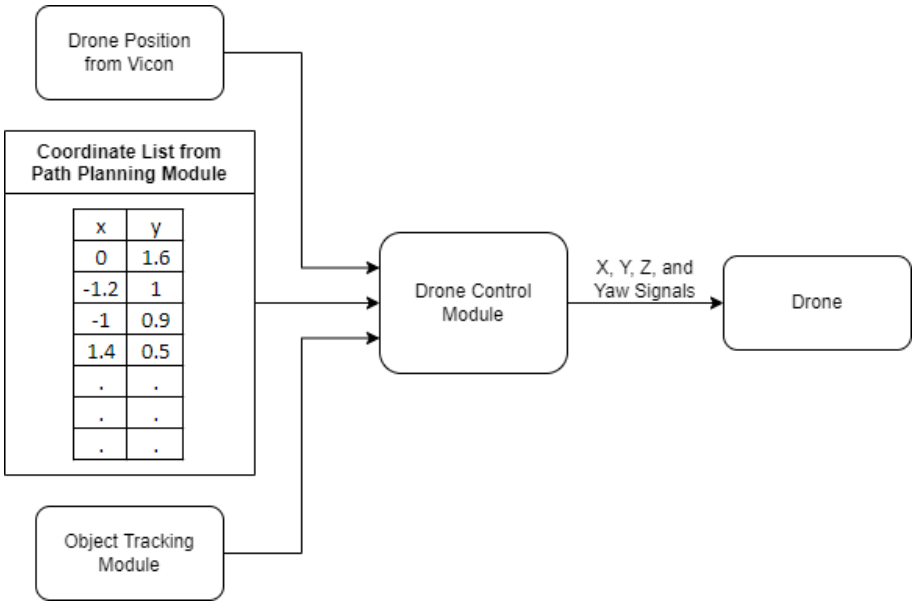


Figure 7.2: Drone Control module operation

To ensure the box accurately follows its predetermined path, a metric for drone response time to payload movement of less than 40ms was put in place. An additional metric states that the drone should fly to within 5cm of each coordinate in the path. These metrics combine to ensure that the payload follows the desired path to within 5cm at all times. Finally, a metric for drone battery life of greater than 8 minutes was used to ensure that the drone would not be hindered by excessive real-time processing or additional movements, as the normal life of the drone's battery is 12 minutes.

7.2 Hardware

The Quanser QDrone and the Vicon Motion Capture System were provided by STARLab. The hardware used is meant to demonstrate a proof of concept at a small scale; however, this hardware can be scaled up or modified to meet the needs of a real-world application with minimal changes to the design of the Drone Control system. The primary reason for selecting this combination of drone and positioning system is due to its availability and compatibility. Simulink models provided by Quanser support Vicon systems out of the box, and STARLab has a flight area used for testing various drones, with a Vicon system already in place.

7.2.1 Quanser QDrone

The Quanser QDrone shown in Figure 7.3 is an autonomous quad-rotor drone with an on-board Intel Aero Compute Board, along with various types of cameras [19]. The Intel Aero Compute Board is responsible for real-time flight control, while also being capable of high-intensity computing such as video processing. This was one of the main factors leading to the choice of the Quanser QDrone, as the Object Tracking module is designed to run in real-time on the drone during flight. The Quanser QDrone is entirely controlled using Simulink, both on the drone side and on the server side, with communication taking place over a private WiFi network. The Quanser QDrone does not come standard with any positioning system, and relies on external hardware to enable autonomous flight.

Figure 7.3: Quanser QDrone [19]

7.2.2 Vicon Motion Capture System

The Vicon Motion Capture System is used to acquire real-time positional data for use in the Drone Control Simulink models. In real-world testing, the positional data provided by these cameras would instead come from a GPS or ideally a Differential GPS (DGPS), without any fundamental changes to the system. Since this is a proof-of-concept system tested in a laboratory, the scope of the project does not need to include these real-world options. STARLab has 4 Vicon cameras placed in the upper four corners of a 3m x 3m x 2.5m volume that acts as the flight area for the drone that is enclosed by a safety net, as shown in figure 7.4. Infrared light reflectors are placed on the object, and the four Vicon

cameras bounce infrared light off of the reflectors in order to determine its location in 3-dimensional space. A Cartesian coordinate system is created within the Vicon software, and the object is located within that space, providing the Simulink models with X, Y, Z, and yaw positions of the object in real time. The Cartesian coordinate space is configured such that the origin is located in the middle of the floor of the flight area, and Vicon coordinates correspond to real-world meters for X, Y, and Z, and radians for yaw. Vicon is also capable of tracking multiple objects at once, as will be covered later when discussing object tracking and drift compensation in section 7.4.4.

Figure 7.4: Vicon-equipped cage used for drone flights

7.3 Quanser Simulink Models

Quanser supplies several Simulink models that enable out-of-the-box flight without any Simulink development required. One of these models, known as the Commander Stabilizer, is run on the QDrone's Intel Aero Compute Board and is responsible for real-time flight control and stabilization. All other Simulink models supplied by Quanser aim to demonstrate the various use-cases for the QDrone, and are known as Mission Servers, which run on the server-side computer known as the base station. Quanser has developed their own Simulink block library known as the QUARC Targets library, which is used for all communication and hardware specific functionality. No changes were made to the Commander Stabilizer, as the drone remained in control without alteration even when dragging a payload. A Mission Server provided by Quanser was built upon in order to enable fully autonomous flight. Part of the Mission Server has been replaced by a Simulink subsystem that is responsible for all autonomous maneuvering of the drone, and this subsystem will be referred to as the Autonomous Trajectory Calculation subsystem. All communication and coordinate translation was left untouched, and only X, Y, Z, and yaw control signal generation was altered.

7.3.1 Server and Drone Communication

The base station computer runs the Mission Server Simulink model while the Quanser QDrone runs the Commander Stabilizer Simulink model. Both of these models contain stream server Simulink blocks from the QUARC Targets library to achieve communication between the base station and the drone. A private WiFi network is created using a router connected to the base station which acts as a wireless access point that the drone connects to. All communication systems are unchanged, as no additional communication lines were required between the the base station and the drone. Among the signals that are transmitted over this network are the X, Y, Z and yaw signals that control the drone's position.

7.3.2 Drone Positioning Using Vicon

In the lab testing environment, the Vicon Motion Capture System is a simple and accurate method of acquiring positional data. The Simulink models provided by Quanser have built-in Vicon compatibility, and the method by which these Simulink models incorporate Vicon positional data into the drone's behaviour should be understood. The Vicon Motion Capture System is connected directly to the base station where Vicon software is run to provide the Mission Server Simulink model with the positional data of the drone. This positional data can be considered the measured pose of the drone, and consists of the drone's real-time X, Y, Z, and yaw position in the Vicon coordinate space, which corresponds to real-world meters and radians. The Autonomous Trajectory Calculation subsystem outputs the real-time desired X, Y, Z, and yaw of the drone, which can be considered the drone's desired pose. Both the measured pose and desired pose of the drone are sent to the drone's Commander Stabilizer Simulink model, which maneuvers the drone accordingly.

The Commander Stabilizer uses the measured and desired poses, along with several other signals from the Mission Server, to move the drone to the desired pose as fast as possible. Therefore, if the desired pose changes instantaneously from 0 to 1 per say, the Commander Stabilizer will attempt to move the drone to the new desired position of 1 as fast as possible. The Commander Stabilizer is using a feedback loop with the measured and desired pose of the drone to control its position in the Vicon coordinate space, as the Vicon positional data is its only point of reference during flight.

From the Simulink model and Vicon positioning information above, it becomes more clear how drone control can be accomplished using the two systems. Within the Autonomous Trajectory Calculation subsystem, the desired pose of the drone in the Vicon coordinate space must be continuously generated in a way that controls both speed and position in real-time. The desired pose can then be sent to the Commander Stabilizer Simulink model over WiFi, which will attempt to move the drone to the desired pose as fast as possible.

7.4 Autonomous Path Following

Autonomous path following is the ability of the drone to move in a way that drags a payload on a predetermined path and accounts for any unexpected movement of the payload. The path is defined by a list of coordinates that correspond to the corners of a path with straight lines between each corner. This list of coordinates is generated by the Path Planning module, and is supplied in a CSV file. Within the Mission Server lies the Autonomous Trajectory Calculation subsystem, which accomplishes autonomous path following. This subsystem must translate the list of coordinates into continuous desired pose signals that can be supplied to the Commander Stabilizer Simulink model on the drone. These signals consist of the desired X, Y, Z, and yaw position of the drone at any given time, and should maneuver the drone in such a way that the dragged payload follows the predetermined path as closely as possible. The Autonomous Trajectory Calculation subsystem is made up of four subsections, including coordinate processing, generation of continuous signals, proximity triggering, and drift compensation. A high level overview of the functionality of this subsystem is shown in Figure 7.5, where the system runs continuously while the drone is in flight.

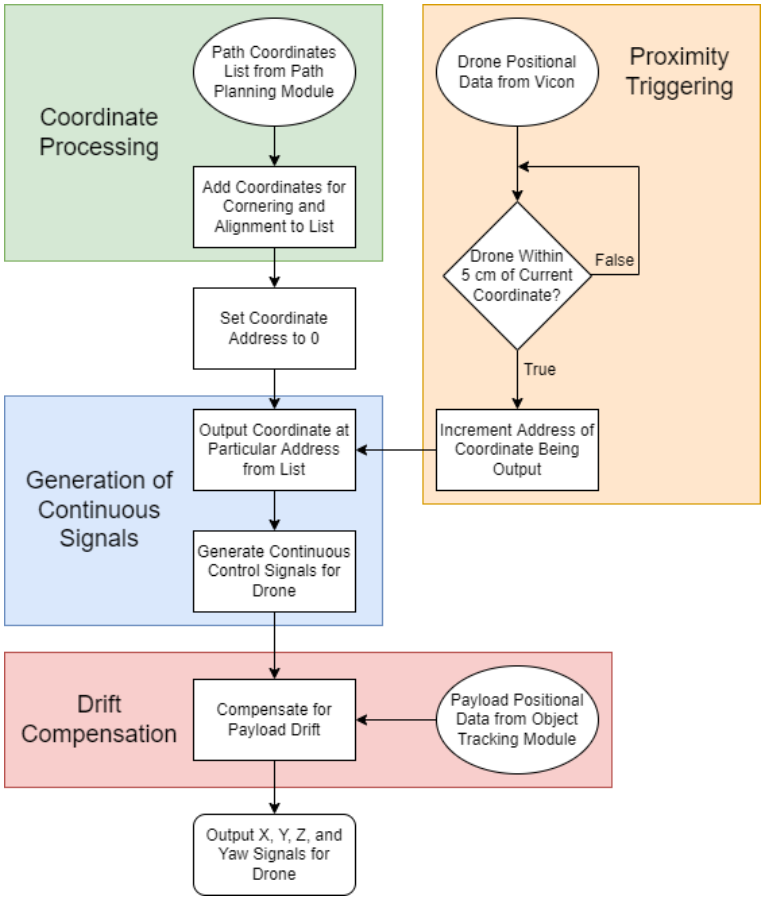


Figure 7.5: Inside the Mission Server Simulink model

7.4.1 Cornering and Coordinate Processing

There are several approaches to dragging a payload around a corner, and the selection of the best method depends on multiple factors. As discussed in Section 2.4.2, GPR testing revealed that the orientation of the antennas does not effect their readings, and therefore the orientation of the payload does not need to be maintained or controlled. This greatly simplifies cornering with a payload, as the rotation of the payload can be ignored. The method of cornering used was to overshoot every coordinate in the path by the same distance as the horizontal offset from the payload to the drone. This ensures that the payload reaches the desired coordinate. The drone then returns to the actual coordinate directly above the payload, and is then able to move on to the next coordinate.

To incorporate the overshoot method of cornering into the Autonomous Trajectory Calculation subsystem, the MATLAB function that runs at initialization of the model is used. An example of a triangular path that could be supplied by the Path Planning module is shown in Table 7.1. For every coordinate in the list, an additional coordinate is created for the cornering sequence. The first coordinate has an overshoot added to it using the yaw of the drone to calculate the X and Y components of the overshoot. The second coordinate is unchanged from the original, and brings the drone back to directly above the payload. The application of the cornering method to the example path in Table 7.1 with an overshoot of 0.2m is shown in Table 7.2.

x	y
0	0
0	1
1	1
0	0

Table 7.1: Coordinate list example

x	y	
0	0	Alignment Sequence
0.4	0	
-0.2	0	
0	0	Coordinate
0	1.2	Overshoot
0	1	Coordinate
1.2	1	Overshoot
1	1	Coordinate
-0.1414	-0.1414	Overshoot
0	0	Coordinate

Table 7.2: Coordinates after processing

A final addition must also be made to the list of coordinates in order to properly align the payload before following the path. By moving the drone to the first supplied coordinate, then moving along an arbitrary axis by twice the overshoot distance, then moving in the opposite direction by three times the overshoot distance, and finally moving back to the original coordinate, the payload and the drone should be aligned on the first coordinate. The alignment sequence on the X-axis is also shown in the coordinate list in Table 7.2, which now represents the full coordinate list of a simple triangular path. A visualization of the drone's path in the XY-plane can be seen in Figure 7.6, where the drone starts from an arbitrary home position of (0,-1).

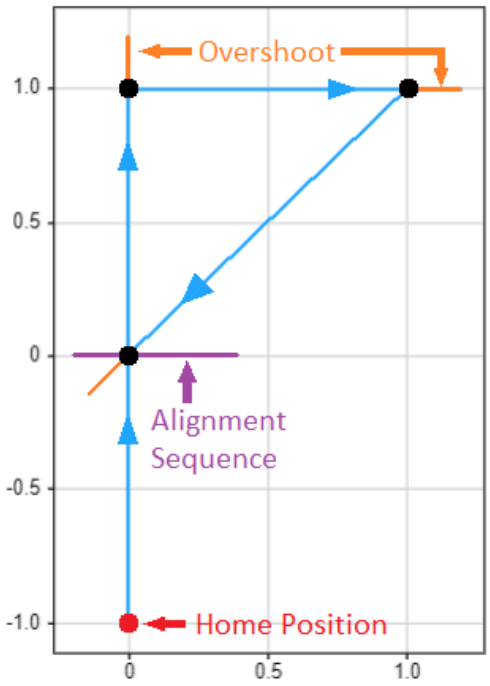


Figure 7.6: Visualization of drone path

7.4.2 Generation of Continuous Signals

Once the path coordinates have been processed and a full ordered list of coordinates has been created, they must be accessed within Simulink. Simulink lookup tables are used to accomplish this by making the list of values addressable and outputting their values as continuous signals. Each address within the lookup tables corresponds to a specific line that the drone must follow, and so for every change in direction that the drone must make, the lookup address must be incremented to output the next set of values. This creates the required continuous signals in Simulink as shown in the first image of Figure 7.7. However, they cannot be sent to the drone directly. These signals instantaneously change when the address is incremented to output the next coordinate, which would cause the drone to attempt to move to the new position as fast as possible. Velocity control must therefore be implemented, as the payload must be able to move at a constant, controllable velocity throughout the flight.

Velocity control translates into limiting the derivative of the signals, resulting in linear slopes being generated in place of the instantaneous changes that come from the lookup tables. These linear slopes form the continuous X, Y, Z, and yaw signals that can be provided to the drone, where the slope of the line is the velocity of the drone in m/s. During model initialization, MATLAB code calculates the maximum velocities for the X, Y, and Z directions for each line that the drone will follow, and the values are again accessed in Simulink using lookup tables. The maximum velocities are calculated such that the drone moves in all three directions over the same period of time. These values are used as the maximum derivatives for each direction to ensure that the drone moves in a straight line from one coordinate to the next, rather than completing its movement in one direction before another. An example of the implementation of rate limiting with calculated maximum velocities is shown in Figure 7.7. The drone's yaw is also limited this way, however its maximum rate of

change is constant throughout flight, and does not require additional calculations.

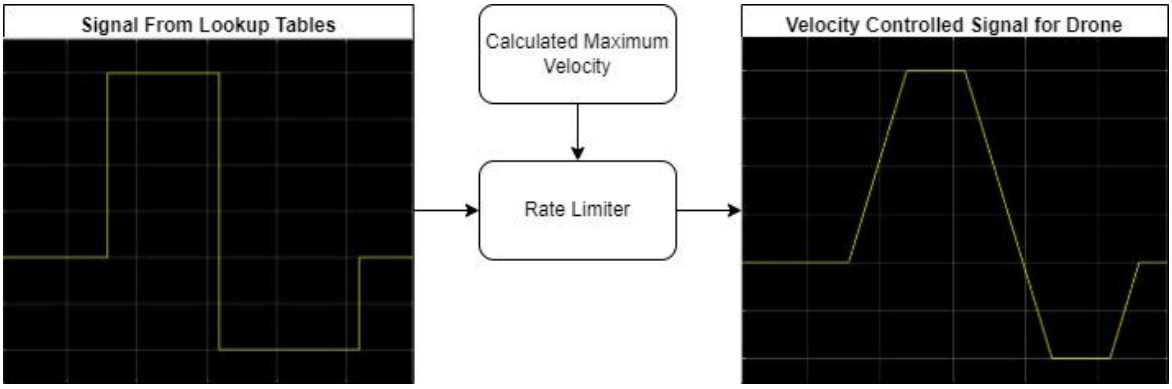


Figure 7.7: Example of rate limiting to control drone velocity

7.4.3 Proximity Triggering

In order for the drone to move sequentially from one coordinate to the next, it must have a method of incrementing the address used by the lookup tables in order for them to output the signals for the next coordinate. This is accomplished using the measured pose of the drone, by determining its proximity to the desired pose that is output by the lookup tables. Once the measured pose of the drone is within a threshold distance of the desired pose, the address supplied to the lookup tables is incremented, causing them to output the signals for the next coordinate. This ensures that the drone meets the metric of getting to within 5cm of each coordinate in the path by setting the threshold value to 5cm. This makes up the second half the cycle required to move the drone from one coordinate to the next in a loop, as shown in Figure 7.8.

Due to the method of integration between the Object Tracking module and the Drone Control module, the front of the drone must always point in the direction it is moving to ensure that the camera orientation does not change in relation to the payload. This requires the drone to change its yaw position between movements, or in other words, the drone cannot move while the yaw is changing and the yaw cannot change while the drone is moving. Yaw proximity is used to implement this into the Autonomous Trajectory Calculation subsystem. Once the drone's measured yaw is within a threshold of its desired yaw, drone movement is enabled and the position proximity system can take over. This repeats for every coordinate the drone visits, and ensures that the yaw of the drone is changed first, followed by the movement of the drone. The entire cycle required to sequentially move the drone between all coordinates is shown in Figure 7.8.

7.4.4 Payload Drift Compensation

While the payload is being dragged by the drone, it can experience different amounts of friction or elevation changes that cause it to go off course. To compensate for the payload drifting off course, the Object Tracking module is used to track the payload using the Quanser QDrone's bottom facing camera, and the Autonomous Trajectory Calculation subsystem

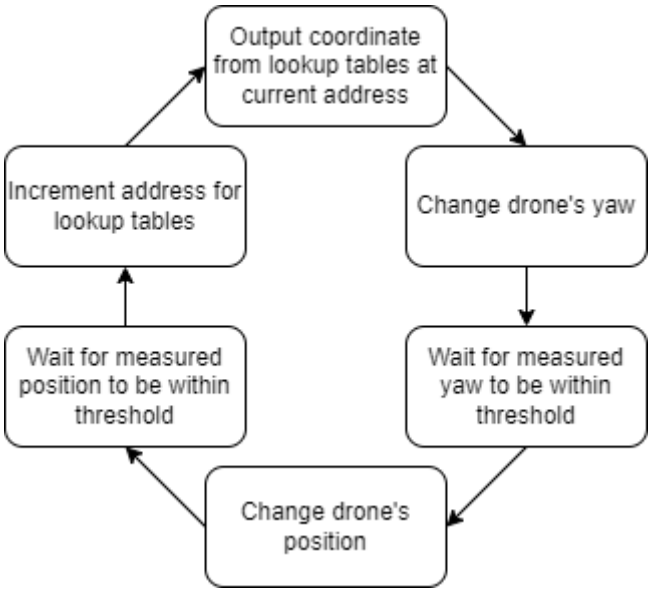


Figure 7.8: Cycle used to sequentially move drone between coordinates

alters the desired position of the drone accordingly. The Object Tracking module takes the line that the drone travels on and determines the distance, or offset, between the centre of the payload and this line. The desired position of the drone (X, Y) and the drone's yaw are also known, and these values can be used in conjunction with the offset to calculate a new optimal position for the drone to compensate for the movement of the payload. As the drone begins to compensate, the camera will move with the drone and will therefore add additional offset which must also be accounted for. The resulting calculation to determine the new position of the drone (X', Y') can be seen in equations 7.1a and 7.1b. This allows the drone to oppose the drift of the payload by an equal and opposite amount after only a few iterations of the equations which are done every time step of 1ms. A representation of the equal and opposite movement of the drone is shown in Figure 7.9, where the payload has drifted off of its intended path by distance d, resulting in the drone compensating in the opposite direction by distance d.

$$X' = X - \frac{\text{offset}}{2} sin(\text{yaw}) \tag{7.1a}$$

$$Y' = Y + \frac{\text{offset}}{2} cos(\text{yaw}) \tag{7.1b}$$

The drift compensation used in the Autonomous Trajectory Calculation subsystem is an altered version of the drift compensation described above. The position of the payload provided by Vicon rather than the offset provided by the Object Tracking module due to complications with integration that will be discussed in Chapter 8. The position of the payload is used to calculate its offset from the line that the drone is traveling on, and the drone compensates in the same way that is described above, as shown in Figure 7.9.

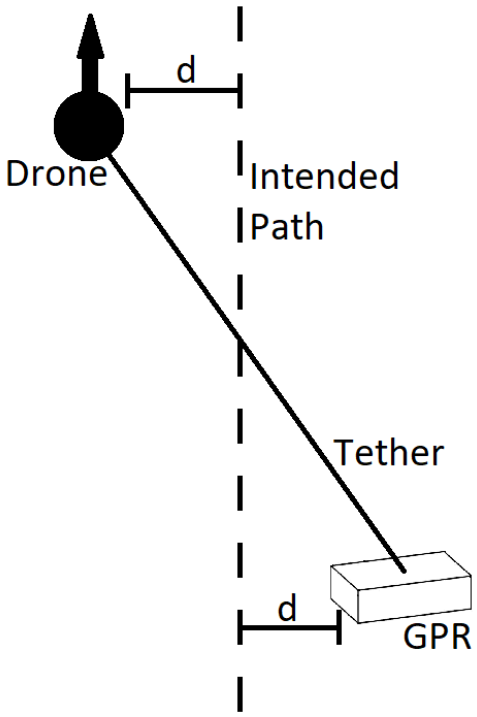


Figure 7.9: Movement of drone for drift compensation

7.5 Drone Control Testing

Testing took place throughout the design process, and can be divided into two major phases. The first phase was the development of the Autonomous Trajectory Calculation subsystem using a simulated testing environment in Simulink. The subsystem was tested throughout development using simulated inputs and drone parameters in order to efficiently develop the model. The second phase was the lab testing of the Autonomous Trajectory Calculation subsystem. This involved the fine tuning of the drone's control parameters, as well as the payload creation, tethering, and drag testing. The goal of testing was to meet all metrics relating to the physical drone outlined in Table 1.2. This includes the drone visiting all coordinates to within 5cm, correcting for payload drift within 40ms, and achieving a battery life of at least 8 minutes.

7.5.1 Coordinate Following Testing

Throughout the development phase of the Autonomous Trajectory Calculation subsystem, all testing was done using a simple list of coordinates that were example output from the Path Planning module. In order to simulate the drone's measured position, the desired position output was delayed by 15ms, and fed back into the system as the measured position to simulate the slight delay that would be expected in lab testing. Other parameters such as the proximity thresholds and velocity values were simulated to match the expected values in the lab. The input from the Object Tracking module was also simulated as a sine wave which was equivalent to the payload swaying from side to side. This method of testing enabled rapid prototyping and simplified the transition to lab testing.

The transition to lab testing involved the integration of the Autonomous Trajectory Calculation subsystem into the Mission Server. This was as simple as replacing the drone positioning subsystem that was included in the Mission Server with the Autonomous Trajectory Calculation subsystem, as all inputs were readily available other than the object tracking data. Simple paths were then supplied to the drone, and the proximity threshold and maximum velocity values were tuned to stabilize flight. A proximity threshold value of 5cm was achieved without the drone requiring excessive time to trigger the next coordinate. The drone was able to sequentially visit all coordinates supplied by the Path Planning module within this 5cm threshold, which meets our coordinate proximity metric of 5cm.

7.5.2 Payload Dragging Testing

To simulate the GPR that would be dragged in a full-scale test, the box shown in Figure 7.10 was filled with weight and tethered to the drone. The size of the box was scaled down to match the scaled-down drone, and its weight was chosen such that it did not put excessive strain on the drone. The Quanser QDrone is rated to lift a 300 g load, and after testing various weights, it was determined that dragging a 200 g box on a rubber floor resulted in 80% load on all four rotors, and so 200 g was selected.

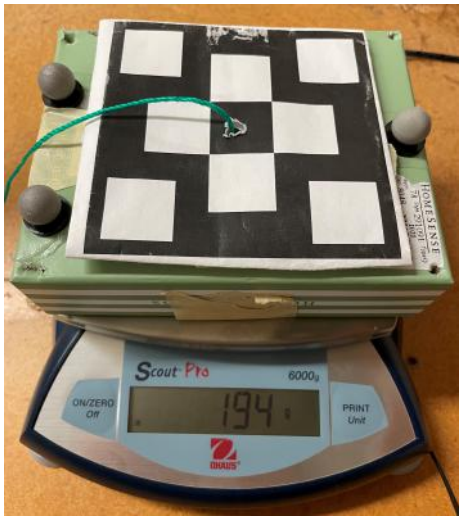
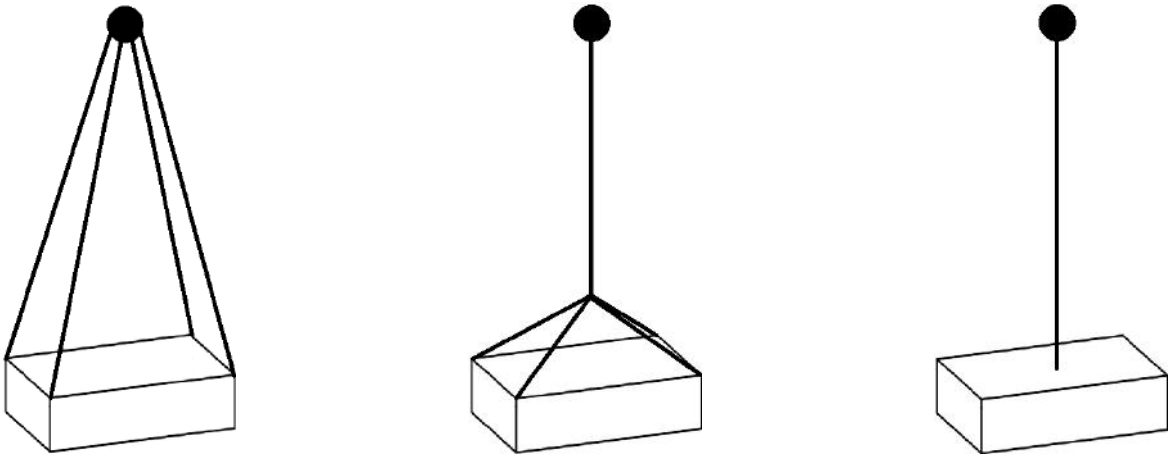


Figure 7.10: Box used as GPR replacement at small-scale

The method of tethering was also tested in various configurations, three of which are shown in Figure 7.11. A single attachment point at the center of the drone was selected as we determined in Section 2.4.2 that the orientation of the payload does not need to be maintained, and it ensures all four rotors experience equal load. Figure 7.11(c) shows the tethering method that was selected, as it minimized the torque applied to the box over all possible orientations that the box could be dragged, resulting in the box flipping less. The tethers in Figures 7.11(a) and 7.11(b) produced additional torque on the box, causing it to flip more often.

Once the box and tethering method were finalized, various test flights took place in order to determine the drone height, the tether length, and the overshoot value. Drone height was selected to be 1 m with a tether length of just over 1.1 m. This ensured that the box



((a)) ((b)) ((c))

Figure 7.11: Possible drone-to-box tethers

did not experience excessive wind from the rotors which could cause the box to lift off the ground, while also ensuring that the box remained within the view of the drone's bottom facing camera for the Object Tracking module. The view from the bottom facing camera while the box is being dragged can be seen in Figure 7.12, with the taut tether being slightly visible going horizontally across the image. The overshoot value could then be calculated using the drone height and tether length, however this value changed throughout testing as the string tether stretched over time.

Figure 7.12: Quanser QDrone bottom camera view

Throughout these test flights, the battery life of the drone was monitored, and a maximum flight duration of approximately 7 minutes was observed. This does not meet our battery life metric of at least 8 minutes, which is 66% of the drones normal battery life of 12 minutes. The load experienced by the rotors while dragging the payload was greater than 1.5 times their normal load, which indicates that a battery life of 7 minutes is within reason.

The final system that required testing was the Drift Compensation subsystem. As previ-

ously stated, due to integration complications, this testing was done by tracking the position of the payload using Vicon rather than the Object Tracking module. A string was attached to the side of the box that could be pulled in order to move the box off of its intended path. The drone was put on a test flight and the box was pulled, at which time the drone was observed to move in the opposite direction of the box movement, eventually bringing it back to its intended path, demonstrating successful drift compensation. A dead-zone was added to the subsystem to disable corrections of less than 2cm, and no other corrections were needed. In Figure 7.13, the box's intended path is shown in orange, and the box has been pulled to the right, causing the drone to move to the left by an equal amount in order to compensate.

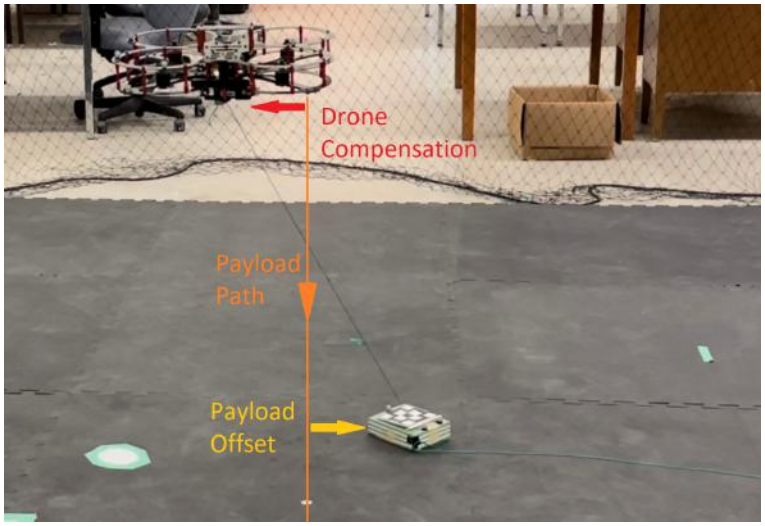


Figure 7.13: Drift compensation testing

Since the implementation of drift compensation was done in Simulink, the drone is able to respond to payload drift within a single time step of 1ms. Our original drift compensation metric aimed for correction to begin within 40ms, as obtaining payload positional data from the Object Tracking module was the limiting factor in the response time. However, since Vicon is used in place of the Object Tracking module due to integration issues, the payload positional data can be obtained within a single time step. The result is drift compensation beginning within 2ms of box drift, which far exceeds our metric. While this can be considered a success, it is impossible to say what the response time would be if the Object Tracking module were used on the drone as originally intended.

Chapter 8

System Integration and Validation

8.1 Integration

The drone system is made up of multiple modules that need to be integrated. The main points for integration are between the camera of the Quanser QDrone and the Obstacle Detection module, the Obstacle Detection and Path Planning modules, and the Path Planning and Drone Control modules. The Object Tracking module was intended to integrate with Drone Control, but due to unforeseen complications we were unable to do so.

The Obstacle Detection module takes input from the Omnivision OV7251 of the Quanser QDrone and proceeds to use a region growing image segmentation algorithm to find obstacles in the image. It then creates a CSV file map of those obstacles to send to the Path Planning module. To integrate the Obstacle Detection module, some changes to the original design had to be made. Originally the intent was for the region growing image segmentation algorithm to be run on the drone, and the output CSV would be sent wirelessly back to the base station computer. In testing we realized that running the image segmentation algorithm on the drone causes it to crash while in flight.

To avoid crashing, the method of integration for the Obstacle Detection module was changed to reduce the computational load on the drone. The drone now only supplies a live camera feed to the base station, where an image is captured once the drone is at the appropriate position in the middle of the drone cage at a height of 2.1 m. After the image is captured, the drone lands safely and the image is processed on the base station. The output CSV file is saved in a known location on the base station for the Path Planning module to use as input. Figure 8.1 shows the process for Obstacle Detection integration testing.

Each value in the CSV file from the Obstacle Detection module represents whether or not an obstacle exists in that area. The Path Planning module uses its algorithm to calculate the final path and translate that path to coordinates in real space. The real space coordinates required some translation to account for the offset of the camera on the drone in order for the path to accurately line up with the real world. It then passes another CSV file of coordinates to visit to the Drone Control module for the drone to use in flight.

We were unable to integrate the Object Tracking module, as the the object tracking

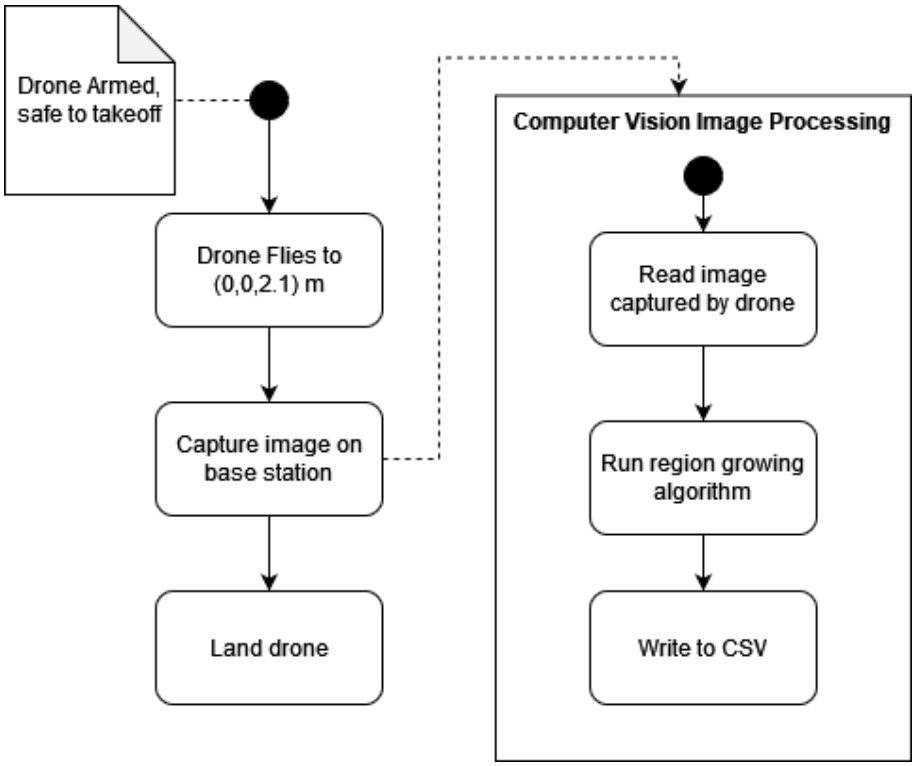
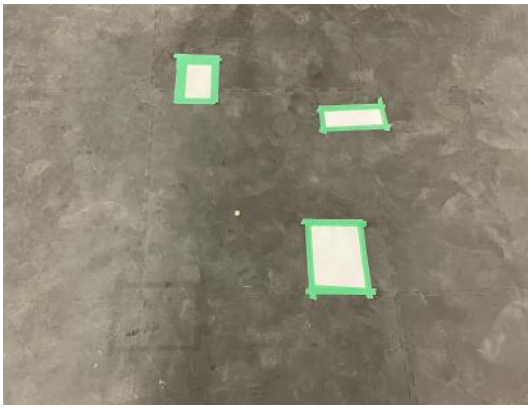


Figure 8.1: Flowchart describing the process for Obstacle Detection integration testing

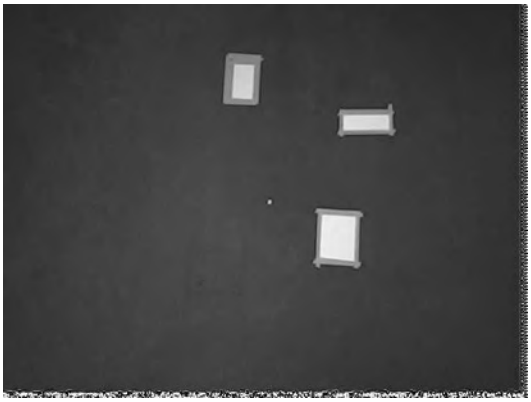
block could not be run on the drone or base station's Simulink models. The Drone Control Simulink models cannot run the extrinsic functions which are necessary for functions used in the implementation of the Object Tracking module, due to software limitations that were not known during the design phase. Due to time restrictions, the testing and validation for our proof of concept was split. The Object Tracking module is now tested by running the object tracking Simulink block with prerecorded video from the drone's flight to validate its ability to detect the location of the object of interest. The path correction of the Drone Control module is tested using STARLab's Vicon system to get the location of the object of interest, rather than using the Object Tracking module. This functionality worked as expected with minor changes to incorporate Vicon.

8.2 Drone System Validation

In order to validate the full drone system integration, a set of white pieces of paper that act as obstacles were placed in the drone flight area, as shown in Figure 8.2(a). The drone then flew to a height of 2.1 m while supplying a video feed to the base station computer where an image was captured of the area to be covered. The image captured by the drone is shown in Figure 8.2(b), and it was then supplied to the Obstacle Detection module.



((a)) Obstacles placed in drone flight area



((b)) Area to be covered captured by drone

Figure 8.2: Obstacles used for full integration testing

Figure 8.3 demonstrates the success of the Obstacle Detection module. Each piece of paper is successfully identified as an obstacle, which satisfies the obstacle detection metric outlined in Table 8.1. The resulting CSV file representing the obstacles was then supplied to the Path Planning module.

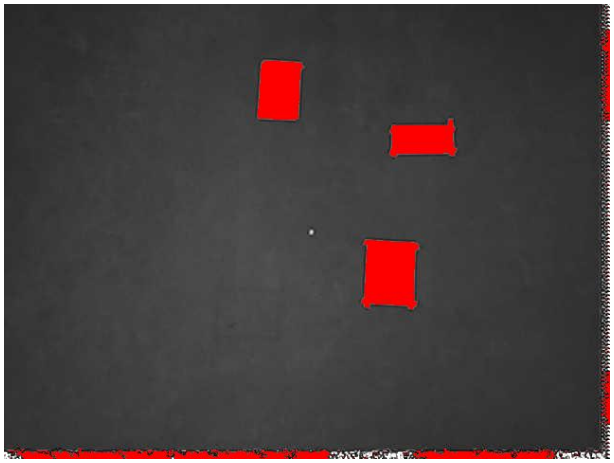


Figure 8.3: Obstacles detected by Obstacle Detection module

Figure 8.4 illustrates the path planned by the Path Planning module using the obstacle map generated by the Obstacle Detection module. Figure 8.4 demonstrates the resizing step of the path planning algorithm and the detection threshold in the top left obstacle. The "lawnmower" pattern is also visible, especially towards the bottom of the image. It can be noted that the plotted path overshoots the obstacle in the upper right when moving upwards by an excessive amount. This is a quirk of the Path Planning module, wherein some circumstances cause the module to be overly cautious in avoiding certain arrangements of obstacles. Since the flight time of the drone is mostly consumed by the drone overshooting and turning, the extra distance covered in these instances does not negatively impact performance by a significant amount.

The path planning algorithm successfully mapped 100% of the area of interest in each test performed. While we cannot say that the Path Planning module would perfectly map every possible arrangement of obstacles, we are confident in saying the system meets the

metric outlined in Table 8.1. However, when translating the path to a real-space flight with the drone, some regions of the area of interest were not fully traversed. The GPR area coverage metric of 99% described in Table 8.1 could not be met, and only approximately 80% coverage could be achieved. However, our initial metric of 99% was more ambitious that it needed to be. It is not necessary for the GPR to traverse the entirety of the area of interest, as the objects we aim to detect are larger than the GPR, allowing for the GPR to skip some stretches of ice and still discover them.

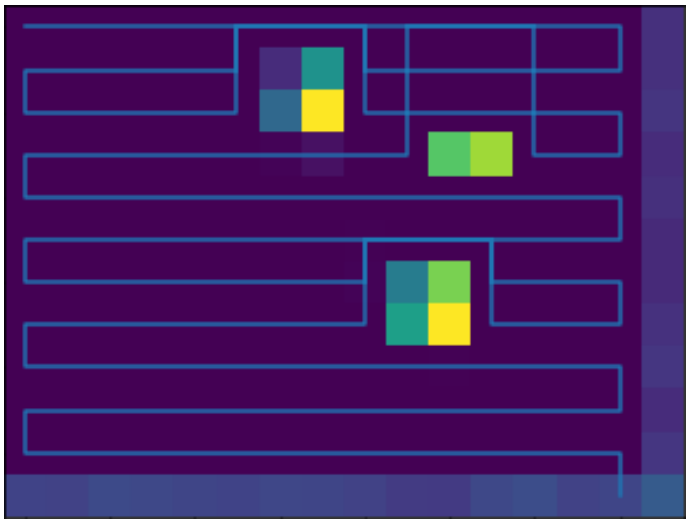


Figure 8.4: Path planned by Path Planning module

The CSV file of coordinates generated by the Path Planning module was then processed by the Drone Control module, and the payload was attached to the drone. The drone then took flight, with drift compensation being integrated using Vicon to track the payload. In this test, the payload did not encounter any of the obstacles, and over the course of all integration tests, 1 out of 13 obstacles were encountered. Compared to the metrics outlined in Table 8.1, the hazard avoidance metric was not met, however insufficient tests were performed to meet this metric, and 1 in 13 obstacles is satisfactory for our purposes.

Figure 8.5 demonstrates that the payload followed its desired path with relatively high accuracy. When comparing the measured position of the payload to its desired path, it successfully came within 5cm of every coordinate. Also, as discussed in Section 7.4.4, drift compensation began within a few milliseconds of the payload travelling outside of its dead-zone. Additionally, drone flight while dragging a payload could only be sustained for just under 7 minutes. When comparing these values to the metrics outlined in Table 8.1, we can conclude that the proximity and compensation metrics have been met, while the battery life metric has not. These results have been discussed in further detail in Section 7.5.

The alignment sequence is shown in the upper left corner of Figure 8.5, where the payload started further up the Y-axis than the drone. The alignment sequence did work to a degree, however an additional alignment in the Y-direction would have significantly improved the starting position of the payload. Systematic error is observed at the left and right ends of the paths, as the payload consistently stops short of the intended path. This could have been caused by an overshoot value that was too low, or due to the 5cm coordinate proximity being

too large, causing the drone to stop early. Random error is also present in how closely the payload follows the path on the longer straight sections. The payload moves from one side to another without predictability, and drift compensation did not compensate for this movement for two primary reasons. Firstly, a dead-zone was in place that disabled drift compensation for any offset of less than 2cm to reduce unnecessary drone movement. Secondly, the drift compensation algorithm does not account for consistent payload movement in one direction, as it reaches its limit and does not compensate further.

Figure 8.5: Measured position of payload compared to its planned path

Since we were unable to integrate the Object Tracking module with the Drone Control module, additional module testing was done with saved video from the full integration test flights to show proof on concept. Similar to the module test that was done in Section 6.3, we were able to track the payload with a per frame processing time of around 100ms - 400ms, depending on hardware being used to run the module. This does not meet our per-frame processing time metric shown in Table 8.1. Had this been incorporated into the Drone Control module, drift compensation could not have been accomplished within the 40 ms metric outlined in Table 8.1 either. Output from these tests is shown in figure 8.6.

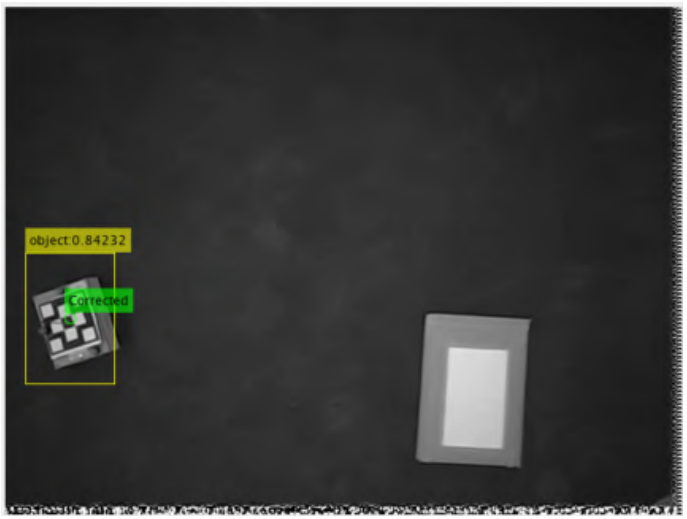


Figure 8.6: Object tracking with saved drone footage

Table 8.1: Metrics for Drone System

Feature	**Metric**	**Target**	**Outcome**
Path-detection	Percentage of area of interest mapped by algorithm	>99%	100%
Path-planning	Percentage of area of interest covered by GPR	>99%	80%
Obstacle detection	Percentage of obstacles detected in lab tests*	>99%	100%
Hazard avoidance	Rate of hazard encounter	<1/100 hazards	1/10 hazards
Object tracking	Per-frame processing time	<33ms	<400ms
Drone control	Maximum drone proximity to each waypoint	5cm	5cm
Drone control	Time to begin correcting drone path to compensate for box drift	<40ms	<10ms
Battery life	Time between required battery change	≥8 minutes	>7 Minutes

* Metric added after design review 2

8.3 GPR System Validation

In order to validate the full GPR system, multiple tests were conducted at the SERF facility in a live environment. Pig carcasses were used in order to get authentic radargrams from the tests. Through viewing the radargram data, we were able to quantify the results and compare them to the metrics set prior.

Initial review of the radargrams indicated a detection rate of 85.71%. This detection rate was calculated by using all the radargrams which were taken over the actual location of the pigs. Eighteen of the 21 radargrams taken had shown the actual presence and location of the pigs, giving us a 85.71% detection rate.

The same format of review was applied for the false detection metric. By the time this metric was measured, more testing was completed and more radargrams had been collected. Of the 39 total radargrams collected following the placement of the piece of plywood, only 6 radargrams detected the plywood, resulting in a false detection rate of 15.38%. This result did not meet our target metric of 10% that was set in design review 2. With a deeper understanding of how a GPR operates, if we were to retest this metric then we would do so in a separate empty tank. The tank would contain objects that are commonly found under water which would allow us to test for a more accurate false detection rate.

Preliminary testing was done to determine the location accuracy metric for the Data Interpretation module. One test showed that the locations of detected objects were determined with a resolution of less than 60cm. Unfortunately, there was not enough time to test

extensively for this metric, and because we were unable to confirm the exact location of the pigs at the time of testing, the accuracy cannot be determined.

The metrics set for the hardware of the GPR, were to confirm proper functionality. It was necessary for the hardware to be running in a consistent manner in different conditions to properly collect data without any external discrepancies. The metrics of antenna locating accuracy and GPR frequency were set up for this reason. The GPR frequency metric was met by conducting proper calibration prior to testing. This ensured that the GPR was attached to the correct antenna such that it would operate at the appropriate center frequency. The antenna locating accuracy was met by observing the radargram scans. This was limited by the GPR's depth range, up to which objects of interest could be detected, but beyond which data cannot be collected. We observed the radargrams we collected in which the pigs were present to see if the objects of interest were within this threshold. All instances satisfied this criterion, and therefore this metric was met.

Table 8.2: Metrics for GPR System

Feature	**Metric**	**Target**	**Outcome**
Data interpretation	Location accuracy/resolution	<60 cm	Not determined
Detection	Object of interest detection rate	>80%	18/21 = 85.71 %
False detection	False detection of object of interest	<10%	6/39 = 15.38%
500MHz antenna locating accuracy	Accuracy of marked location of object of interest	<1 m	Range met
1GHz antenna locating accuracy	Accuracy of marked location of object of interest	<0.5 m	Range met
Data acquisition speed	Speed at which the GPR travels	1m/s - 1.42 m/s	Range met
GPR frequency	Center frequency of GPR	500 MHz & 1 GHz	Metric met

Chapter 9

Future Consideration

9.1 Ground Penetrating Radar

For the GPR, there was discussion regarding how we would go about potentially harnessing it to the drone. Many things had to be considered as carrying and dragging the GPR in real world conditions over the ice would require not only proper harnessing, but also proper placement of the GPR itself. Weight distribution would be a large factor to the placement of the different components. The battery pack alone weighs 3.6 kilograms. This heavy weight would not only cause issues with the payload delivery but also impact the battery life of the drone itself. This would lead to a reduction in the overall operation time of the drone. To counter this, consideration would have to be made to design a smaller battery pack that would weigh less then the original and still be able to provide enough power to last an entire testing session. Another consideration was to secure the components onto the vessel. This would ensure that when the drone is carrying the components, any sudden movements would not lead to them falling due to the uneven distribution in the weight.

Further considerations were made for the data parser. We discussed that the next step for the data parser would be to incorporate one more axis that would be acquired from the GPS location of the drone. This additional axis would be used to plot a three-dimensional map of the area tested, giving us a better idea about what lies below the surface underneath the GPR. We would compare one radargram's dimensions to the next and interpolate the figures accordingly to give us a final 3D plot.

Lastly, the final consideration made for this section was the development of an additional module, the sole purpose of which would be to integrate the drone and GPR. It would deal with the possibility of a remote trigger that would work in tandem with the Path Planning and Drone Control module. This new module would be among the final steps before this project goes from a proof of concept to real-world application.

9.2 Obstacle Detection

Currently this module is capable of differentiating a safe region for the GPR to operate as long as there is a difference in appearance between the operating surface and obstacles. In the lab tests, this is achieved by modelling the obstacles with white printer paper which has a high amount of contrast to the dark grey floor. In a working scenario the assumption is that the surface of the frozen river will appear distinct from any holes in the ice or debris on the surface. In a real test this may not be a safe assumption to make; clear ice may appear similar in colour to holes in the ice and would not be picked up by the region growing image segmentation algorithm.

In the case of a full system integration with a large drone capable of dragging a GPR across ice, a more robust Obstacle Detection module would be advisable. A possible direction to improve the Obstacle Detection module is to take a large amounts of training pictures of frozen lakes and rivers and label any obstacles in the pictures. This data can then be used to fine tune a pre-trained convolutional neural network [20]. With enough data this should be able to accurately detect and classify obstacles.

9.3 Satellite Path Detection

Early in project development, satellite imaging was considered briefly as a method of inputting images to the Path Planning module. In contrast to the drone image we currently use, a satellite image would allow for much larger chunks of planning to be done at once, which would be ideal for a larger drone with a large battery capacity. This would also lower the amount of pre-processing time that would have to be done on the ice, as path planning from a satellite image could be done at home. If this was the case, then the overall flow of the full system would change; the obstacle detection module would likely have to change to a real-time system, as it would no longer be feasible for the path planning module to use obstacle detection as its input.

However, a major issue with performing path planning in this method is that a significant portion of each image is not the area we care about. We wish to plan only on the water in the image, and not on the land. Therefore, a small proof-of-concept script was written to attempt to segment satellite images into land and water regions using OpenCV2. This script grey-scaled the image, then applied a mask based on each pixel's brightness (as water tends to be darker in satellite imagery than land). These chunks of dark pixels were grouped as an object, then contoured to eliminate noise and fuzziness. The specific brightness threshold, contouring method, and contouring aggressiveness would have to be fine-tuned if used in the field. A basic sample of this algorithm is shown in figure 9.1, using a satellite image of the St. Lawrence Seaway.

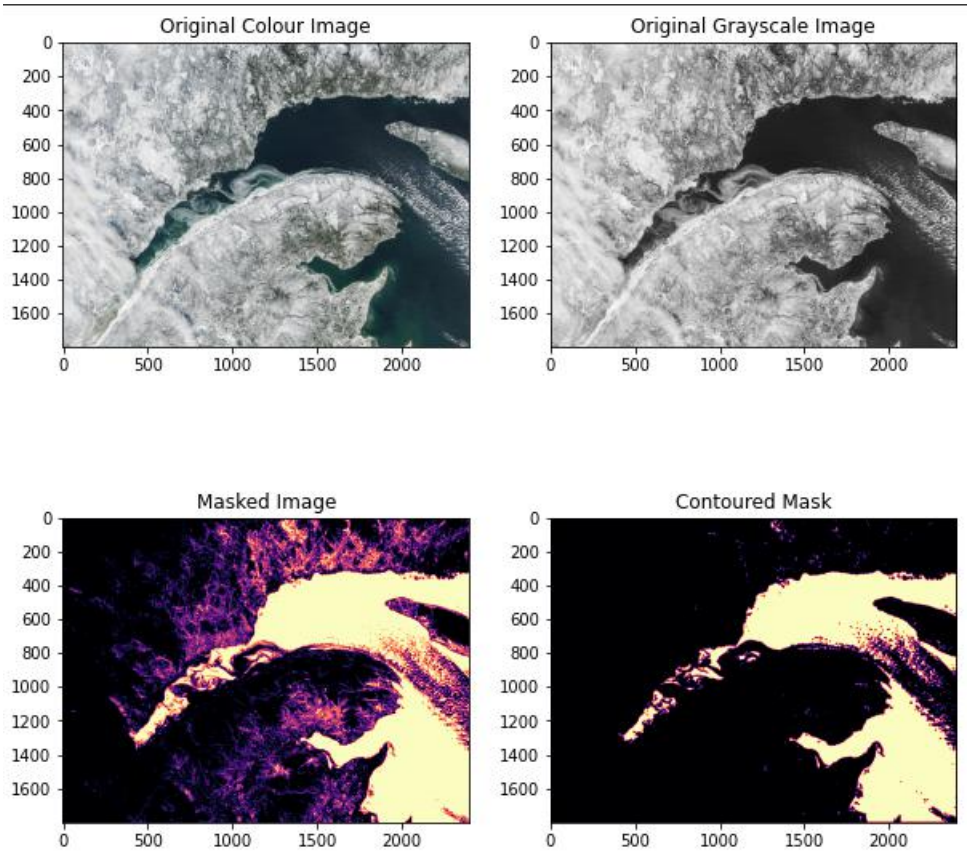


Figure 9.1: Sample satellite path detection using the St. Lawrence Seaway.

9.4 Object Tracking

During the integration phase of this module we were unable to to integrate with the Drone Control module as the module was incapable being run on the Drone. Our current system could be improved by running the module on a separate processing unit and implementing a communication system that is able to communicate with the drone to receive an image and detect the location to be sent back to drone. This improvement would also need to consider the time required to communicate. Therefore, this would require communication protocol that is fast enough to track the payload.

9.5 Drone Control

The Vicon Motion Capture System plays a key role in enabling accurate control of the drone when dragging a payload. However this is not a viable solution for acquiring positional data in outdoor scenarios. A GPS could be used to acquire this positional data depending on the accuracy requirements of the application. Should more accuracy be needed, a DGPS would be a suitable choice, as it can achieve much higher resolution, however more hardware is required.

Another point to consider is the use of DGPS in GPR tracking. The original design of the Drone Control module relied on object tracking of the GPR in order to obtain its location, however if a DGPS is in place, it is trivial to obtain much more accurate positional data of

the GPR this way. This would reduce real-time computational requirements of the drone, and eliminate the requirement for the GPR to be in the view of a camera at all times. This is especially relevant when considering the height at which a full-size drone must fly. Testing was done with a DJI Matrice 600 Pro dragging the GPR over ice, and the drone required an altitude of at least 10m as shown in Figure 9.2. This would require the drone to carry a high-quality camera in order to obtain a live video feed of the GPR that could provide accurate readings to the Drone Control module.

Figure 9.2: Full-scale test dragging GPR using DJI Matrice 600 Pro over ice

A final improvement could be made to the Drift Compensation subsystem. This subsystem utilizes a rudimentary algorithm to oppose the drift of the payload, however this is not robust to many conditions that could be found in real-world scenarios. The use of Proportional Integral Derivative (PID) controls would be able to compensate for drift more consistently and faster than the current solution.

Chapter 10

Conclusion

In summary, this report presents the design and implementation of successful proof of concept of Ground Penetrating Radar Collection from an Autonomous Drone. Even though we did not meet all the metrics that were initially proposed, we were still able to successfully implement both an autonomous drone system along with a GPR system capable of both collection and interpretation of data.

The team was able to complete a functioning drone system prototype that is capable of detecting obstacles in a particular area of interest and planning path around said obstacles. The drone able to take the planned path and follow the given path while dragging a scaled down payload. The drone was able to direct the payload and avoid the detected obstacles. Additionally we were able to track the payload and correct its positioning in real time. Based on the above achievements we believe that we successfully implemented a proof of concept that can be further developed into a full prototype.

We were able to take our GPR system out into the field to conduct controlled environment testing that had further strengthened the feasibility of this proof of concept. With the opportunity to conduct live tests in the field we were able to successfully detect the pigs that had been used as the objects of interest. This achievement has made significant progress towards a fully integrated prototype that can be used in the foreseeable future on frozen lakes and rivers to detect objects of interest underneath ice via an autonomous drone.

Bibliography

[1] *Ground penetrating radar*, 2020. [Online]. Available: https://tech27.com/resources/ground-penetrating-radar/ (visited on 03/06/2022).

[2] GeoSci, *Basic principles¶*. [Online]. Available: https://gpg.geosci.xyz/content/GPR/GPR_fundamental_principles.html (visited on 03/05/2022).

[3] A. Benedetto and F. Benedetto, *Radargram*, 2014. [Online]. Available: https://www.sciencedirect.com/topics/engineering/radargram (visited on 03/18/2022).

[4] A. Benedetto and F. Benedetto, *From a single GPR signal to a radargram*. Science Direct, 2014. [Online]. Available: https://www.sciencedirect.com/topics/engineering/radargram (visited on 03/18/2022).

[5] Sensors and Software, *Pulseekko product manual*, Sep. 2005. (visited on 11/09/2021).

[6] Sensors and Software, *Spidar product manual*, Feb. 2018. (visited on 11/09/2021).

[7] L. B. Schook, *Unraveling the swine genome: Implications for human health*. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/25689318/ (visited on 03/13/2022).

[8] S. S, *What is pandas in python? everything you need to know*, Nov. 2021. [Online]. Available: https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/#:~:text=Pandas (visited on 03/19/2022).

[9] A. Hayes, *What is correlation in finance?* Feb. 2022. [Online]. Available: https://www.investopedia.com/terms/c/correlation.asp (visited on 03/06/2022).

[10] D. Marshall. "Region growing." (Sep. 1996), [Online]. Available: https://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node35.html (visited on 03/16/2022).

[11] D.-J. Kroon. "Region growing." (2022), [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/19084-region-growing (visited on 02/21/2021).

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," Jun. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[13] M. Gupta, *Yolo-you only look once*, May 2020. [Online]. Available: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4 (visited on 12/22/2021).

[14] A. Ahmed and K. Terada, "Object detection and tracking using kalman filter and fast mean shift algorithm.," vol. 1, Jan. 2009, pp. 585–589. (visited on 12/22/2021).

[15] MathWorks, *Object detection using yolo v3 deep learning*. [Online]. Available: https://www.mathworks.com/help/vision/ug/object-detection-using-yolo-v3-deep-learning.html (visited on 12/27/2021).

[16] MathWorks, *Evaluatedetectionprecision*. [Online]. Available: https://www.mathworks.com/help/vision/ref/evaluatedetectionprecision.html (visited on 01/18/2022).

[17] MathWorks, *Use kalman filter for object tracking*. [Online]. Available: https://www.mathworks.com/help/vision/ug/using-kalman-filter-for-object-tracking.html (visited on 01/17/2022).

[18] "Vero: Compact super wide camera by vicon." (Mar. 2022), [Online]. Available: https://www.vicon.com/hardware/cameras/vero/ (visited on 02/18/2022).

[19] "Qdrone." (Apr. 2021), [Online]. Available: https://www.quanser.com/products/qdrone/ (visited on 10/08/2021).

[20] MathWorks, *Object recognition*. [Online]. Available: https://www.mathworks.com/solutions/image-video-processing/object-recognition.html (visited on 03/03/2022).

[21] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, pp. 247–253, 2000, ISSN: 0929-5593. DOI: https://doi.org/10.1023/A:1008958800904. [Online]. Available: https://link.springer.com/article/10.1023/A:1008958800904.

[22] G. Miraglia and L. R. Hook, "A feedback motion plan for vehicles with bounded curvature constraints," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2019, pp. 1180–1186. DOI: 10.1109/UEMCON47517.2019.8993031.

[23] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, 2001, 1927–1933 vol.2. DOI: 10.1109/ROBOT.2001.932890.

[24] *Lecture-10-klt*, Mar. 2019. [Online]. Available: https://www.crcv.ucf.edu/wp-content/uploads/2019/03/Lecture-10-KLT.pdf (visited on 11/07/2021).

[25] D. Wagener and B. Herbst, "Face tracking: An implementation of the kanade-lucas-tomasi tracking algorithm," Jul. 2014. [Online]. Available: https://www.researchgate.net/profile/Ben-Herbst/publication/228883294_Face_Tracking_An_implementation_of_the_Kanade-Lucas-Tomasi_Tracking_algorithm/links/0046353bf0bf3520ae000000/Face-Tracking-An-implementation-of-the-Kanade-Lucas-Tomasi-Tracking-algorithm.pdf (visited on 11/07/2021).

[26] MathWorks, *Vision.pointtracker*. [Online]. Available: https://www.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html (visited on 11/10/2021).

[27] MathWorks, *Extractfeatures*. [Online]. Available: https://www.mathworks.com/help/vision/ref/extractfeatures.html (visited on 12/19/2021).

[28] H. Shuo, W. Na, and S. Huajun, "Object tracking method based on surf," *AASRI Procedia*, vol. 3, pp. 351–356, 2012, ISSN: 2212-6716. DOI: https://doi.org/10.1016/j.aasri.2012.11.055. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212671612002144 (visited on 12/19/2021).

Chapter 11

Appendix

11.1 Appendix A: Alternative Considerations

In developing each module, we made many considerations which were experimented with and tested early in development but were ultimately not followed through for a variety of reasons. These considerations are listed below for completeness.

11.1.1 Data Collection and Interpretation

For the GPR subsystem, one method of visualization which was considered was to use a stacked line plot which would display all radargram scans from one session of testing. The intention had been to take multiple single line plots and have them stacked and identified, such that the lines containing higher peaks would indicate to us the location of the objects of interest. An example of this approach is shown in Figure 11.1. This approach was rejected by the team as it did not allow us to meet our data interpretation metric due to the misinformation it had provided. This is in reference to the peaks seen in Figure 11.1 at the 50th point on the x-axis. There is a great density of peaks, suggesting the presence of an object, but no object is actually present. This can be seen in Figure 3.2 at the same point on the x-axis.

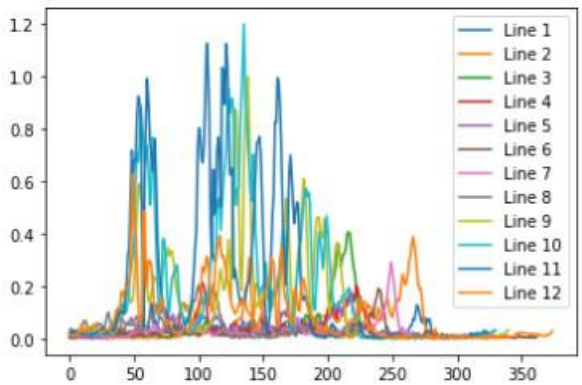


Figure 11.1: All lines plotted from January 13th 1 GHz test session

11.1.2 Obstacle Detection

Two main computer vision techniques for detecting obstacles were considered. The first is an object detection algorithm that classifies objects within the GPR's desired path. The second technique is image segmentation, the one which was eventually used, which segments the image into regions that are either safe for the GPR or an obstacle.

Object detection is a computer vision technique for detecting and locating objects within an image or video [20]. Deep learning is a method that can implement object detection, a common deep learning model being a convolutional neural network (CNN) [20]. To train a CNN to detect desired objects, a large number of labelled photos are required. To use a CNN for obstacle detection, we would need to take hundreds of pictures from the drone while piloting it over frozen lakes and rivers, and label the objects within the images. Due to the difficulty of collecting training data for a CNN that can recognize all obstacles that the system may encounter, this method was not used.

One image segmentation technique which was considered but not used is semantic segmentation, which uses deep learning to classify and group parts of an image. Semantic segmentation has similar shortcomings to a deep learning object detection algorithm, in that it requires a lot of labelled data for training.

11.1.3 Path Planning

Early in this module's development, other more robust algorithms were considered. The path planning algorithm is essentially an area coverage algorithm. Therefore, there are several well-established algorithms that could have been used in place of a developed one.

The first that was examined was an altered A* algorithm named Boustrophedon [21]. A* is a well-known path-planning algorithm using a weighted graph maintained via some cost (eg: distance, time, etc.). This graph is grown until it reaches from the starting node to the termination node, and it selects the minimum-cost path it has discovered between the two. However, A* is designed for a single path between a single starting point and a single ending point. It does not cover an area, which is what the Path Planning module must do. Instead, Boustrophedon decomposes the area into subregions, and performs A* within each region to cover the areas.

The second is Wavefront [22], another point-to-point path planning algorithm that can be adapted to cover an area instead. Beginning from a point of interest, a "wave" of exploratory nodes examine the immediate area around the original point. This process is repeated until the end point is reached. In the area coverage version, each subsequent wave is treated as a new area to cover.

The last is spanning tree coverage [23], which creates a minimum spanning tree between the cells that represent the area to be covered.

However, the issue with these algorithms is that they attempt to minimize distance traveled or total computations made. In our application, while a minimal distance is desired, it is not the single most important metric by which to evaluate a path planning algorithm.

Rather, it is the total amount of turns made. Because of the required overshooting of the drone at each turn, a considerable amount of time is used to change directions. Therefore, it was decided to create a relatively simple in-house algorithm for the Path Planning module. This is the "lawnmower" algorithm as mentioned earlier.

11.1.4 Object Tracking

Initially a Kanade-Lucus-Tomasi (KLT) algorithm was used to track the payload. The KLT algorithm is a feature-based tracking algorithm that starts off by detecting an object in an initial frame by identifying point features which are then tracked across successive features [24]. This algorithm tracks features from one frame to another by calculating the sum of the squared intensity differences between the features of two consecutive frames [25].

The KLT algorithm can be implemented using functions from MATLAB's Computer Vision Toolbox. We had initially looked into implementing this algorithm by using Harris features for identifying initial features and the `vision.PointTracker` function from the Computer Vision Toolbox to track the object from frame to frame [26]. Harris features are commonly used to detect the corner features of an image and can be easily implemented using the `detectHarrisFeatures` function [27]. Although Harris features worked well for tracking features from frame to frame, the function also tracked the corner features of other objects that may also be moving alongside the payload, which was problematic. We also considered using Speed-Up Robust Features (SURF) to identify features to be tracked. SURF is a feature-based object detection algorithm which uses Hessian matrices to detect features [28]. The algorithm works by initially detecting features in a reference image (e.g., a fiducial) and detecting features in the initial frame of the video input. The features from the reference image and the initial frame are matched accordingly. These matched features are then tracked from frame to frame in a similar manner to the Harris features through the `vision.PointTracker` function. The features can be extracted and detected using functions from the Computer Vision Toolbox [27]. Upon initial implementation we found that using SURF features was also not a viable solution, as the algorithm struggled to detect and match enough features accurately in slightly unfavorable conditions.

11.2 Appendix B: Budget

Our project required a Quanser QDrone and Vicon Motion Capture System for the drone subsystem, as well as the required supporting hardware and environment (eg: drone cage net, landing foam, etc.) to perform testing and validation. This was graciously supplied to us by Dr. Philip Ferguson and STARLab at no cost. The drone and supporting hardware, including the Vicon system, originally cost approximately $50,000 CAD.

Additionally, the GPR subsystem required a PulseEkko GPR, a location to perform testing, and the material for the testing (namely, pig carcasses). This was done at SERF, again at no cost to us. The PulseEkko was quoted as costing $29,253.00 CAD and the days of testing at SERF were quoted at approximately $2,500 CAD.

In total, if we were required to purchase the components we used, we would have needed approximately $82,000 CAD.

11.3 Appendix C: Code

A compilation of all code developed and used for this project can be found at the following hyperlink: total code repository.