NELSON MANDELA
UNIVERSITY

# Vision-Based Autonomous Aircraft Payload Delivery System

Masters Research Dissertation

For the qualification towards

MEng (Mechatronics)

Department of Mechatronics

Faculty of Engineering, the Built Environment and Information Technology

Nelson Mandela University

By

James Alderton Sewell

Student Number: 214085996

-

*April 2019*

*Supervisor:* Professor T. van Niekerk (Nelson Mandela University)

*Co-Supervisor:* Professor R. Phillips (Nelson Mandela University)

*Co-Supervisor:* Professor R. Stopforth (University of KwaZulu-Natal)

NELSON MANDELA

UNIVERSITY

**DECLARATION BY CANDIDATE**

**NAME:** *JAMES ALDERTON SEWELL*

**STUDENT NUMBER:** *214085996*

**QUALIFICATION:** *MENG (MECHATRONICS)*

**TITLE OF PROJECT:** *VISION-BASED AUTONOMOUS AIRCRAFT PAYLOAD DELIVERY SYSTEM*

**DECLARATION**:
In accordance with Rule G5.6.3, I hereby declare that the above-mentioned treatise/ dissertation/ thesis is my own work and that it has not previously been submitted for assessment to another University or for another qualification.

**SIGNATURE:** ______________________________________________

**DATE:** ___________________________________________________

## Dedication

I would like to dedicate this dissertation to my grandfather, Barry Crawford Sewell, who passed away during the completion of this research. Thank you for always promoting and encouraging my passion for engineering.

## Acknowledgements

I would like to thank the following people, without their contributions, this research would have not been possible.

I would like to show my gratitude towards the South African Department of Science and Technology's ROSSA programme, the Eskom TESP programme, Nation Research Foundation, MerSETA and Karl du Preez, for providing the necessary funding, which enabled the acquisition of the necessary tools and equipment required for this research.

Professor Theo van Niekerk, for your supervision and guidance throughout my mechatronics career. Thank you for your unrelenting commitment towards the furthering of my academic career and for ensuring that I always remained on course of the final goal.

Professor Russell Phillips, for your guidance throughout my masters and for providing me with your years' worth of experience and expertise. Thank you for your drive and dedication to the furtherment of engineering.

Professor Riaan Stopforth, for your insightful knowledge into new developments in mechatronic technology, which proved invaluable in the development of this research and for providing the necessary academic funding to have allowed me to undertake this research.

Damian Mooney, your knowledge and expertise in drone technology proved invaluable throughout this research. Thank you for your many hours of patience and guidance.

Brogan McArthur, for your years of endless love, support and encouragement, without which I would have never undertaken this research or become the person I am today.

My parents, Mike and Michelle Sewell, for your support and encouragement throughout all of my endeavours.

## Abstract

This research sought to design and develop an autonomous aircraft payload delivery system which utilised an onboard computer vision system for drop-zone identification. The research was tasked at achieving a modular system which could be used in the delivery of a given payload within a 5 m radius of designated drop-zone identifier. An integrated system was developed, where an autonomous flight controller, an onboard companion computer and computer vision system formed the physical hardware utilised to achieve the desired objectives. A Linux-based Robotic Operating System software architecture was used to develop the control algorithms which governed the autonomous flight control, object recognition and tracking through image processing, and payload release trajectory modelling of the system. The hardware and software architectures were integrated into a remote control fixed-wing aircraft for testing. Implementation of the system through simulation and physical testing proved successful and payload delivery was achieved at an altitude of 75 m, within an average displacement of 1.82 m from the true drop-zone location, where drop-zone detection and location were determined through autonomous survey over the approximate drop-zone's location. This research furthered the development of autonomous aircraft delivery systems by introducing computer vision as a means of drop-zone location confirmation and authentication, allowing for greater payload delivery security and efficiency. The results gathered in this research illustrated the possible applications of modular airborne payload delivery systems into Industry 4.0 through the use of such a system in the service delivery sector.

# Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

Industry 4.0 refers to a combination of major innovations in digital technology which are composed to transform the energy and manufacturing sectors. These technologies include: advanced robotics and artificial intelligence, sophisticated sensors, cloud computing, data capture and analytics, digital fabrication, and platforms which use algorithms to direct motor vehicles (for example, autonomous vehicles). These technologies are often used independently. However, when used together, allow for the integration of physical and virtual worlds. Similarly, this research sought to develop and combine an autonomous flight control system with a vision-based identification and recognition system to deliver payloads to targets with non-specific Global Positioning System (GPS) coordinates [19].

To define this system as autonomous, flight control from take-off to mission execution to landing was fully automated, with the only human intervention being that of the aircraft's pre-flight preparation. The system utilised a computer vision system to locate and authenticate a desired drop-zone location, then, with the assistance of onboard computational hardware and software, calculated an appropriate drop trajectory for a given payload, taking into account current flight and environmental conditions. With this information, the system then communicated flight path alterations to the aircraft's flight control unit (FCU) until an acceptable drop trajectory was obtained, from which the system would then release the given payload. The use of the real-time vision system allowed for the system to be able to deliver payloads to targets with nonspecific GPS coordinates.

With each defining element of this research: autonomous flight, vision, and trajectory plotting, having been developed separately and able to work independently, this dissertation focused on the conceptualisation, development and testing of an integrated vision-based autonomous aircraft payload delivery system. Specific focus was placed on the integration of each of these defining elements into a single modular system, requiring low level user input in order to function. This level of autonomy and integration allowed for the combination of independent technologies to be integrated into a system which is capable of contributing to the innovations evolving within Industry 4.0.

## 1.1 Rationale and Motivation

Irrespective of efforts made towards the development of the developing countries, service delivery in developing countries has long been an issue yet to find a solution. This lack of service delivery has subsequently led to the need for an effective and efficient mode of service delivery to be developed. A solution to the service delivery issue would seek to not only promote the development of the service delivery sector but would also look at the fulfilment of basic human rights and the development of the various other sectors, including the manufacturing sector. As such, the development of an effective and efficient service delivery system would seek to provide a solution to political, social, economic and ecological issues. The development of a service delivery system capable of benefiting several developing countries and the sectors within those countries, places such a system within the objective of Industry 4.0 and it is through such a system that, interconnectivity between sectors would be promoted and integrated [12].

This research sought to define the development of an autonomous fixed-wing aircraft that was capable of using a vision-based system to identify and locate desired payload drop-zones and then, adjusting its flight path accordingly, achieve a ballistic trajectory drop of a given payload to said drop-zone. Where, such research seeks to transform the service delivery sector by providing a system capable of the desired service delivery functionality, whilst maintaining a low-cost, efficient and effective design.

Within the framework of this research, four defining components existed: mode of transportation, method of navigation, identification and authentication of service delivery recipient and method of delivery. Each of these components played a role in the development of a system capable of promoting Industry 4.0 within a developing country and the constraints experienced within various sectors. The desired mode of transportation, a fixed-wing aircraft, provided the beneficial functionality of a system which was able to circumvent obstacles faced through land- and sea-based service delivery, and in turn allowing for the delivery of goods and services to time-sensitive or previously inaccessible areas. With the use of such a transportation system, a method of navigation proves necessary to fulfil the delivery process and as such, the introduction of autonomous control into the aircraft provides the ability to optimise the delivery process by mitigating human error. The introduction of autonomy introduces the need for implementation of service delivery confirmation. As such, a method of delivery location identification and authentication becomes necessary to avoid disruption of service delivery based on process-oriented disconnectivity. Finally, with a stable delivery platform in place, the selection of an appropriate delivery method which promotes efficiency and effective service delivery is necessary. Therefore, an airborne method of delivery would prove to be the most feasible option, by providing the required service delivery capabilities without the need for the introduction of additional infrastructure.

However, the application of such a delivery system is not limited to the service delivery sector. There are various ways in which this system could be utilized to fulfil outcomes within several sectors. As such, the applications discussed within this rationale are not exhaustive and should rather be used as a foundation for what could be achieved with the utilization of this system.

## 1.2 Aim, Objectives and Research Contributions

This section discusses the research aim and objectives, and provides an overview of the contributions made through the completion of this research.

### 1.2.1 Aim

The purpose of this research was the development of an autonomous aircraft payload delivery system which made use of vision-based object identification and recognition, in parallel with real-time environmental parameter assessment for application in aircraft flight path alteration and payload drop trajectory modelling to deliver a given payload from altitude to an identified drop-zone location.

### 1.2.2 Objectives

The objectives and sub-objectives pursued in this research include:

➢ The development of a vision-based autonomous aircraft payload delivery system.

   a. Hardware architecture defining the physical infrastructure utilised in achieving an integrated industry standard, compact and modular design.

   b. Software architecture defining the computational infrastructure utilised in achieving autonomous flight control, vision-based identification and location, and payload trajectory computation.

   c. Minimal operator defined inputs for mission execution.

- The development of an autonomous flight control system.
    a. Full mission autonomy must be implemented, where all required flight procedures are to be executed by the onboard hardware and software.
    b. The autonomous flight control system must be able to adjust the aircraft's flight path based on current mission parameters and sensory data.
- The development of a vision-based identification and recognition system.
    a. Development of an image processing algorithm capable of drop-zone identification without the need for operator intervention.
    b. Identify the drop-zone location with respect to the aircraft's current location and provide the onboard flight control unit with a drop-zone location.
- The development of a payload release system.
    a. The development of a mathematical model upon which the ballistic trajectory of a payload could be mapped.
    b. Payload release computation in real-time, as to allow for adaptation of current mission parameters.
    c. Payload delivery within a 5 m radius of the drop-zone location.

### 1.2.3 Research Contributions

In addition to the overall aims and objectives of this research, this research sought to bridge the gap in related research, where applications of similar systems failed to perform efficiently. The areas in which this research contributed towards the furtherment of previous literature were as follows:

- The development of a vision system and image processing algorithm which achieved similar results in target location determination accuracy at an altitude of 75 m when compared to the apparent average accuracy of Hinas et al.'s vision system at an altitude of 40 m [27].
- The development of a fully integrated autonomous aircraft with an integrated autonomous vision system capable of achieving the automated payload release. This research built upon the research conducted by Boura et al. [10]

in the development of an autonomous aircraft with payload release capabilities.

- Based on the work conducted by Boura et al. [10] where payload trajectory modelling was achieved through real-time environmental parameter monitoring, it was found that payload trajectory modelling could be further developed. As such, this research included the development of a two-dimensional two-phase payload release trajectory model for the prediction of a payload's ballistic fall trajectory, which achieved successful displacement prediction results during simulated and physical testing.
- An integrated payload delivery system capable of achieving the same results as achieved by Zipline [50] and DHL Parcelcopter [16], two commercially used systems implementing autonomous aircraft payload delivery. Additionally, the introduction of a vision system, a system the two commercial systems did not possess, increased payload delivery accuracy and security by ensuring payload drop-zone identification.
- The development of a fully automated mission flight plan based on a single user-defined waypoint. Improving upon the DHL Parcelcopter's [16] system of manual user mission designation and subsequently optimising mission waypoint designation.

## 1.3 Hypothesis

A vision-based autonomous aircraft payload delivery system can be used to identify and accurately drop a given payload within a 5 m radius of a given drop-zone location.

## 1.4 Delimitation

As this research consisted of several subsystems, integrated into a single operational system, various limiting factors existed between the subsystems.

- System accuracy and reliability will be limited to the accuracy and reliability of the relevant onboard hardware and computational performance limited by the onboard processing ability.
- Testing limited to duration of onboard battery life.
- Results and testing are limited to weather permitting testing periods.
- Environmental parameters will be modelled to represent a static system, where unforeseen events, such as sudden wind gusts, will be excluded from the relevant computations.

## 1.5 Dissertation Outline

This dissertation was structured as follows. Chapter 1 provides an introductory view into the given research topic, highlighting the rational and significance of the research. Chapter 2 defines the relevant background research pertaining to the application of this research and the relevant technologies applied. Chapter 3 focuses on the developed system architecture, the hardware and software architecture, and the test platform aircraft.

Chapter 4 through to Chapter 6 of the dissertation was structured to follow the order in which an autonomous mission would be executed. Chapter 4 highlights the autonomous flight control system, discussing the processes which constituted an autonomous mission, the flight plan and waypoint allocation. Chapter 5 focuses on the vision system, defining the theoretical approach taken to translate captured imagery into discernible drop-zone information. Chapter 6 discussed the payload delivery system, defining the mathematical foundation which formed the payload trajectory model. Chapter 8 discussed the integration process, defining the method mission initiation and discussing the process of integration implementation in terms of structured execute of the respective control algorithms.

Thereafter, Chapter 8 discussed the testing procedure and results for each subsystem and the integrated system. Finally, Chapter 9 concludes the dissertation, answering the question as to whether the proposed hypothesis was achieved and indicates possible future recommendations and alterations to be made to the current system.

## 1.6 Chapter Conclusion

Chapter 1 represented the introduction to the research, where the research was defined to be the development of an autonomous aircraft vision-based payload delivery system. Aims and objectives were defined, where it was noted that integration between several subsystems was necessary to achieve the desired results. whose design would be modular to allow for variation of application. Elements such as the objectives, hypothesis, delimitation, research timeframe and dissertation outline were discussed. Furthermore, the literature study in Chapter 2 will be used to provide a background in order to achieve the desired outcomes of this research. Providing the reader with the foundation pertaining to relevant technologies and developments in similar fields.

# Chapter 2

# Literature Study

This chapter discussed the application-based approach pertaining to how this research was developed. Research into relevant technologies and commercially available systems capable of the desired functionality were reviewed.

## 2.1 Application-Based Background

As discussed in Chapter 1, this research sought to provide various applications in both the public and private sector, moving from aid relief to privatised package delivery for offshore freighters. The modularity of the system was defined as an overall objective due to the wide variety of applications this research encompassed. To design such a system, insight into the relevant technologies was undertaken. Hassanalian et al. [26] has provided a detailed view of possible drone aircraft applications, as seen in Figure 2.1.

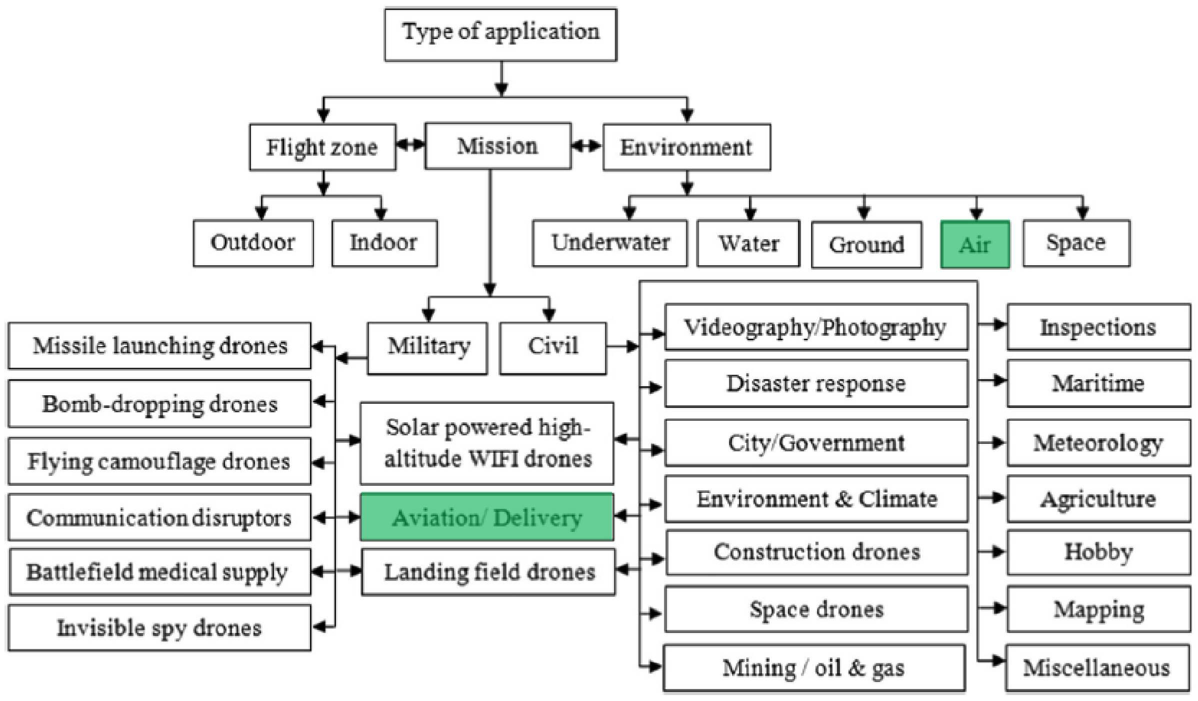


*Figure 2.1: Drone aircraft classifications [26]*

From Figure 2.1, it was noted that applications for delivery systems in drone technology had already been identified and as such, the proposed delivery system was feasible. The applications of drone technology in this research were highlighted in Figure 2.1.

### 2.1.1 Airframes

Prior to the study into airframes, the clarification of the nomenclature utilised in these technologies was necessary to remove any misconceptions. The term drone was first utilised in the early to mid-1900's to classify remotely piloted aircraft used for surveillance, target practice and other military applications. The term for these vehicles changed over the years, until the classification of unmanned aerial vehicles (UAVs) was developed [41]. UAV has become a term that not all aviation authorities have conformed to. The International Civil Aviation Organization (ICAO), among other aviation authorities, employ the term remotely piloted aircraft system (RPAS) when referring to these aircraft systems [29]. Due to the wide variety of terms used to classify these vehicles, this research focussed on and defined the specific type of airframe utilised.

Airframes have been classified based on their features including weight, range, speed, wing span, endurance, production costs and application [26]. As application was the driving factor behind this research, selection of the most appropriate airframe was necessary to achieve the desired results. For the purpose of this research, three airframes were assessed. These frames included: multirotor aircraft, fixed-wing aircraft and quadplane aircraft, and were assessed based on models designed for remote piloting.

#### 2.1.1.1 Multirotor Aircraft

Multirotor aircraft, also known as multicopter aircraft, are aerial vehicles which derive their motion from the control of several thrusting propellers. These vehicles are aerodynamically unstable and require an FCU to ensure stable flight. At their core, multirotors require very few mechanical components to operate and their basic design consists of a frame, control electronics, motors and propellers [5].

Multirotors require thrust to remain airborne, where the failure of a single motor would result in the aircraft becoming unstable and crashing. As such, the power consumption of multirotors is high, where average flight time can range from 5 minutes to 30 minutes depending on the aircraft's size, weight, motors and available power source.

#### 2.1.1.2 Fixed-Wing Aircraft

Fixed-wing aircraft are flying vehicles which possess the capability of flight due to their use of wings to generate lift caused by the forward motion of the aircraft. Fixed-wing aircraft provided both advantages and disadvantages when compared to multirotor aircraft. Fixed-wing aircraft are more flexible in the event of a technical fault and possess natural glide capabilities when under no power. Additionally, fixed-wing aircraft are also capable of carrying greater loads over greater distances under less power when compared to multirotor applications. Areas in which the fixed-wing aircraft does not perform as well as multirotor aircraft include applications where missions with precise objectives are required, as the fixed-wing's need for constant forward motion to produce lift, limits their manoeuvrability [3].

#### 2.1.1.3 QuadPlane Aircraft

QuadPlane aircraft form part of the Vertical Takeoff and Landing (VTOL) class of aircraft. QuadPlane aircraft combine the advantages features of both multirotor and fixed-wing aircraft, where the frame design of the aircraft follows that of a fixed-wing aircraft while possessing four to eight horizontal rotors to provide the aircraft with the functionality of a multirotor. The benefits of QuadPlane aircraft include improved manoeuvrability during precision missions, smaller launch area required and larger range of travel. The disadvantages of QuadPlane aircraft include the need for two integrated control systems to control the multirotor and fixed-wing functionalities respectively. Additionally, the increase in aircraft functionality introduces more onboard weight, complexity and greater possible number modes of failure [4].

### 2.1.2 Airborne Delivery

The concept of airborne delivery of goods and services, specifically the airdrop, was first developed during World War II for the purpose of resupplying troops with limited access to additional support. Many of these techniques are still used today [35]. Various types of airdrops exist, these include:

- Free-Drop. Developed for the delivery of non-delicate loads, such as clothing, Free-Drops are the most uncommon form of airdrop. Figure 2.2 depicts an image of a DC3 Cargo Plane executing a Free-Drop Airdrop, delivering tons of needed food to refugees in a region of South Sudan, inaccessible by trucks [38].
- High-Velocity. Designed for higher altitude deployment, where, at the time, aircraft were less vulnerable to ground-based defence systems. High-velocity

airdrops made use of small stabilizing parachutes attached to the loads to ensure the load remained in an upright position during descent, with energy-dissipating material secured to the base of the load to reduce impact force. These loads would descend at rates between 21 m/s to 27 m/s.

- Low-Velocity. Developed for the delivery of delicate loads. Cargo parachutes were attached to the loads to allow for substantial descent deceleration to no more than 8.5 m/s.
- Low-Altitude Parachute Extraction (LAPE). LAPE airdrops consisted of an airdrop from an aircraft flying between two to three metres above the ground, where the load was rigger with a specially designed airdrop platform and one to three LAPE parachutes to assist with deceleration. Once released from the aircraft, the load then slides across the drop-zone, where the LAPE parachutes ensured the load would not tumble. LAPE airdrops require relatively flat and smooth drop-zones, requiring specialized preparations to be made prior to execution.

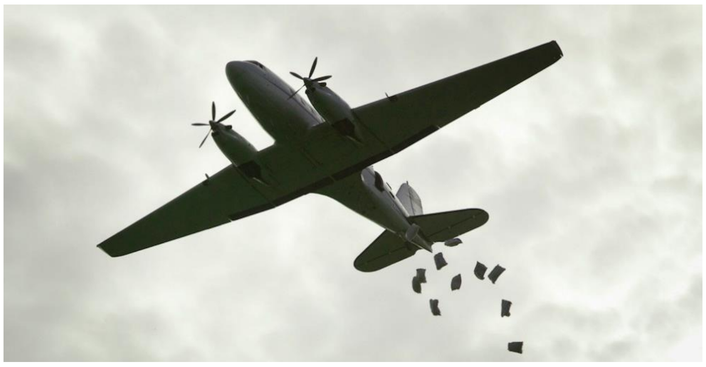


*Figure 2.2: Free-Drop Airdrop from a DC3 Cargo Plane in South Sudan [38]*

In addition to the types of airdrop, there are also various methods of airdrop. These include:

- Extraction. A method used for low-velocity and LAPE platform airdrops, where with the use of an extraction parachute or LAPE parachute, the load would be pulled from the aircraft's cargo compartment.
- Gravity. Prior to the drop, with the cargo restrained in the aircraft by a nylon webbing release gate, the aircraft is flown in a nose-up attitude. With the release gate being cut, the cargo would roll out the aircraft.

Although effective, the limiting factor of airdrops was the need for specialised personnel for rigging and piloting the aircraft to execute them.

## 2.2 Related Research

This research did not represent the first applications of integrated vision and autonomous aircraft. As such, a study into the available literature on existing research within this field was undertaken to provide insight to the available hardware and software for such applications and to identify possible limitations the systems present.

### 2.2.1 Academic Studies

Hinas et al. [27] developed an autonomous multirotor, specifically a quadrotor, for application in vision-based identification and inspection of a red circle. The aircraft flew a survey flightpath over a given area and upon detection of a red circle, would descend to a predefined hover height at which the vision system would inspect the identified red circle. Results from the physical implementation of the autonomous system showed that the system was able to achieve high levels of success up to an altitude of 40 m. This system contributed to high altitude applications of similar research which were previously run at altitudes of 3 m to 5 m. Additionally, from visual inspection of the results of the vision system's ability to determine the x and y coordinates of the red circle when in view, an average accuracy within 2 m of the target's true location was achieved from an altitude of 10 m. The system architecture utilised in the development of this autonomous detection and inspection system highlighted the possible efficiency of low-cost implementation of readily available technologies. These technologies included a Pixhawk FCU for autonomous flight control, a Raspberry Pi 2 for onboard vision processing and a four-node Robotic Operating System software architecture was developed for computational control.

Moving from multirotors to fixed-wing applications of autonomous flight control and vision, Prabowo et al. [36] developed a hardware in the loop (HITL) simulation for the development and testing of target identification influenced servoing of a fixed-wing aircraft. The desired application of such a system was based in the identification and tracking of specific targets, where monitoring of the various avionics and camera parameters were used as an indication of the performance of the system. The system architecture and testing procedure consisted of an integrated hardware-controlled flight simulator, where a simulated fixed-wing was flown on a given course and presented with a target. Upon identification of the desired target, the simulated aircraft would begin to follow the target, both physically in terms of avionics and via its onboard vision system. The subsequent output of the simulation was the monitored motion of the various control servos and the vision system pan and tilt angles. The implemented hardware included a Pixhawk PX4 flight-stack for flight control and a Cubieboard2 for image processing. The results from the HITL simulations showed minimal errors when compared to simulated results and the system was deemed ready for application in a physical aircraft system.

Hochstenbach et al. [28] developed a VTOL aircraft for autonomous parcel delivery which was capable of transporting parcels up to 1 kg in weight. This available onboard cargo weight was optimised to determine the greatest ratio between weight and transportation distance. The VTOL system developed, the VertiKUL, achieved a functional range of 26 km over a sustained flight period of 29 minutes. The system was never fully automated in terms of parcel delivery and parcel delivery was achieved through physical removal of the parcel from the aircraft. To allow for flight control and stability, a Pixhawk FCU was utilised in a mid-level control state where user defined inputs were still used for flight control of the aircraft.

The UAV Challenge Outback Rescue held in Kingaroy, Australia, held between 2007 and 2014, was a search and rescue competition where autonomous aircraft were used to locate a mannequin in the Outback and deliver a bottle of water within 100 m of the mannequin's location. CanberraUAV were the first to successfully complete the challenge in 2014 [46]. Boura et al. [10] represented one of the many teams which attempted the UAV challenge in 2010. This entry into the UAV challenge was not the primary aim of the research conducted by Boura et al. and the initial system was developed for application in a Lockheed Martin Desert Hawk III for the autonomous delivery drop of roughly 1 kg of medical supplies. The onboard hardware utilised included a Piccolo II autopilot for autonomous control and a Gumstix computer-on-module for onboard computation of the most appropriate payload release timing. Determination of the desired drop-zone's location was not done with the use of an automated vision system and the onboard vision system, comprising of a TASE gimbal and Song block camera, relayed the video feed from the aircraft back to the ground station. This lack of vision autonomy was compensated for by the vision system's ability to communicate the GPS location of the point on the ground at which the camera was focussed. From the UAV Challenge, Boura et al. were able to successfully detect the mannequin but upon payload release, a malfunction in the autopilot's waypoint allocation system prevented the payload from being released in the correct location. The resulting displacement of the payload relative to the mannequin was approximately 450 m, even though testing of the system had result in an optimum displacement of approximately 3.5 m.

### 2.2.2 Commercial Systems

This section discussed the commercial implementation of systems which utilised autonomous aircraft as a means of airborne delivery.

#### 2.2.2.1 Zipline

Zipline, an American-based start-up company, was used for delivering units of blood, vaccines and platelets in African countries, Rwanda and Tanzania, where access to necessary medical supplies were limited. Zipline uses custom-built autonomous fixed-wing aircraft to deliver these medical payloads. Once loaded with supplies, the aircraft were launched and would fly a predefined flight path to the requested drop-zone. Upon reaching the designated drop-zone, the aircraft initiates a controlled decent, until the required drop altitude has been acquired. The payload descends to the desired drop-zone with the assistance of a small wax parachute, where the drop-zone of the given payload was defined to be accurate within a 5 m radius. The aircraft then flies back and lands at the distribution centre, where it would then be prepared for the next flight [50]. The launch of a Zipline aircraft can be seen in Figure 2.3 [39].

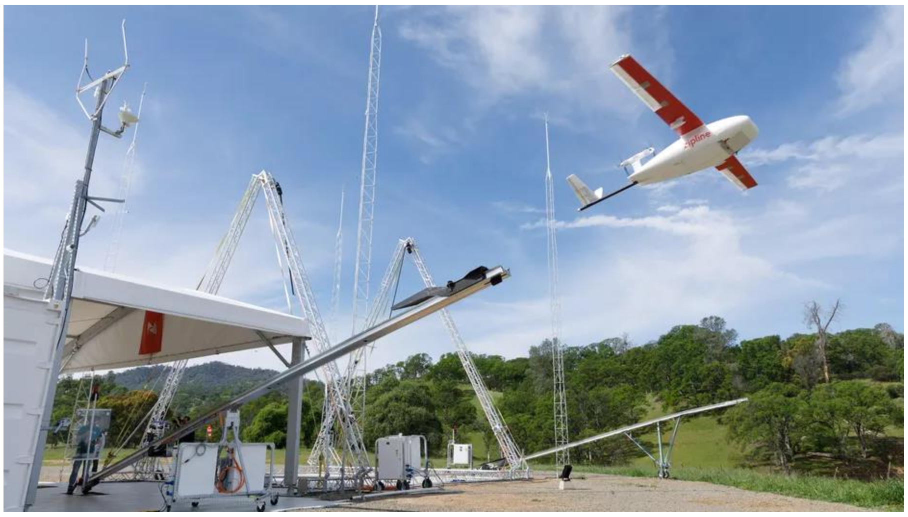


*Figure 2.3: Zipline aircraft launching [39]*

From the available literature, it was deduced that Zipline did not utilise any onboard intelligence that can identify and confirm the drop-zone, other than the initially predefined GPS drop-zone location. This lack of visual confirmation introduced the possibility that Zipline's payloads could be compromised upon delivery in terms of correct recipient collection of the payload.

### 2.2.2.2 DHL Parcelcopter

Much like Zipline, the DHL Parcelcopter was an autonomous aircraft system developed for the transportation of medical supplies in Tanzania. The DHL Parcelcopter project started in 2013 with the manual flight control of a multirotor to deliver medical supplies across the Rhine to the Deutsche Post DHL Group headquarters in Bonn. From this initial concept, the DHL Parcelcopter evolved into an autonomous VTOL aircraft, used for the delivery of medical supplies to inaccessible areas in Tanzania in the 3rd quarter of 2018. Unlike Zipline, whose mission flightpaths were fully automated, DHL's Parcelcopter made use of user defined mission waypoints, including takeoff and landing, in the QGroundControl software. Additionally, the Parcelcopter did not make use of an airdrop to deliver their packages and the VTOL would land and allow for package removal. The DHL Parcelcopter can be seen in Figure 2.4 [16].

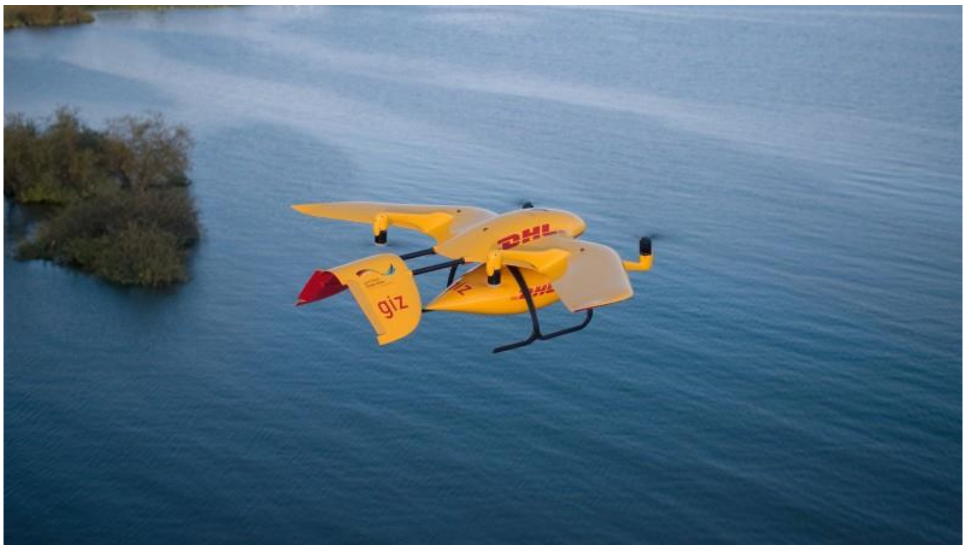


*Figure 2.4: DHL Parcelcopter [16]*

Like Zipline, DHL Parcelcopter did not make use of any onboard vision systems to assist with mission execution and although autonomous landing was a feature, no indication of object avoidance was seen. The significance of this lack of onboard vision or obstacle avoidance was that the missions were user defined and incorrect placement of the landing waypoint could have resulted in the aircraft colliding with physical obstacles. The need for such an avoidance system was only present in DHL's aircraft, as Zipline's aircraft only launched and landed at Zipline ground stations and the implementation of airborne delivery removed the need for additional onboard hardware.

## 2.3 Chapter Conclusion

From the research conducted it was noted that a trend in the use of the Pixhawk FCU flight controller for the introduction of autonomy into various airframes can be seen. Additionally, from the research conducted into the various airframes it was noted that VTOL aircraft seemed to be the theoretical favourite, but upon review of the commercial systems, Zipline's fixed-wing aircraft achieved the desired result equally, if not better than the DHL Parcelcopter VTOL. From the commercially available systems, a lack of onboard vision was noted, providing the opportunity for implementation of such a system to improve the desired performance. Looking at the required infrastructure for the necessary airdrops, the least complex method for implementation on a low-cost system would be the free-drop. As such, the focus of the research has been steered towards integrating and improving upon the research conducted by Hinas et al. and Boura et al. into a single fully autonomous system capable of vision-based autonomous aircraft payload delivery. Chapter 3 follows this literature study with the development process of the system architecture utilised in this research.

# Chapter 3
# System Architecture

This chapter discussed the platform used to develop the vision-based autonomous aircraft payload delivery system. Aspects such as the test bench aircraft, hardware architecture, software architecture and aircraft alternations were discussed.

## 3.1 The Test Bench Aircraft

This research was developed for the purpose of developing a modular design that could easily be transferred between different airframes. Considering this, the test bench model, upon which results were gathered, was selected.

The test bench model aircraft utilised in the development of this research was the Skywalker 2013 (Carbon Fibre Tail Version), as seen in Figure 3.1. The Skywalker frame presented the necessary flexibility and expendability due to its Erythropoietin (EPO) foam composition which allowed for alterations to be made to the physical structure of the aircraft with minimal detriment to the aircraft's structural rigidity. Physical specifications of the aircraft can be seen in Table 3.1.

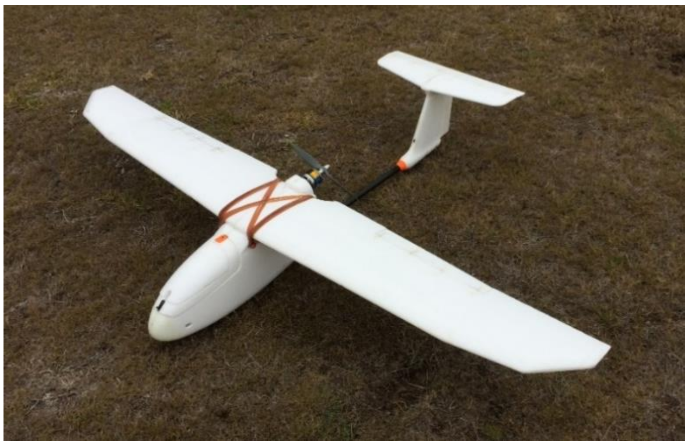


*Figure 3.1: Test bench aircraft - Skywalker 2013*

*Table 3.1: Test bench aircraft physical parameters*

| *Parameter* | *Value* | *Unit* |
|---|---|---|
| Wingspan | 1.88 | m |
| Length | 1.225 | m |
| Height | 0.22 | m |
| Weight | 1.4 | kg |

The aircraft was altered from its given state to accommodate the necessary hardware. These alterations were discussed in Section 3.4.

## 3.2 Hardware Architecture

Several subsystems were required to be integrated into a single operational platform, hardware for each subsystem was selected based on functionality and integrability with the other remaining subsystems. This section focused on the hardware selected and discussed the rationale pertaining to the reasoning for each hardware component and its selection.

### 3.2.1 Aircraft Flight Control

Prior to any additional hardware being introduced into the aircraft to achieve autonomous control, basic flight control hardware was installed to achieve sustained flight and control of the aircraft. This section focused on these devices, as their performance and functionality formed the foundation upon which the remainder of this research was based.

#### 3.2.1.1 Remote-Control

Basic remote-control hardware was installed to develop a functional test platform. The aircraft's remote-control hardware can be seen in Table 3.2.

*Table 3.2: Remote-control hardware onboard the aircraft*

| *Component* | *Model* | *Quantity* |
|---|---|---|
| Control Actuators | 15g Servo | 4 |
| Motor | SunnySky X2814-7 KV: 1100 | 1 |
| Electronic Speed Controller (ESC) | Hobbywing FLYFUN 40A | 1 |
| Receiver | FrSky X8R | 1 |
| Transmitter | FrSkyTaranis QX7 | 1 |
| Batteries | X-Power LiPo 3300mAh 3S | 2 |

### 3.2.1.2 Telemetry Sensors

In addition to the control hardware, telemetry sensors were installed onboard the aircraft to provide sensory information during flight. These sensors did not form part of the final integrated system's sensory package but were kept onboard to provide comparative information for testing. These telemetry sensors were also included due to their compatibility with the remote-control hardware onboard the aircraft. The remote-control telemetry sensors can be seen in Table 3.3.

*Table 3.3: Remote-control sensory hardware onboard the aircraft*

| <u>*Sensor*</u> | <u>*Model*</u> | <u>*Quantity*</u> |
|---|---|---|
| Airspeed sensor | FrSky ASS-70 | 1 |
| Current and Voltage Sensor | FrSky FCS-150A | 1 |
| GPS Sensor | FrSky GPS V2 | 1 |
| Variometer | FrSky Vari-N | 1 |

### 3.2.1.3 Configuration

With the aircraft's remote-control and sensory hardware defined, their configuration in the aircraft could be defined. Within the remote-control hardware, two subsystems existed, namely: the onboard remote-control hardware and the ground-based remote-control hardware. The ground-based remote-control hardware was represented by the transmitter and the onboard remote-control hardware was represented by the remaining remote-control hardware, with its central configuration point being the X8R receiver. The configuration of the onboard remote control and sensory hardware can be seen in Figure 3.2.

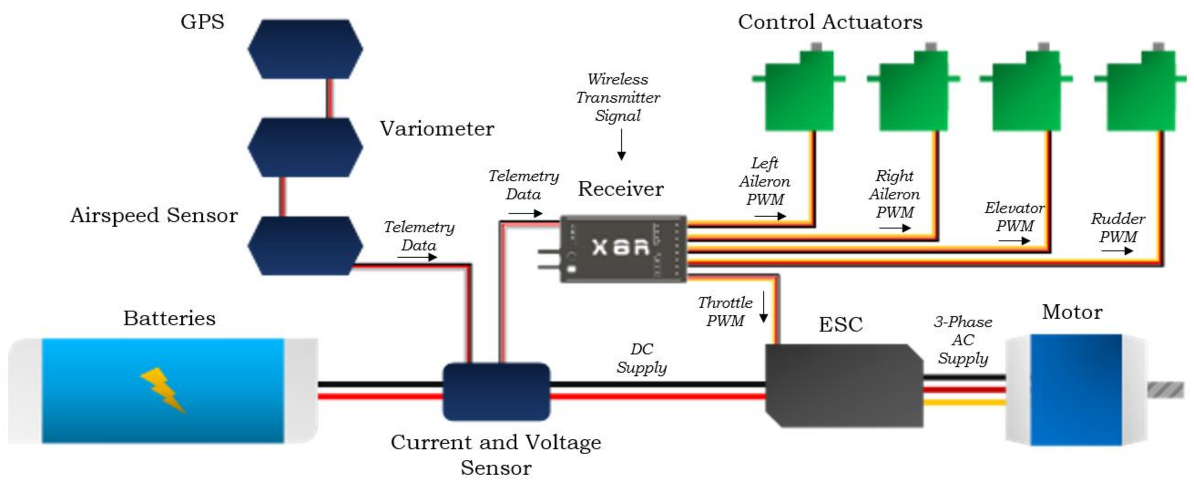


*Figure 3.2: Onboard remote control and sensory hardware configuration*

The configuration of the receiver for the remote-control hardware can be seen in Figure 3.3.

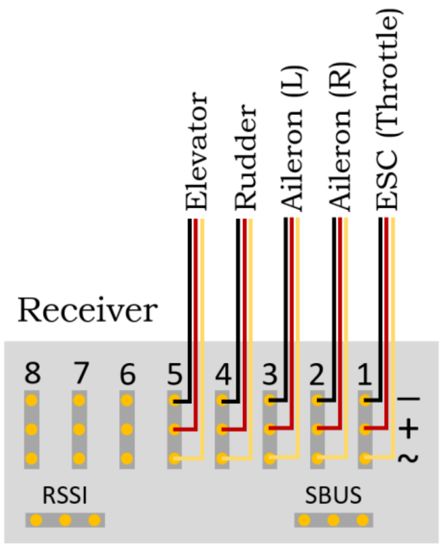


*Figure 3.3: Initial receiver configuration of the control actuators*

### 3.2.2 Autonomous Flight Hardware

With the basic flight control hardware defined, the hardware introduced to convert the system into a fully autonomous aircraft could be defined. During the installation of the autonomous flight control hardware, none of the aircraft's remote-control hardware and sensory hardware was removed.

#### 3.2.2.1 The Autopilot

The Pixhawk 1 Flight Controller (The Pixhawk), an open hardware, industry standard autopilot system, was selected as the desired autopilot. The Pixhawk was a commonly used and readily available system, whose integration into the current flight control configuration did not require third party hardware. The Pixhawk can be seen in Figure 3.4 and the list of the Pixhawk's specifications can be seen in Appendix C1 [37].

*Figure 3.4: The Pixhawk 1 [37]*

The Pixhawk, with the assistance of a ground control station (GCS), was capable of several manual and autonomous flight control modes. These modes ranged from *MANUAL* flight mode, where the Pixhawk acted like a remote-control receiver in terms of full actuator control. Thereafter, *GUIDED* flight mode, a semi-autonomous flight mode where the aircraft would follow user defined waypoints either through a mission statement or via a click-and-fly approach in the GCS. Finally, *AUTO* flight mode, where the aircraft could receive and execute an entire mission statement from the GCS. These flight modes include variations for both GPS dependent and non-GPS dependent flight paths. The Pixhawk also came with a variety of peripheral devices for sensory feedback and extended functionality, several of which were used within this research and were discussed later in this chapter.

The Pixhawk provided several autonomous flight control functions, for a wide variety of airframes, however it was not an autonomous flight controller as it possessed no onboard intelligence, in terms of situational decision making other than that of predefined failsafe procedures. This lack of additional intelligence meant that the Pixhawk would only execute commands presented to it by the GCS and the transmitter. As such, the introduction of onboard intelligence was necessary to transform the aircraft into a fully autonomous system. Therefore, the next phase of the autonomous flight hardware was introduced, the companion computer.

#### 3.2.2.2 The Companion Computer

Companion computers were used to interface and communicate with flight controllers through the Micro Air Vehicle Link (MAVLink) protocol, allowing for onboard intelligence to be integrated into the aircraft. The level of onboard intelligence introduced into the flight controller was only limited by the capabilities of the selected companion computer. Tasks, such as vision processing and mapping, were a few of the capabilities of these companion devices. These devices had limitations, with issues such as additional power consumption, weight and available onboard space all became relevant factors when developing a sustainable and feasible autonomous system.

Several possible Pixhawk compatible companion computer systems existed. The selection of the most applicable system was based on application, computational requirements and functionality. The available companion computers that offered the necessary functionality include:

- Raspberry Pi
- ODroid
- NVidia Jetson TX1

Considering the available companion computers, a comparison based on cost, processing capability and functionality was conducted to determine the applicable system for selection. Functionality implied the integrability of the companion computer with additional onboard hardware. The results of these comparisons can be seen Table 3.4 and Figure 3.5.

*Table 3.4: Companion computer comparison merit table*

| | *Cost* | *Processing Capabilities* | *Functionality* | *Total* |
|---|---|---|---|---|
| *Weighting* | *0.3* | *0.5* | *0.2* | |
| Raspberry Pi | 8 | 5 | 7 | 6.67 |
| ODroid | 6 | 9 | 7 | 7.33 |
| NVidia Jetson TX1 | 1 | 7 | 5 | 4.33 |

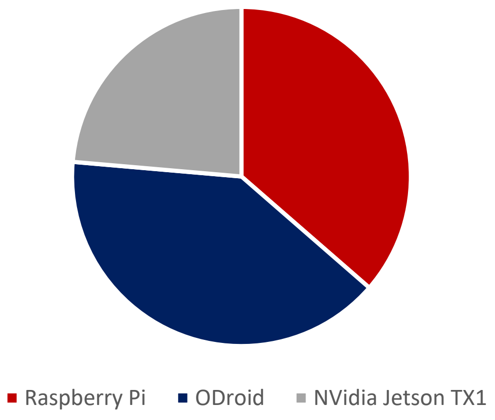


*Figure 3.5: Companion computer weighted comparison*

Noting the results gathered from Table 3.4 and Figure 3.5, it was found that the largest weighting was placed on processing capabilities. The ODroid-XU4, as seen in Figure 3.6, was selected as the desired companion computer for this research. The list of the ODroid-XU4's specifications can be seen in Appendix D1 and the ODroid-XU4's schematic can be seen in Appendix D2 [25].

Figure 3.6: The ODroid-XU4 [25]

To establish a communication link between the ODroid and the Pixhawk, a six-way Future Technology Devices International (FTDI) Universal Serial Bus (USB) to Serial converter cable was used. This USB to serial cable, more commonly referred to as an FTDI cable, was used to establish the link between the ODroid's USB port and the TELEM2 port of the Pixhawk. The FTDI cable operated by converting the USB communication from the ODroid to Transistor-Transistor Logic (TTL) level Serial communication for the Pixhawk. In addition to the FTDI cable, further functionality was added to the ODroid with the introduction of an embedded Multi-Media Controller (eMMC). The eMMC acted as the ODroid's high speed storage device.

#### 3.2.2.3 Peripheral Devices

The Pixhawk provided a wide variety of peripheral devices for both data capture and data transmission. Several of these devices were included in the configuration of the autonomous flight controller, but not all of the peripheral devices utilized in this research were discussed in this section as they were not applicable to the autonomous flight of the aircraft. These additional peripheral devices were discussed in their respective sections. With the available peripheral devices for the Pixhawk considered, the devices utilized in the development of the autonomous flight control of this research can be seen in Table 3.5.

Table 3.5: Pixhawk peripheral devices

| *Peripheral Device* | *Quantity* |
|---|---|
| GPS and Compass | 1 |
| Digital Airspeed Sensor | 1 |
| 933MHz Wireless Telemetry Modem | 1 |
| Battery Monitor | 1 |
| I$^2$C Splitter | 1 |

#### 3.2.2.4 Configuration

The configuration of the autonomous flight control hardware was integrated with the remote-control flight control configuration as the remote-control system was kept in place as a safety. Various other sub-configurations in the autonomous flight hardware architecture were also present within the autonomous flight control configuration. These included the Pixhawk's peripheral devices' configuration, the FTDI cable wiring configuration for the Pixhawk's TELEM2 port and the receiver-Pixhawk configuration. These configurations can be seen in Figure 3.7, Figure 3.8 and Figure 3.9, respectively. The integrated configuration of the autonomous flight control hardware can be seen in Figure 3.11. The Pixhawk port pinouts can be seen in Appendix C2 and the Pixhawk's schematic can be seen in Appendix C3.

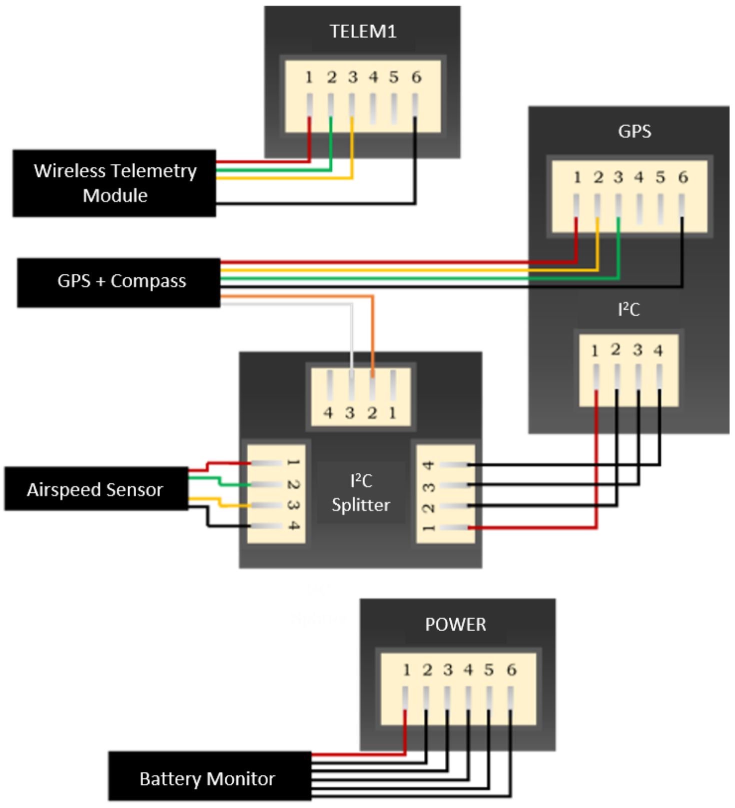
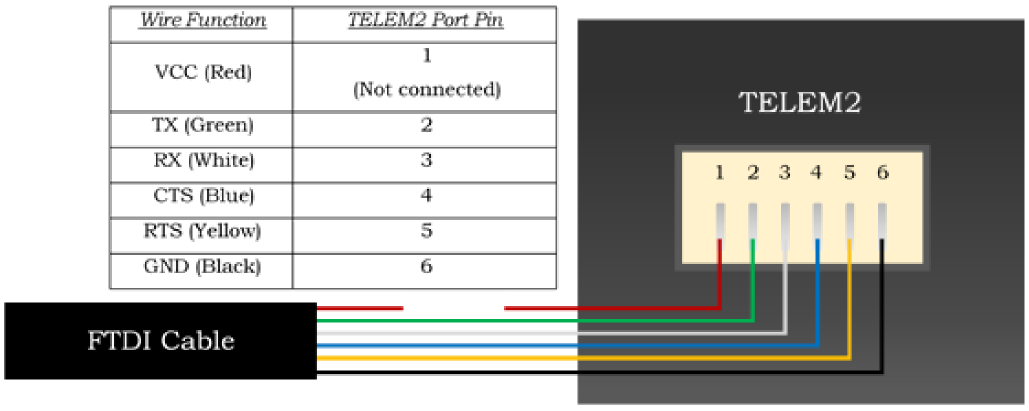


*Figure 3.7: Pixhawk's peripheral devices' configuration*

The FTDI cable's serial port connector did not match the Pixhawk's TELEM2 port and as such, the correct six-pin male JST ZH connector was used to replace the FTDI cable's serial port connector. The corresponding wiring configuration of the correct serial port connector to the Pixhawk's TELEM2 port can be seen in Figure 3.8. A wiring convention table was also included, as the FTDI cable and the Pixhawk did not share the same wiring colour conventions.



| Wire Function | TELEM2 Port Pin |
|---|---|
| VCC (Red) | 1 (Not connected) |
| TX (Green) | 2 |
| RX (White) | 3 |
| CTS (Blue) | 4 |
| RTS (Yellow) | 5 |
| GND (Black) | 6 |

*Figure 3.8: FTDI cable wiring configuration for Pixhawk TELEM2 port*

To allow for manual control of the aircraft to be taken at any time, the remote-control receiver was connected to the Pixhawk. This connection was established between the receiver's Serial BUS (SBUS) port and the Pixhawk's RC IN port using a three-wire servo lead with female JR connectors at both ends.

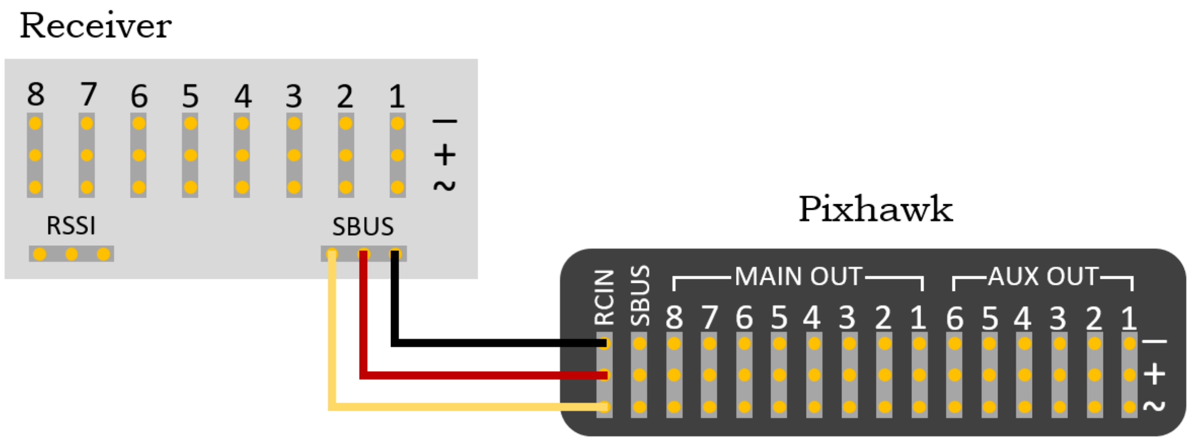


*Figure 3.9: Receiver – Pixhawk configuration*

Due to the integration of the Pixhawk into the autonomous flight control system, the control actuators were required to be rewired as per the Pixhawk's control surface configuration. The Pixhawk control actuators configuration can be seen in Figure 3.10.

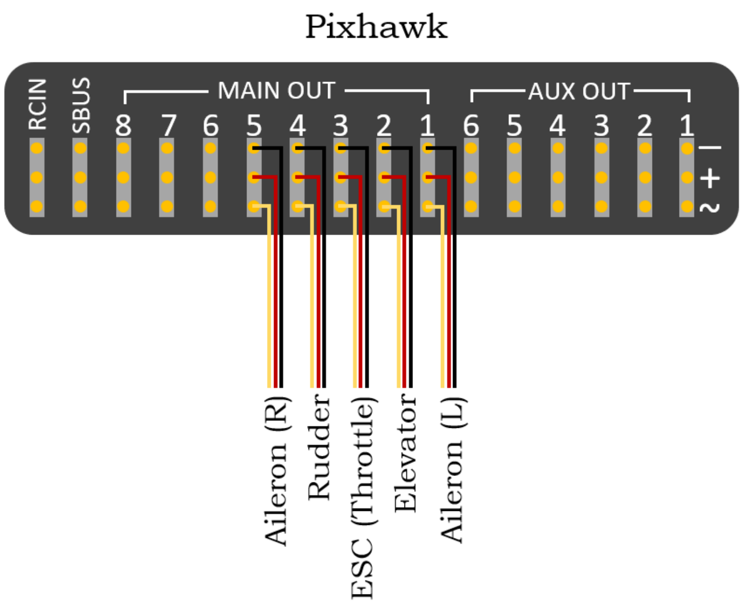


*Figure 3.10: Pixhawk control actuator configuration for a fixed-wing aircraft*

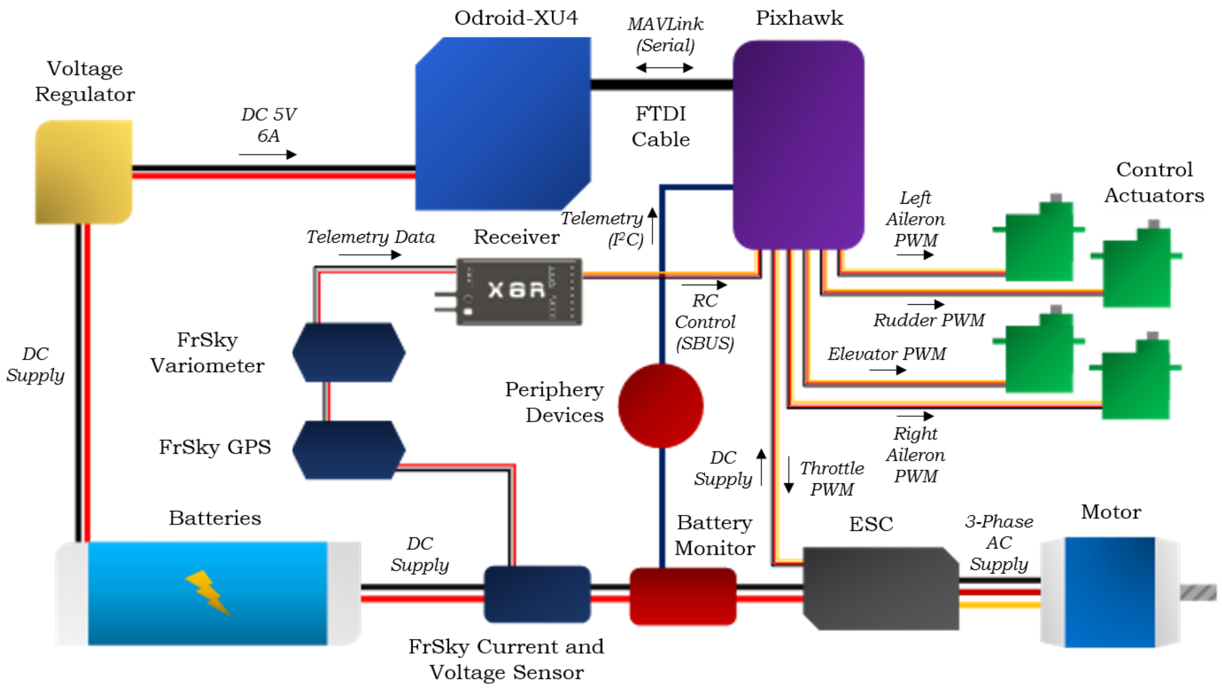


*Figure 3.11: Integrated autonomous flight control hardware configuration*

### 3.2.3 Vision System Hardware

The following phase of an autonomous payload delivery mission involved the identification of the desired drop-zone location, therefore the required hardware to establish such a system was discussed in this section.

#### 3.2.3.1 The Processor

The ODroid-XU4 was selected as the main processing unit. The defining reason for selecting the ODroid was due to its vision processing capabilities. As such, the ODroid formed part of the vision system hardware.

#### 3.2.3.2 The Camera

The key element of the vision system was the selection of an appropriate camera. The selection was based on the application of the camera and the system requirements of the current hardware and future software in terms of integrability. In addition to these criteria, the size and weight of the camera were also considered due to the limited space onboard the aircraft. A comparison of available camera systems was conducted. Due to the wide variety of possible solutions, the list was refined for the final selection. The initial comparison of camera systems can be seen in Table 3.6 and Figure 3.12.

*Table 3.6: Camera type comparison merit table*

| | *Cost* | *Size & Shape* | *Application Suitability* | *Integrability* | *Total* |
|---|---|---|---|---|---|
| *Weighting* | *0.2* | *0.1* | *0.4* | *0.3* | |
| Webcam | 5 | 5.5 | 5 | 8 | 5.875 |
| GoPro | 2 | 8 | 8 | 5 | 5.75 |
| CMOS Camera | 8 | 8 | 7 | 8 | 7.75 |

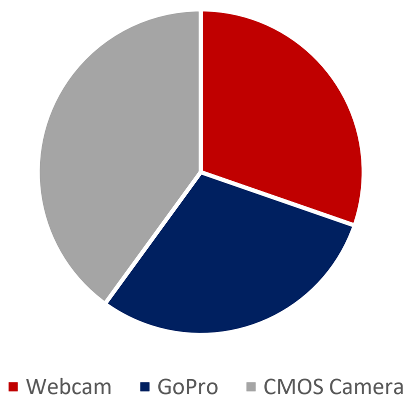


*Figure 3.12: Camera type weighted comparison*

From the comparison in Table 3.6 and Figure 3.12, a complementary metal-oxide semiconductor (CMOS) camera was selected as the desired type of camera system. With the type of camera now selected, the search field could now be narrowed to optimise the selection of the most appropriate CMOS camera. Refining the search noted the application specific criteria. A criterion for the camera system was to possess a USB connector to interface with the ODroid's USB ports. To narrow the search field further, the selection of possible camera systems was focussed at a single CMOS camera manufacturer, ELP, who specialised in USB camera modules. The final camera system comparison can be seen in Table 3.7 and Figure 3.13.

*Table 3.7: ELP CMOS camera comparison merit table*

| | *Resolution* | *FOV* | *Functional Range* | *Functionality* | *Total* |
|---|---|---|---|---|---|
| *Weighting* | *0.4* | *0.1* | *0.4* | *0.1* | |
| ELP-USB500W04AF-A60 | 7 | 8 | 9 | 8 | 8 |
| ELP-USB500W02M-L21 | 7 | 4 | 9 | 8 | 7 |
| ELP-USB8MP02G-L75 | 9 | 6 | 7 | 7 | 7.25 |

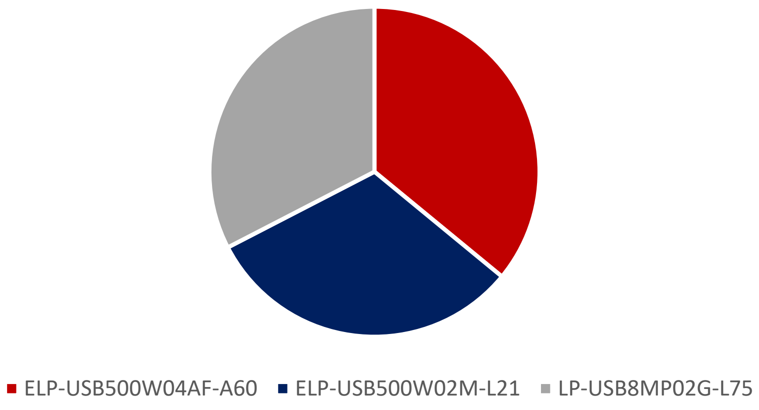


*Figure 3.13: ELP CMOS camera weighted comparison*

From Table 3.7 and Figure 3.13, the highest weighting was given to resolution and functional range of the camera system. Based on the final weighted comparison, the ELP-USB500W04AF-A60 was selected as the desired camera system for the onboard vision system, as seen in Figure 3.14 [17]. The full list of the ELP-USB500W04AF-A60 camera's specifications can be seen in Appendix E1 and the ELP-USB500W04AF-A60 camera's schematic can be seen in Appendix E2. Some of the ELP-USB500W04AF-A60 camera's specifications can be seen in Table 3.8 [17].

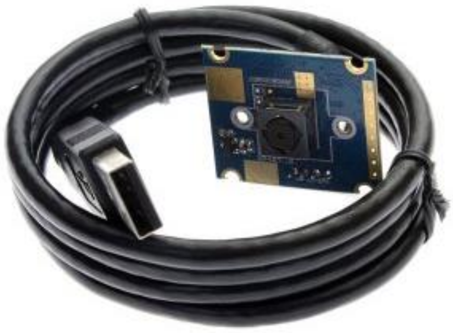


*Figure 3.14: ELP-USB500W04AF-A60 CMOS camera [17]*

*Table 3.8: ELP-USB500W04AF-A60 camera key specifications*

| *Specification* | *Value* | *Unit of Measurement* |
|---|---|---|
| Resolution (Max) | 2592 (H) X 1944 (V) | pixels |
| Functional Range | 0.05 to 100 | m |
| Focus | Automatic | - |
| Operational Temperature | Operation: -20 to 70<br>Stable Image: 0 to 60 | ˚C |

### 3.2.3.3 The Gimbal

The overall performance of the vision system was predominantly based on the selection of an appropriate camera system, however the vision system would have been insufficient during flight without image stabilisation. As such, a camera gimbal was incorporated into the vision system to account for the CMOS camera's lack of image stabilisation capabilities. The TAROT GoPro 3D V metal 3-axis gimbal was selected as the desired gimbal for the vision system.

Although designed for the GoPro HERO 5 BLACK camera, the TAROT gimbal was viable for use with the CMOS camera. The TAROT gimbal was mass-balanced according to the size and weight specifications of the GoPro HERO 5 BLACK. As such, the alterations required to allow the CMOS camera to function in the gimbal correctly was to mass-balance the CMOS camera according to the design of the GoPro. The CMOS camera mass-balancing adapter was discussed in Section 6.3.

As image stabilisation was necessary with respect to the motion of the aircraft, the gimbal was connected to the Pixhawk's AUX ports. Two of the gimbal's axes were controllable, the TILT and the PAN axes. The roll axis was compensated for by the gimbal's built-in controller. The wiring configuration of the gimbal and the Pixhawk can be seen in Figure 3.15. The TAROT GoPro 3D V metal 3-axis gimbal specifications can be seen in Appendix F.

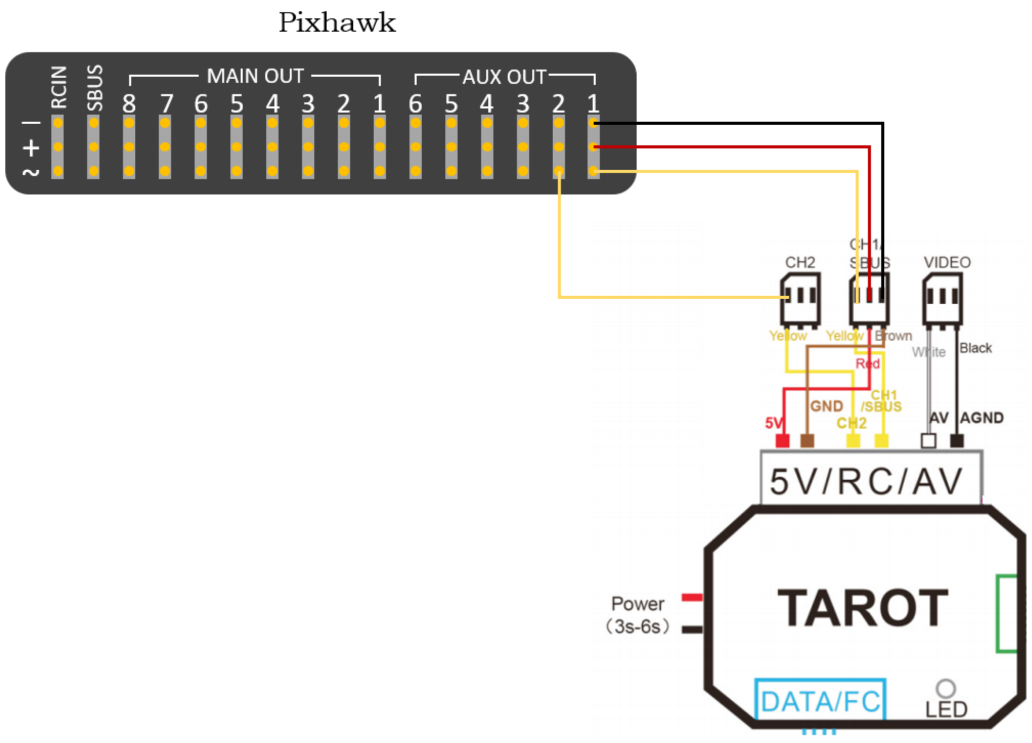


Figure 3.15: Gimbal - Pixhawk configuration

### 3.2.3.4 Configuration

With the selection of the vision system components complete, the integration of the vision system to aircraft could take place. The vision system's integrated configuration with the integrated autonomous flight control hardware configuration can be seen in Figure 3.16.

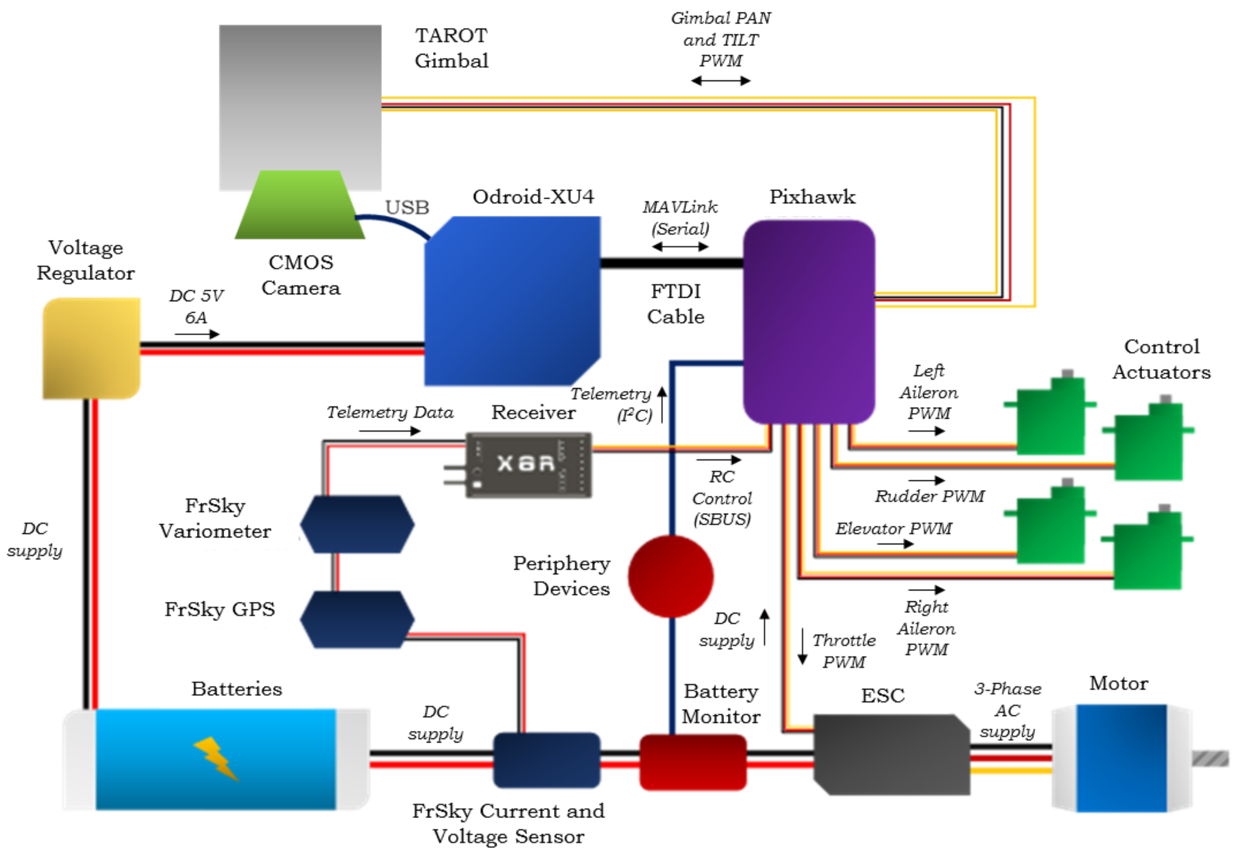


*Figure 3.16: Vision system integrated with autonomous flight control hardware configuration*

### 3.2.4 Payload Release Hardware

The hardware required for the payload release mechanism was an actuating servo release system. The payload release system was integrated with the other subsystems' hardware and the required payload release signal was triggered by the Pixhawk when instructed to do so by the ODroid.

#### 3.2.4.1 Release System

An initial concept of a single servo holding the payload was devised for this system. After testing, it was found that the servo was under continuous load when attempting to retain the payload. The solution to this payload release system problem was to make use of a predesigned release mechanism. The predesigned release mechanism was designed to remove any load from the actuating servo until payload release was necessary. A conceptual view of the actuation of the final release mechanism can be seen in Figure 3.17.

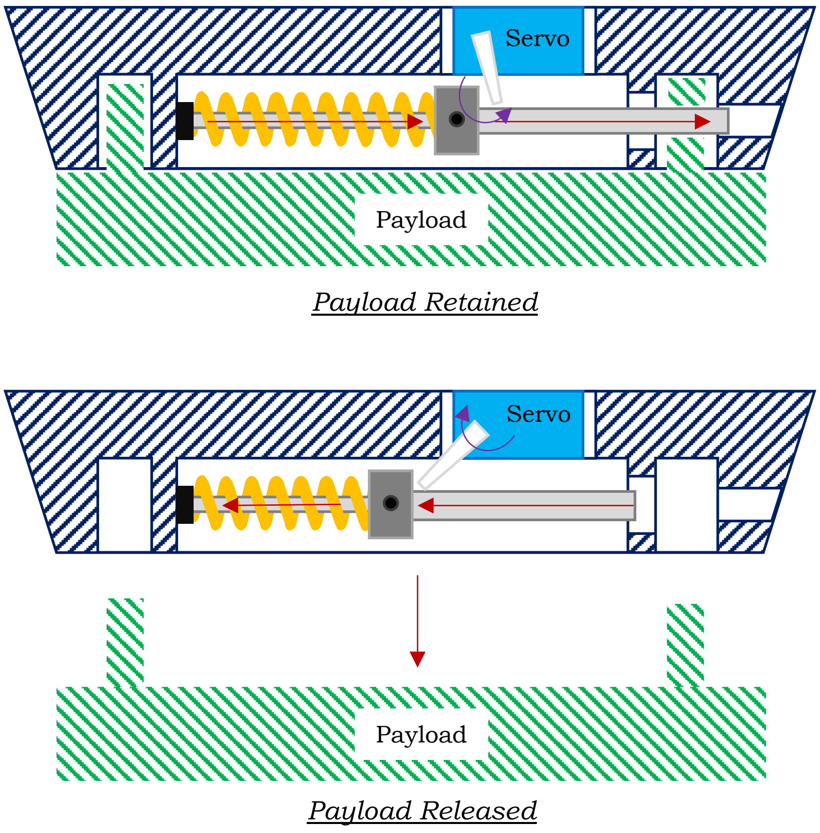


Figure 3.17: Payload release mechanism actuation diagram

Although a new payload release mechanism was selected, the required electromechanical hardware remained unchanged from the initial concept and only a single servo was required to actuate the new mechanism.

#### 3.2.4.2 Configuration

The remaining attributes of the payload release system were based on software conditions to determine when the payload should be released. As such, the final hardware attribute of the payload release system was the configuration of the payload release servo with respect to the Pixhawk, which can be seen in Figure 3.18. The payload release mechanism was then secured to the underside of the aircraft's fuselage.

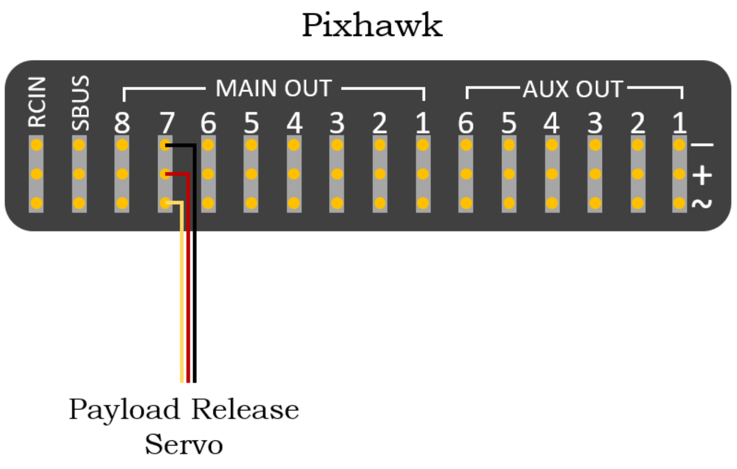


*Figure 3.18: Pixhawk configuration for the payload release servo*

### 3.2.5 Additional Hardware Considerations

The operational temperature of the hardware devices was taken into consideration due to Lapse rate. Where, Lapse rate was the rate at which atmospheric temperature decreased with respect to an increase in altitude. Lapse rate was applicable due to the increase in altitude the hardware would experience during flight. As such, noting the standard Lapse rate for altitudes up 100m, the maximum change in the ambient temperature could be approximated to 1˚C below the ground level ambient temperature. Due to the location at which this research was conducted, this variation in ambient temperature was within the operational limits of the onboard hardware. Application of this research in areas with lower ambient temperatures could prove to generate inconsistent results and component insulation could be considered to rectify this [9].

## 3.3 Software Architecture

The majority of the hardware utilised in this research provided preinstalled firmware and software, however the computational control algorithms required for autonomous flight, vision, payload release and additional onboard processing were required to be developed. These processes were required to run concurrently and communicate with one another and although the ODroid provided the necessary hardware to achieve this type of computational control, this hardware did not imply the necessary software was present to achieve these processes. As such, this section discussed the software and software architecture used to develop the aforementioned algorithms.

### 3.3.1 Operating System

When selecting the ODroid, the option to preinstall an operating system (OS) on the eMMC was selected. The ODroid came with an Android OS already preloaded on the ODroid's built-in storage, but a Linux OS was selected to be installed on the eMMC. Linux offered a more user-friendly interface and was selected based on its compatibility with the computational middleware used for the required algorithms, the Robotic Operating System (ROS). The Linux OS installed on the ODroid was Ubuntu MATE 18.04, a companion computer supported OS. A similar Linux OS was installed on a separate computer system running virtual machine software. This secondary Linux OS was used for off-board testing and simulations of the various algorithms.

### 3.3.2 Robotic Operating System

ROS was selected as the middleware for this research. ROS allowed for several nodes to be executed concurrently, where ROS nodes represented the programmes and algorithms which performed computations. These nodes were written in the C++ programming language, where application specific ROS libraries and headers were added to these nodes to allow for extended functionality. Five separate nodes were used in this research, namely: the MAVROS node, the autonomous flight node, the image capture node, the image processing node and the payload release node. These nodes were discussed in detail in their respective chapters. The ROS node architecture can be seen in Figure 3.19.

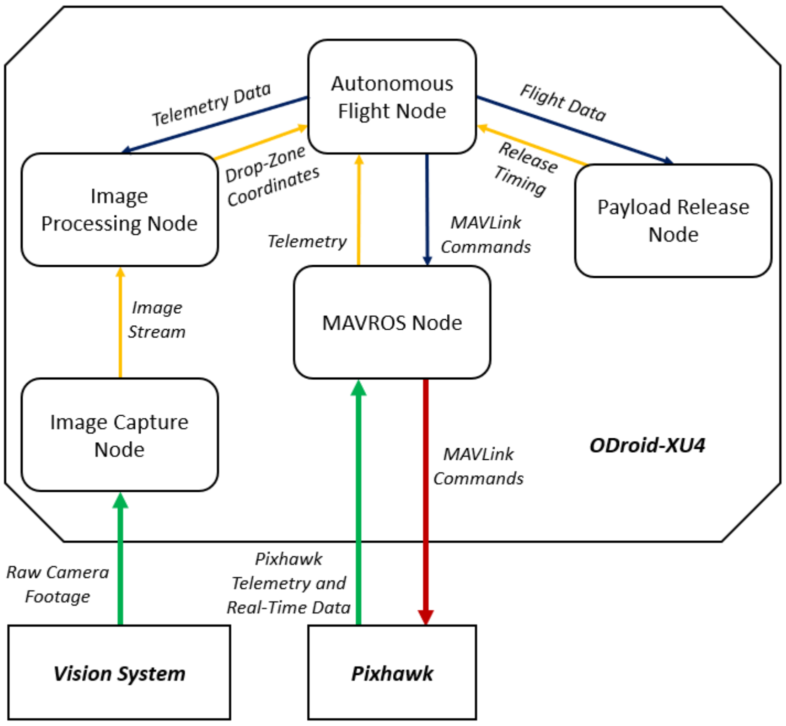


Figure 3.19: System Software Architecture

### 3.3.2.1 MAVROS Node

The MAVROS node was a pre-written header-only message marshalling library that was developed for establishing the MAVLink communication protocol with unmanned air vehicles' FCUs and in this case, the Pixhawk. No additional data was required to be added to the MAVROS node in order to achieve MAVLink communication with the Pixhawk. In order for the other nodes to communicate with the Pixhawk via the MAVROS node, each node was required to make use of MAVROS library headers. These headers were developed for vehicles enabled with the MAVLink communication protocol which was not limited to fixed-wing aircraft and as such, several of these headers were not compatible with fixed-wing aircraft.

### 3.3.2.2 MAVROS Headers and MAVLink

MAVLink represented the method of communication with the Pixhawk and the MAVROS node handled all encoding and decoding of the MAVLink messages to and from the Pixhawk. The MAVLink message structure can be seen in Figure 3.20.

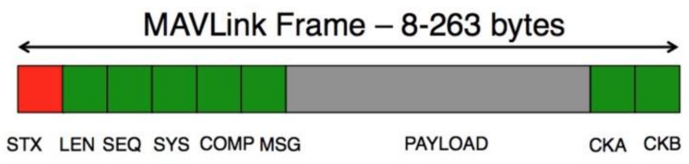


Figure 3.20: MAVLink message structure [47]

From Figure 3.20, the MSG byte represented the Message ID byte which defined which MAV_CMD messages were being transmitted. The MAV_CMD messages represented the possible commands available to be transmitted via MAVLink to a supporting FCU. The PAYLOAD byte represented the parameters within the given MAV_CMD being transmitted. The remaining bytes represented system parameters and identifiers which ensured that the communication link would only exist between the desired FCU and the MAVLink transmitting device. Ensuring the correct link was established created an end-to-end encryption between the transmitting and receiving devices. The MAVROS library worked in such a way as to request that the user define the MSG and PAYLOAD bytes by defining the corresponding variables for the available byte parameters. The remaining bytes were handled by the MAVROS node upon connection with the desired FCU. These bytes could be altered manually if necessary [47].

Considering the MAVROS header structure, it was found that the MAVROS headers required the use of the ROS node structure of a Subscriber and Publisher node. These node structures implied that, in order to transmit data from a node to the FCU via the MAVROS node, a publisher was to be written for each MAVROS header with a node wanting to be transmitted information. In the case of receiving data from the FCU, a subscriber was to be written in a node to request the desired MAVROS header data from the MAVROS node. Each MAVROS header defined which of its variables would require a publisher or subscriber to be written to execute the desired command. These MAVROS header subscriber and publisher structures were defined from the FCU's frame of reference and as such, the opposing publisher and subscriber node structure was to be written by the transmitting device, in this case the ODroid's ROS nodes. The process of transmitting and receiving data upon request via the ROS subscribers and publishers allowed for several processes to run simultaneously and was the defining feature of ROS which solidified it as the desired middleware for the required computational control [48][49].

MAVROS headers made use of the ROS node structure of Services and Clients. Where, like the subscriber and publisher structures, the MAVROS header variables requiring services and clients were defined within each MAVROS header description. Command requests and replies were executed with the use of ROS services and clients. Unlike the subscribers and publishers, the MAVROS headers only made use of the service node structure. As such, functions could be requested from the Pixhawk and no additional functionality could be added. The defining differences between subscribers and publishers, and services and clients were that subscribers and publishers were many-to-many one-way transmissions, which originated from the transmitting device. Services and clients were one-to-one, two-way communication. Several nodes could make use of subscribers and publishers, but only a single service and client node could exist, a master node.

#### 3.3.2.3 Autonomous Flight Node

The autonomous flight node was responsible for flight mode selection, aircraft arming, mission waypoint allocation and vehicle diagnostic monitoring. The MAVROS headers, although developed for various vehicles, presented limited information as to the operation and application of the various header variables, especially in the case of fixed-wing aircraft. The majority of the available data on the application of the MAVROS header was for multirotor vehicles and as such, various fixed-wing flight scenarios could not be programmed using the available headers. The full description of the autonomous flight node can be seen in Chapter 4.

#### 3.3.2.4 The Image Capture Node and the Image Processing Node

The image capture node was used to interface with the camera and for capturing the image stream from the camera. The image processing node, upon receiving the captured image stream, was responsible for image processing, making use of the Open Computer Vision Library (OpenCV) header to achieve the desired object recognition and object tracking. Upon successful identification of the drop-zone, the image processing node would communicate the drop-zone's coordinates to the autonomous flight node and payload release node. The full description of the image capture node and image processing node can be seen in Chapter 5.

#### 3.3.2.5 Payload Release Node

The payload release node was responsible for the computation of environmental parameters, such as the headwind velocity and direction. Additionally, the payload release node was responsible for the computation of the payload release trajectory and subsequently the payload release location. The full description of the payload release node can be seen in Chapter 6.

## 3.4 Aircraft Alterations

Due to the hardware required to conduct this research, physical alterations were made to the aircraft to allow for the fully integrated system to function correctly.

### 3.4.1 Nose Cone and Fuselage

To house the vision system, ODroid and telemetry devices, a new nose cone and fuselage were designed. The new nose cone and fuselage were required due to the limited internal space offered by the standard Skywalker airframe. The additional space required was to primarily accommodate the vision system, where a gimbal mount and a viewing window for the camera were incorporated into the new nose and fuselage. The new nose and fuselage were designed as a three-tier system, as seen in Figure 3.21. The first tier was designed to accommodate for the camera gimbal to move about and housed the camera viewing window, the second tier supported the camera gimbal and the ODroid, and the third tier housed the telemetry devices. Above these three-tiers was a fibreglass reinforced foam hood which served to secure the three tiers to the aircraft. The design of the new nose and hood did not improve upon the aerodynamic profile of the aircraft and was purely for additional hardware storage.

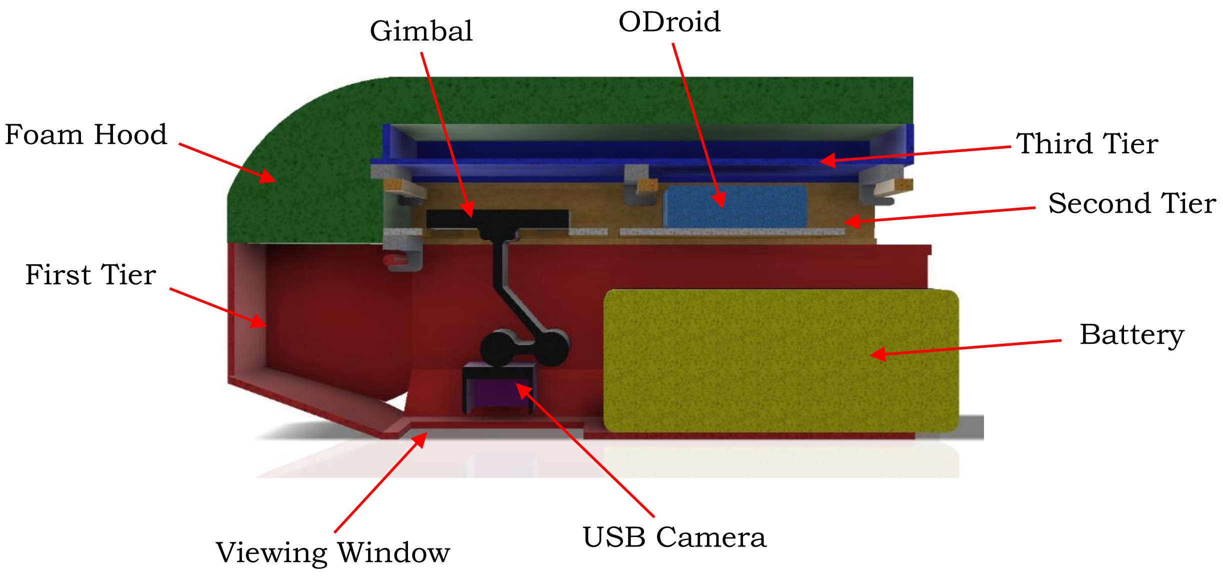


*Figure 3.21: New nose and fuselage conceptual design*

The new nose and fuselage were constructed from balsa and plywood to ensure minimal additional weight was introduced into the aircraft. Removing the original nose from the aircraft, the first tier was constructed and secured to the remaining aircraft fuselage. A comparison between the new nose first tier and the original nose can be seen in Figure 3.22.

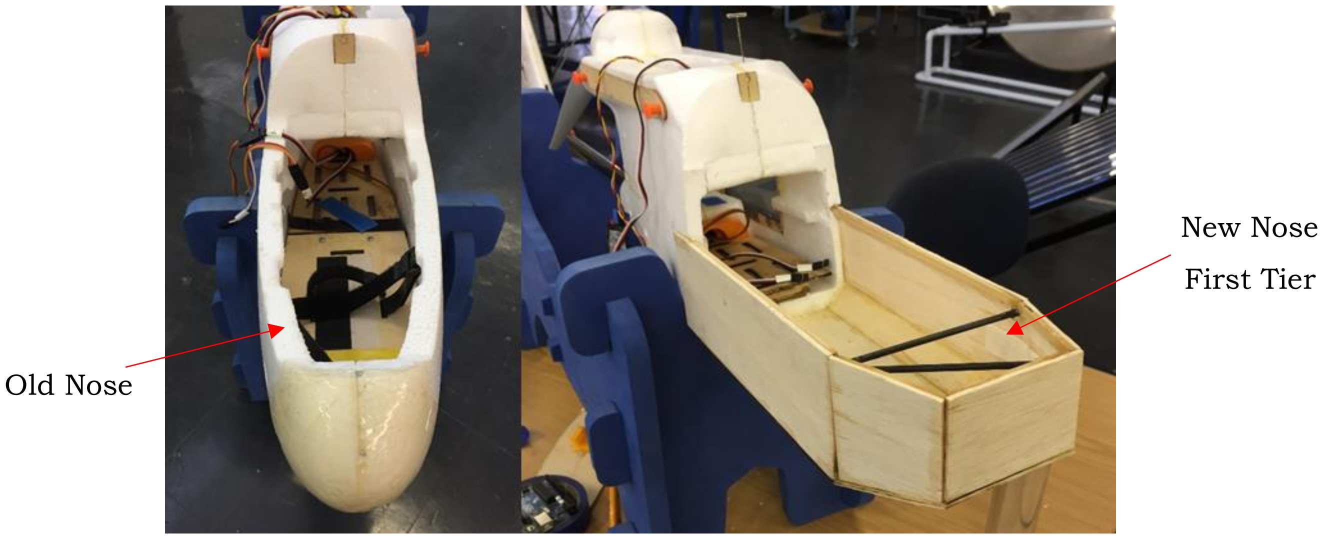


*Figure 3.22: Comparative view of the aircraft's old nose and new nose*

The second tier was designed to interlock with the first tier via forward facing 3D printed hooks. These hooks connected with the first tier's support rods, as seen in the lower right-hand corner of Figure 3.22. In addition to the forward-facing hooks, two latches were added to the rear of the second tier and engaged with two reinforced holes in the aircraft's fuselage. The second tier attached to the first tier can be seen in Figure 3.23.

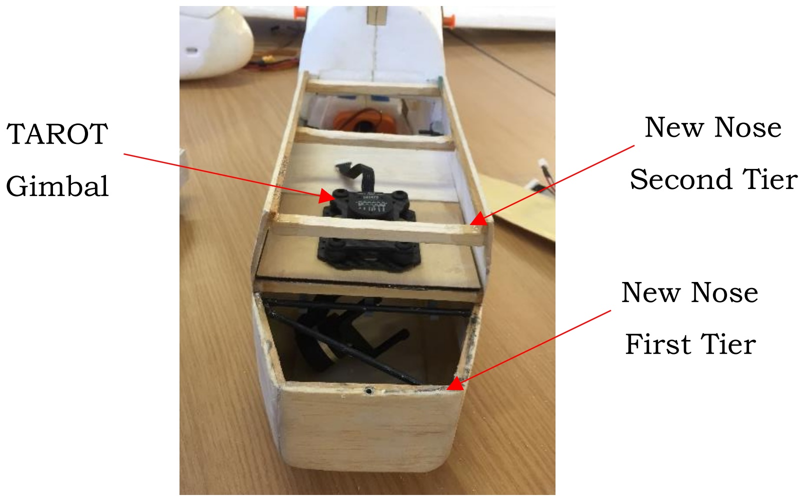


*Figure 3.23: Second tier of the new nose with the camera gimbal mounted*

The third tier of the new nose interlocked with the second tier via three sets of rear-facing hooks and was locked into place with the use of the fibreglass reinforced hood. The hood was then secured to the aircraft via two latches, one which interlocked the front of the hood with the first tier and the second latch interlocked with the aircraft upper fuselage. Figure 3.24 illustrates the third tier of the new nose, Figure 3.25 illustrates the hood during fibre-glassing and Figure 3.26 illustrates the fully assembled new nose and fuselage.

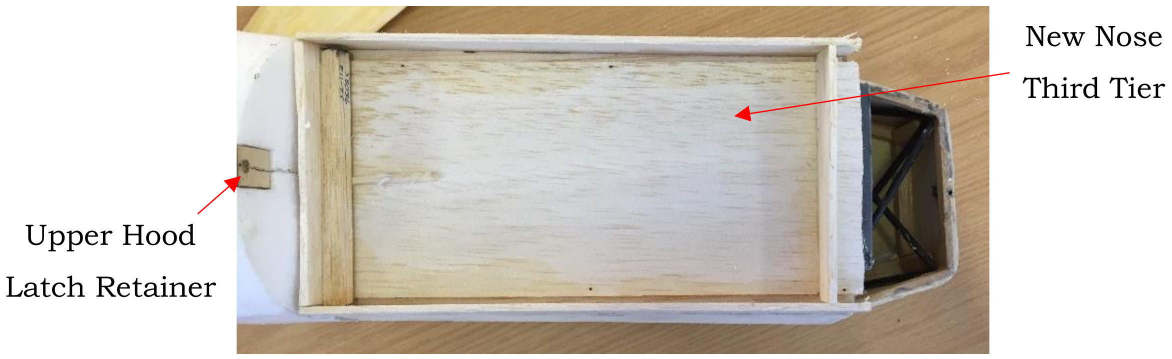


*Figure 3.24: The third tier of the new nose*

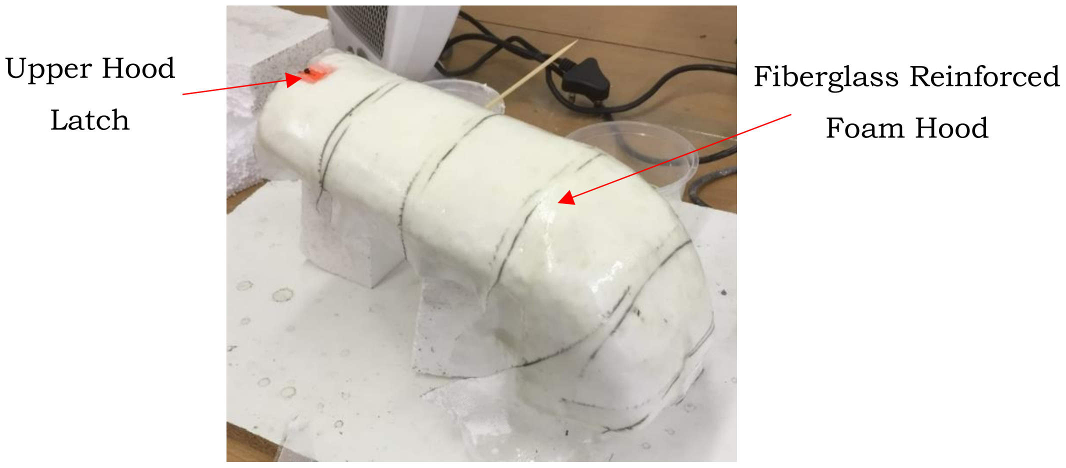


*Figure 3.25: Fibreglass reinforcing the new nose hood*

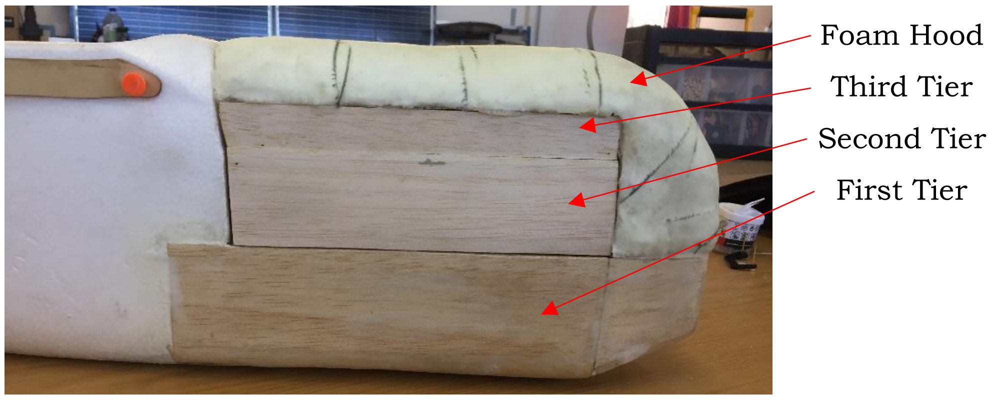


*Figure 3.26: Fully assembled new nose and fuselage*

## 3.4.2 New Wing

With the addition of the onboard hardware, the new nose and fuselage, the mass of the aircraft increased and as such, the Skywalker's standard wing was not suited to support the additional load. To assist with this additional mass, a new wing was developed for the aircraft. This new wing was implemented for an application specific aircraft, as the original Skywalker wing was designed in a polyhedral fashion. Polyhedral implied a wing with multiple stages of dihedral, where dihedral refers to the dihedral angle a wing possesses with respect to the local horizontal. The added benefits of dihedral come in the form of roll stability, where if the aircraft experiences slight displacement in the roll axis, it will naturally return to its original attitude [24]. The added benefit of roll stability from a dihedral design comes at the sacrifice of lift and drag. As such, the polyhedral design of the Skywalker's wing sought to maximise roll stabilisation and available lift by implementing a steep dihedral angle at the wing tips and a shallower dihedral angle at the wing root. Where, the steeper dihedral angle at the wing tips promoted roll stabilisation and the shallower dihedral angle promoted lift. As autonomous flight control of the aircraft was taken during missions, the need for roll stabilisation was compensated for by the FCU. As such, optimising the available lift of the wing was the primary objective for the new wing's design. The Skywalker's polyhedral wing design can be seen in Figure 3.27.

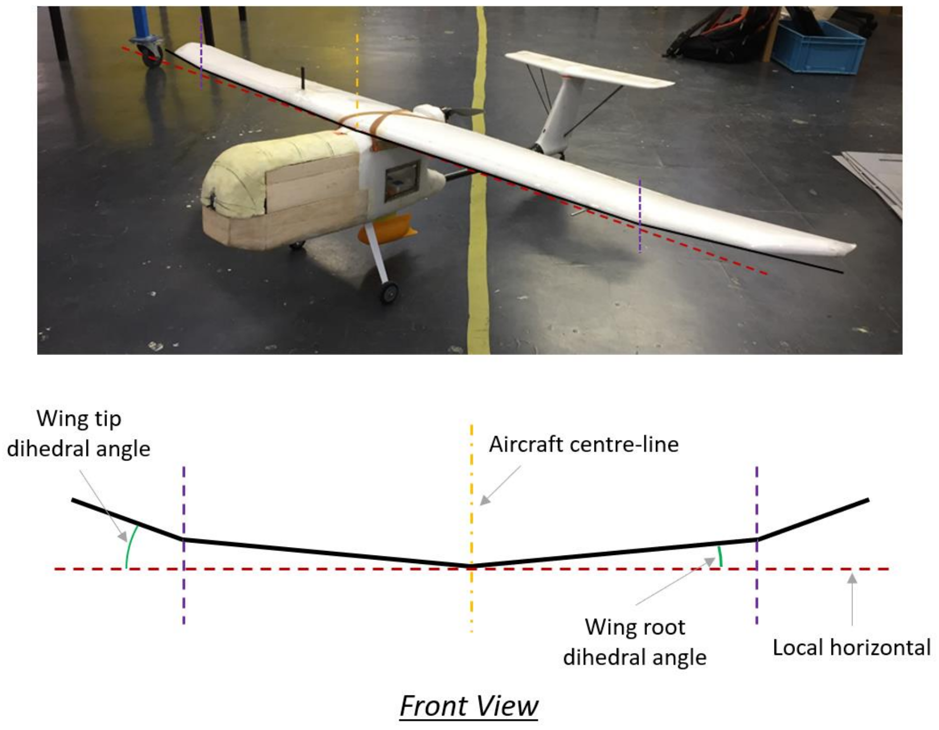


*Figure 3.27: Skywalker polyhedral wing structure*

The first stage of designing the new wing for the aircraft involved the selection of the aerofoil profile of the new wing. The Clark Y (clarky-il) aerofoil was selected as the desired profile for the new wing, as seen in Figure 3.28. The Clark Y aerofoil was a flat bottom aerofoil for application in general purpose aircraft designs and the flat bottom design provided improved lift characteristics.

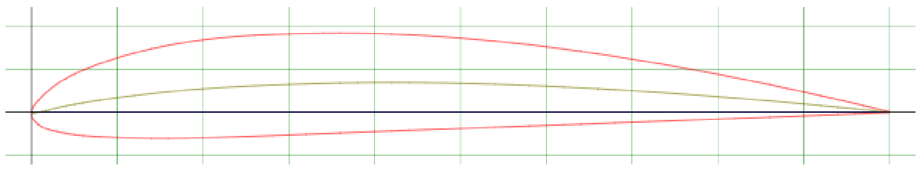


*Figure 3.28: Clark Y (clarky-il) aerofoil*

With the desired aerofoil selected, two wing halves were cut from closed cell expanded polystyrene (EPS) foam. With the wing halves prepared, the next stage of the new wing design was to determine the required spar cap design to allow for the wing to be able to endure wing loading. The process of determining the required design of the spar caps was done through the determination of the required moment of inertia of the spar caps. To determine the required moment of inertia, a wing strength analysis was conducted by simulating wing loading. Only half the wing was analysed, as wing loading was symmetric about the centreline of the aircraft. As such, the wing was broken into ten equidistant portions, with the final two wing segments being broken into five further segments each. The division of the wing for analysis can be seen in Figure 3.29.

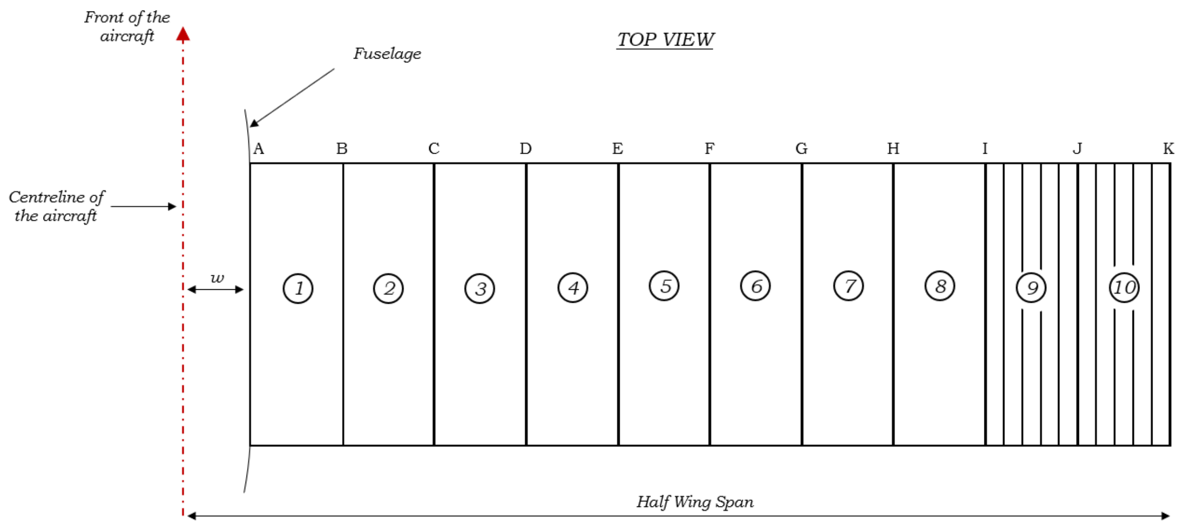


*Figure 3.29: Wing division for wing loading analysis*

For the purpose of this research, elliptical loading was used to determine the required moment of inertia of the new wing's spar caps, where the applied loading was the bending moment experienced by the wing. The theoretical elliptical loading experienced by the new wing can be seen in Figure 3.30.

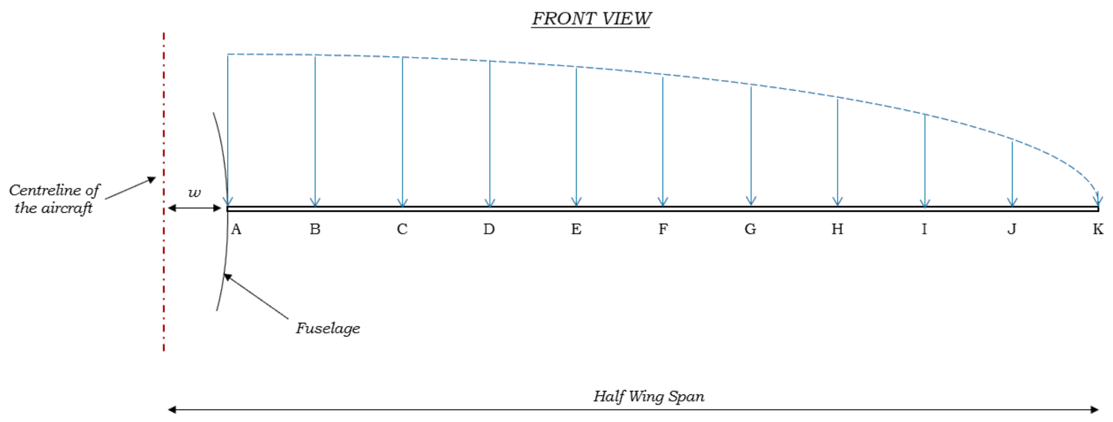


*Figure 3.30: Elliptical loading of new wing*

From Figure 3.30, it could be seen that towards the wing tip segments, the elliptical wing loading changed rapidly as the slope of the elliptical wing loading decreased. Due to the rapid change in wing loading, the final segments of the wing were divided into smaller portions for analysis. To determine the required spar cap moment of inertia, the total wing load, $F_{WL}$, to be expected was determined, where the total load was represented by the area of the quarter elliptical wing loading function, as seen in Figure 3.31.

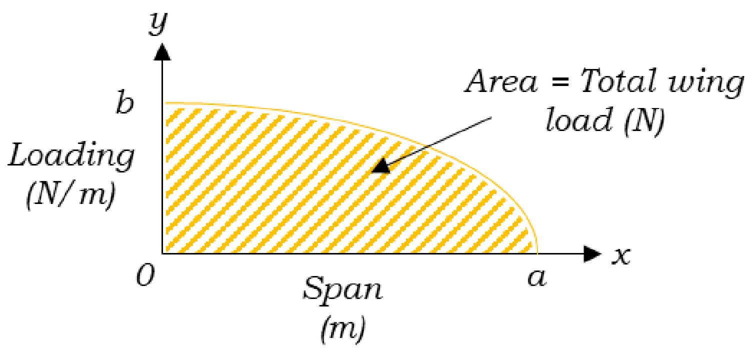


*Figure 3.31: Total wing load area within an elliptical loading function*

To determine this area, three additional aircraft parameters were required to be known: the gross mass of the aircraft, $m_a$, the mass of a single wing, $m_w$, and the load factor, $n$, for the wing. Equation 3.1 described the area of an ellipse, Equation 3.2 described the equation of an ellipse, Equation 3.3 defined the total wing load and Equation 3.4 represented the effective wing area the wing load would act upon.

$$A_{ellipse} = \frac{\pi ab}{4} \tag{3.1}$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3.2}$$

$$F_{WL} = \frac{gn(m_a - 2m_w)}{2} \tag{3.3}$$

$$a = \frac{Span}{2} - w \tag{3.4}$$

Where, $a$ represented the loaded span length and $g$ was the gravitational acceleration. The maximum bending moment, $b$, was now determined by setting Equation 3.1 equal to Equation 3.3, as all equation variables were now known. The variables for these equations were found to be as follows:

$$F_{WL} = 90.25\ N$$

$$a = 0.93\ m$$

$$b = 136.54\ N/m$$

From the aforementioned variables, the maximum wing load was defined to be $F_{WL}$ = 100 $N$ to allow for a factor of safety. As such, with the use of Equation 3.2, the elliptical wing loading function was found to be:

$$y = 136.54\sqrt{1 - \frac{x^2}{0.87}} \tag{3.5}$$

For each wing segment, numbered 1 – 10, the net force exerted over the wing area was determined. The net force per segment was represented by the area of the elliptical wing loading function, which was initially determined through the application of a Left Riemann Sum of the elliptical load function. Subdivisions for the Riemann Sum were taken to be the wing segment divisions. The elliptical wing loading function graph, with the Left Riemann Sum superimposed, can be seen in Figure 3.32.

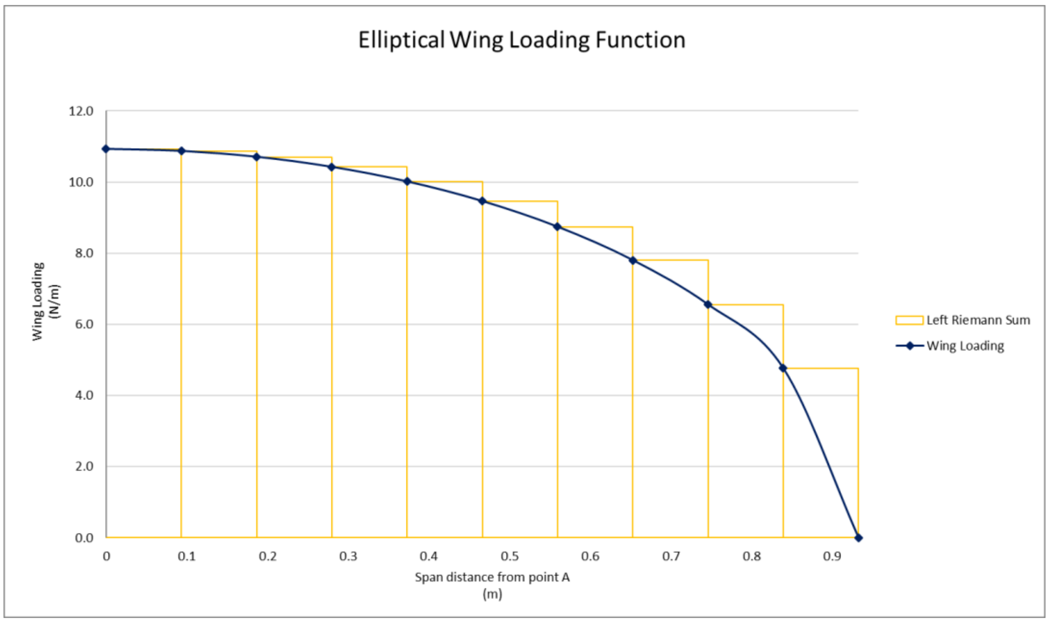


*Figure 3.32: Elliptical wing loading function with Left Riemann Sum Superimposed*

From Figure 3.32, it was noted that the Left Riemann Sum resulted in an overshoot in terms of the net force exerted on the wing and as such, the finite integral of the elliptical wing loading function over the wingspan was taken to deduce the net force. Each wing segment required its own finite integral. Equation 3.6 defined the general form of the finite integral used to determine the net force exerted over the wing.

$$F_{net} = \int_{d0}^{d1} \left( 136.54 \sqrt{1 - \frac{x^2}{0.87}} \right) dx \tag{3.6}$$

Where, $d0$ and $d1$ represented the segment start and end distances in relation to point A on the wing. With the net force for each wing segment known, the bending moment ($M$) and shear force ($V$) at the inside edge of each wing segment could be determined.

The bending moment was determined by taking the net force applied to the centre of each wing segment. Equation 3.7 defined the bending moment of each wing segment. The reference for the bending moment fulcrum was wing point $K$. The bending moment was used to determine the ability of the spar cap to endure the desired wing loading.

$$M_{Seg} = \sum_{n=K}^{Seg} F_{net_n} d_n \tag{3.7}$$

Where, $M_{Seg}$ represented the bending moment of a particular wing segment, $F_{Seg_n}$ represented the net force exerted on a wing segment starting at wing point $K$ and working towards desired wing segment, and $d_n$ represented the distance from wing point $K$ to the current wing segment's centre. The new wing bending moment curve can be seen in Figure 3.33.

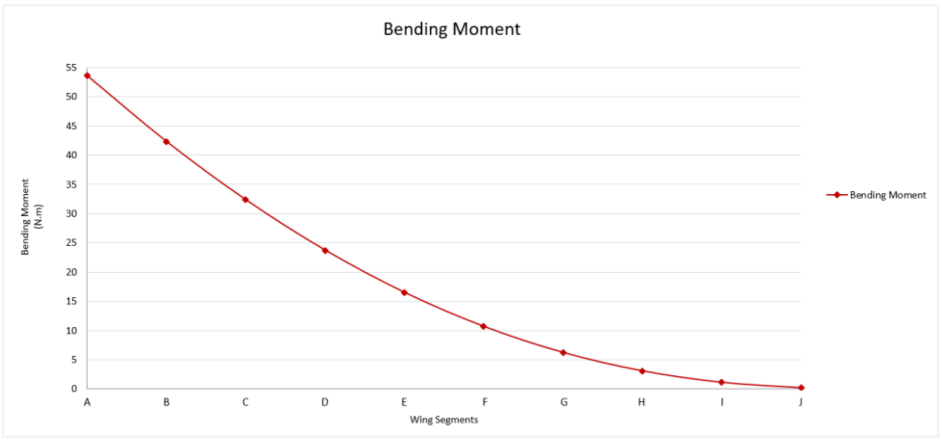


*Figure 3.33: New wing bending moment curve*

From Figure 3.33, it was noted that the bending moment diminished across the length of the wingspan, where the maximum bending moment was found to be 53.58 N.m at the wing root (wing point A). With the bending moment for each segment known, the required moment of inertia of the spar cap was determined. Equation 3.8 described the moment of inertia of the spar cap.

$$I = \frac{My}{\sigma} \tag{3.8}$$

Where, $I$ represented the moment of inertia, $M$ represented the bending moment, $y$ represented the vertical distance away from the neutral axis of the wing's aerofoil and $\sigma$ represented the allowable bending stress of the spar cap material. During the cutting of the aerofoil, the spar cap position was placed at 1 mm above the neutral axis of the aerofoil. The desired spar cap material was selected to be pultruded carbon fibre, with a bending stress of 1.6 GPa. The required moment of inertia of each wing segment was determined, these moments of inertia can be seen in Table 3.9.

Table 3.9: Wing segment moments of inertia

| Wing Segment | Moment of Inertia ($m^4$) |
|---|---|
| 1 | 8.37 x$10^{-12}$ |
| 2 | 6.61 x$10^{-12}$ |
| 3 | 5.06 x$10^{-12}$ |
| 4 | 3.71 x$10^{-12}$ |
| 5 | 2.58 x$10^{-12}$ |
| 6 | 1.67 x$10^{-12}$ |
| 7 | 0.97 x$10^{-12}$ |
| 8 | 0.48 x$10^{-12}$ |
| 9 | 0.17 x$10^{-12}$ |
| 10 | 0.03 x$10^{-12}$ |

From Table 3.9, it was noted that the required moment of inertia of the spar cap decreased over the span of the wing. As such, a variable spar cap design would have been optimal in terms of the given moments of inertia. Due to the complexity and cost of manufacturing a variable spar cap, pultruded carbon fibre tubing and rectangular extrusion was used to construct the spar caps. As wing loading decreased outward from the fuselage of the aircraft towards the wing tips, the spar cap design was varied in terms the length of the carbon fibre tube utilised. The tubing was present for the first 0.5 m of a wing's spar cap and the rectangular extrusion was present in two strips, at the top and bottom of the aerofoil, for the full length of the wing. The spar cap cross-sections can be seen in Figure 3.34.

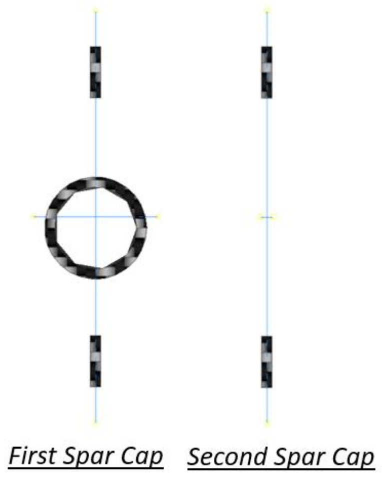


Figure 3.34: Spar cap cross-sections

The required shear stress ($\tau$) of the wing was determined by deducing the applied shear force on the wing. The required shear stress of the wing was an indication of the required shear stress of the material from which the wing body was made. In this case, EPS foam. Equation 3.9 described the required shear stress.

$$\tau = \frac{V}{A} \tag{3.9}$$

Where, $A$ represented the area of the upon which the shear force acted. The shear force curve can be seen in Figure 3.35.

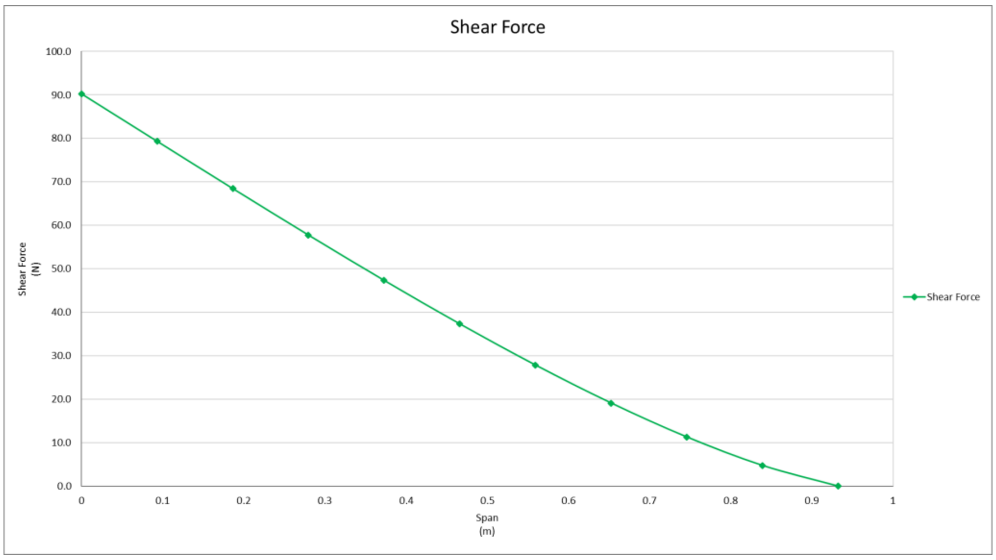


*Figure 3.35: New wing shear force curve*

From Figure 3.35, it was noted that the shear force decreased over the length of the wingspan and the maximum shear force was found to be 90.3 N at the wing root. Due to the EPS foam design of the wing, the required shear stress of the wing was defined to be the shear strength of the EPS foam between the spar cap segments. The required shear stress of the new wing was defined to be the shear stress required at the point of maximum shear force along the wing, but due to the change in spar cap design throughout the wing, two shear stress measurements were required. One from the first spar cap section and one from the second spar cap section. From these two shear stress measurements, the measurement with the maximum required shear stress was defined to be the wing's minimum required shear stress. The minimum required shear stress was found to be 407.07 kPa with a shear force of 90.3 N applied to a shear area of 0.22x10$^{-3}$ m$^2$ at wing point $A$. The shear stress of the EPS foam used in the construction of the wing was 482.63 kPa and as such, the wing's design was within the required specifications to endure the desired wing loading. The spar cap and new wing design calculations can be seen in Appendix G.

Upon construction of the wing, it was found that due to the hot wire foam cutter's spar cap cutting toolpath, the spar caps separated from the wing during tension and compression testing. This separation implied that the tension and compression experienced by the wing was not transferred across the wingspan via the spar caps. An additional wooded spar cap was placed in the middle portion of the wing, extending 0.6 m symmetrically about the centre of the wing. The wooden spar cap possessed a rectangular cross-section and extended through the entire thickness of the new wing. The wing was then sealed with clear adhesive tape to provide torsional strength.

With the new wing constructed, it could be compared to the original Skywalker wing. An aspect ratio comparison was undertaken to compare the two wings. Where, aspect ratio ($AR$) was an indication of the square of the wingspan ($S^2$) divided by the area of the wing ($A$), as defined by Equation 3.10 [24].

$$AR = \frac{S^2}{A} \tag{3.10}$$

Higher aspect ratio wings provided higher lift to drag ratio, allowing them to carry a given weight easier than their lower aspect ratio wing counterparts. It was found that the original Skywalker wing possessed an aspect ratio of 6.1639 and the new wing possessed an aspect ratio of 6.48.

In addition to aspect ratio, a comparison in wing loading capabilities of the two wings showed that the new wing resulted in an 12.60% reduction in wing loading when compared to the original Skywalker wing. This wing loading comparison was done by working backwards with the moment of inertia of the spar caps and calculating the allowable loading force each wing would experience. As the spar caps were over designed for the new wing, they provided additional strength.

With the physical parameters of the wing defined, the control surfaces and additional features could be added to the wing. The control surfaces included ailerons and flaps and the additional features included a wireless telemetry module and an airspeed sensor. The new wing, with the control surfaces included, can be seen in Figure 3.36 and the new wing attached to the aircraft can be seen in Figure 3.37. The new wing's specifications can be seen in Table 3.10.

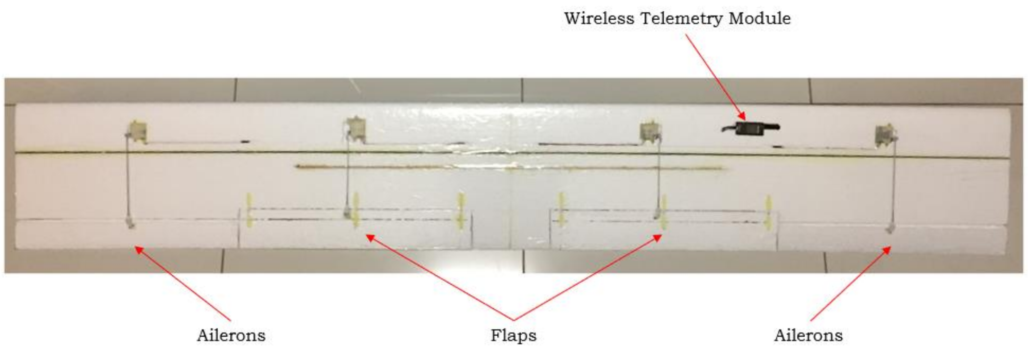


*Figure 3.36: New wing with wireless telemetry and control surfaces included*

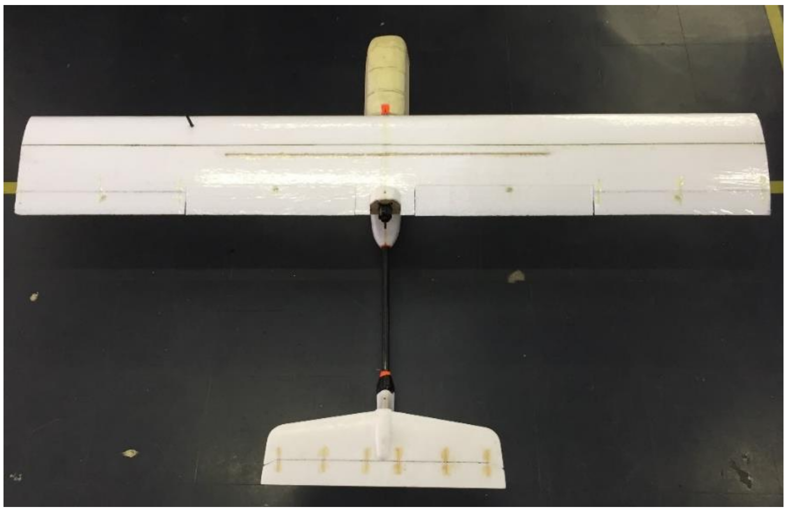


*Figure 3.37: New wing attached to the aircraft*

*Table 3.10: New wing specifications*

| *Parameter* | *Value* | *Unit* |
|---|---|---|
| Aerofoil | Clark Y | - |
| Chord Length | 0.31 | m |
| Thickness | 0.04 | m |
| Span | 1.98 | m |
| Aspect Ratio | 6.48 | - |
| Spar Cap Centre Position | From Leading Edge: 0.102<br>From Lower Surface: 0.018 | m<br>m |
| Wing Material | Expanded Polystyrene | - |
| Spar Cap Material | Carbon Fibre | - |
| Spar Cap Moment of Inertia | First Spar Cap: $2.31 \times 10^{-9}$<br>Second Spar Cap: $2.01 \times 10^{-9}$ | $m^4$<br>$m^4$ |
| Wing Load Factor | 4 | - |
| Wing Loading Reduction | 12.60 | % |
| Full Wing Mass | 0.4 | kg |

### 3.4.3 Additional Alterations

With the new nose, fuselage and wing completed, several additional changes were introduced. These alterations included the addition of landing gear, including a tail wheel for ground steering during takeoffs. Another alteration included the addition of battery and FCU shelves inside the fuselage. A viewing window was installed on the side of the fuselage to allow for visual confirmation that the FCU was responding correctly in terms of its onboard LED indicator. These additional features concluded the alterations made to the aircraft.

## 3.5 Chapter Conclusion

Chapter 3 defined the system architecture utilised in this research. Elements such as the airframe, onboard hardware and onboard software were discussed. Component selection comparisons were undertaken and the Pixhawk FCU, ODroid-XU4 and ELP-USB500W04AF-A60 CMOS camera were some of the selected hardware for use in this research. A Linux-based, C++ programming language ROS node architecture was developed as the system's software architecture, where five nodes were developed for the purpose of achieving the desired research objectives defined in Chapter 1. The final section of Chapter 3 discussed the alterations made to the aircraft in order to achieve the desired system functionality. The development of a new nose, fuselage and wing were discussed. With the conclusion of Chapter 3, Chapter 4 discussed the first subsystem, the autonomous flight control system.

# Chapter 4
# Autonomous Flight

This chapter discussed the details pertaining to the autonomous flight node and has been structured to describe the various phases of the autonomous flight process. In addition to this node structure, the defining Pixhawk parameters required to achieve the defined flight procedures were also discussed.

## 4.1 Autonomous Flight Node

As discussed in Chapter 3, the autonomous flight node made use of the MAVROS header to define all flight control variables. These processes include pre-flight calibration and safety checks, motor arming, flight mode selection, take-off and landing, waypoint allocation, environmental parameter determination and payload release approach. Prior to the configuration of the autonomous flight node, the computational logical flow was mapped out, as seen in Figure 4.1. The autonomous flight node code can be found in Appendix A1.

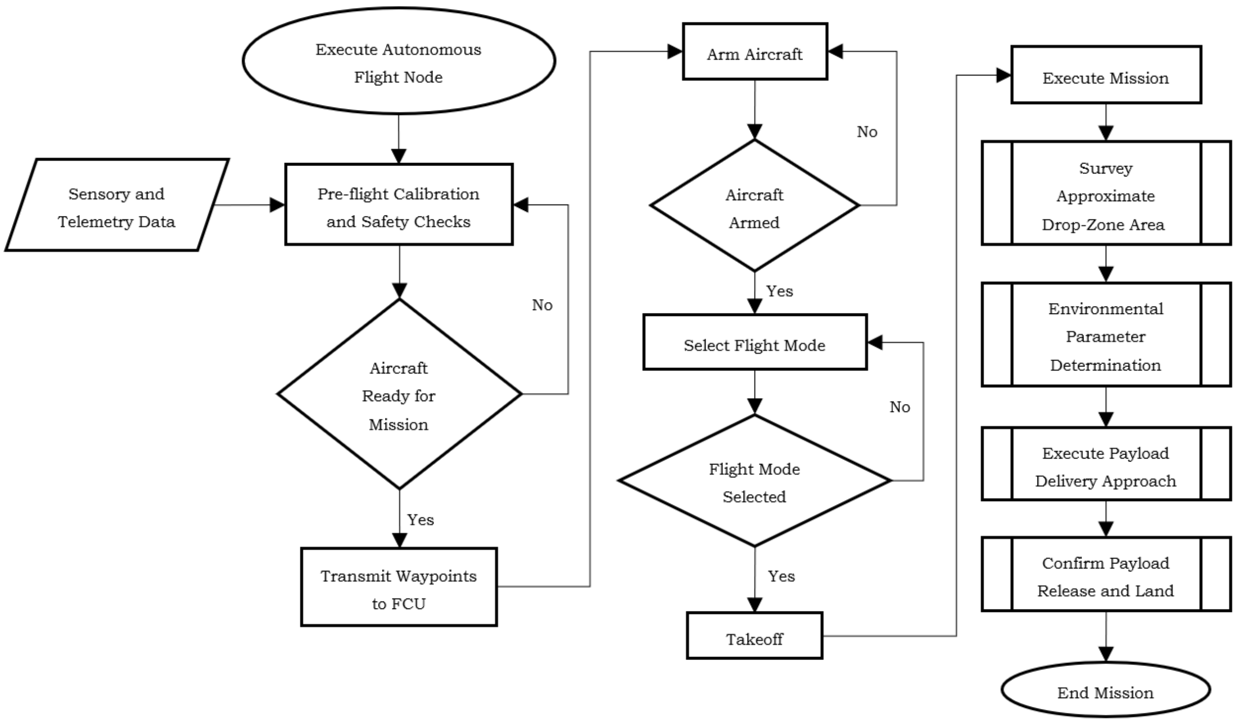

Execute Autonomous Flight Node
Sensory and Telemetry Data
Pre-flight Calibration and Safety Checks
Aircraft Ready for Mission
No
Yes
Transmit Waypoints to FCU
Arm Aircraft
Aircraft Armed
No
Yes
Select Flight Mode
Flight Mode Selected
No
Yes
Takeoff
Execute Mission
Survey Approximate Drop-Zone Area
Environmental Parameter Determination
Execute Payload Delivery Approach
Confirm Payload Release and Land
End Mission

*Figure 4.1: Autonomous flight node computational flow diagram*

### 4.1.1 Pre-flight Calibration and Safety Checks

The aircraft was placed through a series of pre-flight safety checks and calibration tests. These checks included the response of the various control surfaces, battery state of charge, airspeed sensor feedback, GPS lock, altitude disparity, response of the vision system and release mechanism. These checks all fell within a greater check which ensured that a stable node build was running and that a sustainable communication between the FCU and the MAVROS node was established. This broader check was run throughout the flight control process to ensure sustained control over the aircraft by the companion computer.

The Pixhawk possessed an onboard, pre-arm parameter evaluator which, when enabled, prevented the aircraft from being armed unless all selected pre-arm checks were successful. This pre-arm evaluator was used to ensure the airspeed sensor, GPS and the battery's state of charge were responding correctly. The remaining pre-arm checks were assessed by subscribing and publishing to the MAVROS headers. These checks included the testing of the response of the control surfaces and payload release mechanism. For the vision system, the establishment of successful connections to the image capture node and the image processing node was defined as a successful response.

### 4.1.2 Arming and Disarming

Based on the success of the pre-flight arming checks, an arming client was defined which requested the aircraft's arming sequence to take place. This arming client fell part of the MAVROS library's *CommandBool* header which requested the changing of the arming state of the aircraft. To ensure the aircraft successfully armed, an FCU state subscriber was defined to listen to the aircraft's published state using the MAVROS Library's *State* header, where the published state of the aircraft defined the current arming state [48][49].

The arming of the aircraft was required prior to selection of the flight mode due to a predefined safety feature of the Pixhawk which prohibited the selection of an autonomous flight mode prior to arming the aircraft.

## 4.1.3 Flight Mode Selection

With the use of the Pixhawk, a variety of flight modes were available for selection. However, the MAVROS header library's limited application for fixed-wing aircraft, this list of flight modes was reduced to a select few. These flight modes fell under the MAVROS library's *CustomModes* header. See Table 4.1 for the list of available flight modes [43][48][49].

*Table 4.1: List of available flight modes for the APM Plane Pixhawk firmware*

| *Numeric* | *String* | *Description and notes* |
|---|---|---|
| 0 | MANUAL | |
| 1 | CIRCLE | |
| 2 | STABILIZE | |
| 3 | TRAINING | |
| 4 | ACRO | |
| 5 | FBWA | Fly by wire A |
| 6 | FBWB | Fly by wire B |
| 7 | CRUISE | |
| 8 | AUTOTUNE | |
| 10 | AUTO | |
| 11 | RTL | Return to Launch |
| 12 | LOITER | |
| 14 | LAND | *not in list now* |
| 15 | GUIDED | |

From the Table 4.1, the AUTO flight mode was selected for the mission. The MAVROS command for the selection of the aircraft's flight mode was not iterated repeatedly throughout the mission and only a single instance of the flight mode was defined at the beginning of the mission. The reason for this single declaration of the flight mode was to allow for manual control, STABILIZE flight mode, of the aircraft to be taken at any point should the aircraft's autonomous flight control fail. The change from AUTO flight mode to STABILIZE flight mode was setup as a function of the aircraft's remote control, requiring a flip of the switch to interchange between the two modes. Once back within stable control and at the operator's discretion, AUTO flight mode could be reactivated by switching the transmitter flight mode switch back to its initial position. Upon reactivation, the FCU would pick up the mission where it had last left off after changing flight modes.

### 4.1.4 Takeoff

Although the MAVROS Library possessed a header for the takeoff of an aircraft, *CommandTOL*, the header provided limited parameter allocation for the takeoff procedure of a fixed-wing aircraft and as such, the takeoff was defined as a takeoff waypoint. Several fixed-wing takeoffs were possible with the Pixhawk: all were defined in the same manner in terms of defining the take-off waypoint but differed in terms of their respective FCU parameters, where these FCU takeoff parameters were defined based on the aircraft's physical capabilities and the type of takeoff desired. The available Pixhawk takeoff methods include [1]:

- Hand Launching,
- Catapult Launching,
- Bungee Launching, and
- Runway Takeoffs - Conventional Takeoff and Landing (CTOL).

#### 4.1.4.1 Hand Launching

During the initial testing stages, the HAND LAUNCHING takeoff was utilised as the addition of the aircraft's undercarriage had not yet been completed. Figure 4.2 illustrates *Hand Launching* during the testing phase. *Hand Launching* was used with the assistance of manual flight control.

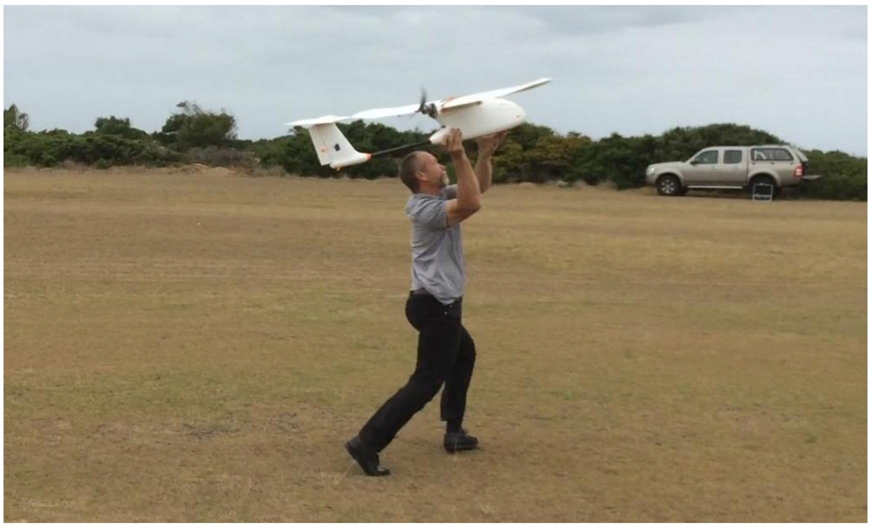


*Figure 4.2: Hand launching the aircraft during the testing phase*

### 4.1.4.2 Catapult Launching

An attempt to automate the takeoff process resulted in the development of a catapult system to make use of the aircraft's *Catapult Launching* takeoff. Table 4.2 defines the design criteria for the catapult.

*Table 4.2: Catapult design criteria*

| *Criteria* | *Value* | *Unit* |
|---|---|---|
| Launch Angle ($\alpha$) | 15 | Degrees |
| Minimum Launch Velocity ($V_L$) | 18 | kt |
| Maximum Catapult Length ($L$) | 2 | m |
| Launch Mass (Aircraft + Payload) | 3 | kg |

From the catapult criteria, the required spring constant for the catapult's launch mechanism was determined. To determine the elastic potential required, a force diagram representation of the catapult was developed, as seen in Figure 4.3.

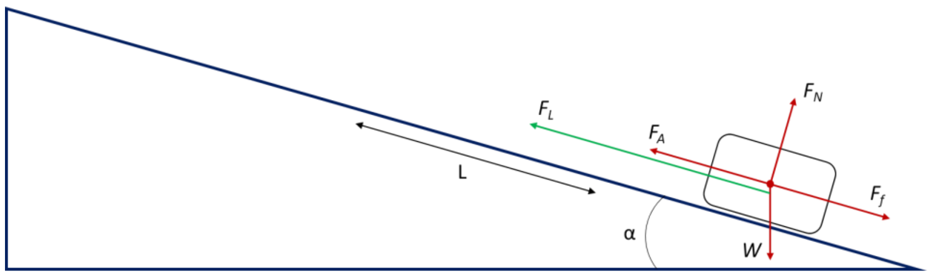


*Figure 4.3: Catapult force diagram*

From Figure 4.3, $F_L$ represented the required launching force, $F_A$ represented the minimum required force for the aircraft to remain in equilibrium (the preload force), $F_N$ represented the normal force of the aircraft, $W$ represented the weight of the aircraft and $F_f$ represented the frictional force of the catapult.

The required preload force and launching force were deduced to be as follows:

- $F_A = 18.39\ N$
- $F_L = 64.31\ N$

Where, $F_L$ was the average launch force over the length of the catapult. From these forces, the required spring constant of the launching mechanism was deduced with the use of Equation 4.1, Hooke's Law [30].

$$F_L = kx \tag{4.1}$$

Where, $k$ represented the spring constant of the launch mechanism and $x$ represented the displacement of the launch mechanism.

To assist the launch mechanism, the preload force was compensated for by applying thrust from the aircraft's motor. The available thrust of the aircraft was deduced by placing the aircraft against a scale and measuring the additional weight generated by the aircraft whilst applying thrust. In addition to this preload thrust, the launch mechanism's available displacement ($x$) was increased by introducing a two-stage pulley mechanism. With the additional displacement for the launch mechanism being defined to 1.8 m, the required spring constant for the launch mechanism was found to be, with the assistance of Equation 4.1, $k = 36$.

With the spring constant known, latex rubber tubing was selected as the desired catapult launching mechanism. The rubber tubing was tested to deduce its elastic properties. This testing involved measuring the required force to extend the tubing to various lengths. The results of this testing can be seen in Table 4.3 and Figure 4.4.

*Table 4.3: Latex rubber elastic properties testing results*

| *Pieces of Rubber Tubing* | *Extension Distance (m)* | *Extension Ratio* | *Required Force for Extension (N)* |
|---|---|---|---|
| Single Band (Initial length – 4.7 m) | 7 | 1.49 | 34.34 |
| | 10 | 2.13 | 49.05 |
| | 13.40 | 2.85 | 68.67 |
| Double Band (Initial length – 2.35 m) | 3.10 | 1.32 | 58.86 |
| | 4.23 | 1.80 | 88.29 |
| | 5.75 | 2.45 | 117.72 |

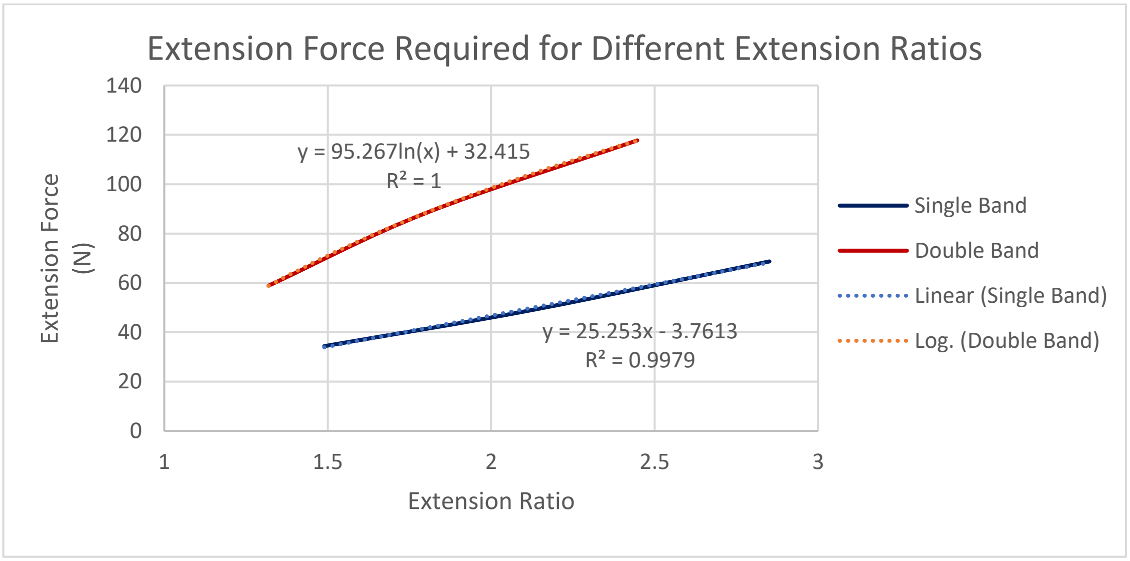


*Figure 4.4: Latex rubber extension force as a function of extension ratio*

Looking at Table 4.3, it was deduced that the use of a double band launch mechanism would result in the greatest launching force with the smallest required extension length. From the extension force trend equation in Figure 4.3, it was deduced that an extension ratio of 1.40 was required to achieve the required average launch force of the length of the catapult. This extension ratio implied that a latex rubber double band of length 1.29 m was required. It was decided that the length of the rubber tubing would be decreased to 1 m to allow for an additional preload force to be introduced into the system. Refer to Appendix H1 for the catapult launcher calculations.

The final catapult was designed and built as per the calculated forces. To cradle the aircraft on the catapult, a launch sled was designed to fit the rear of the aircraft, allowing for the tail and propeller of the aircraft to pass through without collision during launch. A release mechanism for the catapult was designed into the launch sled. The launch sled design can be seen in Figure 4.5.

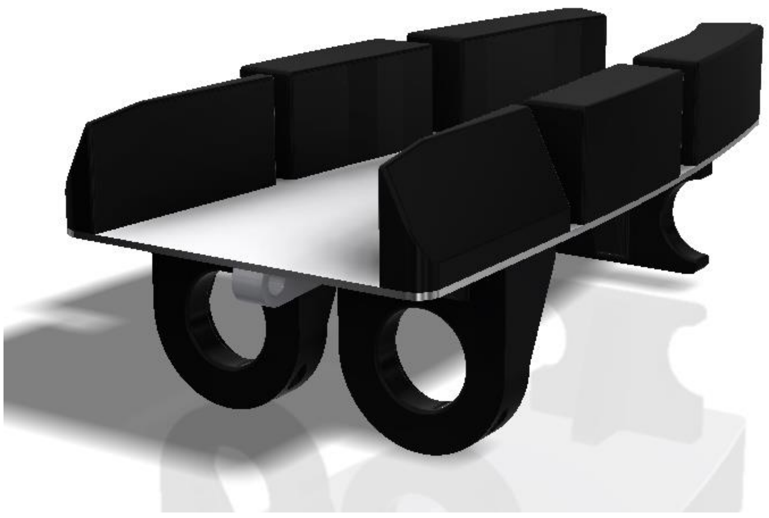


*Figure 4.5: Catapult launch sled CAD model design*

The catapult proved to successfully launch the aircraft but limited the available room to mount the payload release system. As such, the catapult was used during the testing stage, as the aircraft's autonomous flight control was being developed but was replaced by the addition of landing gear on the aircraft. The aircraft being launched from the catapult can be seen in Figure 4.6. Refer to Appendix H2 for the CAD designs of the launch sled and catapult.

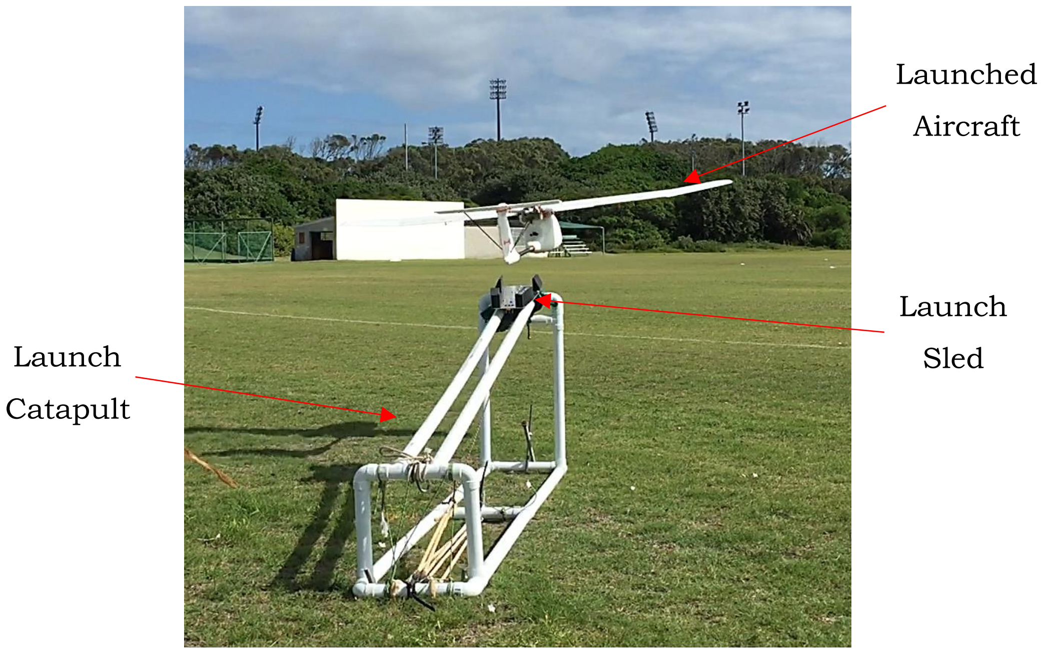


*Figure 4.6: Aircraft launched from the catapult*

#### 4.1.4.3 Runway Takeoff

For the final integrated aircraft system's takeoff, the addition of landing gear allowed for the selection of the *Runway Takeoff*. As stated previously, the takeoff for the aircraft was defined as a takeoff waypoint, which was done with the use of the MAVROS Library's header *Waypoint* and *WaypointPush*. The *Waypoint* header provided the ability to define the desired waypoint type and corresponding parameters. The takeoff waypoint parameters can be seen in Table 4.4 [47].

*Table 4.4: Takeoff waypoint parameters*

| *MAV_CMD_NAV_TAKEOFF* | |
|---|---|
| *Parameter* | *Description* |
| Mission Param #1 | Minimum pitch |
| Mission Param #2 | - |
| Mission Param #3 | - |
| Mission Param #4 | Yaw angle |
| Mission Param #5 | Latitude |
| Mission Param #6 | Longitude |
| Mission Param #7 | Altitude |

The location of the takeoff waypoint was automatically updated based on the location of the aircraft prior to takeoff and as such, only the desired altitude of the takeoff was necessary to be defined in the takeoff waypoint parameters. After the parameters of the takeoff waypoint had been defined, the *WaypointPush* header was used to publish the waypoint to the FCU. In addition to the declaration of the takeoff waypoint, the *Runway Takeoff* required that the aircraft was faced in the direction of the headwind prior to takeoff and that the corresponding FCU takeoff parameters for the *Runway Takeoff* be defined, as seen in Table 4.5. The use of the *Runway Takeoff* allowed for the aircraft to support the payload below the aircraft, as opposed to on the side of the aircraft, reducing the need to counter balance the aircraft [2][43][48][49].

*Table 4.5: FCU takeoff parameters*

| *Parameter* | *Value* | *Units* | *Description* |
|---|---|---|---|
| TKOFF_TDRAG_ELEV | 100 | % | Amount of elevator to apply in the initial stage of takeoff |
| TKOFF_TDRAG_SPD1 | 4 | m/s | Airspeed at which to stop holding the tail down during takeoff |
| TKOFF_THR_SLEW | 33 | %/s | Percentage throttle change per second during takeoff |
| TKOFF_ROTATE_SPD | 11 | m/s | Airspeed at which the aircraft will begin to climb during takeoff |
| TECS_PITCH_MAX | 15 | Degrees | Maximum pitch in autonomous throttle modes |
| STEER2SRV_P | 4 | m | Ground turning radius of the aircraft |
| GROUND_STEER_ALT | 5 | m | Altitude at which ground steering of the rudder takes effect |

### 4.1.5 Waypoint Allocation

The designation of all waypoints took place prior to the arming of the aircraft during the pre-flight checks. Defining all of the mission waypoints prior to the mission was possible due to the consistency of how each mission was structured. Defining each waypoint prior to the mission also allowed for only their parameters to be updated during the mission, mitigating the need to create new waypoints. All waypoints where their final location was to be determined during the mission were defined with default coordinates of (0,0), as the waypoint would not be executed if not updated or within acceptable range and the aircraft would enter *RTL* flight mode. The automatic *RTL* feature was beneficial in the sense that the aircraft would not continue with the mission should mission waypoints not be within the achievable range of the aircraft.

#### 4.1.5.1 Survey Waypoints

Due to variation in each mission's drop-zone location, a low input waypoint allocation method was developed for the allocation of survey waypoints, where only the drop-zone's approximate location was to be defined prior to the mission's execution and the autonomous flight node would expand the waypoints about this point.

The survey area followed a square serpentine pattern of six defining coordinates, expanded about the drop-zone's approximate location. The spacing between these survey points was dependent on the FOV of the onboard vision system and aircraft's surveying altitude. Knowing the correct FOV for the planned surveying altitude, the survey waypoints were defined with the use of Equation 4.2 and Equation 4.3 [5].

$$Lat_S = \sin^{-1}(\sin(Lat_{dz}) \cos\left(\frac{d_{FOV}}{R_E}\right) + \cos(Lat_{dz}) \sin\left(\frac{d_{FOV}}{R_{Earth}}\right) \cos(\gamma)\ ) \tag{4.2}$$

$$Lon_S = Lon_{dz} + arctan2\left(\sin(\gamma) \sin\left(\frac{d_{FOV}}{R_E}\right) \cos(Lat_{dz}), \cos\left(\frac{d_{FOV}}{R_E}\right) - \sin(Lat_{dz})\ \sin(Lat_S)\right) \tag{4.3}$$

Where, $(Lat_S, Lon_S)$ represented the respective survey waypoint coordinates, $(Lat_{dz}, Lon_{dz})$ represented the coordinates of the drop-zone, $d_{FOV}$ represented the horizontal camera FOV distance, $\gamma$ represented the bearing of the survey waypoint with respect to the approximate drop-zone coordinates and $R_E$ represented the radius of the Earth. The survey area was designed to ensure the FOV of the camera overlapped the other survey coordinates to ensure full coverage and as such, the effective surveyed area with respect to the survey waypoints can be seen in Figure 4.7.

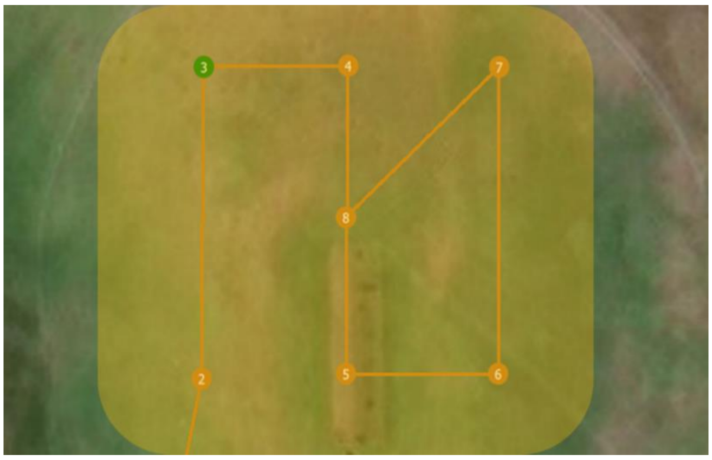


*Figure 4.7: Effective survey area and survey waypoints expanded about the drop-zone's approximate location (waypoint 8)*

After testing this survey system, it was found that aircraft was unable to successfully achieve the desired survey pattern without overshooting the desired survey points, specifically at the turn-around waypoints (Waypoints 3-4 and 5-6). As such, the minimum turning radius of the aircraft was determined based on the predefined maximum bank angle of the aircraft, $\varphi$, and the airspeed of the aircraft, $v_{air}$, where the maximum bank angle FCU parameter was defined to be LEVEL_ROLL_LIMIT = 30˚. With these variables and with the use of Equation 4.4, the minimum turning radius of the aircraft during flight, $R_{min}$, could be determined. It should be noted that for Equation 4.4, the airspeed of the aircraft needed to be in knots [44].

$$R_{min} = \frac{381 v_{air}^2}{14075 \tan \varphi} \tag{4.4}$$

The minimum turning radius did not account for wind. With the turn radius known, the issue of survey waypoint spacing and the aircraft's turn radius was accommodated for by designating an additional waypoint between the turn-around waypoints. This additional waypoint followed the trajectory of the survey pattern prior to the required turn and extended away from the survey area, such that the aircraft could successfully make the turn and achieve stable flight prior to re-entering the survey area. Figure 4.8 illustrates the extended survey area flight path.

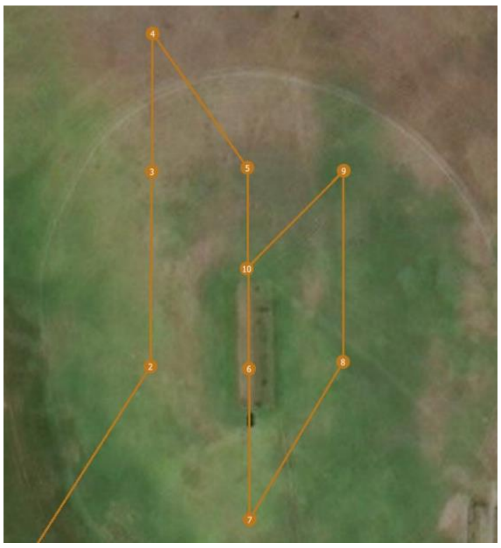


*Figure 4.8: Survey area waypoints with additional turn-around waypoints (waypoint 4 and waypoint 7)*

#### 4.1.5.2 Environmental Parameter Determination Waypoint

Once the vision system had successfully identified the drop-zone's location, the environmental parameters were determined. From previous testing, it was found that taking the airspeed upon the drop approach yielded inaccurate results as the average wind speed measured lower than the overall average windspeed, resulting in inaccurate drop trajectories. As such, the average windspeed was determined during the headwind direction determination. Both the headwind velocity and direction were determined with the use of a three loop loiter waypoint about the location of the target. These three loiter loops were executed with the use of the MAVROS Library's *Waypoint* and *WaypointPush* headers, where the required parameters to be declared was the centre coordinate of the loiter, the loiter altitude and the number of loiter loops. The centre of the loiter was defined to be the drop-zone location and the loiter altitude was defined to be the altitude at which the payload release would occur. The rationale behind the three loiter involved an initial redundant loop to stabilize the aircraft into a successful loiter flight path, the second loiter loop for environmental data capture, and the final loop for drop approach and trajectory computation and allocation.

To physically capture the headwind data from the aircraft's airspeed sensor, the MAVROS Library's *VFR_HUD* header was used, as it published telemetry from the aircraft and as such, a telemetry subscriber was written to read the published data. Table 4.6 lists the available data published by the *VFR_HUD* header [47][48][49].

*Table 4.6: VFR_HUD header published data*

| *VRF_HUD Data* | *Units* |
|---|---|
| Airspeed | m/s |
| Groundspeed | m/s |
| Heading | Degrees |
| Throttle | Normalized from 0.0 to 1.0 |
| Altitude | m (MSL – Mean Sea Level) |
| Climb | m/s |

The true headwind velocity, $v_{w\_diff}$, was determined by taking the difference between the aircraft's groundspeed, $v_{ground}$, and airspeed, $v_{air}$.

$$v_{w\_diff} = v_{ground} - v_{air} \tag{4.5}$$

$$v_w = \frac{\sum_{i=1}^{n} v_{w\_diff_i}}{n} \tag{4.6}$$

From Equation 4.5 and Equation 4.6, the average headwind velocity, $v_w$, was determined. The true headwind direction was determined by taking the heading just prior of the heading at which the lowest groundspeed was recorded. The headwind velocity and direction were determined in the payload release node. The generic waypoint parameters used for the survey waypoints and the multiple turn loiter waypoint parameters used for the headwind determination can be seen in Table 4.7 and Table 4.8 [47].

*Table 4.7: Generic waypoint parameters*

| *MAV_CMD_NAV_WAYPOINT* | |
|---|---|
| *Parameter* | *Description* |
| Mission Param #1 | - |
| Mission Param #2 | Waypoint acceptance radius in meters |
| Mission Param #3 | 0 to pass through the WP, if > 0 radius in metres to pass by WP. Positive value for clockwise orbit, negative value for counter-clockwise orbit. |
| Mission Param #4 | Desired yaw angle at waypoint |
| Mission Param #5 | Latitude |
| Mission Param #6 | Longitude |
| Mission Param #7 | Altitude |

*Table 4.8: Multiple turn loiter waypoint parameters*

| *MAV_CMD_NAV_LOITER_TURNS* | |
|---|---|
| *Parameter* | *Description* |
| Mission Param #1 | Number of turns |
| Mission Param #2 | - |
| Mission Param #3 | Radius around waypoint, in meters. (Positive value = clockwise, negative value = counter-clockwise) |
| Mission Param #4 | Loiter exit location (0 for centre of loiter, 1 for tangential exit location, else the desired yaw angle) |
| Mission Param #5 | Latitude |
| Mission Param #6 | Longitude |
| Mission Param #7 | Altitude |

#### 4.1.5.3 Payload Release Waypoints

Once the drop-zone's location had been successfully located by the vision system and the headwind velocity and direction had been determined, the appropriate payload release approach was defined. The process of defining the payload release approach was done by taking the drop-zone's location and defining three waypoints along the same heading as the headwind. The first waypoint was a lineup waypoint (waypoint 12) and was used to allow the aircraft to reach the payload release approach in the desired attitude. The second payload release waypoint represented the payload release approach waypoint (waypoint 13). The final payload release waypoint represented the payload release point (waypoint 14) and was defined by the payload release node to be the displacement upwind or downwind of the drop-zone to achieve the desired release trajectory. Flying directly into the headwind allowed for the headwind to be modelled as a two-dimensional entity and was discussed in Chapter 6.

The process of computing the payload release approach was done within the payload release node and the approach waypoints were published to the FCU during the third loiter loop of the environmental parameter determination phase of the mission. The payload release approach waypoints can be seen in Figure 4.9, where the direction of the increasing waypoint index number was defined to be the opposing direction of the wind.

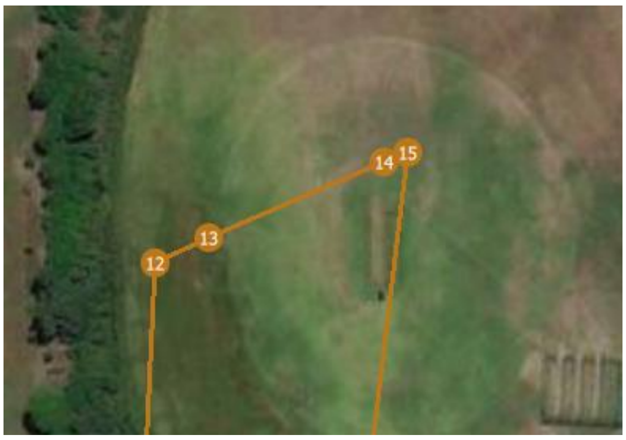


*Figure 4.9: Payload release approach waypoints, with the drop-zone location defined by waypoint 15*

### 4.1.5.4 Confirmation and Landing

Once the payload had been released, the aircraft would circle back and execute the payload release approach again, this time recording the drop-zone. The recording was done to evaluate the success of the release and as a means of payload delivery confirmation should the system be implemented in a market related application, such as goods delivery. This footage was not processed onboard the aircraft and was left to post-flight processing, should the aircraft operator wish to evaluate the footage.

To land the aircraft, the MAVROS Library's *Waypoint* and *WaypointPush* headers were used. QGroundControl offered the option of selecting a fixed-wing landing waypoint, but this was not the case when working from first principles and as such, the basic waypoint elements of the fixed-wing landing waypoint were identified and defined separately. A fixed-wing waypoint had two distinct elements, namely: a loiter to altitude and landing waypoint. Out of these two elements, the loiter to altitude defined the majority of the landing parameters, such as landing approach altitude and the heading of the aircraft during landing. The landing heading corresponded with the second element of the landing, the landing waypoint, where the landing waypoint was defined at a given distance away from the aircraft tangential to when the aircraft exited the loiter and was defined in the direction of the heading defined during the loiter to altitude waypoint. The landing waypoint indicated the location at which the aircraft was theorised to land and as such, the aircraft would execute a landing flare just prior to this location. A landing flare represented the landing stage between landing approach and touchdown where the nose of the aircraft would raise up, decreasing the rate of descent, and the aircraft attains the correct touchdown attitude. The designation of fixed-wing landing waypoints can be seen in Figure 4.10.

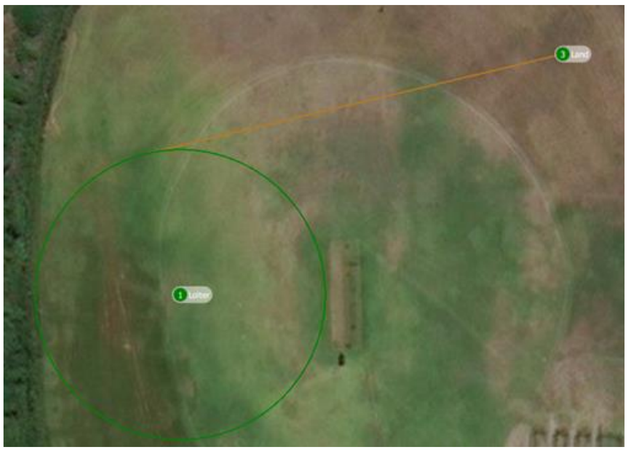


*Figure 4.10: Fixed-wing landing waypoints*

Provided the landing parameters and landing waypoints were defined correctly, the FCU would take full control over the landing. Along with the aforementioned landing waypoints, FCU parameters were also defined in terms of the aircraft's physical capabilities as seen in Table 4.9. Figure 4.11 depicts the aircraft flying overhead during landing [2][47][48][49].

*Table 4.9: FCU landing parameters*

| *Parameter* | *Value* | *Units* | *Description* |
|---|---|---|---|
| LAND_FLARE_ALT | 3 | m | Altitude at which to lock heading and flare to the LAND_PITCH_CD pitch |
| LAND_FLARE_SEC | 2 | s | Vertical time before landing point at which to lock heading and flare with the motor stopped |
| LAND_PITCH_CD | 0 | Centidegrees | The minimum pitch in the final stage of landing (after the flare) |
| TECS_LAND_ARSPD | -1 | m/s | The goal airspeed during the landing approach |
| TECS_LAND_SPDWGT | -1 | - | Weighting applied to speed control during landing |

*Figure 4.11: Fixed-wing flying overhead during landing*

### 4.1.5.5 Generic Mission Waypoint List

With the description of each waypoint type used within any mission, the list of generic mission waypoints can be seen in Table 4.10 [47].

*Table 4.10: Generic mission waypoints*

| *Waypoint Index* | *ROS Node Designation* | *MAV_CMD Type* | *Description* |
|---|---|---|---|
| 1 | wp_takeoff | MAV_CMD_NAV_TAKEOFF (#22) | Takeoff waypoint |
| 2 | wp_sl | MAV_CMD_NAV_WAYPOINT (#16) | Survey alignment waypoint |
| 3 | wp_s1 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 1 |
| 4 | wp_s2 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 2 |
| 5 | wp_s3 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 3 |
| 6 | wp_s4 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 4 |
| 7 | wp_s5 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 5 |
| 8 | wp_s6 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 6 |
| 9 | wp_s7 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 7 |
| 10 | wp_s8 | MAV_CMD_NAV_WAYPOINT (#16) | Survey waypoint 8 |
| 11 | wp_hw | MAV_CMD_NAV_LOITER_TURNS (#18) | Headwind 3 turn loiter |
| 12 | wp_dal | MAV_CMD_NAV_WAYPOINT (#16) | Drop approach alignment waypoint |
| 13 | wp_da1 | MAV_CMD_NAV_WAYPOINT (#16) | Drop approach waypoint 1 |
| 14 | wp_da2 | MAV_CMD_NAV_WAYPOINT (#16) | Drop approach waypoint 2 |
| 15 | wp_target | MAV_CMD_NAV_WAYPOINT (#16) | Target location waypoint |
| 16 | wp_landing1 | MAV_CMD_NAV_LOITER_TURNS (#18) | Landing loiter waypoint |
| 17 | wp_landing2 | MAV_CMD_NAV_WAYPOINT (#16) | Landing location waypoint |

## 4.2 Chapter Conclusion

Chapter 4 discussed the development of the autonomous flight control functionality of the aircraft. Different approaches to aircraft takeoff were assessed and the development of an aircraft launch catapult was discussed. From the different takeoff procedures assessed, the *Runway Takeoff* procedure was selected. Where, runway takeoffs, and subsequent, landings were found to be more conducive towards the desired payload release system mounting arrangement. The various types of waypoints utilised in this research were also discussed. The functionality of the autonomous flight node in defining mission waypoints was discussed, where the necessary operator input prior to execute was the GPS coordinates of the approximate drop-zone location. All other mission waypoints were defined by the FCU or through the input of the vision system. With the conclusion of the autonomous flight control comes the next integrated subsystem, the vision system. Chapter 5 discussed the details pertaining to the vision system, where object recognition and object tracking were discussed in the form of the image capture and image processing algorithms. Additionally, drop-zone global frame location determination was discussed in Chapter 5.

# Chapter 5

# Vision System

This chapter discussed the details pertaining to image capture and image processing. The vision system allowed for the detection of the drop-zone and the determination of the drop-zone location with respect to the aircraft, through implementation of the image capture and image processing algorithms.

## 5.1 Image Capture Node and Image Processing Node

A two-stage image capture and image processing algorithm was developed, where image capture was executed in a separate node, the image capture node, to the node in which image processing was executed, the image processing node. The reason for the separation of the image capture and image processing was to allow for a constant flow of images to the image processing node. Initially, a combined image capture and processing node was constructed but it was found that a large latency in image feedback was experienced. This latency in image feedback was due to the execution of the image capture being dependent on the time taken for the image processing algorithm to finish with the previous image stream. Figure 5.1 illustrates the computational flow diagram of the image capture and image processing algorithm.

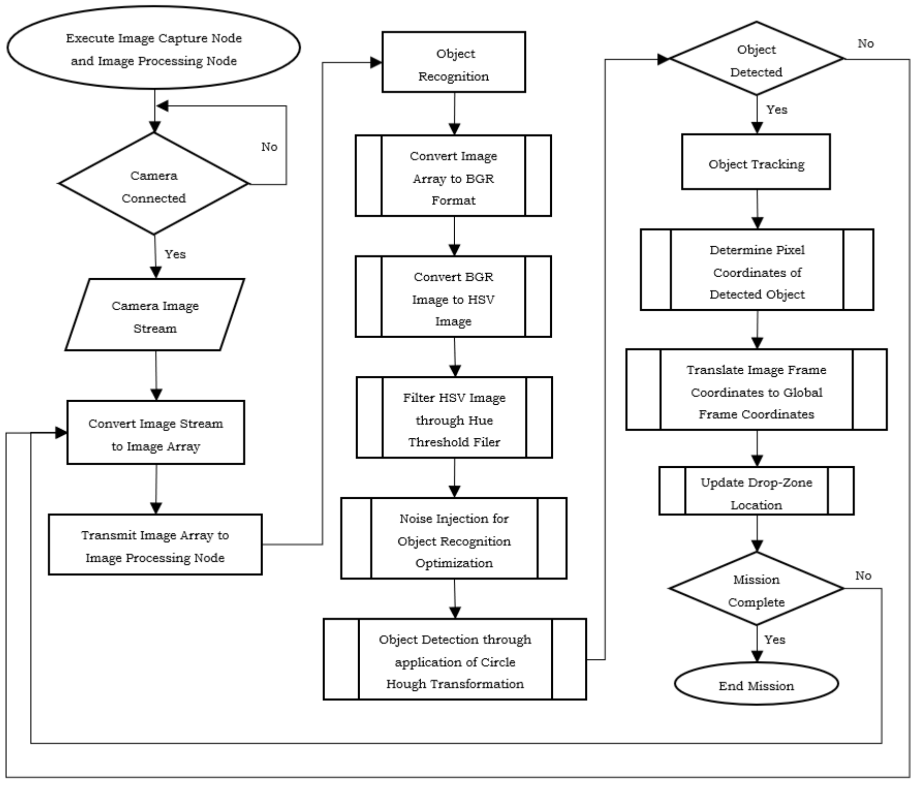

Execute Image Capture Node and Image Processing Node
Camera Connected
No
Yes
Camera Image Stream
Convert Image Stream to Image Array
Transmit Image Array to Image Processing Node
Object Recognition
Convert Image Array to BGR Format
Convert BGR Image to HSV Image
Filter HSV Image through Hue Threshold Filer
Noise Injection for Object Recognition Optimization
Object Detection through application of Circle Hough Transformation
Object Detected
No
Yes
Object Tracking
Determine Pixel Coordinates of Detected Object
Translate Image Frame Coordinates to Global Frame Coordinates
Update Drop-Zone Location
Mission Complete
No
Yes
End Mission

*Figure 5.1: Image capture node and the image processing node computational flow diagram*

The image capture node and the image processing node were derived from the ROS C++ image publisher and image subscriber examples, where the alterations made to these example nodes was the adaptation of the code to accept video stream from the onboard camera [20][34]. The refresh rate of the image stream was altered to accommodate the amount of area covered by the aircraft during flight. A slower refresh rate resulted in regions of the survey area being omitted from the image capture process due to latency.

## 5.2 Image Processing

The image processing node consisted of two distinct phases, namely: the object recognition phase and the object tracking phase. Object recognition would occur for all images transmitted from the image capture node and object tracking would only occur upon successful detection of the drop-zone identifier.

### 5.2.1 Object Recognition

Object recognition and object tracking formed part of the image processing node, where in order to be able to achieve object tracking, object recognition was required first. The object to be recognised was defined to be a red circle. Hinas et al. [27] suggested that blue was a more suited colour for object recognition, but due to the application of this system, red was selected as the testing colour due to its contrast on both land and sea.

Object recognition made use of OpenCV to execute a multistage image processing algorithm, as defined in Figure 5.1. Where, these multiple stages were: Blue-Green-Red (BGR) image formatting applied to the input image stream, converting from BGR image to Hue-Saturation-Value (HSV) image, assigning the upper and lower thresholds for image colour filtering, introduction of noise to optimise object detection, circle detection with the assistance of the Circle Hough Transform, and detected circle image overlay onto original image. The image processing algorithm was based on an existing red circle detection algorithm [40].

#### 5.2.1.1 BGR Formatting

From the image capture node, the video stream was transmitted as individual images, represented in the form of an n-dimensional array by the OpenCV *Mat* class. The captured image was represented in this array by the number of pixels present in the image and was structured according to the image's pixel rows and columns. The image array, in this case the two-dimensional image array, defined the captured image's colour in a BGR format, implying that each element of the image array was defined to be the corresponding numerical value for the BGR colour of the pixel in the image. The BGR colour scale was defined to be three numbers, one for each colour, where each colour value ranged from 0 to 255. Figure 5.2 illustrates an example of the OpenCV BGR colourspace array of dimension $n \times m$ [32].

| | Column 0 | | | Column 1 | | | Column ... | | | Column m | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row 0 | 0,0 | 0,0 | 0,0 | 0,1 | 0,1 | 0,1 | ... | ... | ... | 0, m | 0, m | 0, m |
| Row 1 | 1,0 | 1,0 | 1,0 | 1,1 | 1,1 | 1,1 | ... | ... | ... | 1, m | 1, m | 1, m |
| Row ... | ...,0 | ...,0 | ...,0 | ...,1 | ...,1 | ...,1 | ... | ... | ... | ..., m | ..., m | ..., m |
| Row n | n,0 | n,0 | n,0 | n,1 | n,1 | n,1 | n,... | n,... | n,... | n, m | n, m | n, m |

*Figure 5.2: OpenCV BGR colourspace array [32]*

### 5.2.1.2 Converting from BGR to HSV

HSV image formatting was used for image processing due to its ability to isolate objects based on colour through a single variable, hue, as opposed to the BGR image formatting where colour identification required variables to be defined for each BGR element. The difference between the HSV and BGR colourspace was their geometric representation, where the BGR colourspace was represented as a cubic geometry structure and the HSV colourspace was represented as a hexcone geometric structure, as seen in Figure 5.3. As such, the BGR image was converted to the HSV image format with use of the OpenCV *cvtColor* class.

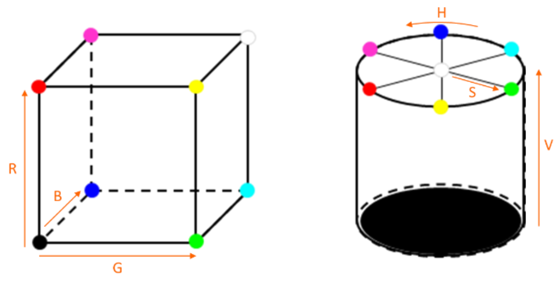


*Figure 5.3: BGR cubic colourspace (left) and HSV hexacone colourspace (right)*

### 5.2.1.3 Image Threshold Filtering

Image threshold filtering was used to define the colour range of the objects within the captured image to be processed, where this filtering was based on the hue of the HSV image. As such, the lower hue range and upper hue range of the desired object's colour were defined. These upper and lower hue ranges were defined as binary images where if an object within the given colour ranges was detected, these objects would be displayed in white in the binary image. All colours outside of the hue ranges would be displayed as black in the binary images, as seen in Figure 5.5. Both the lower and upper hue range binary images were combined into a single output binary image. The filtering of the HSV images to binary images was done with the use of the OpenCV *inRange* function and the OpenCV *Scalar* class. The final binary image was used in the object detection algorithm by defining the objects meeting the desired colouring criteria. Objects not meeting the colouring criteria were then disregarded for the remainder of object recognition algorithm.

#### 5.2.1.4 Noise Injection

Noise injection was used to reduce the object recognition algorithm's susceptibility to false object detection. Prior to noise injection, the binary threshold image was smoothed with the use of the OpenCV *medianBlur* function. Noise injection was executed with the assistance of the OpenCV *GaussianBlur* function, where a Gaussian filter was applied to the captured image, distorting the shape geometry. With the application of the Gaussian filter, the object recognition algorithm could be refined to filter both simulated and capture image interference and noise [33].

#### 5.2.1.5 Hough Circle Transformation

With the application of the threshold and Gaussian filters, the final process of the object recognition algorithm could take place, the detection of circular objects. Detection of circular objects was done with the use of the OpenCV *HoughCircles* function. The OpenCV *HoughCircles* function makes use of the Circle Hough Transform (CHT) via an adaptation of the Hough Gradient method.

The standard CHT made use of a three-dimensional accumulator for circle detection within a greyscale image where the radius of the circle was unknown. This three-dimensional accumulator stored the votes of the edge points of the circle, $x$ and $y$, and the radius of the circle, $r$. These votes represented the numerical value assigned to each pixel's suitability at representing acceptable circle geometry, determined by the pixel's colour within the greyscale, in this case binary, image. Each pixel's vote was determined with the use of the mathematical expression defined in Equation 5.1.

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{5.1}$$

Where, $x_c$ and $y_c$ represented the location of the pixel coordinated used to satisfy Equation 5.1 for each edge coordinate and radius stored within the accumulator. Due to the computational strain of the standard Circle Hough Transformation method of circle detection, OpenCV's adaptation of the Hough Gradient method for the *HoughCircles* function was implemented. The Hough Gradient method monitors the change in pixel colour gradient, reducing the number of votes required to deduce the location of the centre of a possible circle. Determination of the circle's centre coordinates would allow for the image processing algorithm to proceed with the next phase, object tracking [11][31]. The OpenCV *HoughCircle*s function and its parameter discretion can be seen in Equation 5.2 and Table 5.1.

$$HoughCircles(Image, Circles, Method, dp, minDist, param1, param2, minRadius, maxRadius\ ) \quad (5.2)$$

*Table 5.1: OpenCV HoughCircles function parameters*

| *Parameter* | *Variable Type* | *Description* |
|---|---|---|
| Image | InputArray | 8-bit, single-channel, grayscale input image. |
| Circles | OutputArray | Output vector of found circles. |
| Method | int | Detection method to use. |
| dp | double | Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height. |
| minDist | double | Minimum distance between the centres of the detected circles. If the parameter is too small, multiple neighbour circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed. |
| param1 | double | First method-specific parameter. In case of *CV_HOUGH_GRADIENT*, it is the higher threshold of the two passed to the *Canny()* edge detector. |
| param2 | double | Second method-specific parameter. In case of *CV_HOUGH_GRADIENT*, it is the accumulator threshold for the circle centres at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first. |
| minRadius | int | Minimum circle radius |
| maxRadius | int | Maximum circle radius |

### 5.2.2 Object Tracking

Upon successful detection of the drop-zone identifier, the image processing algorithm would proceed with object tracking phase. Objecting tracking involved the transformation of the object's image plane coordinates, pixel coordinates, to global plane coordinates, relative GPS coordinates.

#### 5.2.2.1 Determination of the Object's Pixel Coordinates

The image plane represented the plane upon which object tracking and object recognition was executed. The image plane was divided into four quadrants, in which the location of the drop-zone identifier was determined relative to the global frame, as seen in Figure 5.4. The reason for the image plane zoning was to define location of the drop-zone relative to the centre of the image plane, as the centre of the image plane did not represent the origin of the image plane but represented the $x$ and $y$ origin of the camera frame. The origin of the image plane, also defined to be the image frame, was assigned by OpenCV to be the upper left-hand corner of the visible image. Relating the image frame to camera frame was necessary due to the camera frame and global frame sharing the same $x$ and $y$ axes. The only dimensional difference between the camera frame and global frame was a $z$-axis offset, $Z_{CG}$, due to the mounting arrangement of the vision system relative to the GPS module.

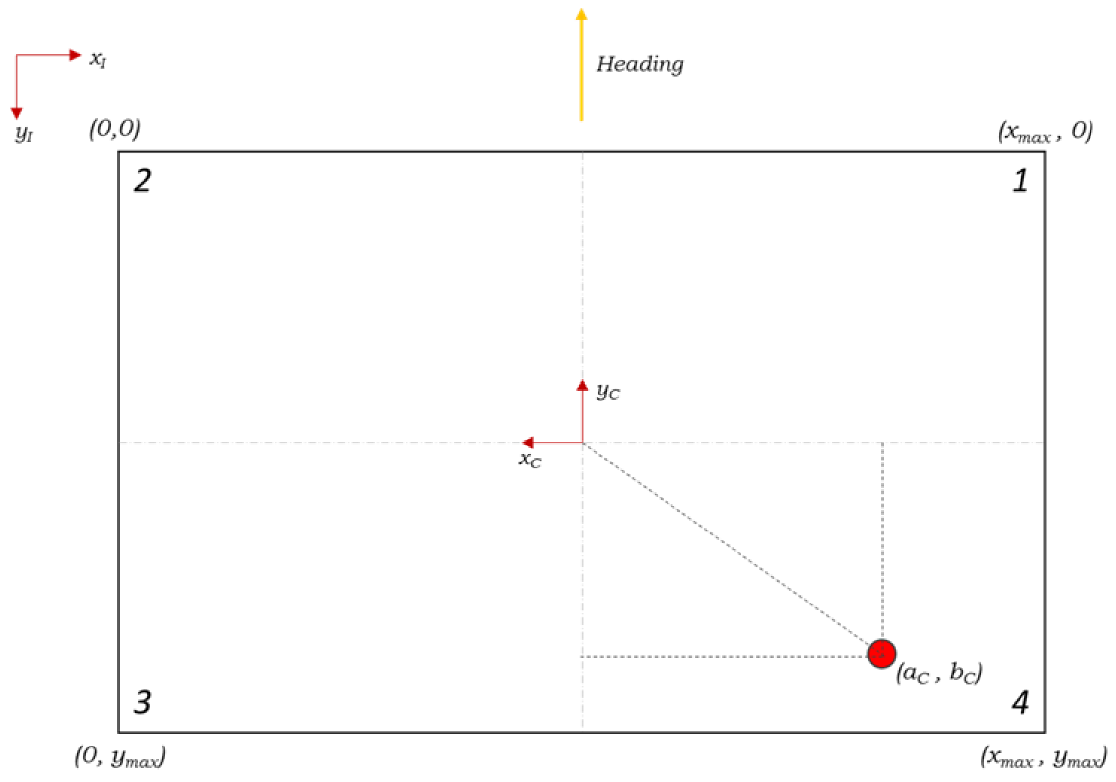


*Figure 5.4: Image plane quadrant view*

The designation of the image plane quadrant ranges were as follows:

$$1.\quad \frac{x_{max}}{2} < x \leq x_{max}, \text{ and } 0 \leq y \leq \frac{y_{max}}{2} \tag{5.3}$$

$$2.\quad 0 \leq x \leq \frac{x_{max}}{2}, \text{ and } 0 \leq y \leq \frac{y_{max}}{2} \tag{5.4}$$

$$3.\quad 0 \leq x \leq \frac{x_{max}}{2}, \text{ and } \frac{y_{max}}{2} < y \leq y_{max} \tag{5.5}$$

$$4.\quad \frac{x_{max}}{2} < x \leq x_{max}, \text{ and } \frac{y_{max}}{2} < y \leq y_{max} \tag{5.6}$$

From Figure 5.4, the drop-zone's pixel coordinates could be defined. These pixel coordinates were used to determine the drop-zone's global frame location by overlaying the pixel coordinates onto the ground plane. The image plane pixels were defined relative to the image frame, but the drop-zone's pixel coordinates were defined relative to the camera frame for convenience of conversion to global frame coordinates. An illustration of the various coordinate frames can be seen in Figure 5.5.

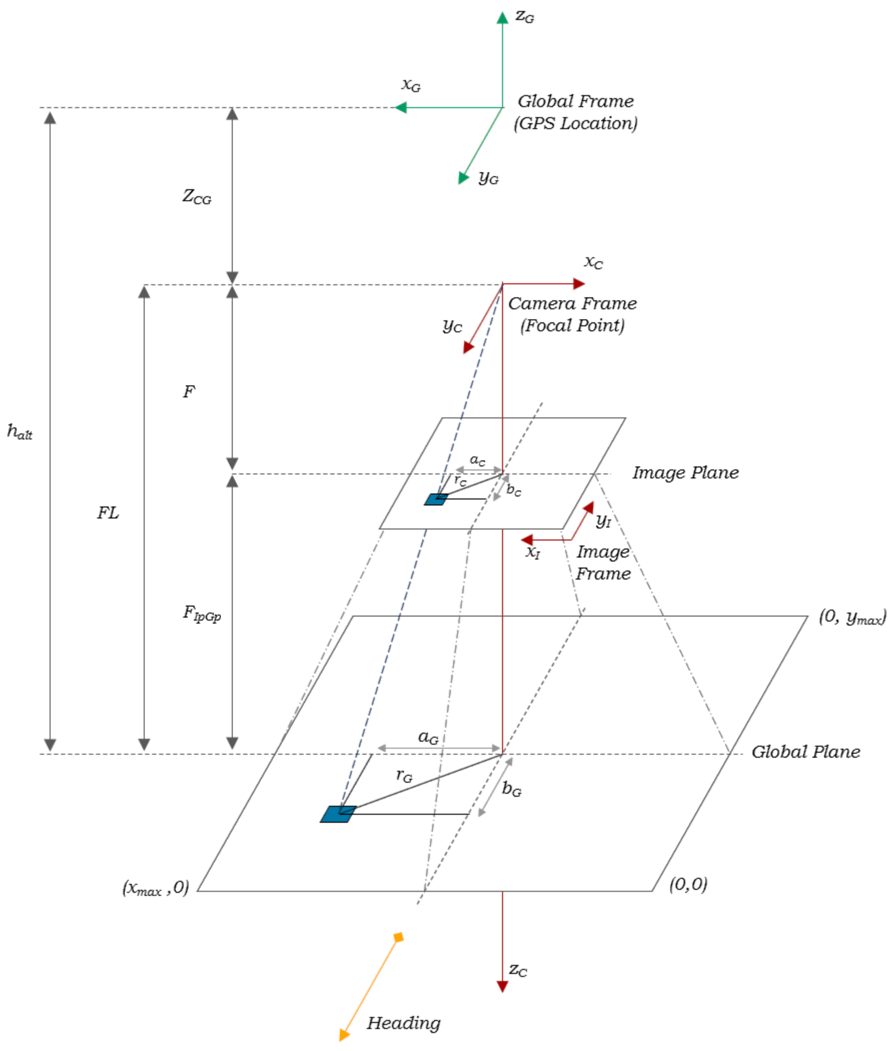


*Figure 5.5: Relative coordinate frames*

As seen in Figure 5.4, four quadrants were defined within the image plane. The allocation of quadrants was to allow for ease of drop-zone location determination within the image plane. These four quadrants allowed for the bearing, $\gamma$, of the drop-zone's location relative to the aircraft's heading to be determined, as seen in Figure 5.6. This drop-zone bearing was taken to be a new heading for the aircraft, as heading was required for the computation of the global coordinates of the drop-zone. Each quadrant was defined with a rule for determining the bearing heading of the drop-zone relative to the aircraft's heading, where the aircraft's heading was defined to be the direction of the camera's $y$ axis. These bearing heading rules for each quadrant were as follows:

1. $$\gamma_{1\_heading} = heading - \tan^{-1}\left(\frac{a_c}{b_c}\right) \tag{5.7}$$
2. $$\gamma_{2\_heading} = heading + \tan^{-1}\left(\frac{a_c}{b_c}\right) \tag{5.8}$$
3. $$\gamma_{3\_heading} = heading + \left(180 - \tan^{-1}\left(\frac{a_c}{b_c}\right)\right) \tag{5.9}$$
4. $$\gamma_{4\_heading} = heading + \left(180 + \tan^{-1}\left(\frac{a_c}{b_c}\right)\right) \tag{5.10}$$

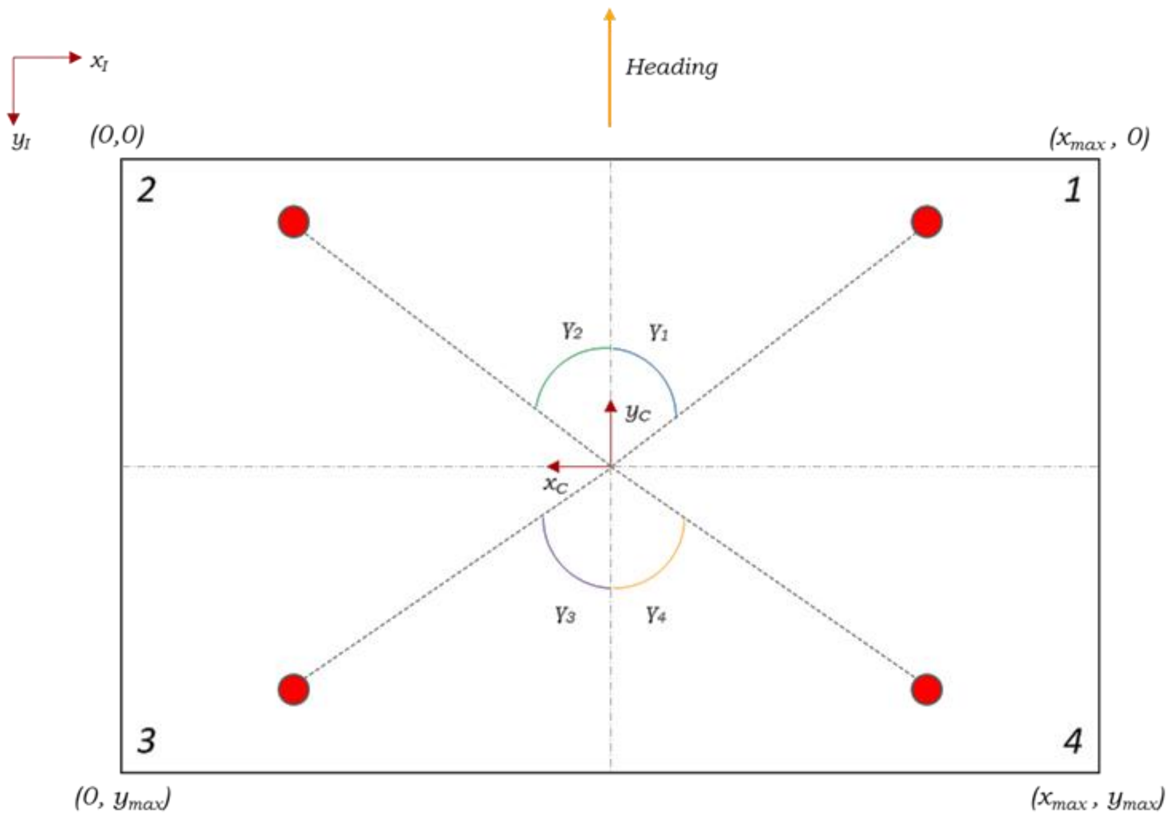


*Figure 5.6: Image plane quadrant bearings*

### 5.2.2.2 Conversion from Image Frame to Global Frame

To determine the relationship the between camera's image frame and the global frame, the true FOV of the camera was to be determined. Determining the FOV of the camera was done with the use of two calibration tests, one test at close range and another at far range. Both calibration tests involved placing the camera at a known distance from a fixed vertical plane, *FL*, and then measuring the maximum visible image area of the plane, ($x_{max}$, $y_{max}$). Determining this maximum visible area allowed for the calculation of the camera's FOV. An illustration of the calibration tests can be seen in Figure 5.7.

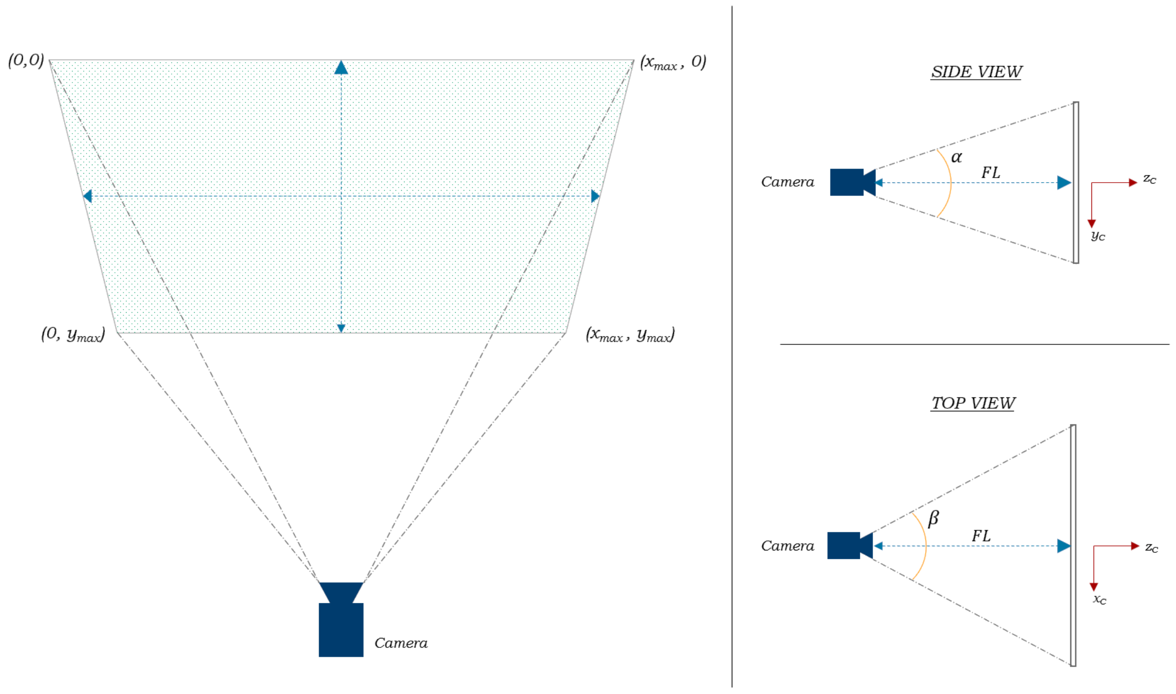


*Figure 5.7: Camera FOV calibration illustration*

From Figure 5.7, the FOV of the camera was defined to be two angles, $\alpha$ and $\beta$. Equation 5.11 and Equation 5.12 define how these angles were deduced.

$$\alpha = 2\tan^{-1}\left(\frac{\left(\frac{y_{max}}{2}\right)}{FL}\right) \tag{5.11}$$

$$\beta = 2\tan^{-1}\left(\frac{\left(\frac{x_{max}}{2}\right)}{FL}\right) \tag{5.12}$$

Where, $\alpha$ represented the vertical FOV angle of the camera and $\beta$ represented the horizontal FOV angle of the camera. Knowing these angles, the conversion from pixel coordinates to GPS coordinates was now possible.

### 5.2.2.3 Defining the GPS Coordinates of the Drop-Zone

From the true FOV angles of the camera, the pixel coordinates of the drop-zone and the bearing of the drop-zone relative to the camera frame, the relationship between the image plane and the ground plane could be defined. This relationship allowed for the GPS coordinates of the drop-zone relative to the aircraft to be defined. Based on the FOV determined for the camera and knowing the altitude of the aircraft and the camera frame to global frame $z$ axis offset, $Z_{CG}$, the camera's visible image area on the ground plane could be defined. The drop-zone's pixel coordinates could be superimposed upon the camera's ground plane visible area and the drop-zone's global frame offsets could be determined. These offsets, $a_G$ and $b_G$, along with the bearing of the drop-zone relative to the camera frame, were then used to determine the GPS location of the drop-zone, $(Lat_T, Lon_T)$ with the use of Equation 5.14 and Equation 5.15. With the assistance of Equation 5.13 and offsets $a_G$ and $b_G$, the magnitude of the distance vector between the global frame origin and the drop-zone location could be determined [44].

$$r_G = \sqrt{a_G^2 + b_G^2} \tag{5.13}$$

$$Lat_T = \sin^{-1}(\sin(Lat_A)\cos\left(\frac{r_G}{R_E}\right) + \cos(Lat_A)\sin\left(\frac{r_G}{R_{Earth}}\right)\cos(\gamma)) \tag{5.14}$$

$$Lon_T = Lon_A + arctan2\left(\sin(\gamma)\sin\left(\frac{r_G}{R_E}\right)\cos(Lat_A), \cos\left(\frac{r_G}{R_E}\right) - \sin(Lat_A)\ \sin(Lat_T)\right) \tag{5.15}$$

Equation 5.14 and Equation 5.15 made use of three characteristics to determine the drop-zone's GPS coordinates, namely: the GPS location of the aircraft upon drop-zone identification $(Lat_A, Lon_A)$, the bearing heading of the drop-zone relative to the aircraft, $\gamma$, and the magnitude of the distance vector between the aircraft's GPS coordinates and the drop-zone's location, $r_G$.

With the GPS coordinates of the drop-zone relative to the aircraft now known, the previously defined approximate drop-zone location could be updated. The image capture node and image processing node code can be found in Appendix A2 and Appendix A3.

## 5.3 Camera Mass-Balancing

As discussed in Chapter 3, the image stabilisation gimbal utilised in this research was designed for a GoPro HERO 5 BLACK camera. As such, the CMOS camera required a mass-balancing adapter to be design in order for the gimbal to function correctly. In order to mass the CMOS camera, the centre of mass of a GoPro HERO 5 BLACK camera was to be determined. The specifications of the GoPro can be seen in Table 5.2 [21].

*Table 5.2: GoPro HERO 5 physical specifications*

| *Parameter* | *Value* | *Unit* |
|---|---|---|
| Mass | 0.12 | kg |
| Length | 62 | mm |
| Height | 44 | mm |
| Thickness | 24 | mm |
| Location of centre of mass (From front bottom left corner) | x: 32<br>y: 24<br>z: 10 | mm<br>mm<br>mm |

From Table 5.2, the CMOS camera holder was designed to fit the physical dimensions of a GoPro. As the camera mount was to be 3D printed, the mount's weight did not match that of the GoPro and a compartment was included within the camera mount's design to house additional weights to balance the camera. The final CMOS camera mount can be seen in Figure 5.8 and the design of the camera mount can be seen in Appendix I.

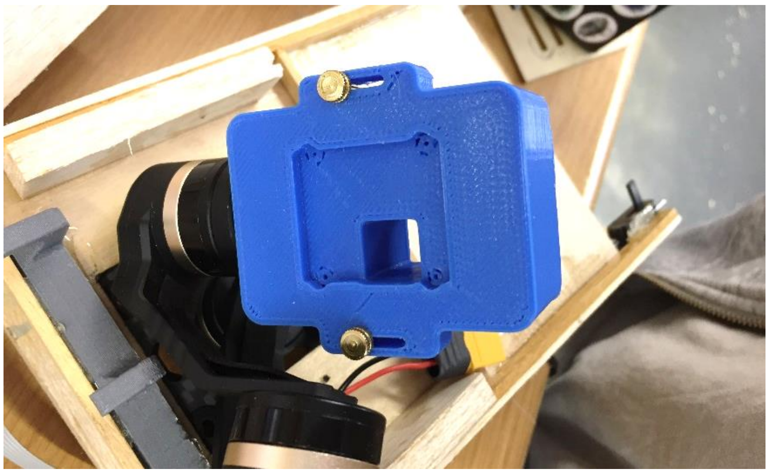


*Figure 5.8: CMOS camera mount attached to the gimbal*

## 5.4 Chapter Conclusion

From Chapter 5, the foundation of the image capture and image processing node algorithms was discussed. The process through which captured images were converted between colourspaces was discussed, where it was found that application of the HSV colourspace was conducive to image filter and object detection. Additionally, it was found that the image processing algorithm possessed three phases for the determination of the drop-zone identifier's location. These phases included: image hue filtering for the removal of unwanted colours from the search process, application of the OpenCV *HoughCicrles* function for detection of the drop-zone identifier's circle geometry and transformation of the drop-zone identifier's pixel coordinates from image frame to global frame coordinates. A camera mount was designed and 3D printed to match the physical properties of a GoPro HERO 5 BLACK camera, the intended camera for the TAROT gimbal. Chapter 6 discussed the final computational subsystem of integrated system, the payload release system. Elements of payload release modelling and payload design were discussed in Chapter 6.

# Chapter 6

# Payload Delivery

This chapter discussed the various environmental and structural factors influencing the payload during its ballistic projectile motion, along with the mathematical foundation upon which the payload trajectory was based. The design and modelling of the payload was also discussed, along with the structure of the payload release node.

## 6.1 Environmental Factors

This research considered the real-time environmental parameters which would affect the trajectory of the payload during the drop, where these parameters included the headwind velocity and direction experienced during flight. The consideration of these environmental parameters was based on the flight path of the aircraft during its drop approach, which would see the aircraft flying directly into the headwind. As such, the headwind and subsequent projectile motion could be modelled in two dimensions.

Due to the unpredictability of wind gusts, the assumed influence of the wind on the payload was perceived to follow the wind gradient, also known as wind shear, as defined by Equation 6.1. Equation 6.1 was used to extrapolate the wind horizontal velocity with respect to the elevation [14].

$$v_w(h) = v_{w_{alt}} \left(\frac{h}{h_{alt}}\right)^{\frac{1}{\alpha}} \tag{6.1}$$

Where, $v_w(h)$ represented the wind velocity at elevation $h$, $v_{w_{alt}}$ the wind velocity at the drop elevation, $h_{alt}$ the elevation at the moment of release and $\alpha$ the exponential wind coefficient. This equation holds only when $0 < h < h_{alt}$. Equation 6.1 represented a generalisation of the predicted wind velocity at various heights of a building, as the desired testing altitudes of this system fell within the same height in which these structures were designed to withstand wind loading. Literature on wind gradient effects on aircraft were found to only describe the process in which wind shear occurs and methods of avoidance [45].

## 6.2 Mathematical Foundation

This subsection discussed the mathematical foundation upon which the ballistic projectile motion of the payload was derived, where ballistic motion has been defined as the falling motion of an object due only to the force of gravity [42].

### 6.2.1 Ballistic Projectile Motion in a Vacuum

Two-dimensional projectile motion was expressed through the equations of motion. These equations of motion describe the projectile motion of an object within a vacuum. As defined by Hall [22] and Jewett and Serway [30], the equations of motion used for this research were as follows:

$$v_f = v_i + gt \tag{6.2}$$

$$v_f{}^2 = v_i{}^2 + 2g\Delta x \tag{6.3}$$

$$\Delta h = v_i t + \frac{1}{2}gt^2 \tag{6.4}$$

$$\Delta h = \left(\frac{v_f + v_i}{2}\right)t \tag{6.5}$$

Where, $v_i$ and $v_f$ described the initial and final velocity of the payload, $g$ the gravitational acceleration, $\Delta x$ the displacement of the payload and $t$ the time taken. Projectile motion within a vacuum follows a parabolic flight path, which implies that trajectory has symmetry about its axis of highest elevation, as seen in Figure 6.1. In Figure 6.1 it can be seen that, an initial launch angle of 45 degrees (red line) results in the largest projectile displacement for projectile motion within a vacuum, whereas angles greater and less than 45 degrees result in smaller displacements.

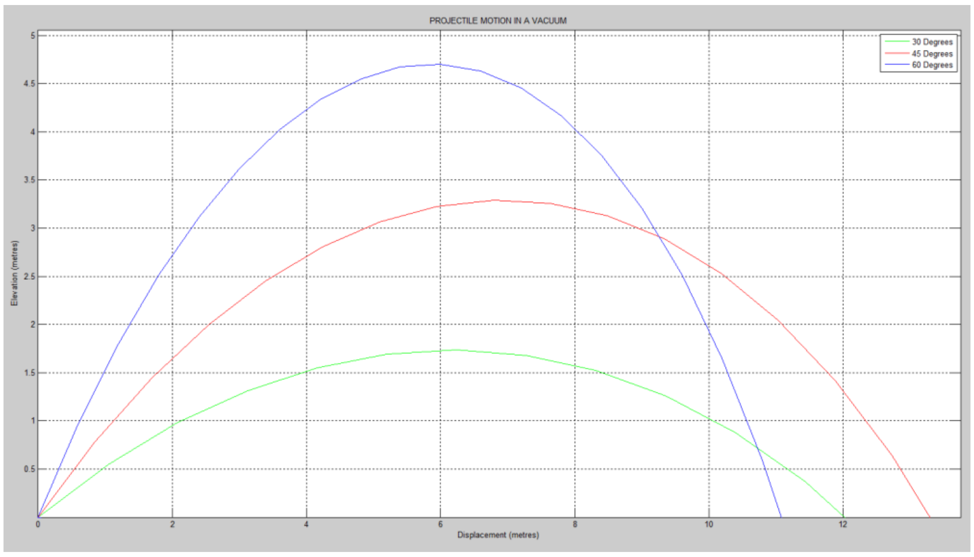


*Figure 6.1: Projectile motion of an object in a vacuum under various test parameters*

## 6.2.2 Ballistic Projectile Motion with Drag

The equations of motion mentioned previously were not used as the final product of the ballistic projectile motion due to lack of consideration of drag which the projectile would experience when traveling through air. As such, drag, specifically quadratic drag, was taken into account when determining the ballistic projectile motion of the payload.

$$F_D = \frac{1}{2}\rho A C_d v^2 \tag{6.6}$$

Where, $F_D$ represented the drag force, $\rho$ the density of the medium the payload moved through, $A$ the cross-sectional area of the payload, $C_d$ the drag coefficient and $v$ the velocity of the payload relative to the fluid. Equation 6.6 formed one of the final forces required to develop the projectile motion of the payload for this research [8].

Unlike projectile motion within a vacuum, projectile motion with drag did not follow a parabolic trajectory path. Projectile motion with drag had two distinct phases, namely vertical ascent, Phase 1, and vertical descent, Phase 2, as seen in Figure 6.2. Where, unlike the horizontal displacement of projectile motion in a vacuum, the horizontal displacement of projectile motion with drag was not symmetrical between the two phases.

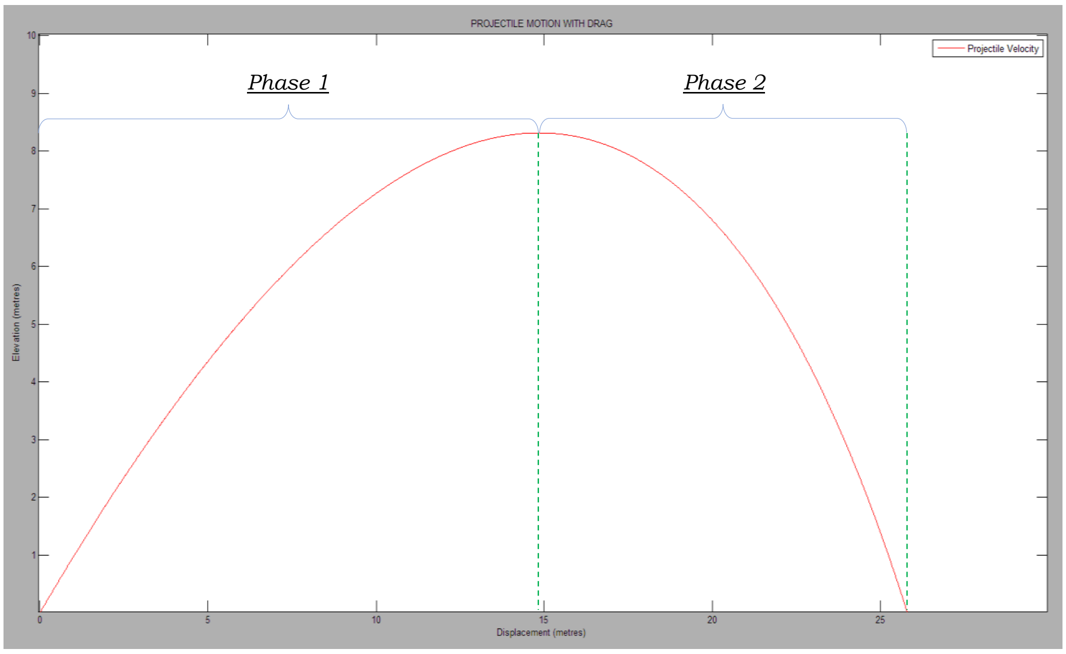


*Figure 6.2: Projectile motion of an object with drag under test parameters*

For the purpose of this research, only vertical descent, Phase 2, was considered as the aircraft was assumed to be in level flight and at altitude at the moment of release. As such, the derivation of the vertical descent equations of motion were as shown in Equation 6.7:

$$v_x(t) = \frac{{V_T}^2 v_{xo}}{{V_T}^2 + g v_{xo} t} \tag{6.7}$$

Where, $v_x(t)$ described the payload's horizonal velocity component with respect to time, $v_{xo}$ the payload's initial horizontal velocity component, and $V_T$ the terminal velocity [23]. The terminal velocity of an object describes the maximum velocity said object could achieve during free-fall and was defined as shown in Equation 6.8:

$$V_T = \sqrt{\frac{2m_p g}{C_d A \rho}} \tag{6.8}$$

Where $m_p$ defined the mass of the payload.

The second equation of motion described the horizontal displacement of the payload, $x(t)$. This horizontal displacement was with respect to the initial point of release and as such, this initial point represented zero displacement.

$$x(t) = \frac{V_T^2}{g} \ln\left(\frac{V_T^2 + g v_{xo} t}{V_T^2}\right) \tag{6.9}$$

The third equation of motion described the payload's vertical velocity component with respect to time, $v_y(t)$, as seen in Equation 6.10a [23]:

$$v_y(t) = V_T \frac{v_{yo} - V_T \tan\left(\frac{gt}{V_T}\right)}{V_T + v_{yo} \tan\left(\frac{gt}{V_T}\right)} \tag{6.10a}$$

Since the aircraft was assumed to be in level flight at the moment of release, the payload's initial velocity vertical component, $v_{yo}$, would be zero and therefore, Equation 6.10a could be redefined as Equation 6.10b [23]:

$$v_y(t) = -V_T \tan\left(\frac{gt}{V_T}\right) \tag{6.10b}$$

The fourth equation of motion described the elevation of the projected payload with respect to time, $h(t)$, as seen in Equation 6.11 [23]. This elevation was with respect to the ground and as such, the ground represented a zero elevation.

$$h(t) = \frac{V_T^2}{2g} \ln\left(\frac{V_T^2}{v_y(t)^2 + V_T^2}\right) \tag{6.11a}$$

With these equations of motion, the velocity and position of the payload could be determined for any time throughout the drop. A comparison of the projectile motion of a payload in a vacuum and in the presence of drag can be seen in Figure 6.3. In Figure 6.3 it was noted that, projectile motion within a vacuum (green) results in a greater horizontal displacement when compared to the horizontal displacement of the projectile motion with drag (red).

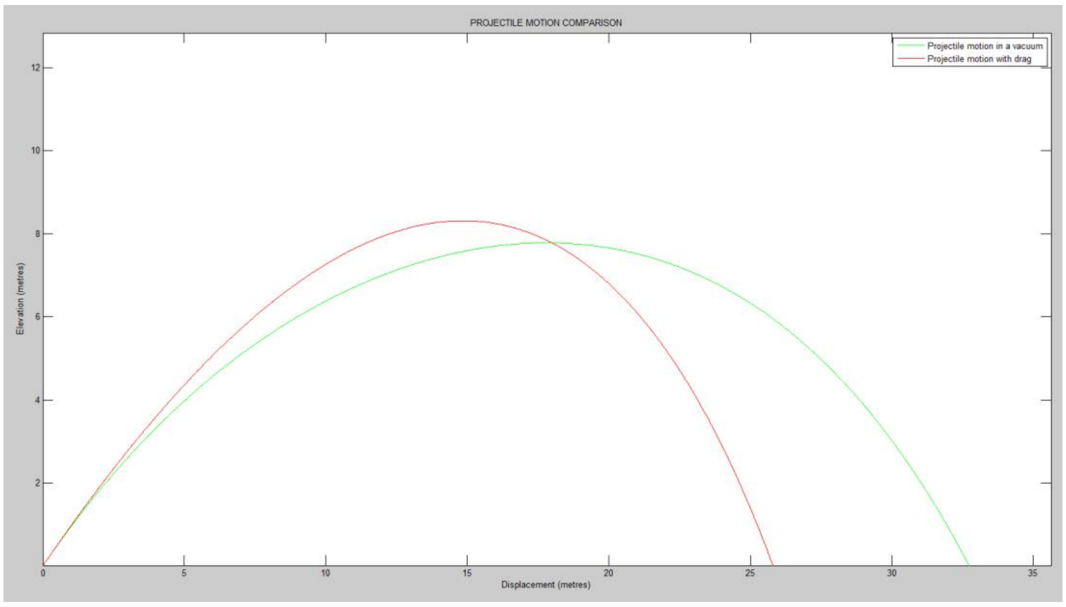


*Figure 6.3: Comparison of projectile motion in a vacuum and with drag under test parameters*

### 6.2.3 Ballistic Projectile Motion with Quadratic Drag and Headwind

Using the information gathered from the fundamentals of the environmental factors and the ballistic projectile motion to be experienced during the payload delivery, the final equations were defined. These equations describe the horizontal displacement of the payload during the ballistic projectile motion.

As the aircraft was moving through the air, with the payload still attached, the payload was yet to reach a state of horizontal equilibrium within the fluid, air. Where, in this case the term state of horizontal equilibrium implied a steady state within the air where the effects of the aircraft's thrust were no longer imparted on the aircraft. This state of horizontal equilibrium was hereafter referred to as 'the state of equilibrium'. Within this state, the horizontal displacement of the projectile would be determined through the use of velocity vector addition. This state of imbalance between the payload and the fluid was due to the aircraft imparting thrust upon the payload at the moment of release. As such, the projectile motion of the payload was defined by two distinct phases, namely Phase A and Phase B. Phase A represented the period in which the payload reached a state of equilibrium within the fluid, such that the payload's motion was no longer defined by the aircraft's imparted thrust. Phase B represented the time taken for the payload to reach the desired drop location through velocity vector addition. The payload would not always enter Phase B. The reasoning behind this payload phase dependency was discussed later in this subsection.

#### 6.2.3.1 Phase A of the Final Ballistic Projectile Motion

Phase A of the payload's projectile motion was defined to be the period in which the payload reached a state of equilibrium within the fluid, such that its motion was no longer defined by the aircraft's imparted thrust. The period of this equalization was defined to begin at the moment of release and was defined to end at the moment when the payload's horizontal acceleration, in this case deceleration, would equal zero. The rationale for this terminating state for Phase A was based on the payload decelerating horizontally due to drag and headwind, from the moment of release. The moment when the deceleration of the payload reached zero was the point at which the horizontally imparted thrust from the aircraft would have been cancelled out and the payload would be accelerated by the wind.

The desired result of Phase A was to solve for the time at which the payload enters equilibrium ($t_{Eq}$) and the change in elevation and horizontal displacement of the payload ($\Delta h$ and $\Delta x$) with respect to the initial launch position ($h_{alt}$ and $x_0$). These variables could be solved with the use of Equation 6.9 and Equation 6.11, once the equilibrium time had been deduced.

To solve for the equilibrium time constant, the projectile was modelled as a second-order system, as seen in Figure 6.4. Several conditions were assumed throughout the derivation of the equilibrium time constant and were defined accordingly.

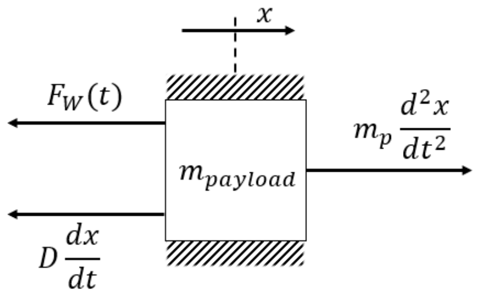


*Figure 6.4: Free-body diagram of the payload during phase 1 of the final ballistic projectile motion*

From the free-body diagram shown in Figure 6.4, the second-order differential equation could be defined. This equation described the acceleration of the payload, due to the imparted thrust from the aircraft, under the decelerating effects of both drag and headwind. Equation 6.12 and Equation 6.13 described this second-order differential equation.

$$F_W(t) = m_p \frac{d^2x}{dt^2} - D\frac{dx}{dt} \tag{6.12}$$

$$\therefore \frac{d^2x}{dt^2} = \frac{F_W(t)}{m_p} + \frac{D}{m_p}\frac{dx}{dt} \tag{6.13}$$

Where,

$$D\frac{dx}{dt} = F_D(t) \tag{6.14}$$

Thus,

$$D(t) = \frac{1}{2}\rho A C_d v_p(t) \tag{6.15}$$

Several variables of Equation 6.13 could be defined by Equation 6.15, a redefinition of Equation 6.7. The imparted force of the headwind onto the payload was modelled with use of Equation 6.15, such that:

$$F_W(t) = \frac{1}{2}\rho A C_d v_w(t)^2 \tag{6.16a}$$

Although Equation 6.1 described the headwind, $v_w(h)$, it was defined in terms of elevation, which would not suit the headwind velocity with respect to time, $v_w(t)$, required by Equation 6.15. As such, Equation 6.1 was redefined to fulfil this criterion.

Through substitution of Equation 6.10b into Equation 6.11a, Equation 6.11a could be expanded to its full variable equivalency:

$$h(t) = \frac{V_T^2}{2g} \ln\left(\frac{V_T^2}{\left(-V_T \tan\left(\frac{gt}{V_T}\right)\right)^2 + V_T^2}\right) \tag{6.11b}$$

The final equation for headwind velocity could be defined by substituting Equation 6.11b into Equation 6.1:

$$v_w(t) = v_{w_{alt}}\left(\frac{\frac{V_T^2}{2g} ln\left(\frac{V_T^2}{\left(-V_T\, tan\left(\frac{gt}{V_T}\right)\right)^2 + V_T^2}\right)}{h_{alt}}\right)^{\frac{1}{a}} \tag{6.17}$$

Substituting Equation 6.17 into Equation 6.16a yielded the equation for headwind force with respect to time, Equation 6.16b.

$$F_W(t) = \frac{1}{2}\rho A C_d \left(v_{w_{alt}}\left(\frac{\frac{V_T^2}{2g} \ln\left(\frac{V_T^2}{\left(-V_T \tan\left(\frac{gt}{V_T}\right)\right)^2 + V_T^2}\right)}{h_{alt}}\right)^{\frac{1}{a}}\right)^2 \tag{6.16b}$$

Expanding Equation 6.13:

$$\frac{d^2x}{dt^2} = \frac{\frac{1}{2}\rho A C_d v_w(t)^2}{m_p} + \frac{\frac{1}{2}\rho A C_d v_p(t)}{m_p}\frac{dx}{dt} \tag{6.18}$$

Where, $\frac{dx}{dt} = v_p(t)$ and as such, Equation 6.13 was redefined as follows:

$$\therefore \frac{d^2x}{dt^2} = \frac{\rho A C_d}{2m_p}\left(\left(v_{w_{alt}}\left(\frac{\frac{V_T^2}{2g} \ln\left(\frac{V_T^2}{\left(-V_T \tan\left(\frac{gt}{V_T}\right)\right)^2 + V_T^2}\right)}{h_{alt}}\right)^{\frac{1}{a}}\right)^2 + \left(\frac{V_T^2 v_{xo}}{V_T^2 + g v_{xo} t}\right)^2\right) \tag{6.19}$$

Equation 6.19 was used to solve for the time at which the payload would reach equilibrium. To compute Equation 6.19 onboard the aircraft proved costly in terms of computation time. To mitigate this computational issue, lookup tables were developed with predetermined equilibrium times for various headwind velocities. These lookup tables were developed with several parameters predefined, including drop altitude, airspeed and payload mass. Using these lookup tables, a polynomial fit of the graph trend was developed to allow for extrapolation of data and ease of onboard computation. Figure 6.5 depicts an example of the acceleration curve of the payload used in the development of the lookup tables, where the time at which the payload reached zero acceleration has been indicated. Interpolation and linearization of the lookup tables trends were used to determine equilibrium data between tabled values. The MATLAB code used to determine these equilibrium times and displacement can be seen in Appendix B. Noting the structure of Equation 6.19, the payload's mass was significant in terms of it influence on the rate of deceleration experienced by the payload. The payload's mass was found to be inversely proportional to the payload's deceleration and as such, a heavier payload would have a smaller rate of deceleration when compared to lighter payloads and would therefore take longer to decelerate.

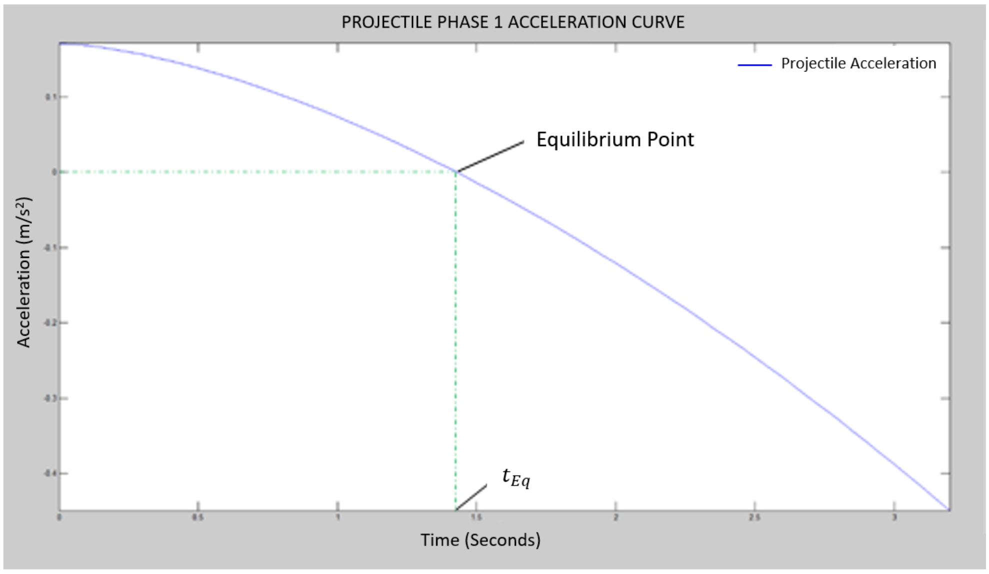


*Figure 6.5: Payload acceleration curve under test parameters*

With the equilibrium time and the acceleration curve of the payload known, the displacement of the payload could be determined with the use of an adapted Equation 6.4.

$$\Delta x_A = v_i t + \frac{1}{2}\left(\int_{t_o}^{t_{Eq}} a \, dt\right) t \tag{6.20}$$

Where, $a = \frac{d^2x}{dt^2}$. Considering that the acceleration integral would generate a negative value due to the payload's deceleration, the full effects of drag and headwind would be compensated for in Equation 6.20. Thus, from Equation 6.20, the Phase A payload displacement could be determined.

### 6.2.3.2 Phase B of the Final Ballistic Projectile Motion

Phase B of the projectile motion assumed that the payload was in a state of equilibrium in the air and the resulting motion of the projectile could be defined by vector addition of the projectile's velocity and the headwind's velocity.

Prior to this vector addition, it was determined as to whether the payload would have entered the state of equilibrium due to the available drop time from the given altitude, $t_{fall}$. Determining this drop time was done with the use of Equation 6.21. Assuming the aircraft was in level flight, Equation 6.3 could be rearranged to form Equation 6.21. Rearranging Equation 6.3 to form Equation 6.21 was possible due to the assumption that all external forces exerted on the payload were horizontal forces and the only vertical force acting upon the payload was gravity.

$$t_{fall} = \sqrt{\frac{2h_{alt}}{g}} \tag{6.21}$$

If this fall time was found to be greater than the equilibrium time, the payload would enter Phase B of the final projectile motion, as the payload would be in equilibrium prior to reaching the ground. If the fall time was found to be less than the equilibrium time, the payload would not enter Phase B and would only be modelled with the equations defined in Section 6.3.3.1 in terms of horizontal displacement.

With the fall time and the equilibrium time constants known, the cumulative horizontal displacement of the payload could be determined. Furthermore, the subsequent final projectile displacement can be determined with the use of Equation 6.22, through substitution of Equation 6.17 and Equation 6.20.

$$\Delta x_T = \Delta x_A + \int_{t_{Eq}}^{t_{fall}} v_{w_{alt}} \left( \frac{\frac{V_T^2}{2g} ln\left( \frac{V_T^2}{\left(-V_T\, tan\left(\frac{gt}{V_T}\right)\right)^2 + V_T^2} \right)}{h_{alt}} \right)^{\frac{1}{a}} dt \tag{6.22}$$

With Equation 6.22, the final displacement of the payload from its release point could be determined and as such, the ideal release point could be defined relative to the desired drop-zone location. Equation 6.22 would hold if the payload were to enter Phase B, otherwise the final displacement of the payload would be defined by Equation 6.20.

## 6.3 Payload Design

With the mathematical foundations for the payload trajectory defined, the design of the payload was conducted.

### 6.3.1 Design Concepts

The desired outcome of the payload design phase was to ensure a reduced drag coefficient of the payload. An estimation of drag coefficients for various shapes can be seen in Figure 6.6 [8], where the effect of the projectile's shape on the projectile's drag coefficient can be noted.

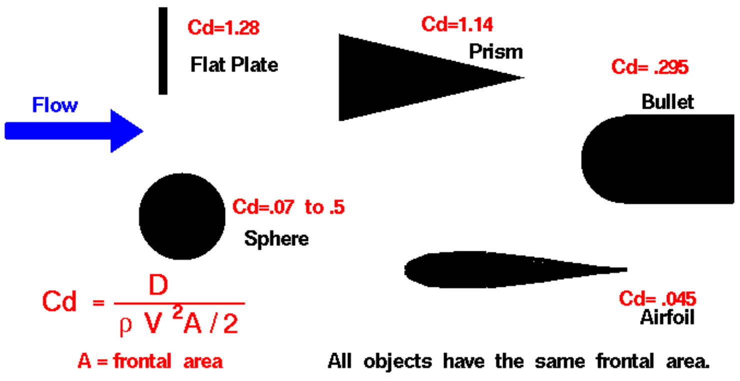


*Figure 6.6: Various shapes' drag coefficients*

#### 6.3.1.1 First Concept – Spherical

The first concept consisted of a two-part spherical payload vessel, which was designed around ensuring a constant drag coefficient, irrespective of its orientation during flight. Each hemisphere would be hollow to allow for the desired payload items to be placed inside. The hemispheres would interlock with the use of locating pins.

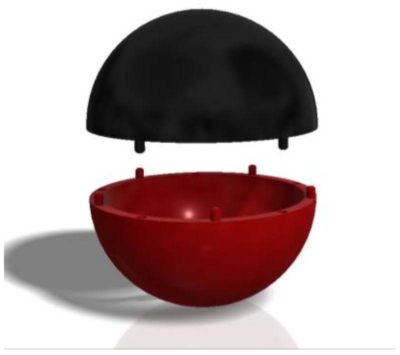


*Figure 6.7: First payload design concept*

#### 6.3.1.2 Second Concept – Three-stage Aerodynamic

This concept focussed on an aerodynamic approach, as opposed to that of the first concept, with the development of a three-stage payload with a nose cone and stabilising tail fins. The design was based on the premise of using readily available plastic piping as a payload body, allowing for a generic nose cone and tail to be designed to fit any length of pipe.

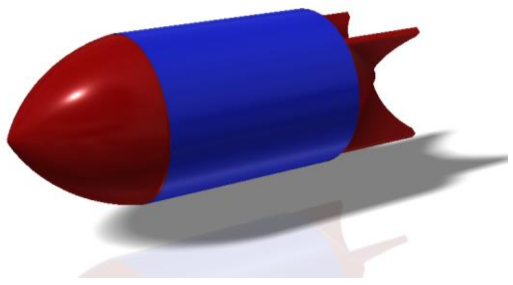


*Figure 6.8: Second payload design concept*

#### 6.3.1.3 Third Concept – Two-stage Aerodynamic

The third concept followed suit of the three-stage payload in that it also made use of the aerodynamic nose cone and tail design. However, this concept consisted of a hollow two-stage nose cone and tail, which interlocked with the use of pressure fit between the two components.

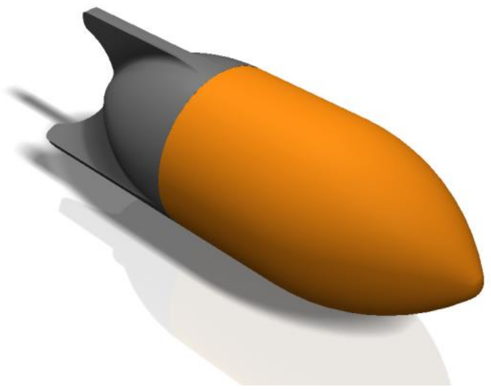


*Figure 6.9: Third payload design concept*

### 6.3.2 Final Design and Design Refinement

The desired outcome of the design phase was the optimisation of the payload’s drag coefficient. The first concept was eliminated from the available selections, due to the improvement in reduced drag that the nose cone designs offered. From the two remaining concepts, the decision was made to use the third concept, the two-stage nose cone and tail design. This decision was made as the full design of the payload could be altered to provide the lowest possible drag coefficient.

There were considerations to be taken when designing the payload, namely: the centre of pressure, the centre of gravity and the aerodynamic centre. Allowing for unrestricted design alterations to be made to the payload presented the opportunity to ensure that the payload’s design aspects were optimised. Although the second concept’s design was more accommodating, in terms of ease of manufacturability and available internal space, the cylindrical body of the payload would not ensure that the desired design aspects’ relationships were satisfied. The significance of the centre of pressure versus centre of gravity design criteria was based on ensuring stabile flight of the payload and preventing the payload from tumbling during free fall. Centre of pressure was not a static variable. Changing the angle of attack of the payload during free fall resulted in a changing centre of pressure. This change in centre of pressure would create a torque and subsequently result in the aforementioned tumble of the payload during free fall. The payload tumbles about its centre of gravity. The aerodynamic centre represented the location through which the aerodynamic force components, lift and drag, acted through. As such, the correct placement of the aerodynamic centre would allow for the aerodynamic force components to induce a torque about the aerodynamic centre, aerodynamic moment, which would counteract the induced torque of the dynamic centre of pressure and subsequently stabilise the payload through free fall. For rocket and missile designs, the aerodynamic centre traditionally sits at the aerodynamic centre of the stabilising fins. Due to the induced aerodynamic moment, the centre of pressure could be deduced to be the product of the weighted centre of the areas of the payload’s features and their centre distances to a defined reference line. Figure 6.10 and Equation 6.23 describe the method of weighted centre of pressure determination [6][7].

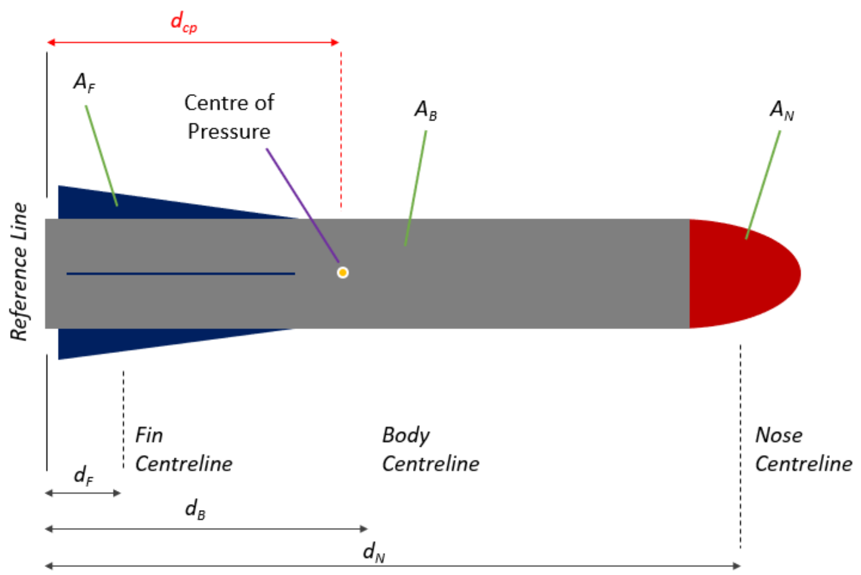


*Figure 6.10: Determination of centre of pressure*

$$A_{Total} d_{cp} = A_F d_F + A_B d_B + A_N d_N \tag{6.23}$$

Where, the respective areas were $A_F$, $A_B$, $A_N$ and $A_{Total}$. where $A_{Total} = A_F + A_B + A_N$. The displacements from the reference line to each feature centre were $d_F$, $d_B$, $d_N$ and $d_{cp}$ [6].

Computational fluid dynamics (CFD) simulations were conducted to determine the final payload's optimal design and drag coefficient. The drag coefficient was calculated by setting a simulation goal of the aerodynamic force exerted on the payload and using Equation 6.6, the average drag coefficient was determined. Figure 6.11 and Figure 6.12 show the flow results of the flow simulation. The payload was modelled in various orientations to ensure the greatest drag coefficient of the design was known. This changing orientation implied that the payload would not remain in the same orientation throughout freefall. One of these orientations can be seen in Figure 6.11. The final payload design's drag coefficient was found to be 0.096.

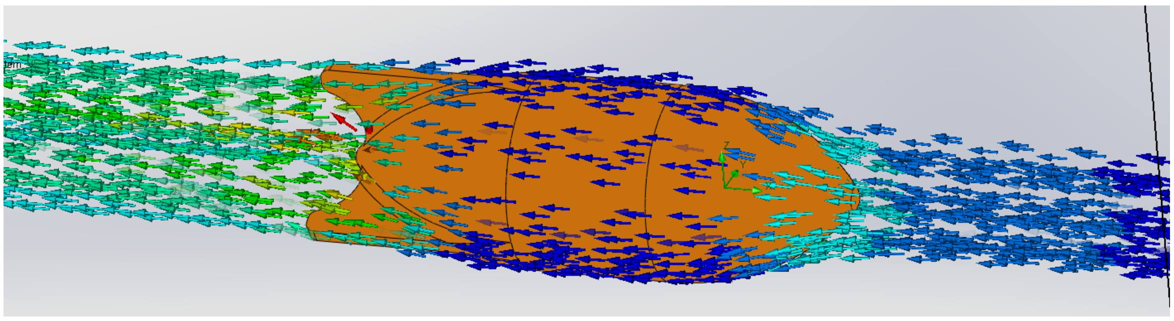


*Figure 6.11: CFD drag coefficient calculation simulation*

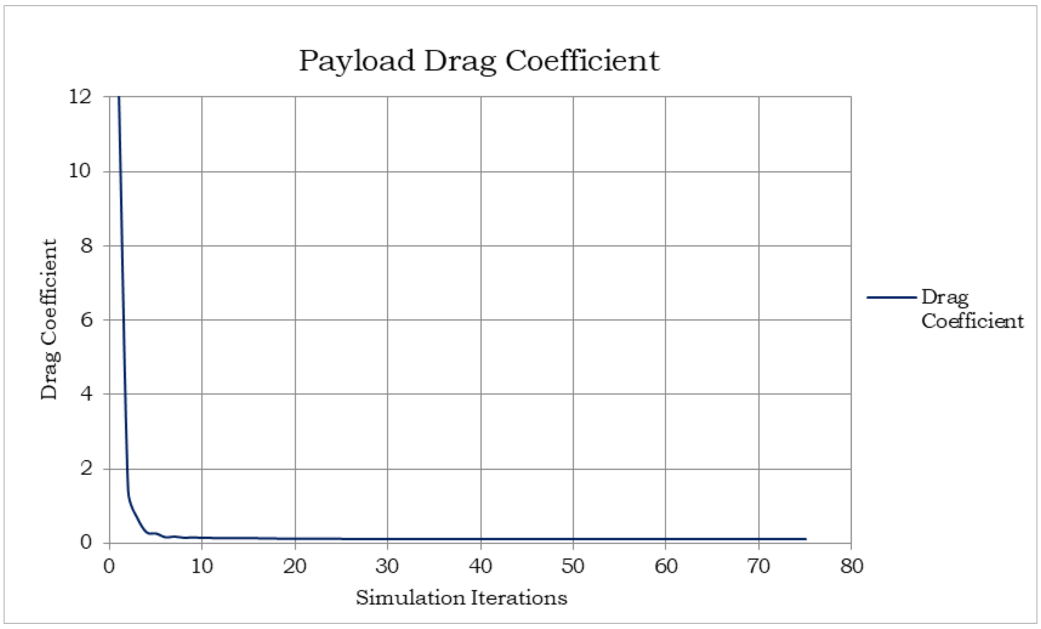


*Figure 6.12: Drag coefficient plot from SolidWorks CFD simulation*

From Figure 6.11, it was noted that there was an increase in drag experienced across the payload, as the change in the force vector colour from blue to green indicated a decrease of the magnitude of the force vector. Figure 6.12 indicated the changing of the drag coefficient during the simulation. The decrease in drag coefficient was due to the decrease of the aerodynamic force over the length of the payload. It was noted that the drag coefficient converged to its average value as the simulation progressed, indicating that the payload had reached equilibrium within the flow simulation. The final payload design utilised in this research can be seen in Figure 6.13.

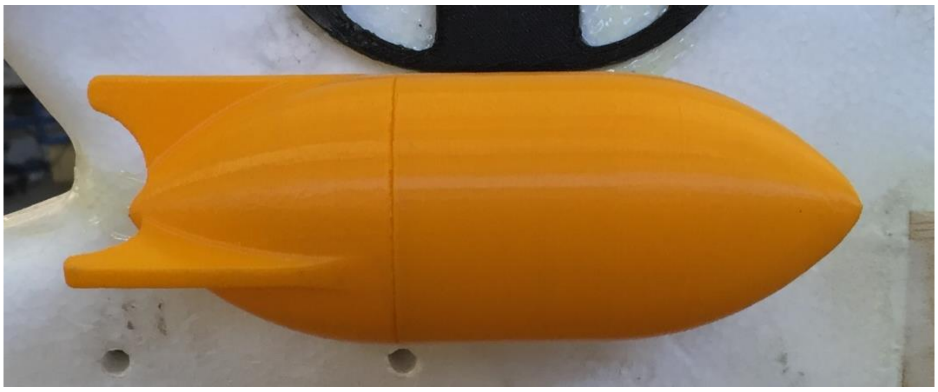


*Figure 6.13: Final payload design*

The final payload design was 3D printed with a minimal periphery layers to reduce the additional weight the payload vessel would possess above that of its contents. The final payload weight was found to be 0.11 kg and the overall length and major diameter of the payload was found to be 0.17 m and 0.06 m, respectively. The final payload design can be seen in Appendix K.

## 6.4 Payload Release Node

The payload release node was responsible for the computation of the headwind velocity and direction. Based on these headwind variables, the payload release node was then responsible for the interpolation and extrapolation of the lookup table values. Once these values had been determined, the payload release approach was defined. Figure 6.14 illustrates the computational flow diagram for the payload release node. The code for the payload release node can be found in Appendix A4.

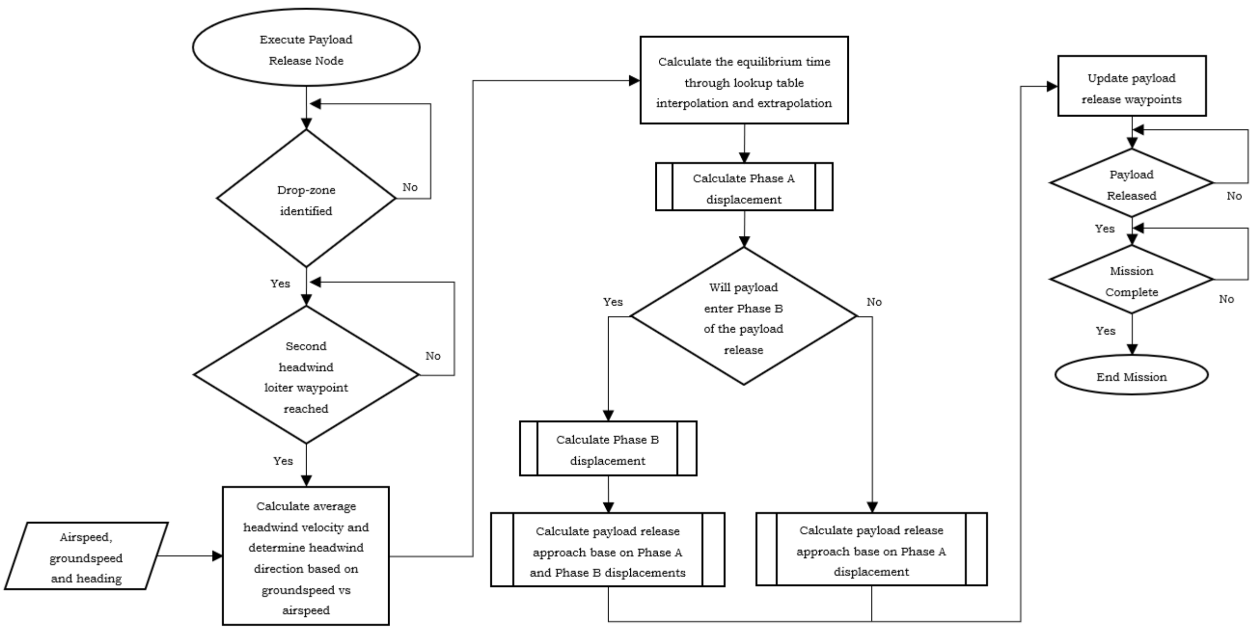

Execute Payload Release Node
Drop-zone identified
No
Yes
Second headwind loiter waypoint reached
No
Yes
Airspeed, groundspeed and heading
Calculate average headwind velocity and determine headwind direction based on groundspeed vs airspeed
Calculate the equilibrium time through lookup table interpolation and extrapolation
Calculate Phase A displacement
Will payload enter Phase B of the payload release
Yes
No
Calculate Phase B displacement
Calculate payload release approach base on Phase A and Phase B displacements
Calculate payload release approach base on Phase A displacement
Update payload release waypoints
Payload Released
No
Yes
Mission Complete
No
Yes
End Mission

*Figure 6.14: Payload release node computational flow diagram*

### 6.4.1 Determination of the Environmental Parameters

As discussed in Chapter 4, the headwind velocity and headwind direction were deduced during the second loop of a three loop loiter around the detected drop-zone location waypoint. During this second loop, the average headwind velocity was deduced by taking the sum of $n$ headwind measurements and dividing it by $n$, as defined by Equation 4.6. Where, the headwind was defined to be the difference between the airspeed and groundspeed, as described by Equation 4.5. The direction of the headwind was defined to be the heading of the aircraft just prior to the lowest groundspeed reading during the second loiter. The airspeed, groundspeed and heading of the aircraft was determined with the use of the MAVROS Library's *VFR_HUD* header [48].

$$v_{w\_diff} = v_{ground} - v_{air} \tag{4.5}$$

$$v_w = \frac{\sum_{i=1}^{n} v_{w\_diff_i}}{n} \tag{4.6}$$

## 6.4.2 Lookup Tables

As discussed in Section 6.2.3.1, lookup tables were developed to reduce computational strain during flight by tabulating predetermined equilibrium times and Phase A displacements. As such, three lookup tables were developed, each for a different altitude to allow for data extrapolation. Each lookup table possessed the same headwind velocity and aircraft airspeed values to allow for data interpolation. Figure 6.15 and Figure 6.16 illustrates the graphed lookup table values for the equilibrium times for the payload released from an altitude of 50 m. Corresponding lookup tables for the Phase A displacement of the payload were also developed. The respective lookup tables for both the equilibrium time and displacement of the payload under different testing conditions can be seen in Appendix J1 and Appendix J2.

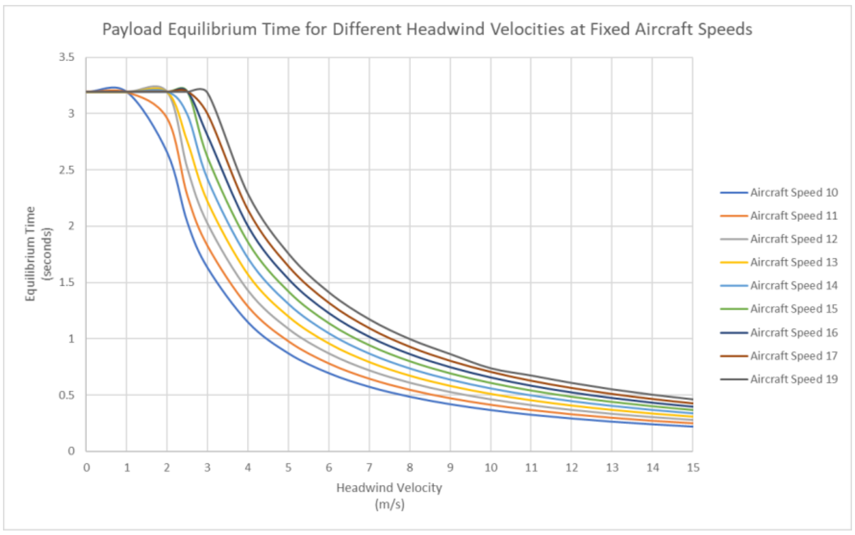


*Figure 6.15: Equilibrium times for different headwind velocities at fixed aircraft speeds at 50m*

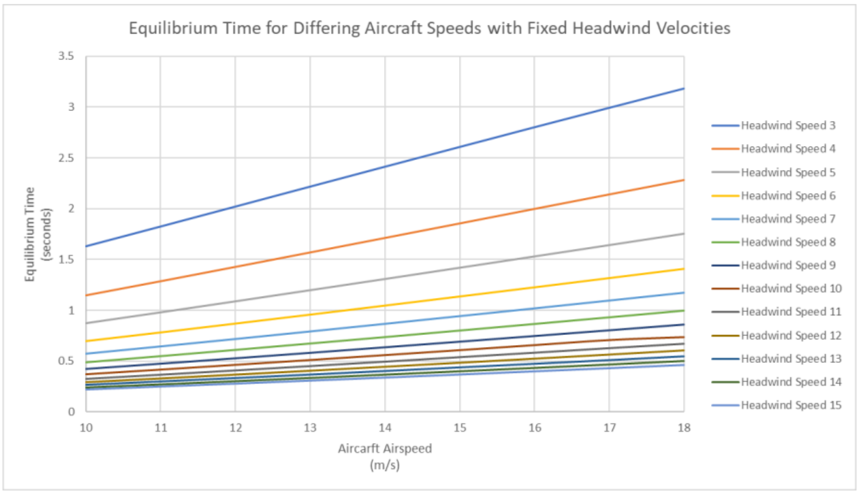


*Figure 6.16: Equilibrium times for different aircraft speeds at fixed headwind velocities at 50m*

From Figure 6.15, the trend of the equilibrium time followed a power regression model. In the upper left side of Figure 6.15, the equilibrium time trends were seen to plateau and was due to the equilibrium times exceeding the fall time for the given release altitude, 50 m. With the increment of each aircraft airspeed value, a consistent shift in the equilibrium time trends appeared. Due to the complexity of the power regression trend, the graph in Figure 6.16 was generated, where Figure 6.16 illustrates the same equilibrium times as in Figure 6.15. The defining difference between Figure 6.15 and Figure 6.16 was that the fixed variable for each dataset in Figure 6.15 was the aircraft's airspeeds and in Figure 6.16, the headwind velocities was the fixed variable. From Figure 6.16, the datasets were found to possess linear trends. These trends were also found to possess decreasing gradients as the headwind increased between datasets. In addition to these decreasing gradients, the vertical shift between the linear trends of the datasets decreased. The linear trends of the 50 m altitude lookup table data from Figure 6.16 was approximated by Equation 6.24.

$$y = (0.7248x^{-1.175})x + (-0.0017x^2 + 0.0495x - 0.4433) \tag{6.24}$$

Where, the gradient of the linear trend was represented by the power function $0.7248x^{-1.175}$ and the *y-axis* intercept was defined by the quadratic equation $-0.0017x^2 + 0.0495x - 0.4433$. With the use of the linear trend equation from each lookup table, the equilibrium time of the payload could be deduced through substitution of the aircraft airspeed. Interpolation was used to deduce data between altitude values of the lookup table linear trend equations. The standard form of linear interpolation was defined by Equation 6.25.

$$y_{desired} = y_{less} + \frac{x_{desired} - x_{less}}{x_{greater} - x_{less}}(y_{greater} - y_{less}) \tag{6.25}$$

Where, $y_{desired}$ represented the desired equilibrium time, $x_{desired}$ represented the desired equilibrium time's aircraft airspeed, $y_{less}$ and $x_{less}$ represented a known equilibrium time lower than the desired result and its corresponding aircraft airspeed, and $y_{greater}$ and $x_{greater}$ represented a known equilibrium time greater than the desired result and its corresponding aircraft airspeed.

With the equilibrium time deduced, the displacement of the payload could then be deduced through application of Equation 6.20 and Equation 6.22, based on the outcome of Equation 6.21 as to whether the payload would enter Phase B of the payload release. All mathematical computations executed within the payload release node made use of the C++ *cmath* header, *math.h*. With the payload release trajectory now computed, the payload release waypoints could be updated. The updating of these waypoints was discussed in Chapter 4.

## 6.5 Chapter Conclusion

Chapter 6 presented the mathematic foundation upon which the payload release was modelled. Environmental parameters of the payload release were accommodated for by ensuring the aircraft flew directly into the headwind during the payload release approach. Flying into the headwind allowed for the payload release to be modelled in two dimensions. Modelling the payload release in two dimensions, it was found that two trajectory phases existed, Phase A and Phase B. Where, not all payload releases would result in the payload entering Phase B of the payload release. In addition to modelling the trajectory of the payload, the design of the payload was discussed. It was found that a two-stage aerodynamic payload design was the optimal for the desired application. Application of the mathematical foundation for the payload release trajectory was discussed in the form of the payload release node. The development of the payload release lookup tables and their mathematical relationships was discussed within the node structure. With the conclusion of Chapter 3 to Chapter 6, the method of subsystem communication could be discussed. Chapter 7 discussed the final integrated system achieved through subsystem communication.

# Chapter 7

# Integrated System

This chapter discussed the process through which integrated communication was achieved between the developed subsystems.

## 7.1 Integrated Communication

With the clarification of the functionality of the three defining subsystems: autonomous flight, vision and payload release, and their respective hardware and software configurations, the integrated communication between each subsystem could be defined.

As discussed in Chapters 3, 4, 5 and 6, intelligence was introduced to the aircraft via the addition of the ODroid and ROS. Where, ROS, running on the ODroid, provided the necessary processing capabilities in order to execute each subsystem's computational node simultaneously. Although the communication protocol between the various ROS nodes and hardware has been discussed, the clarification pertained to how these processes were executed initially has yet to be discussed.

### 7.1.1 Communication with the ODroid and ROS

During the development of this research, the majority of the ROS testing phase was developed offboard the aircraft and communication with the ODroid was established with the use of a keyboard, mouse and display monitor. This configuration allowed for ROS nodes to be constructed and tested manually with the use of the display monitor for performance evaluation. Due to the method of development, each ROS node required manual execution, and a specific order of execution of each ROS node was necessary based on prior node conditions. The ROS node execution order and conditions were as follows:

1. Node: MAVROS Node
    - Condition: FTDI cable connected to FCU and FCU initialisation complete.
2. Node: Image Publisher Node
    - Condition: Upon successful MAVLink connection with the FCU via the MAVROS node.
3. Node: Image Processing Node
    - Condition: Image publisher node executed.

4. Node: Payload Release Node
    - Condition: Upon successful MAVLink connection with the FCU via the MAVROS node.
5. Node: Autonomous Flight Node
    - Condition: Upon successful MAVLink connection with the FCU via the MAVROS node and all other nodes have been executed.

Noting the ROS node execution order and conditions, the autonomous flight node was defined to be the final node to be executed. The final order position for the autonomous flight node was necessary to ensure that the aircraft would not begin its mission without the remaining nodes running. From the ROS node execution order, it was noted that the execution of the image publisher node and the payload release node could be alternated as their conditional statements were the same. Above all node executions, MAVLink communication via the MAVROS node was necessary first as the MAVROS node hosted the remaining nodes.

#### 7.1.1.1 Secure Shell (SSH)

The ROS node development configuration of the ODroid was not feasible throughout a mission onboard the aircraft and as such, a method of wireless communication was developed in order to execute the ROS nodes remotely. Wireless communication was established between a laptop and the ODroid via the Secure Shell (SSH) protocol, where the SSH protocol represented an encrypted communication link between the SSH client (laptop) and the SSH Server (ODroid). This secure link ensured the data transfer between the two devices would remain protected. Figure 7.1 illustrates the SSH protocol [13].

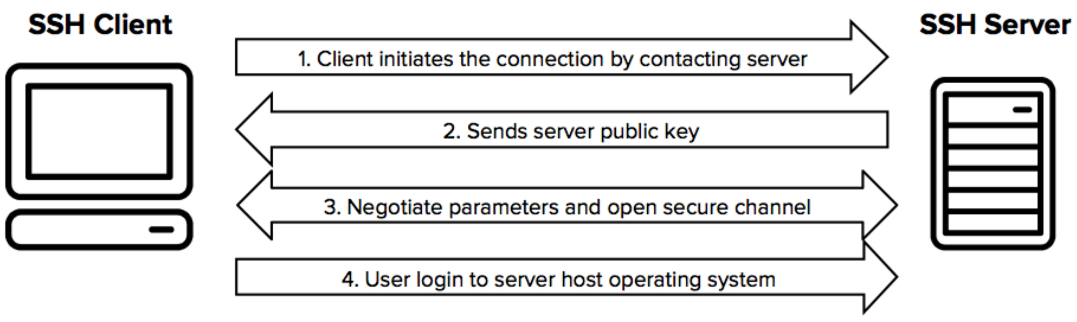
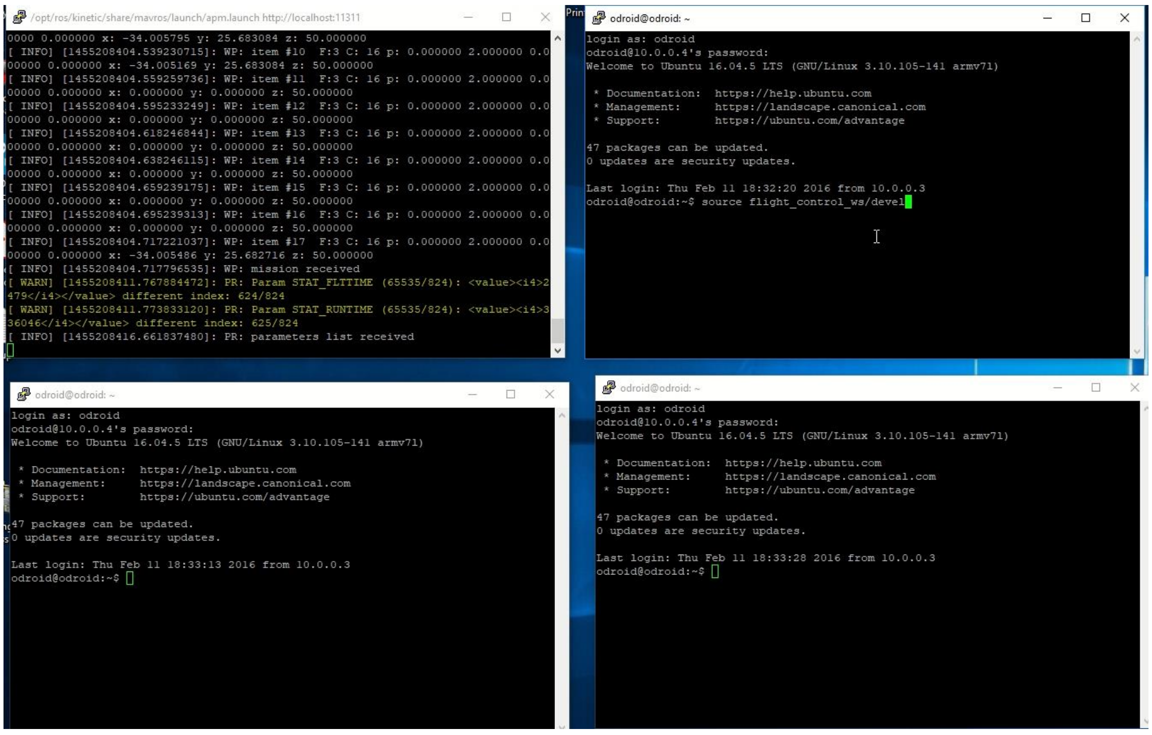

SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

*Figure 7.1: SSH protocol*

In order to establish the SSH communication between the client and server, client software was necessary. As the client's OS was Windows-based and the server's OS was Linux, client software capable of running on both systems was selected. PuTTY was the selected client software. In addition to the client software, a wireless network was required between the client and server in order for communication to be established. As such, a Wi-Fi adapter was added to the ODroid to provide it with the necessary wireless capabilities and an external wireless router was introduced to provide the required common network link between the client and server. Looking at the client software, PuTTY, a singular element was necessary in order to establish the desired SSH protocol link, the server's internet protocol (IP) address. Upon successful connection with the server, the client would be prompted to enter the necessary server security credentials. Once logged into the server, the client was capable of executing the desired command line protocols to execute the ROS nodes. The PuTTY SSH protocol client interface during ROS node execution can be seen in Figure 7.2.


/opt/ros/kinetic/share/mavros/launch/apm.launch http://localhost:11311

```
0000 0.000000 x: -34.005795 y: 25.683084 z: 50.000000
[ INFO] [1455208404.539230715]: WP: item #10  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: -34.005169 y: 25.683084 z: 50.000000
[ INFO] [1455208404.559259736]: WP: item #11  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.595233249]: WP: item #12  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.618246844]: WP: item #13  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.638246115]: WP: item #14  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.659239175]: WP: item #15  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.695239313]: WP: item #16  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1455208404.717221037]: WP: item #17  F:3 C: 16 p: 0.000000 2.000000 0.0
00000 0.000000 x: -34.005486 y: 25.682716 z: 50.000000
[ INFO] [1455208404.717796535]: WP: mission received
[ WARN] [1455208411.767884472]: PR: Param STAT_FLTTIME (65535/824): <value><i4>2
479</i4></value> different index: 624/824
[ WARN] [1455208411.773833120]: PR: Param STAT_RUNTIME (65535/824): <value><i4>3
36046</i4></value> different index: 625/824
[ INFO] [1455208416.661837480]: PR: parameters list received
```

```
login as: odroid
odroid@10.0.0.4's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 3.10.105-141 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

47 packages can be updated.
0 updates are security updates.

Last login: Thu Feb 11 18:32:20 2016 from 10.0.0.3
odroid@odroid:~$ source flight_control_ws/devel
```

```
login as: odroid
odroid@10.0.0.4's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 3.10.105-141 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

47 packages can be updated.
0 updates are security updates.

Last login: Thu Feb 11 18:33:13 2016 from 10.0.0.3
odroid@odroid:~$
```

```
login as: odroid
odroid@10.0.0.4's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 3.10.105-141 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

47 packages can be updated.
0 updates are security updates.

Last login: Thu Feb 11 18:33:28 2016 from 10.0.0.3
odroid@odroid:~$
```

*Figure 7.2: PuTTY SSH protocol client interface during ROS node execution*

From Figure 7.2, the ROS node execution interface through the SSH protocol could be seen. The upper left window in Figure 7.2 depicts the execution of the MAVROS node and the transmission of mission waypoints. The remaining windows in Figure 7.2 represented the remaining nodes to be tested, namely: the autonomous flight node, the image capture node and the image processing node.

Use of the SSH protocol provided the necessary tools for testing the aircraft remotely during ground testing of integrated system and provided a command line diagnostic as to the state of the ROS nodes. The problem identified with the SSH protocol noted that it did not execute the ROS nodes onboard the ODroid and merely played host to the nodes in the client. This hosting of the ROS nodes resulted in the client rejecting some of the desired ROS and MAVROS functionality that the ODroid was capable of executing, such as the payload release command and the updating of mission waypoints. In addition to these limited functionality issues, the hosting of the ROS nodes was only present whilst the wireless connection between the client and server was present. As such, when the aircraft flew outside of the wireless network's range, the ROS nodes would stop running, proving that the client only housed the ROS nodes and did not execute them onboard the ODroid. As the mission waypoints were saved to the FCU upon execution of the autonomous flight node, the aircraft would continue upon the desired mission, but elements such as MAVLink communication, vision and payload release would cease. Due to this limited range of functionality, the SSH protocol was not used in the final system and although long range wireless extenders and devices could have been introduced into the aircraft to continue the hosted execution of the ROS nodes, the risk of communication failure and subsequent ROS node failure was too prominent to overlook. The SSH protocol proved useful when in proximity to the wireless network and contributed to the development phase of this research.

#### 7.1.1.2 Physical Connection

With the limitations the SSH protocol presented, a solution was required to ensure the ROS nodes were executed and hosted onboard the ODroid to ensure the entire integrated system was running during a mission. The final method of communication established with the ODroid to execute the required ROS nodes was done through the reimplementation of the ROS node development configuration. A keyboard, mouse and display monitor were connected to the ODroid in order to execute the ROS nodes. The alterations made to the integrated system to allow for this configuration were software based and the keyboard, mouse and display monitor were removed prior to mission execution. The software alterations involved the removing of the aircraft's ability to arm via the autonomous flight node's arming client. Removing the arming functionality allowed for all ROS nodes to be executed and the necessary hardware to be removed without the aircraft attempting to takeoff. The aircraft was then prepared for flight and placed in the direction of the headwind.

Once ready, the aircraft was manually armed and would begin the mission. Use of the ROS node development configuration succeeded in ensuring all ROS nodes were executed during a mission. The only feature that the ROS node development configuration lacked when compared to the SSH protocol approach was that the SSH protocol allowed for live monitoring of the ROS nodes. In the case of the ROS node development configuration, the discernible feedback as to the success of the system during a mission was the aircraft following the mission waypoints and the appearance of the updated waypoint coordinates during the mission. This loss of node monitoring was not detrimental to the performance of the system and resulted in the need for post mission analysis of the ROS nodes. The ROS node development configuration can be seen in Figure 7.3.

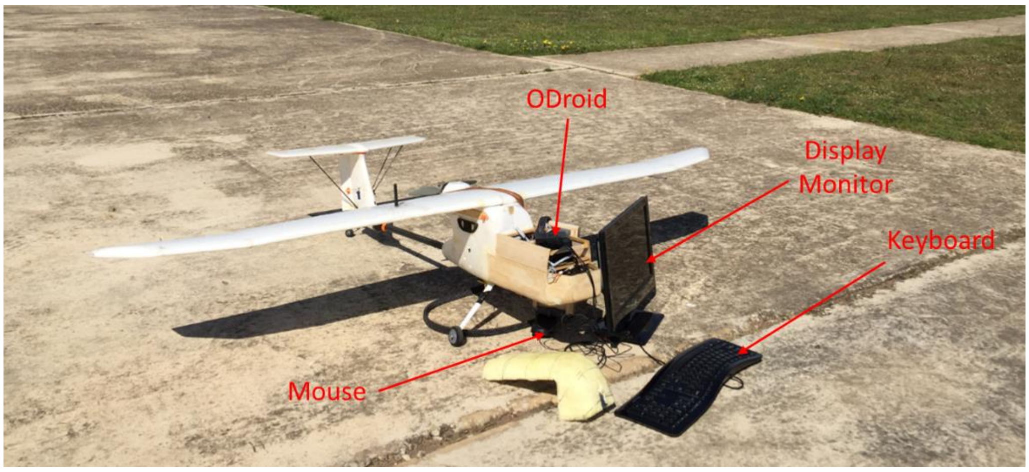


*Figure 7.3: ROS node development configuration of the ODroid onboard the aircraft*

## 7.2 Chapter Conclusion

Chapter 7 served as an explanation as to the method in which communication was established with the ODroid in order to execute the ROS nodes. The method of the SSH protocol was discussed and although this communication protocol was not utilised in the final system, it was extensively used in the remote ground testing of the ODroid and ROS nodes. Chapter 3 through to Chapter 7 defined the subsystems of the final integrated system, which was used to obtain results through testing. Chapter 8 discussed the results obtained through the implementation of the integrated system developed.

# Chapter 8

# Results and Discussion

This chapter discussed the testing and results of the research design. Each of the three defining elements of this research were discussed independently. The integrated system's performance was also discussed.

## 8.1 Focus Area of Research Design and Methodology

As this research focussed on the application of the developed system, testing and integration of each subsystem was the focus in achieving the defined objectives. The research design and methodology focussed on defining how each subsystem was tested. These tests served to provide an indication of the success of this research in achieving the defined objectives.

Looking at the research objectives, the three subsystems: autonomous flight, vision and payload release, were seen to each possess individual objectives. Due to the diversity of each subsystem, various testing scenarios were necessary to test individual and integrated functionality. Noting that the hardware and software architecture of this research have already been stated, the research design and methodology of this research sought to define how the application of said hardware and software was used in order to achieve the research objectives defined in Chapter 1.

### 8.1.1 Method for Testing

As a mechatronic system has been defined to comprise of several systems integrated into a single system, testing was required for not only the final integrated system, but also each subsystem. Individual subsystem testing allowed for each subsystem's performance to be evaluated. Where, subsystem performance allowed for the cumulative performance of the integrated system to be determined, highlighting the accuracy and repeatability limitations of the integrated system. These tests served as the results of this research.

#### 8.1.1.1 Autonomous Flight Control Testing

In order to achieve the objectives defined for autonomous flight control, two testing scenarios were developed. These testing scenarios included the testing of the aircraft's ability to execute an autonomous mission, where the aircraft's ability to follow a predefined flight path represented the success of the test. Elements such as waypoint overshoot and ability to maintaining altitude and airspeed were used to quantity the accuracy and repeatability of the autonomous flight control system. The second testing scenario included the aircraft's ability to allocate new flight path waypoints based on mission parameters. Testing of the aircraft's ability to allocate new flight path waypoints formed part of an integrated test, as input sensory data capable of influencing the change in the aircraft's flight path was represented in the form of the vision system's ability to detect the drop-zone's location.

Physical testing required a baseline upon which to quantify its results, as such simulations of the autonomous flight of the aircraft were conducted. As autonomous flight of the aircraft was influenced by testing environment specific parameters, such as wind speed and direction, simulations were conducted after physical testing to ensure equivalent testing parameters.

#### 8.1.1.2 Vision System Testing

Vision formed the defining element of this research in terms of sensory feedback, the accuracy and repeatability of this system remained pertinent to the success of the integrated system. Considering the vision objective for this research, several requirements of the vision system were noted. From these requirements, the pure functionality of the vision system was defined to be the recognition of the drop-zone location. The remaining requirements formed part of the integrated system, where the relative location of the drop-zone with respect to the aircraft and the updating of the drop-zone's true location saw the vision system interacting with the aircraft's sensory feedback peripheral devices and the FCU. The integrated requirements of the vision system relied on the accuracy and repeatability of the vision system's ability to identify the drop-zone's location. Testing of the vision system took the form of accuracy tests, both simulated and physical, to ensure the vision system was capable of detecting the drop-zone identifier. Testing of the vision system's ability to define the location of the drop-zone relative to the aircraft involved an integrated system test. Where, the accuracy and repeatability of this test was defined by the vision system's ability to determine the drop-zone's location with respect to a known location, highlighting any measurement errors that could be extrapolated over a larger distance.

As with autonomous flight control testing, simulation formed a key role in testing the vision system. Simulation was primarily used for the calibration of the vision system, as desktop testing was possible with the vision system and simulated drop-zone identifiers could be replicated for offboard testing.

#### 8.1.1.3 Payload Release Testing

Physical payload release testing formed part of an integrated test as autonomous flight control and drop-zone identification was necessary in order to achieve the defined objectives. As such, payload release validation relied on simulation results, where different testing parameters were evaluated to determine their influence on the payload release trajectory. Physical payload release drops provided an indication of the payload release system's accuracy and repeatability. Due to the level of integration of the payload release system in the final system, accuracy and repeatability results were dependant on the cumulative accuracy and repeatability of the integrated system. Testing of the payload delivery system's accuracy and repeatability not only fulfilled the research objectives for the payload release system, but also contributed to the testing and fulfilment of the vision and autonomous flight control systems.

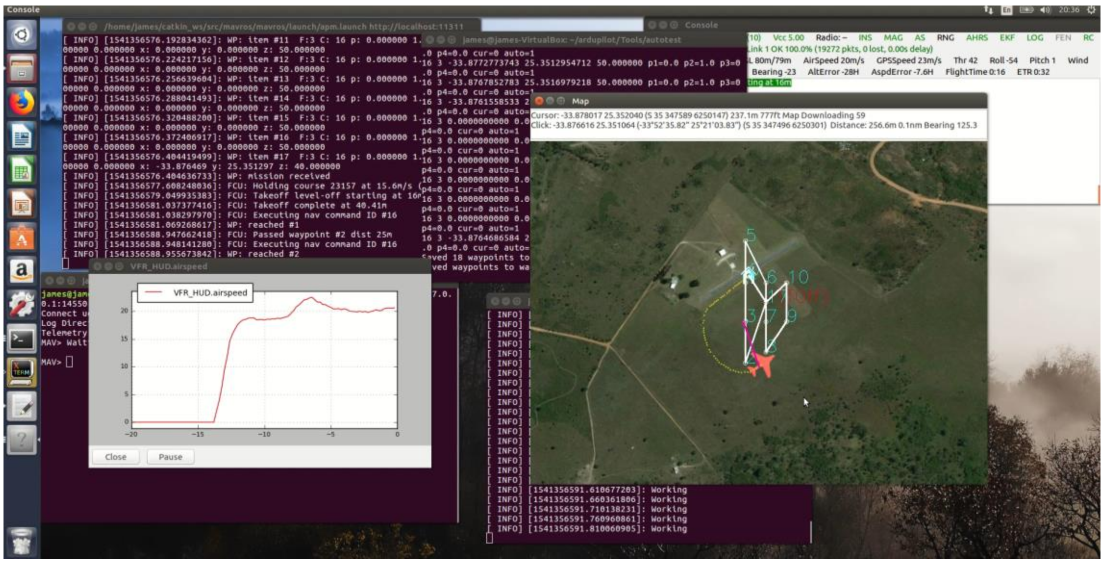
## 8.2 Autonomous Flight Testing

This section covered both the simulated and the physical testing and results of the autonomous flight control portion of this research.

### 8.2.1 Simulation

In order to determine whether the autonomous flight system would function, without putting the aircraft at risk, missions were simulated with the use of a Linux based software in the loop (SITL). The SITL enabled the simulation of a fully functioning aircraft which communicated via the MAVLink protocol. The same aircraft profile and parameter list utilised on the physical aircraft's Pixhawk were uploaded and tested. As such, the SITL could simulate the full aircraft and any additional hardware without any physical hardware being present. In conjunction with the SITL simulator, MAVProxy was also simultaneously implemented. MAVProxy represented a fully functioning GCS which could be utilised for both SITL simulations and for real-time mission monitoring of physical aircraft communicating under the MAVLink protocol. The difference between MAVProxy and a GCS, such as QGroundControl, was that MAVProxy was open-source in terms of defining the additional plugin modules that would be opened when the GCS launched and MAVProxy acted as the MAVROS node.

With SITL running, simulation parameters such as headwind direction and velocity could be implemented, allowing for the advanced testing of how the aircraft would theoretically react in a given flight scenario. The flight simulation was then used to refine the aircraft's flight parameters and allowed for fault finding prior to execution of the physical mission. Further implementation of the SITL software allowed for implementation of the ROS nodes and provided the ability to test each node's effect on the flight of the aircraft. The primary ROS node tested in these situations was the autonomous flight node, as it allowed for the calibration of the required line-up and turnaround distances for the mission survey area. Further testing with the simulated flight paths allowed for the testing of the vision nodes and the payload release node, by simulating the detection of the drop-zone during the survey period. In addition to the flight path simulation, physical properties of the flight were monitored, such as the theorised power consumption of the given flight. A SITL simulation of the autonomous flight node can be seen in Figure 8.1.


Console
/home/james/catkin_ws/src/mavros/mavros/launch/apm.launch http://localhost:11311
james@james-VirtualBox: ~/ardupilot/Tools/autotest
Console
Map
VFR_HUD.airspeed
VFR_HUD.airspeed
Close
Pause
20:36

```
[ INFO] [1541356576.192834362]: WP: item #11  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.224217156]: WP: item #12  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.256639604]: WP: item #13  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.288041493]: WP: item #14  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.320488200]: WP: item #15  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.372406917]: WP: item #16  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: 0.000000 y: 0.000000 z: 50.000000
[ INFO] [1541356576.404419499]: WP: item #17  F:3 C: 16 p: 0.000000 1.
00000 0.000000 x: -33.876469 y: 25.351297 z: 40.000000
[ INFO] [1541356576.404636733]: WP: mission received
[ INFO] [1541356577.608248036]: FCU: Holding course 23157 at 15.6m/s (
[ INFO] [1541356579.049935383]: FCU: Takeoff level-off starting at 16r
[ INFO] [1541356581.037377416]: FCU: Takeoff complete at 40.41m
[ INFO] [1541356581.038297970]: FCU: Executing nav command ID #16
[ INFO] [1541356581.069268617]: WP: reached #1
[ INFO] [1541356588.947662418]: FCU: Passed waypoint #2 dist 25m
[ INFO] [1541356588.948141280]: FCU: Executing nav command ID #16
[ INFO] [1541356588.955673842]: WP: reached #2
```

```
.0 p4=0.0 cur=0 auto=1
16 3 -33.8772773743 25.3512954712 50.000000 p1=0.0 p2=1.0 p3=0
.0 p4=0.0 cur=0 auto=1
16 3 -33.8767852783 25.3516979218 50.000000 p1=0.0 p2=1.0 p3=0
.0 p4=0.0 cur=0 auto=1
16 3 -33.8761558533 2
.0 p4=0.0 cur=0 auto=
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 0.0000000000 0.0
p4=0.0 cur=0 auto=1
16 3 -33.8764686584 2
.0 p4=0.0 cur=0 auto=
Saved 18 waypoints to
ved waypoints to wa
7.0.
```

```
james@jam
0.1:14550
Connect u
Log Direc
Telemetry
MAV> Wait

MAV>
```

```
(10)  Vcc 5.00  Radio: --  INS  MAG  AS  RNG  AHRS  EKF  LOG  FEN  RC
Link 1 OK 100.0% (19272 pkts, 0 lost, 0.00s delay)
L 80m/79m  AirSpeed 20m/s  GPSSpeed 23m/s  Thr 42  Roll -54  Pitch 1  Wind
 Bearing -23  AltError -28H  AspdError -7.6H  FlightTime 0:16  ETR 0:32
ting at 16m
```

```
Cursor: -33.878017 25.352040 (S 35 347589 6250147) 237.1m 777ft Map Downloading 59
Click: -33.876616 25.351064 (-33°52'35.82" 25°21'03.83") (S 35 347496 6250301)  Distance: 256.6m 0.1nm Bearing 125.3
```

```
[ INFO] [1541356591.610677203]: Working
[ INFO] [1541356591.660361806]: Working
[ INFO] [1541356591.710138231]: Working
[ INFO] [1541356591.760960861]: Working
[ INFO] [1541356591.810060905]: Working
```

*Figure 8.1: SITL simulation of the autonomous flight node*

## 8.2.2 Flight Tests

Autonomous flight testing was undertaken in several stages to ensure the system was fully calibrated for an autonomous mission. In the initial stages of testing, no autonomy was introduced into the aircraft and the system was flown via the onboard remote-control hardware. These manual flights were used to calibrate the aircraft in terms of control surface trims and to allow for comparative data to be collected to determine the efficiency of the autonomous FCU versus manual remote piloting. These manual flights were also used to determine the necessary alterations required after the nose and fuselage of the aircraft were replaced. These alterations included balancing the aircraft accordingly due to the shift in the aircraft's centre of gravity with the additional onboard weight. These manual flight tests were necessary in ensuring a stable platform was prepared for autonomous flight.

Once a stable platform had been developed, tested and calibrated, the testing of the Pixhawk's built-in autonomous flight functionality was undertaken. As the autonomous takeoffs and landings were the final autonomous flight control processes tested, the aircraft was manually launched and landed for the majority of the autonomous flight control testing. Once manual takeoff and stable flight had been achieved, the aircraft was switched into *AUTO* flight mode, executing the autonomous mission. Whilst in *AUTO* flight mode, the aircraft's onboard hardware was tested in terms of accuracy and repeatability. Where, the accuracy of an autonomous mission was based on the aircraft's ability to replicate the SITL simulation flight paths. Additionally, a *LOITER* waypoint was used to test the aircraft's ability to maintain a desired airspeed and altitude, and to test the aircraft's repeatability when executing the same loiter repeatedly.

The final testing stage involved the testing of the fully integrated system, involving mission autonomy from takeoff to payload delivery to landing. Final stage testing was an indication of the success of the autonomous flight node in allocating waypoints and updating mission parameters. The final test relied on the other subsystem's inputs to ensure a full mission was executed. However, the performance of the autonomous flight node was still able to be evaluated independently.

### 8.2.3 Flight Test Results

As the manual flight test were for calibration and development of the test bench aircraft into a stable system, the flight manual tests were not discussed. The various parameters altered due to these tests were discussed in Chapter 3 and Chapter 4.

The results of the second testing phase of the autonomous flight control system sought to identify the aircraft's performance in maintaining a specific altitude, airspeed and flight path with no additional hardware onboard. This second test was undertaken by executing a loiter about a non-specific waypoint, where the loiter radius was defined to be 100 m and the setpoint altitude and airspeed were defined to be 75.5 m and 12.5 m/s respectively. Six loiter turns were executed about the given waypoint and the mapped flight path of the aircraft can be seen in Figure 8.2, where the theoretical loiter circle was been superimposed. The results of the aircraft's ability to maintain the setpoint altitude and airspeed can be seen in Figure 8.3 and Figure 8.4.

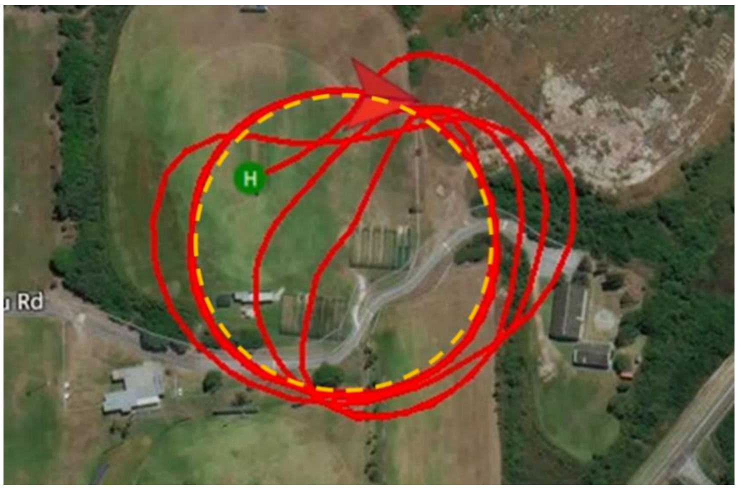


*Figure 8.2: Autonomous flight control loiter with theoretical loiter circle overlaid*

From Figure 8.2, it was noted that the aircraft was able to maintain the desired loiter circle successfully over all six rotations. The additional flight path lines that do not follow the theoretical loiter flight path represented the aircraft's flight paths prior to entering the loiter. These prior flight paths were not considered during the assessment of the aircraft's ability to maintain the desired flight path. From a hardware perspective, it was noted that although the GPS module onboard the aircraft was not as reliable as other available positioning systems, in terms of GPS location determination with respect to true GPS location. The aircraft's ability to maintain a consistent loiter about the same GPS location indicated that the GPS module was able to maintain a consistent error. This consistency implied that the aircraft's GPS module was accurate with respect to itself and this ability to maintain internal accuracy was found to be ideal in the application of the payload release system, where all drop-zone coordinates were defined relative to the aircraft's GPS location and not the true location.

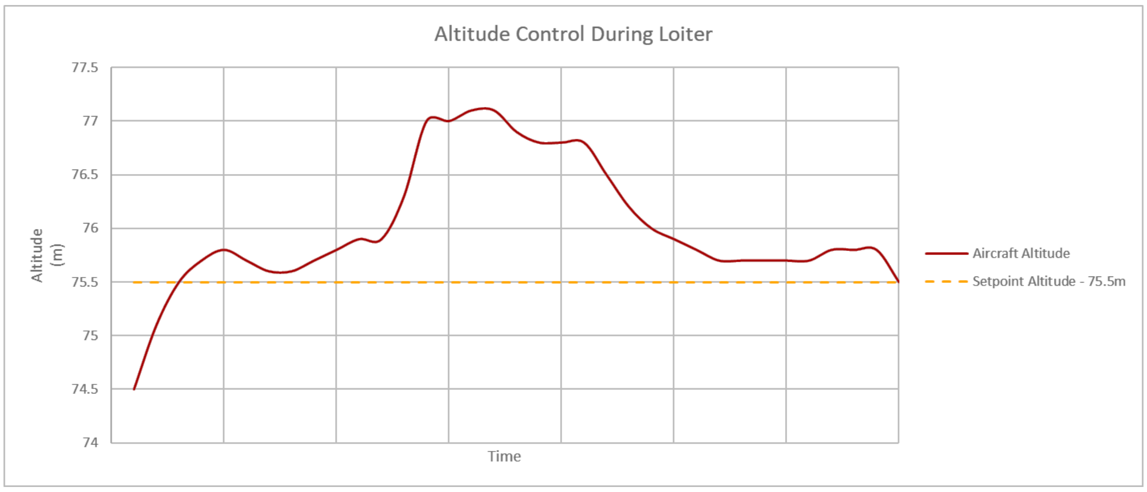


*Figure 8.3: Aircraft altitude control during loiter flight test*

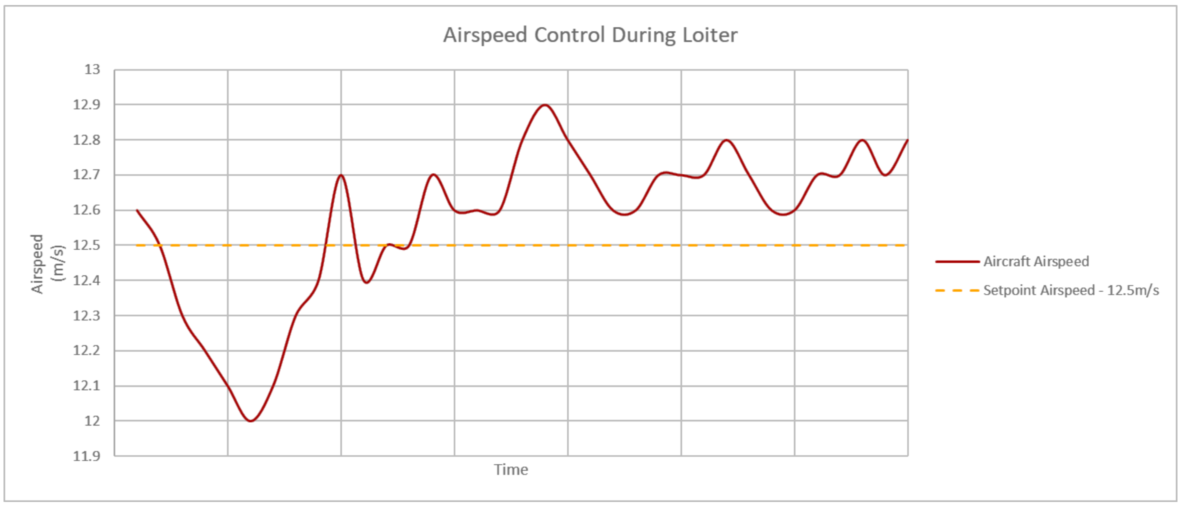


*Figure 8.4: Aircraft airspeed control during loiter flight test*

From Figure 8.3 and Figure 8.4, it was noted that the aircraft's ability to maintain the desired setpoint altitude and airspeed seemed inconsistent. It was noted that the deviation of these results were within close proximity to the setpoint value. This observation was further validated by deducing the average altitude and average airspeed the aircraft attained during the loiter flight path. The average altitude was found to be 76.01 m and the average airspeed was found to be 12.57 m/s. These averages indicated that the aircraft maintained an average altitude error within 0.68% of the desired altitude and maintained an average airspeed error within 0.57% of the desired airspeed. Equation 8.1 and Equation 8.2 described the method in which accuracy was determined. Where, Figliola et al. [18] defined the accuracy of a component to be an indication of how close a produced reading was to its actual value.

$$e = Measured\,Value - Reference\,Value \tag{8.1}$$

$$A = \frac{|e|}{Reference\,Value} \times 100 \tag{8.2}$$

Equation 8.1 described the error ($e$) between the measure value and the true value, and Equation 8.2 described the accuracy ($A$) of the error with respect to the true value.

Comparisons were made based on the performance of the autonomous flight control of the aircraft when compared to the simulated flight control and manual flight control of the aircraft. The first of comparison was based on the instantaneous power consumed during takeoff and during flight, where instantaneous power provided an indication of throttle control. The instantaneous power consumed during takeoff can be seen in Figure 8.5 and the instantaneous power consumption during flight can be seen in Figure 8.6.

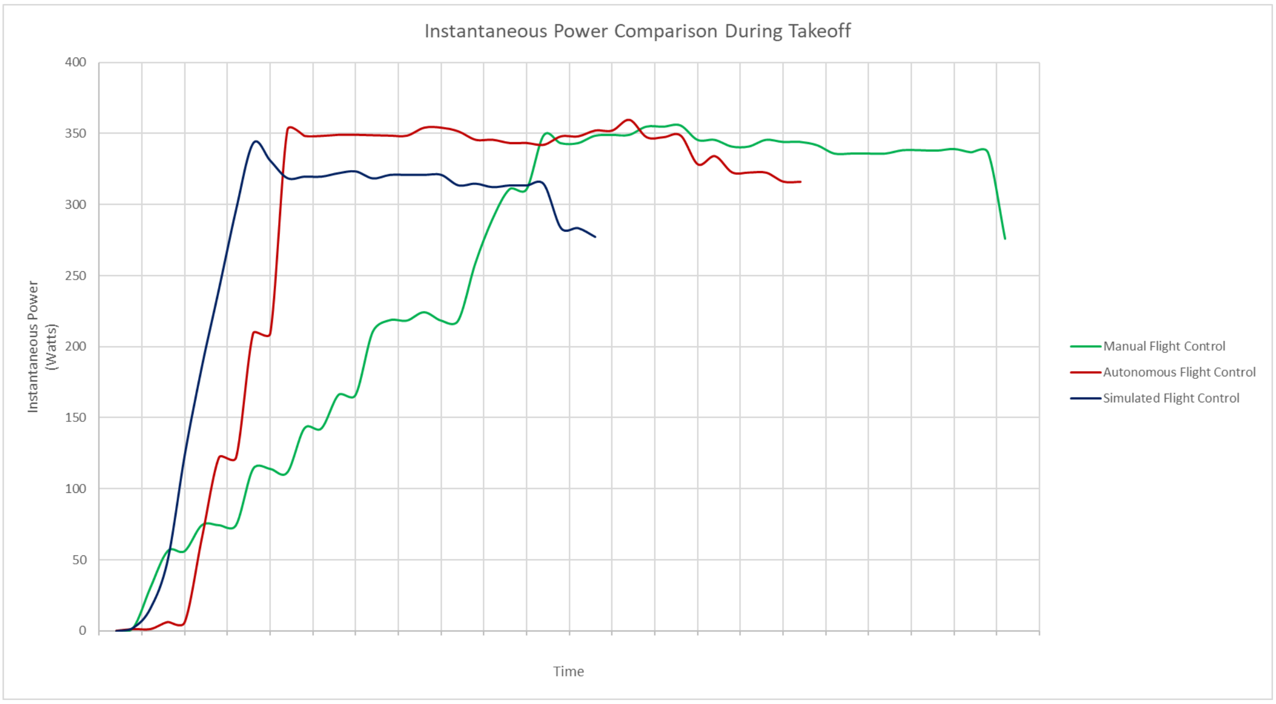


*Figure 8.5: Instantaneous power consumption comparison during takeoff*

From Figure 8.5, it was noted that the simulated flight control and autonomous flight control shared similar instantaneous power trends, where an initial spike in power was present. This initial power spike indicated that both the simulator and the FCU implement high throttle power as soon as a takeoff was initiated and once at the desired upper throttle value, the simulator would keep the throttle power constant until takeoff was complete. Looking at the instantaneous power of the manual flight control, it was noted that a less aggressive throttle ramp was implemented until the desired throttle position was attained and maintained until takeoff was complete. Takeoff was noted to be the moment when the instantaneous power started to decrease consistently.

Assessing these takeoff power trends, it was noted that the simulated flight control increased the throttle power at a steady, almost linear, rate. When compared to the autonomous flight control and manual flight control, incremental steps were noted during throttle ramp. Focussing on the autonomous flight controller, this difference in throttle ramp was deduced to be due to hardware capabilities, where possible throttle ramp functions and multipliers were present to reduce the amount of instantaneous current introduced into the motor and ESC. These same functions would have been present for the manual flight control, but throttle control would have been at the operator's discretion and as such, the inconsistency in throttle ramp was noted.

Additionally, the time taken for simulated flight control and autonomous flight control were less than that of the manual flight control. Simulated flight control was seen to possess the fastest takeoff. The decrease in required takeoff time for manual flight control was to be expected due to the reduced throttle ramp and the moment of takeoff was also defined by the operator. This manual control implied that the given takeoff period for manual flight control could have been conservative. The difference between the simulated flight control and autonomous flight control takeoff period was deduced to be due to the effects of friction as the aircraft was executing the takeoff. The simulator did not replicate surface texture and the additional friction experienced during physical testing resulted in the autonomous flight control takeoff period increasing.

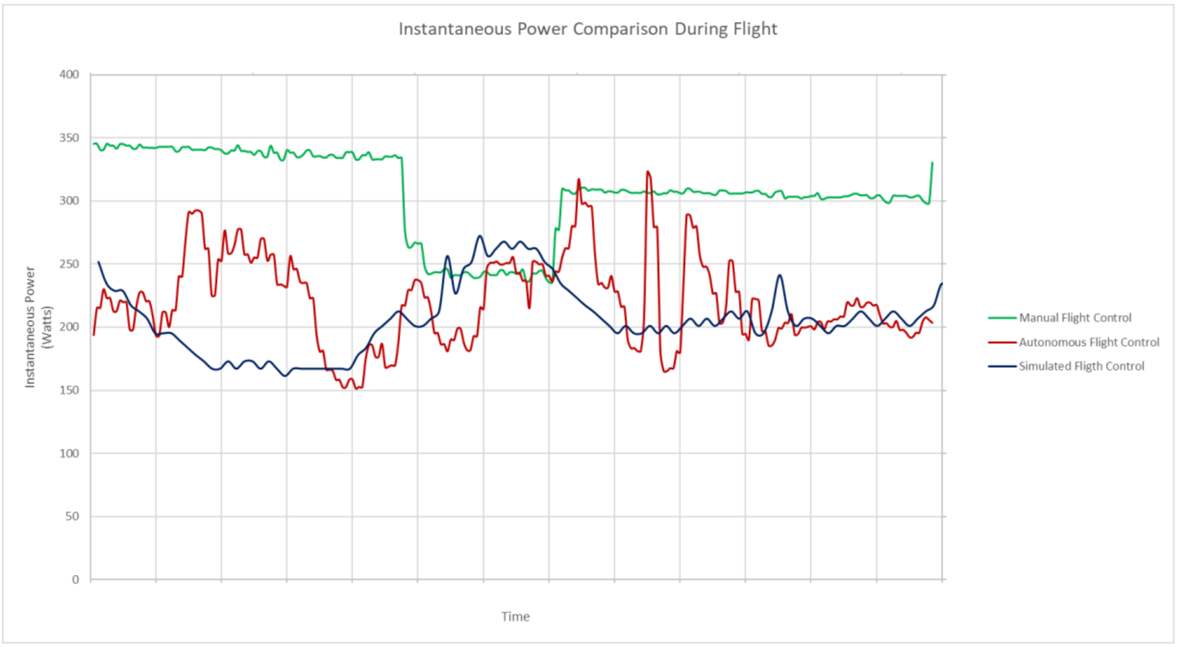


*Figure 8.6: Instantaneous power consumption comparison during flight*

From Figure 8.6 it should be noted that, the data captured for manual flight control of the aircraft was a recording of a non-specific flight path flown by the operator. However, simulated flight control and autonomous flight control executed the same flight plan and their results provide a comparative view as to the simulated and the actual flight control of the aircraft. However, the manual flight control results provided a comparative view of the throttle control of the aircraft, with the additional flight control parameters for the simulated flight control and autonomous flight control being the introduction of an airspeed goal and waypoint goals.

Regarding the manual flight control results gather in Figure 8.6, three flight manoeuvres were noted. For the first five vertical segments of the graph, the aircraft was noted to fly into the wind, for the proceeding two vertical graph segments the aircraft executed a turn and for the remaining vertical segments of the graph, the aircraft flew with a tailwind. Throughout manual flight control, steady throttle control was noted, where no attempt at airspeed maintenance was made. During the first flight manoeuvre, it was noted that the highest instantaneous power was recorder and was believed to be due to need for additional power to be introduced during flight to allow the aircraft to fly through the wind. Upon entering the second flight manoeuvre, the instantaneous power was seen to decrease and then stabilise at the manual flight control's minimum recorded instantaneous power. The decrease in power indicated that upon entering the turn, the aircraft was being accelerated by the wind and as such, less power was required to achieve the desired turn. The stabilisation of the power indicated that the operator was still providing some throttle control to the aircraft during the turn. At the end of the second flight manoeuvre, the power was seen to increase again, this increase indicated that the turn had concluded and that the operator was regaining speed. During the third flight manoeuvre, the power was seen to remain steady, as seen in the first flight manoeuvre, but at a lower level than that of the first flight manoeuvre. The lower power during the final manoeuvre indicated that the operator had attained the desired speed and required less power input than the first flight manoeuvre due to the assistance of the tailwind.

Assessing the simulated flight control and the autonomous flight control, it was noted that the introduction of the mission goals brought about large variations in power control. Two flight manoeuvres were present during the simulated flight control and autonomous flight control, namely: a turn and stabilised flight. The turn was represented by the first eight vertical segments of the graph in Figure 8.6 and the remaining vertical segments of the graph in Figure 8.6 describe the stabilisation of flight. These two flight manoeuvres' flight paths for both the simulated flight control and the autonomous flight control can be seen in Figure 8.7.

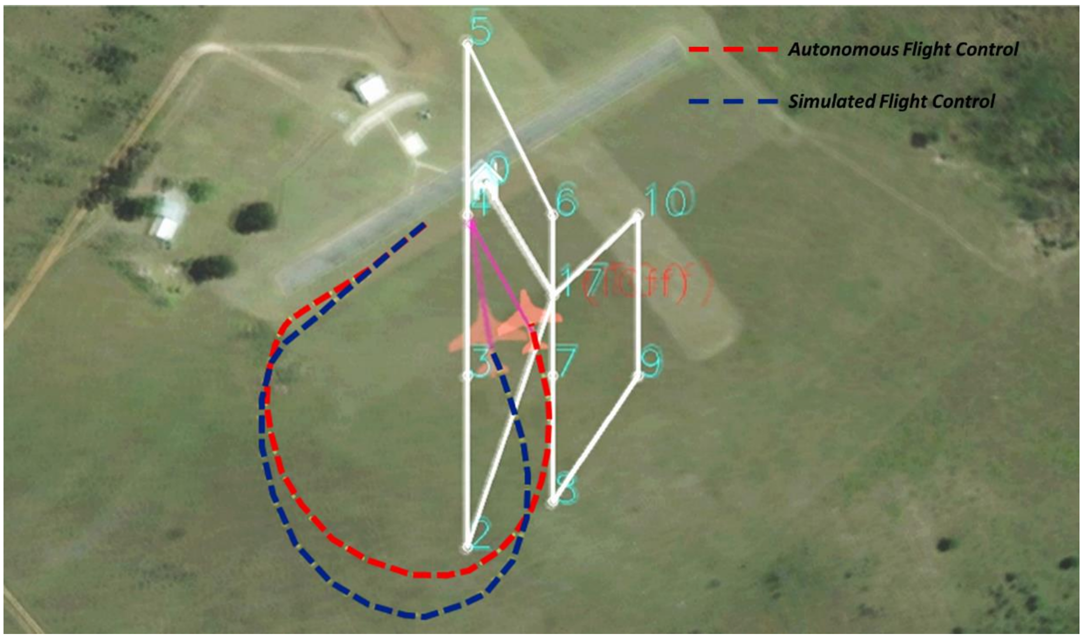


*Figure 8.7: Simulated flight control and autonomous flight control flight path comparison*

From the simulated flight control results from Figure 8.6, it was noted that upon entering the turn, the instantaneous power decreased. This decrease of power during entry of the turn can also be noted in Figure 8.7, as the aircraft took a larger flight path approach, but was able to achieve a sharper turn. This sharper turn allowed for the simulated aircraft to achieve a more linear flight path approach to the waypoints. Upon completion of the turn, the instantaneous power of the simulated flight control was seen to increase. This increase in power was believed to have been due to the aircraft requiring additional power to attain the defined airspeed and waypoint goals, where each waypoint goal also possessed an altitude goal. Entering the second manoeuvre of the flight path, the simulated flight control power was seen to decrease and then stabilise. This reduction in instantaneous power was credited to the simulated control system compensating for the overshoot in power upon leaving the turn.

Looking at the autonomous flight control instantaneous power consumption it was noted that, unlike the simulation, the aircraft increased power when entering the turn. The result of this increased power was noted in Figure 8.7, where the autonomous aircraft overshot the turn and resulted in the autonomous flight control decreasing the power to achieve the turn, as noted in the fourth vertical segment of the graph in Figure 8.6. Upon completing the turn, the autonomous flight control power was seen to increase intermittently, and it was believed that this intermittent increase in power was to ensure the mission goals were met. Entering the second flight manoeuvre, the autonomous flight control was seen to also decrease its power and much like the simulated flight control, this was due to the autonomous flight control overshooting the mission goals. The autonomous flight control power levelled out like in the simulated flight control, but substantial peaks in the power curve were noted. These spikes deduced to be the FCU's control gain values varying from those of the simulated aircraft.

In addition to the results gathered in Figure 8.7 during the comparison of the simulated flight control and the autonomous flight control, Figure 8.8 illustrates the ability of the simulated flight control and autonomous flight control to maintain the altitude goal during the turn and stabilisation.

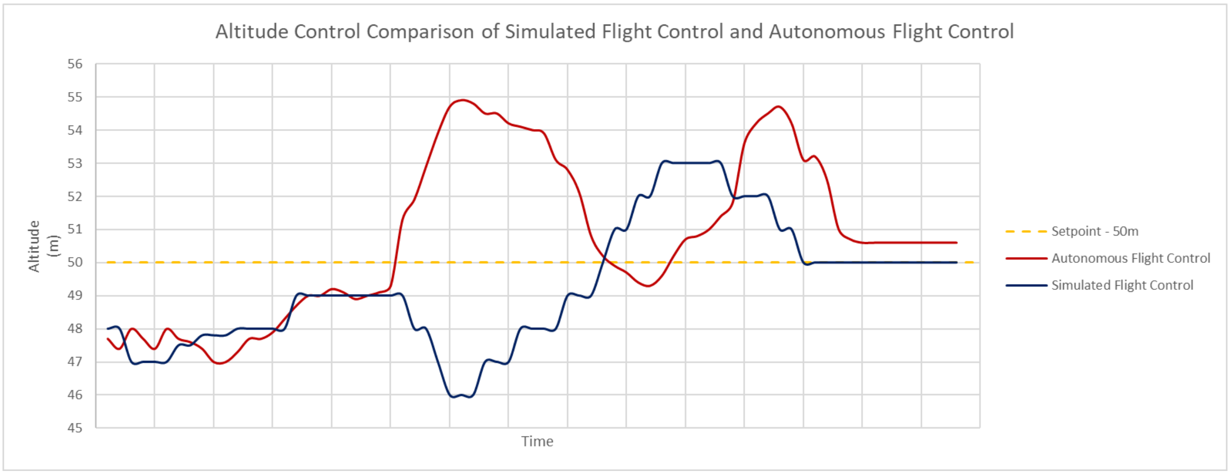


*Figure 8.8: Simulated flight control and autonomous flight control altitude control comparison*

Assessing the resulted captured in Figure 8.8, it was noted that the simulated flight control and autonomous flight control initiated different flight trajectories when entering the turn. The simulated flight control was seen to decrease altitude during the turn approach and then increase altitude during the turn follow through and flight stabilisation. This flight trajectory was complemented by the instantaneous power curve of the simulated flight control, as the power decreased during the turn. The turn flight trajectory altitude control of the autonomous aircraft took the opposite approach and increased altitude when entering the turn. This increase in aircraft altitude was also complemented by its instantaneous power curve from Figure 8.6, as the power was seen to increase. Upon completing the turn, the aircraft was seen to decrease altitude and subsequently the power curve was noted to decrease at this point.

The results captured in Figure 8.6, Figure 8.7 and Figure 8.8 complement one another in terms of visible trends from both a hardware level and a physical level. Variations in the simulation's flight path approach and the autonomous aircraft's flight path could be the result of differing firmware versions. Newer versions of the firmware, including the one uploaded to the autonomous aircraft, could incorporate the use of gravity in assisting with mission waypoint flight paths by allowing the aircraft to glide into the desired airspeed and waypoint goals. Variations in all three flight control functions illustrated the defining differences between one another. Increased throttle control was noted for the simulated aircraft and autonomous aircraft, when compared to the manually piloted aircraft. A similarity between the simulated aircraft and the autonomous aircraft was noted in that both systems were unable to fly the desired missions without overshoot, both horizontally and vertically about the defined waypoints. These overshoots indicated that a level of mission tolerance was necessary when defining waypoints, as overshoots would be present and the need for survey line-up and turnaround waypoints was validated in ensuring the desired survey area would be monitored.

To emphasise the improvements that the autonomous flight control system introduced over the manual flight control, total power consumption of each system during takeoff and flight was assessed, the results of which can be seen in Table 9.1. From the power consumption results gathered in Table 9.1, it was found that the autonomous flight control provided a 14.91% decrease in power consumption during takeoff and a 27.64% decrease in power consumption during flight.

*Table 8.1: Power consumption comparison between manual flight control and autonomous flight control*

| *Power Consumption (mAh)* | | | |
|---|---|---|---|
| | *Manual Flight Control* | *Autonomous Flight Control* | *Difference* |
| Takeoff | 94.64 | 80.53 | 14.11 |
| Flight | 567.53 | 410.66 | 156.87 |

## 8.3 Vision Testing

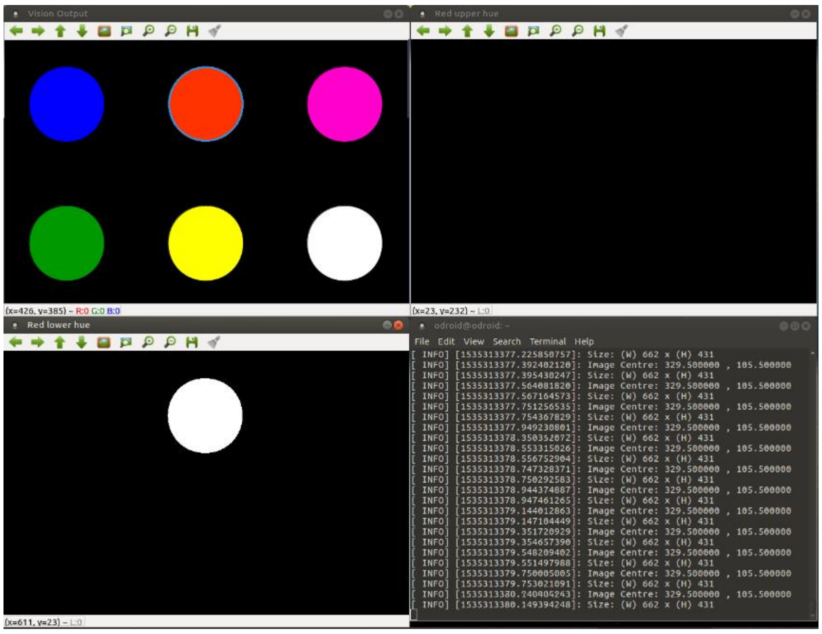
This section covered the onboard and offboard testing and results of the vision system, where both the simulation and physical testing were conducted.

### 8.3.1 Offboard Testing

Prior to integration with the aircraft's onboard hardware, the vision system was developed and tested using both simulated video footage and live video footage. The simulated video footage depicted basic shapes of different colours for algorithm development and progressed to pre-recorded aerial drone footage for theoretical performance testing. Live video footage was used for further system refinement based on differing testing conditions, including lighting effects and moving object recognition and tracking.

#### 8.3.1.1 Simulated Footage Results

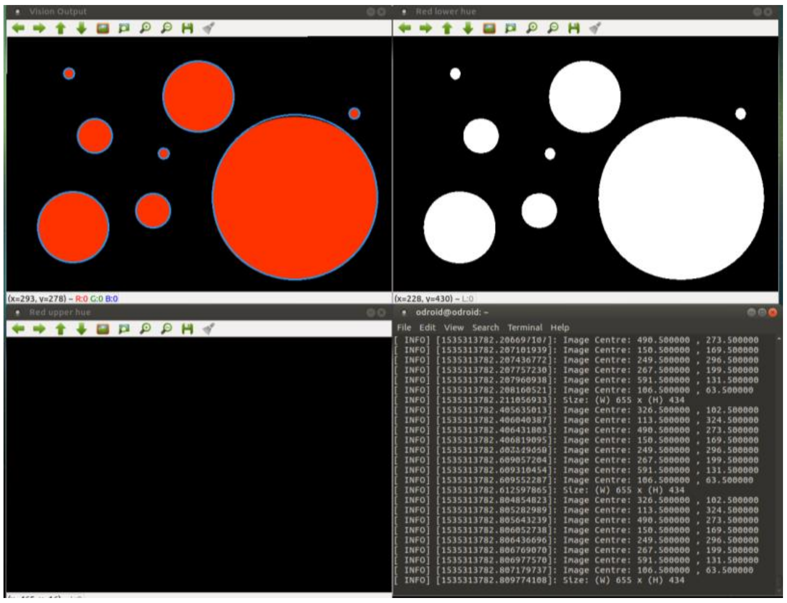
Once an initial vision system had been developed, the first sets of test footage processed were images of basic shapes and colours. These tests deduced the ability of the vision system to detect the proposed red circle drop-zone identifier. The results from the vision system tests were displayed as four-windowed screenshots of each testing scenario, where the upper left window depicted the output image that was processed by the vision system, the upper right window depicted the upper red hue range of the test image, the lower left window depicted the lower red hue range of the test image and the lower right window depicted the image size, in pixels, and the centre coordinates of any detected red circles. Successful detection of circles within the permitted red hue range of the object recognition algorithm were circled by the object recognition algorithm by a blue circle. These initial shape and colour tests can be seen in Figure 8.9 to Figure 8.14.



```
[ INFO] [1535313377.225850757]: Size: (W) 662 x (H) 431
[ INFO] [1535313377.392402120]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313377.395430247]: Size: (W) 662 x (H) 431
[ INFO] [1535313377.564081820]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313377.567164573]: Size: (W) 662 x (H) 431
[ INFO] [1535313377.751256535]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313377.754367829]: Size: (W) 662 x (H) 431
[ INFO] [1535313377.949230801]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313378.350352872]: Size: (W) 662 x (H) 431
[ INFO] [1535313378.553315026]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313378.556752904]: Size: (W) 662 x (H) 431
[ INFO] [1535313378.747328371]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313378.750292583]: Size: (W) 662 x (H) 431
[ INFO] [1535313378.944374887]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313378.947461265]: Size: (W) 662 x (H) 431
[ INFO] [1535313379.144012863]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313379.147104449]: Size: (W) 662 x (H) 431
[ INFO] [1535313379.351720929]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313379.354657390]: Size: (W) 662 x (H) 431
[ INFO] [1535313379.548209402]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313379.551497988]: Size: (W) 662 x (H) 431
[ INFO] [1535313379.750005005]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313379.753021091]: Size: (W) 662 x (H) 431
[ INFO] [1535313380.240405243]: Image Centre: 329.500000 , 105.500000
[ INFO] [1535313380.149394248]: Size: (W) 662 x (H) 431
```

*Figure 8.9: Vision system test scenario 1 - Red circle detection amongst other circles of differing colours*

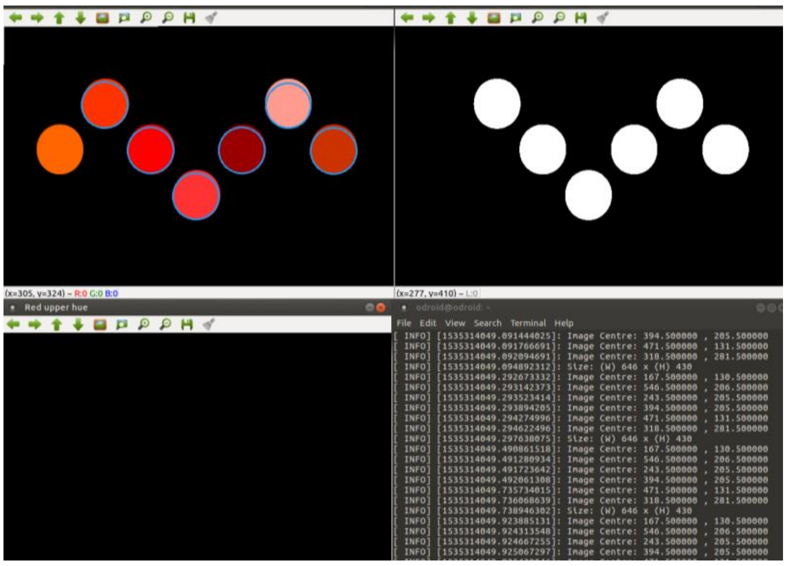
From Figure 8.9, the desired result was to ensure the object recognition algorithm was capable of detecting a red circle amongst other circles of different colours. As can be seen in Figure 8.9, the desired result was achieved. The red circle was filtered through the upper and lower hue filters, allowing only the red circle to appear in the binary image and subsequently be circled in blue by the object recognition algorithm.



```
[ INFO] [1535313782.206697107]: Image Centre: 490.500000 , 273.500000
[ INFO] [1535313782.207101939]: Image Centre: 150.500000 , 169.500000
[ INFO] [1535313782.207436772]: Image Centre: 249.500000 , 296.500000
[ INFO] [1535313782.207757230]: Image Centre: 267.500000 , 199.500000
[ INFO] [1535313782.207960938]: Image Centre: 591.500000 , 131.500000
[ INFO] [1535313782.208160521]: Image Centre: 106.500000 , 63.500000
[ INFO] [1535313782.211056933]: Size: (W) 655 x (H) 434
[ INFO] [1535313782.405635013]: Image Centre: 326.500000 , 102.500000
[ INFO] [1535313782.406040387]: Image Centre: 113.500000 , 324.500000
[ INFO] [1535313782.406431803]: Image Centre: 490.500000 , 273.500000
[ INFO] [1535313782.406819095]: Image Centre: 150.500000 , 169.500000
[ INFO] [1535313782.603149650]: Image Centre: 249.500000 , 296.500000
[ INFO] [1535313782.609057204]: Image Centre: 267.500000 , 199.500000
[ INFO] [1535313782.609310454]: Image Centre: 591.500000 , 131.500000
[ INFO] [1535313782.609552287]: Image Centre: 106.500000 , 63.500000
[ INFO] [1535313782.612597865]: Size: (W) 655 x (H) 434
[ INFO] [1535313782.804854823]: Image Centre: 326.500000 , 102.500000
[ INFO] [1535313782.805282989]: Image Centre: 113.500000 , 324.500000
[ INFO] [1535313782.805643239]: Image Centre: 490.500000 , 273.500000
[ INFO] [1535313782.806052738]: Image Centre: 150.500000 , 169.500000
[ INFO] [1535313782.806436696]: Image Centre: 249.500000 , 296.500000
[ INFO] [1535313782.806769070]: Image Centre: 267.500000 , 199.500000
[ INFO] [1535313782.806977570]: Image Centre: 591.500000 , 131.500000
[ INFO] [1535313782.807179737]: Image Centre: 106.500000 , 63.500000
[ INFO] [1535313782.809774108]: Size: (W) 655 x (H) 434
```

*Figure 8.10: Vision system test scenario 2 - Red circle detection of varying size*

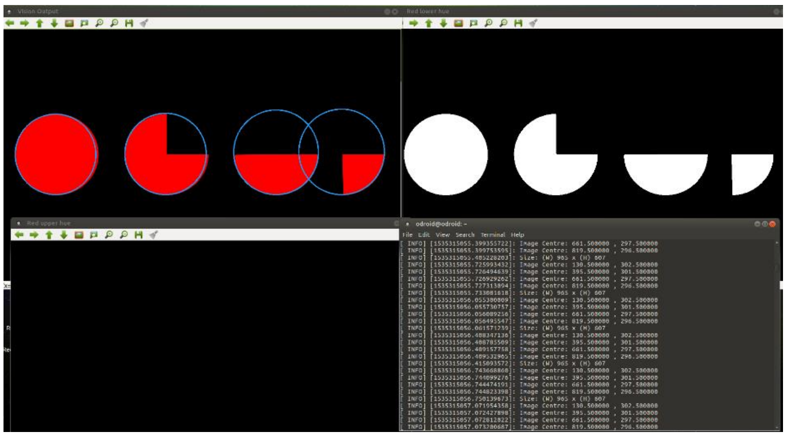
The test conducted in Figure 8.10 was to test the object recognition algorithm's ability to detect multiple red circles simultaneously. These red circles varied in size to test the limits of the algorithm. As seen in Figure 8.10, the desired result was achieved, and the object recognition algorithm was able to detect all eight of the red circles, highlighting them in blue. This result indicated that the algorithm was capable of processing an image in its entirety for any identifiers which met its search criteria. The significance of the algorithm's ability to detect red circles of various sizes was necessary due to the fluctuation in aircraft altitude during flight and as such, the drop-zone identifier would appear larger or smaller respectively.


(x=305, y=324) ~ R:0 G:0 B:0
(x=277, y=410) ~ L:0
Red upper hue
odroid@odroid: ~
File Edit View Search Terminal Help

```
[ INFO] [1535314049.091444025]: Image Centre: 394.500000 , 205.500000
[ INFO] [1535314049.091766691]: Image Centre: 471.500000 , 131.500000
[ INFO] [1535314049.092094691]: Image Centre: 318.500000 , 281.500000
[ INFO] [1535314049.094892312]: Size: (W) 646 x (H) 430
[ INFO] [1535314049.292673332]: Image Centre: 167.500000 , 130.500000
[ INFO] [1535314049.293142373]: Image Centre: 546.500000 , 206.500000
[ INFO] [1535314049.293523414]: Image Centre: 243.500000 , 205.500000
[ INFO] [1535314049.293894205]: Image Centre: 394.500000 , 205.500000
[ INFO] [1535314049.294274996]: Image Centre: 471.500000 , 131.500000
[ INFO] [1535314049.294622496]: Image Centre: 318.500000 , 281.500000
[ INFO] [1535314049.297638075]: Size: (W) 646 x (H) 430
[ INFO] [1535314049.490861518]: Image Centre: 167.500000 , 130.500000
[ INFO] [1535314049.491280934]: Image Centre: 546.500000 , 206.500000
[ INFO] [1535314049.491723642]: Image Centre: 243.500000 , 205.500000
[ INFO] [1535314049.492061308]: Image Centre: 394.500000 , 205.500000
[ INFO] [1535314049.735734015]: Image Centre: 471.500000 , 131.500000
[ INFO] [1535314049.736068639]: Image Centre: 318.500000 , 281.500000
[ INFO] [1535314049.738946302]: Size: (W) 646 x (H) 430
[ INFO] [1535314049.923885131]: Image Centre: 167.500000 , 130.500000
[ INFO] [1535314049.924313548]: Image Centre: 546.500000 , 206.500000
[ INFO] [1535314049.924667255]: Image Centre: 243.500000 , 205.500000
[ INFO] [1535314049.925067297]: Image Centre: 394.500000 , 205.500000
```

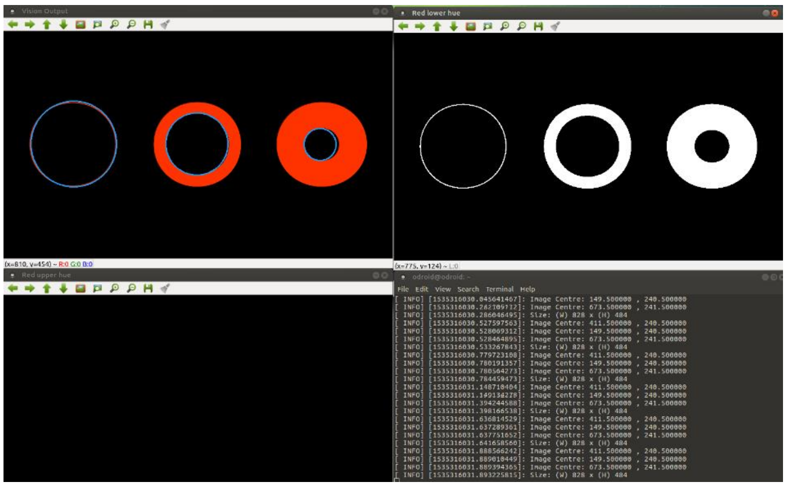
*Figure 8.11: Vision system test scenario 3 - Red circle detection of varying red hue*

Figure 8.11 depicted the results captured when testing the lower hue range of the object recognition algorithm. Seven identifiers of equal size and shape were placed within the processed image. The colour of each circle was varied in the lower red hue range to determine which colours would be detected by the algorithm. The significance of this hue range result was to determine whether objects of similar colour and shape would trigger the detection of the drop-zone, allowing for the hue range to be altered to ensure the correct identifier was detected. Considerations of the acceptable hue range were possible due to the change in appearance of the drop-zone identifier during the survey based on lighting conditions and the relative angle of the drop-zone identifier with respect to the aircraft. The results captured in Figure 8.11 noted that the resulting blue circle did not follow the outline of the identifiers in the test image and this was due to the identifiers not being exact circles, in terms of constant radii. The identifiers were purposefully elongated to test the algorithm's ability to detect the drop-zone when not flying directly above it.



```
[ INFO] [1535315055.399355722]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315055.399753595]: Image Centre: 819.500000 , 296.500000
[ INFO] [1535315055.405228203]: Size: (W) 965 x (H) 607
[ INFO] [1535315055.725993432]: Image Centre: 130.500000 , 302.500000
[ INFO] [1535315055.726494639]: Image Centre: 395.500000 , 301.500000
[ INFO] [1535315055.726929262]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315055.727313894]: Image Centre: 819.500000 , 296.500000
[ INFO] [1535315055.733081618]: Size: (W) 965 x (H) 607
[ INFO] [1535315056.055380809]: Image Centre: 130.500000 , 302.500000
[ INFO] [1535315056.055730757]: Image Centre: 395.500000 , 301.500000
[ INFO] [1535315056.056089256]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315056.056495547]: Image Centre: 819.500000 , 296.500000
[ INFO] [1535315056.061571239]: Size: (W) 965 x (H) 607
[ INFO] [1535315056.408347136]: Image Centre: 130.500000 , 302.500000
[ INFO] [1535315056.408785509]: Image Centre: 395.500000 , 301.500000
[ INFO] [1535315056.409157758]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315056.409532965]: Image Centre: 819.500000 , 296.500000
[ INFO] [1535315056.415093572]: Size: (W) 965 x (H) 607
[ INFO] [1535315056.743668868]: Image Centre: 130.500000 , 302.500000
[ INFO] [1535315056.744099276]: Image Centre: 395.500000 , 301.500000
[ INFO] [1535315056.744474191]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315056.744823398]: Image Centre: 819.500000 , 296.500000
[ INFO] [1535315056.750139673]: Size: (W) 965 x (H) 607
[ INFO] [1535315057.071954358]: Image Centre: 130.500000 , 302.500000
[ INFO] [1535315057.072427898]: Image Centre: 395.500000 , 301.500000
[ INFO] [1535315057.072812822]: Image Centre: 661.500000 , 297.500000
[ INFO] [1535315057.073280687]: Image Centre: 819.500000 , 296.500000
```

*Figure 8.12: Vision system test scenario 4 - Red circle detection of varying circle proportion*



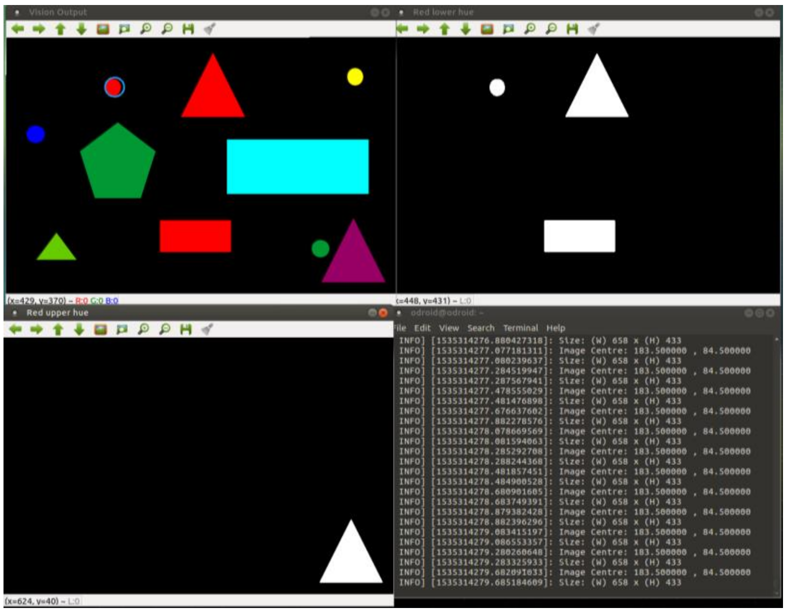
```
[ INFO] [1535316030.045641467]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316030.282709712]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316030.286046495]: Size: (W) 828 x (H) 484
[ INFO] [1535316030.527597563]: Image Centre: 411.500000 , 240.500000
[ INFO] [1535316030.528069312]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316030.528464895]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316030.533267843]: Size: (W) 828 x (H) 484
[ INFO] [1535316030.779723108]: Image Centre: 411.500000 , 240.500000
[ INFO] [1535316030.780191357]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316030.780564273]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316030.784459473]: Size: (W) 828 x (H) 484
[ INFO] [1535316031.148710404]: Image Centre: 411.500000 , 240.500000
[ INFO] [1535316031.149134278]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316031.394244588]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316031.398166538]: Size: (W) 828 x (H) 484
[ INFO] [1535316031.636814529]: Image Centre: 411.500000 , 240.500000
[ INFO] [1535316031.637289361]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316031.637751652]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316031.641658560]: Size: (W) 828 x (H) 484
[ INFO] [1535316031.888566242]: Image Centre: 411.500000 , 240.500000
[ INFO] [1535316031.889010449]: Image Centre: 149.500000 , 240.500000
[ INFO] [1535316031.889394365]: Image Centre: 673.500000 , 241.500000
[ INFO] [1535316031.893225815]: Size: (W) 828 x (H) 484
```

*Figure 8.13: Vision system test scenario 5 - Red circle detection of varying wall thickness*

From Figure 8.12 and Figure 8.13, the functionality of the object recognition algorithm's Hough Circle Transformation was noted. The desired test outcome was to determine how much of the drop-zone identifier had to be present in order to achieve successful detection. The results captured in Figure 8.12 depicted the variation of circle portions, starting from a full circle and ending with a quarter circle, to test circle portion detection. The circle portion detection test results showed that all of the testing images were detected and provided insight as to the *HoughCircles* function's functionality. With the use of the Hough Gradient method, the algorithm was noted to require several edges of a circle to be present, with the same radius from a given centre coordinate. The result in Figure 8.12 contributed to the possible survey environment, where the drop-zone identifier would not need to be fully visible in order for the algorithm to detect it. Practical problems identified, such as shadows and partial concealment of the drop-zone identifier would not render the vision system useless.

In Figure 8.13, with the deduction of how much of the circle's geometry was to be present in order to achieve a successful identification now known, the further testing of what the algorithm defined to be the drop-zone identifier was tested. In the case of Figure 8.13, the drop-zone identifier was converted from a solid circle to a cross-section of a hollow cylinder, donut shape. This new identifier geometry aimed to test the required edge thickness of a red circle in order to achieve detection and as such, the inner circle of the new geometry was coloured black and the edge of the circle was coloured red. During this edge detection test, a consistent result was obtained. Looking at Figure 8.13, this result was the detection of the inner edge of the hollow cylinder's cross-section. The visible result, as displayed in Figure 8.13, alluded to the assumption that a black circle had been detected, but the result was indeed the detection of a red circle. This seemingly incorrect black circle detection was found to be the correct result and the result depicted the algorithm detecting the first red circle edge, providing insight into the functionality of the *HoughCircles* function. This functionality illustrated how the algorithm would allocate a theorised centre coordinate of a circle and work outward from said centre. Upon detection of a pixel meeting the algorithm's search criteria, a radius would be defined between the centre and the pixel. This process of pixel detection would iterate repeatedly until the required number of pixels at the same radius from the centre point were detected. This observation was contradictory, as all solid red circles would possess several internal radii that meet this requirement and as such, it was believed that a secondary detection criterion was present. The pixel detected meeting the necessary radius and colour criteria was required to also be an edge pixel of the circle, either inward or outward of the given radius.

The result captured in Figure 8.13 follows that of the functionality available with the use of the OpenCV Hough Circle Transformation. The detection of the edge colour instead of the entire colour of the identifier was not problematic in this research but a proposed solution to this method of detection would be to execute a secondary algorithm after a circle had been detected. This secondary algorithm could then be used to search about the same centre point for another red circle with a larger diameter than the first. Detection of a second circle about the same centre would indicate the detection of the cross-section of a hollow cylinder and eliminate the detected circle.



```
INFO] [1535314276.880427318]: Size: (W) 658 x (H) 433
INFO] [1535314277.077181311]: Image Centre: 183.500000 , 84.500000
INFO] [1535314277.080239637]: Size: (W) 658 x (H) 433
INFO] [1535314277.284519947]: Image Centre: 183.500000 , 84.500000
INFO] [1535314277.287567941]: Size: (W) 658 x (H) 433
INFO] [1535314277.478555029]: Image Centre: 183.500000 , 84.500000
INFO] [1535314277.481476898]: Size: (W) 658 x (H) 433
INFO] [1535314277.676637602]: Image Centre: 183.500000 , 84.500000
INFO] [1535314277.882278576]: Size: (W) 658 x (H) 433
INFO] [1535314278.078669569]: Image Centre: 183.500000 , 84.500000
INFO] [1535314278.081594063]: Size: (W) 658 x (H) 433
INFO] [1535314278.285292708]: Image Centre: 183.500000 , 84.500000
INFO] [1535314278.288244368]: Size: (W) 658 x (H) 433
INFO] [1535314278.481857451]: Image Centre: 183.500000 , 84.500000
INFO] [1535314278.484900528]: Size: (W) 658 x (H) 433
INFO] [1535314278.680901605]: Image Centre: 183.500000 , 84.500000
INFO] [1535314278.683749391]: Size: (W) 658 x (H) 433
INFO] [1535314278.879382428]: Image Centre: 183.500000 , 84.500000
INFO] [1535314278.882396296]: Size: (W) 658 x (H) 433
INFO] [1535314279.083415197]: Image Centre: 183.500000 , 84.500000
INFO] [1535314279.086553357]: Size: (W) 658 x (H) 433
INFO] [1535314279.280260648]: Image Centre: 183.500000 , 84.500000
INFO] [1535314279.283325933]: Size: (W) 658 x (H) 433
INFO] [1535314279.682091033]: Image Centre: 183.500000 , 84.500000
INFO] [1535314279.685184609]: Size: (W) 658 x (H) 433
```

*Figure 8.14: Vision system test 6 - Red circle detection amongst various shapes of differing shape geometry and colour*

Figure 8.14 illustrates the final of the simulated shape and colour testing scenarios utilised in the development of the vision system. Several geometric shapes of different colours, including circles of different colours, were supplied in the test image. The test aimed to show the object recognition algorithm's ability to detect the red circle amongst various other geometric shapes. The result of the test in Figure 8.14 illustrated the algorithm's ability to filter colours in both the upper and lower hue ranges and through this filtering, detect the red circle. The significance of this test was to replicate possible object geometry surrounding the drop-zone during survey. With the success of the result in Figure 8.14, the testing of the vision system with simulated shapes and colours was concluded, and a movement towards video image processing was taken.

For the second portion of the simulated footage testing of the vision system, aerial footage, recorded on a DJI Phantom 4, of the drone flying over the red circular drop-zone identifier at different speeds and altitude was input into the object recognition algorithm for analysis. This aerial footage included footage from altitudes up to 75 m above the drop-zone identifier, speeds up to 60 km/h (~17 m/s) and included footage where the drop-zone identifier was not within the centre of the camera frame. To adjust for the camera discrepancy between the Phantom 4 and the proposed USB camera, the footage from the fly-overs was compressed to the same size as what was to be expected on the USB camera.

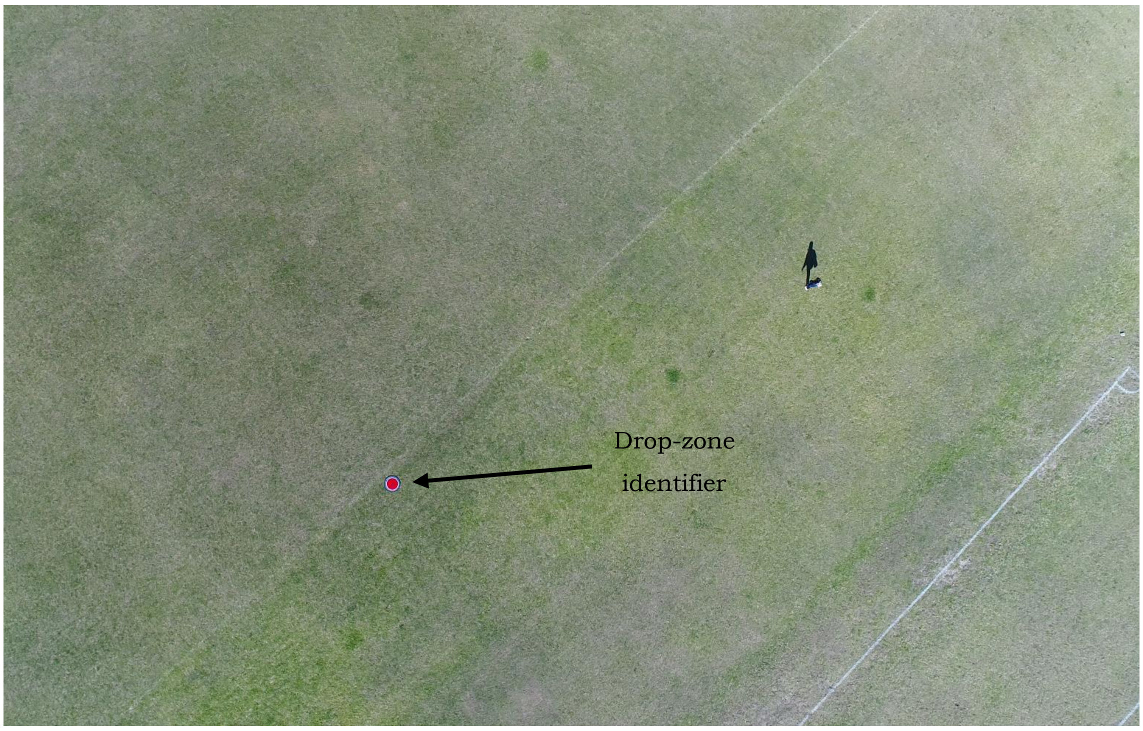


*Figure 8.15: Flight test footage captured from a DJI Phantom 4*

From Figure 8.15, the drop-zone identifier could be seen as the DJI flew overhead. The flight path of the DJI was purposefully set off-centre of the drop-zone identifier to test the object recognition algorithm's ability to detect the drop-zone identifier when not flying directly overhead.

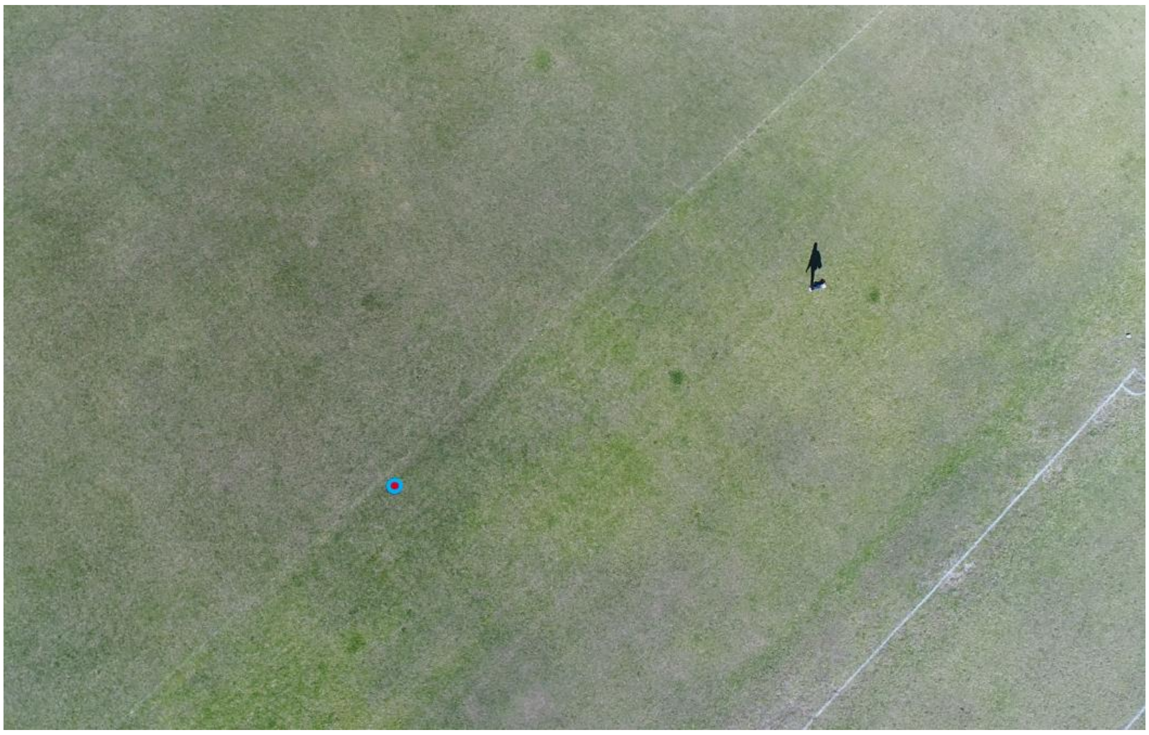


*Figure 8.16: Vision system test 7 – Drop-zone identification with the use of simulated footage recorder on a DJI Phantom 4*

From Figure 8.16, the result of the object recognition algorithm in detecting the drop-zone identifier was noted, with the algorithm circling the drop-zone identifier in blue. As all the recorded footage could not be displayed in these results, the full results obtained with the recorded footage could not be fully illustrated. As such, during the processing of the DJI footage, the displaying of the detected circle's pixel coordinates was not possible due to the large image high resolution of the DJI camera's recording, where the object recognition algorithm struggled to process and overlay the results of the footage at the captured framerate. This lack of coordinate display was not detrimental in the output result of the object recognition algorithm, as the algorithm was still capable of detecting the drop-zone identifier, as seen in Figure 8.16, and the setback to this lack of feedback was for accuracy testing purposes. This limitation of the ODroid's image processing power for larger resolution images validated the use of the selected vision system, as the DJI recorded imagery was in full HD resolution (1920x1080) and the CMOS camera recorded imagery at a resolution of 640x480, resulting in a pixel density ratio of 6.75 less than that of the DJI. As such, the performance of the object recognition algorithm when processing the CMOS camera footage resulted in a far lower processing load for the ODroid. This lower processing requirement was necessary to ensure vision, flight control and payload release computation were possible during flight. The lack of visual processing feedback was also deemed admissible, as the final implementation of the image processing node was executed with all display functionality disabled. Other than the limited processing capabilities of the ODroid being noted, the object recognition algorithm was still able to achieve the desired result in the detection of the drop-zone identifier.

In order to fully test the vision system's capabilities and to refine the system, a final pre-recorded aerial footage test was undertaken. The footage was recorded with the use of the proposed USB camera. During an autonomous mission test, the USB camera was installed into the gimbal onboard the aircraft and the drop-zone identifier was placed within the survey region of the aircraft. The identifier was not placed directly below one of the flight paths, but on the outer limits of what was the theorised camera FOV. Flight parameters included an airspeed of 15 m/s and a flight altitude of 50 m. Upon completion of the flight test, the footage was subjected to the object recognition algorithm. An image captured during the flight test, with the drop-zone identifier indicated, can be seen in Figure 8.17 and the resulting output of the object recognition algorithm can be seen in Figure 8.18.

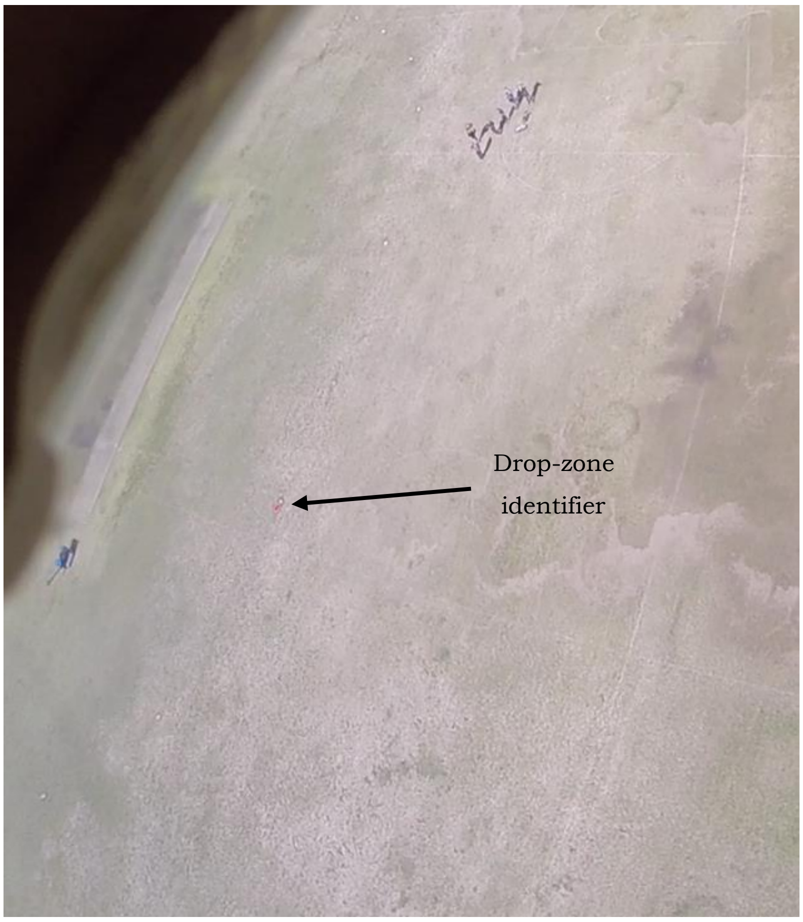


*Figure 8.17: Flight test footage captured from onboard the aircraft with the USB camera*

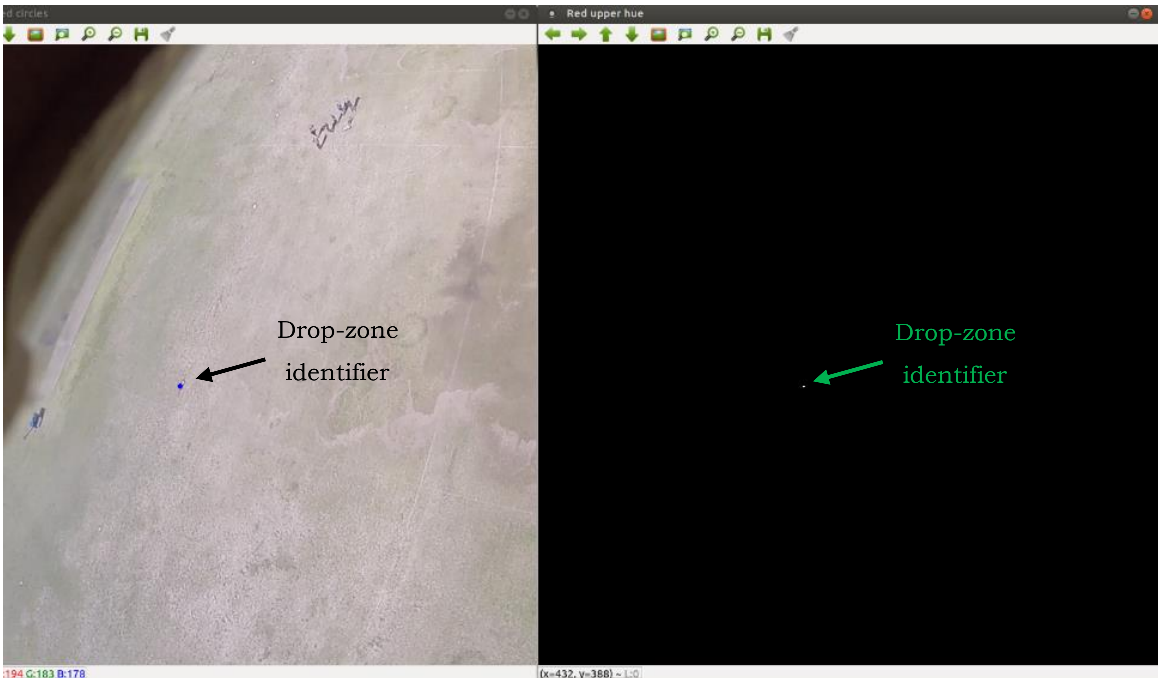


*Figure 8.18: Vision system test 8 – Drop-zone identification from onboard the aircraft (Left – Resulting image overlay, Right – Upper red hue result)*

Although seemingly indistinguishable from the surrounding scenery, the drop-zone identifier can be noted in the centre of the image depicted in Figure 8.17. Figure 8.17 represented the farthest view of the drop-zone identifier from the surveyed flight plan, where the identifier was placed at the edge of the CMOS camera's FOV and on the final survey flight path. The aircraft was also noted to be banking when the footage was captured, as the edge of the camera viewing window was noted in the upper left corner of the image.

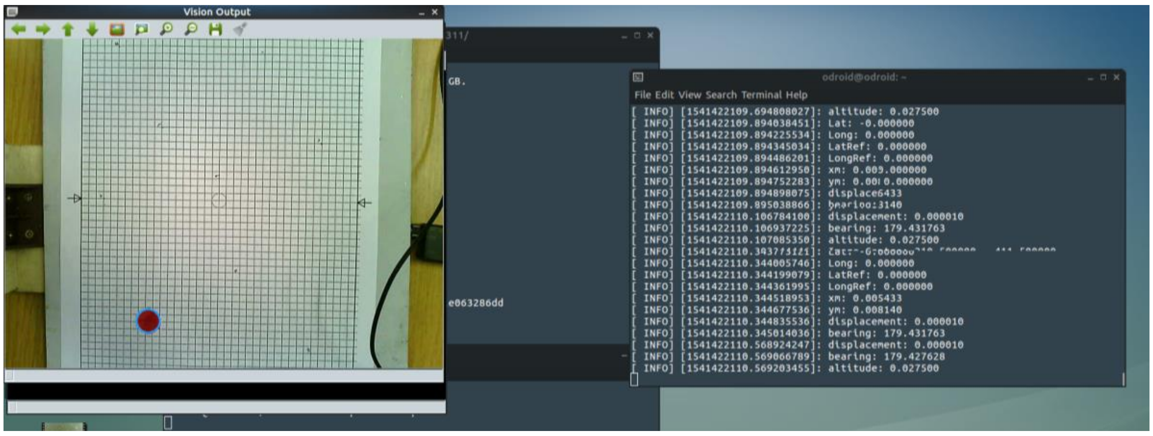
The result captured in Figure 8.18 was the most substantial of all the results gathered from the recorded CMOS camera footage, as the sheer size of the drop-zone identifier in the image alluded to the assumption that the algorithm would fail in its detection. However, the object recognition algorithm was able to successfully identify the drop-zone identifier. The successful detection of the drop-zone identifier was noted in the upper hue filter window in Figure 8.18, where only a few white pixels representing the drop-zone identifier were noted to have been filtered. Through the filtering of the image hue, the algorithm was able to discern the circular pattern of the identifier and the result overlay, depicted in the left-hand window of Figure 8.18, showed the drop-zone identifier being circled in blue.

The detection of such a small drop-zone identifier was noted as a successful result. However, the detection of the identifier whilst the aircraft was banking called for a revaluation of the object recognition algorithm. The concern with the successful result was the possibility that small, incorrect, objects could have been mistaken for the drop-zone and as such, false locations of the drop-zone could have been transmitted. Additionally, the detection of the drop-zone identifier during a bank implied that the coordinate frame of the image plane was not parallel to the ground plane, implying that the translation of pixel coordinates to GPS coordinates would be incorrect. This mismatch of coordinate frames was not true for all bank angles of the aircraft, as the camera gimbal would stabilise the camera for smaller bank angles, but this would not have been the case with steep bank angles. With these issues considered, the object recognition algorithm was calibrated to only detect larger circles by altering the *HoughCircles* function *minRadius* parameter, as seen in Table 5.1. Altering the *minRadius* parameter resulted in successful detection of the drop-zone identifier when the aircraft was in level flight, where the gimbal was able to stabilise the camera, and removed the detection of the drop-zone identifier when the aircraft was banking.

### 8.3.1.2 Live Footage Results

To determine whether the ODroid was capable of executing the object recognition and object tracking algorithms, live footage was streamed from the camera. To deduce the vision system's capabilities, three tests were conducted. The first two tests were conducted in order to calibrate the camera and image processing algorithms and the final test was to calibrate the camera for onboard application.

The first test was used to determine the vision system's accuracy and repeatability. To determine these vision system parameters, a test setup was established with the camera placed perpendicular to a fixed surface at a known distance. A similar testing setup was utilised in the determination of the camera's FOV, with the only difference being the use of a red circle, 16.5 mm diameter, to simulate the drop-zone identifier. The test was conducted to show the accuracy of the vision system by repeatedly placing the red circle at a known location within the image plane. In order to determine the repeatability of the vision system, the accuracy test was repeated at various locations within the image plane. Figure 8.19 depicts a screenshot taken during the vision system accuracy and repeatability testing. On the left-hand side, the red circle can be seen encircled in blue by the image processing algorithms and, on the right-hand side, the determined coordinates of the red circle within the image plane can be seen. The accuracy and repeatability testing template can be seen in Figure 8.20 and the corresponding testing point coordinates can be seen in Table 8.2. The coordinate origin was defined to be the centre of the testing template, indicated by the centre of the circle drawn on the testing template. The first accuracy and repeatability tests were conducted with the camera displaced from the testing template at a distance of 0.275 m.



```
[ INFO] [1541422109.694808027]: altitude: 0.027500
[ INFO] [1541422109.894038451]: Lat: -0.000000
[ INFO] [1541422109.894225534]: Long: 0.000000
[ INFO] [1541422109.894345034]: LatRef: 0.000000
[ INFO] [1541422109.894486201]: LongRef: 0.000000
[ INFO] [1541422109.894612950]: xm: 0.009.000000
[ INFO] [1541422109.894752283]: ym: 0.00 0.000000
[ INFO] [1541422109.894898075]: displace6433
[ INFO] [1541422109.895038866]: bearioo:3140
[ INFO] [1541422110.106784100]: displacement: 0.000010
[ INFO] [1541422110.106937225]: bearing: 179.431763
[ INFO] [1541422110.107085350]: altitude: 0.027500
[ INFO] [1541422110.343771121]: [illegible]
[ INFO] [1541422110.344005746]: Long: 0.000000
[ INFO] [1541422110.344199079]: LatRef: 0.000000
[ INFO] [1541422110.344361995]: LongRef: 0.000000
[ INFO] [1541422110.344518953]: xm: 0.005433
[ INFO] [1541422110.344677536]: ym: 0.008140
[ INFO] [1541422110.344835536]: displacement: 0.000010
[ INFO] [1541422110.345014036]: bearing: 179.431763
[ INFO] [1541422110.568924247]: displacement: 0.000010
[ INFO] [1541422110.569066789]: bearing: 179.427628
[ INFO] [1541422110.569203455]: altitude: 0.027500
```

*Figure 8.19: Vision system accuracy and repeatability testing screenshot*

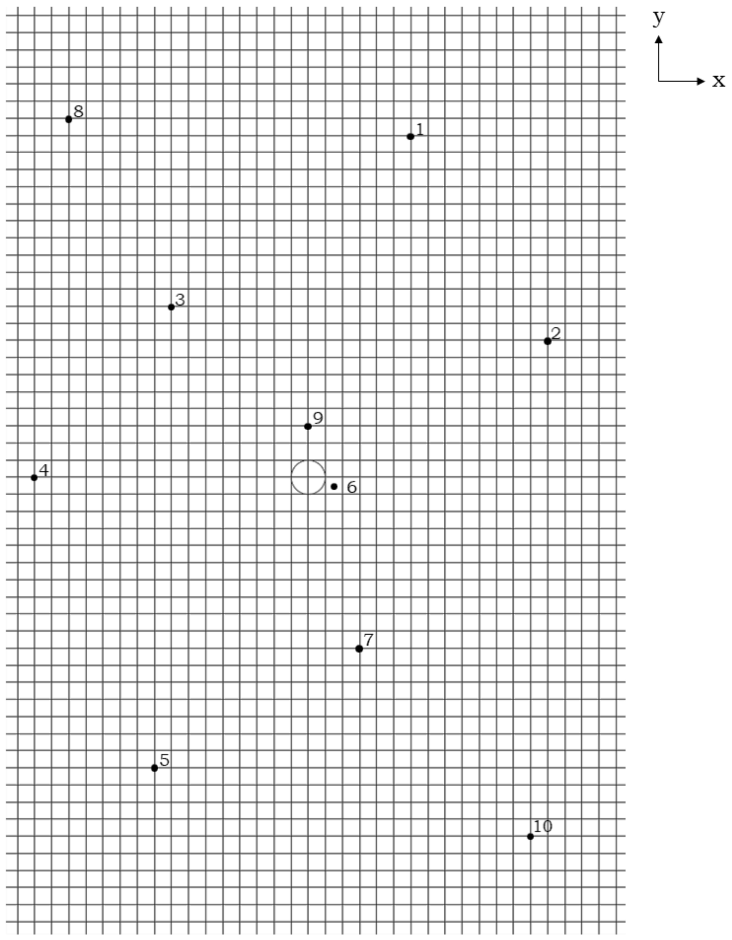


*Figure 8.20: Accuracy and repeatability testing template*

*Table 8.2: Accuracy and repeatability testing point coordinates at 0.275 m camera displacement*

| *Testing Point* | *X Coordinate* | *Y Coordinate* |
|---|---|---|
| 1 | 0.0295 | 0.096 |
| 2 | 0.067 | 0.0385 |
| 3 | -0.0385 | 0.048 |
| 4 | -0.077 | 0 |
| 5 | -0.043 | -0.0815 |
| 6 | 0.007 | -0.0025 |
| 7 | 0.0145 | -0.048 |
| 8 | -0.0675 | 0.101 |
| 9 | 0 | 0.015 |
| 10 | 0.063 | -0.101 |

Figure 8.21 illustrates the results of the accuracy and repeatability testing, where the results were defined relative to their respective testing point coordinates to illustrate the deviation of the results from the reference data.

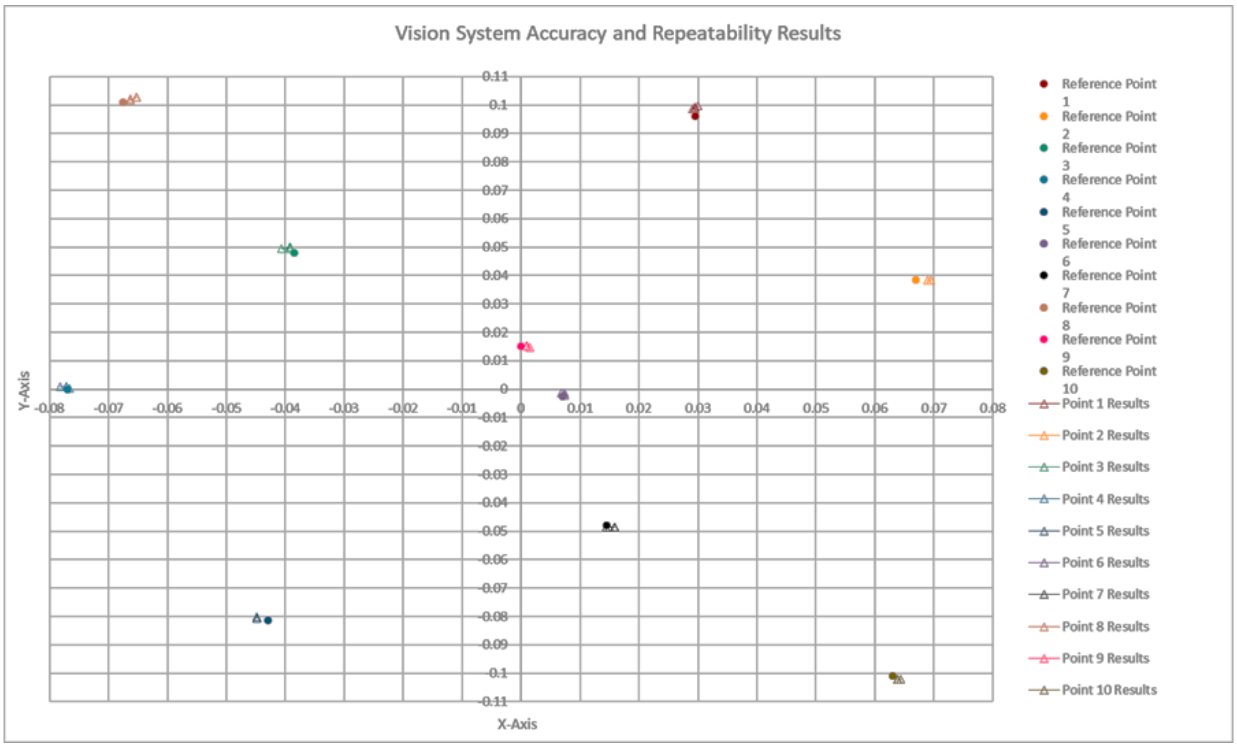


*Figure 8.21: Vision system accuracy and repeatability results on testing template*

From the results gathered in Figure 8.21, the accuracy and repeatability of the vision system was determined. Discrepancies in the accuracy and repeatability results were not immediately considered, as the accuracy of the experimental setup was first to be determined. The theoretical performance of the vision system was defined by determining the design-stage uncertainty of the vision system. This uncertainty determination sought to define the minimum uncertainty in the results obtained. The vision system possessed two components which contributed to the uncertainty in results captured. These two components included the camera and the gimbal. For the purpose of this research, the 3D printed camera mount was assumed to present no error in the results as the additional uncertainty in the mount's design would require an intricate uncertainty analysis of the 3D printer utilised in the manufacture of the mount. The manufacturer specified performance of the camera can be seen in Table 8.3 and the manufacturer specified performance of the gimbal can be seen in Table 8.4.

*Table 8.3: Manufacturer defined performance of the ELP-USB500W04AF-A60 camera*

| *Performance Parameter* | *Value* | *Unit* |
|---|---|---|
| Pixel Size | 1.4×1.4 | µm×µm |
| Resolution | 640×480 | Pixels |
| Field of View | 60(H)×40(V) | Degrees |
| Functional Range | 0.05 - 100 | m |

*Table 8.4: Manufacturer defined performance of the TAROT GoPro 3D V metal 3 axis gimbal*

| *Performance Parameter* | *Value* | *Unit* |
|---|---|---|
| Control Accuracy | ±0.02 | Degrees |
| Controllable Range (PAN) | ±330° | Degrees |
| Controllable Range (TILT) | -135° to +45° | Degrees |

Design-stage uncertainty, as defined by Figliola et al. [18], represented a test to determine the minimum uncertainty to be expected based on available information. As such, design-state uncertainty ($u_d$) was defined by Equation 8.3.

$$u_d = \sqrt{u_0^2 + u_c^2}\ (P\%) \tag{8.3}$$

Where, $u_0$ was the *zero-order uncertainty* which represented the uncertainty of an instrument's measurement result with respect to only the instrument's resolution, as seen in Equation 8.4. $u_c$ represented the *instrument uncertainty* which was the combined uncertainty estimate of all identifier instrument errors [18]. $P\%$ represented the probability confidence interval.

$$u_0 = \frac{1}{2} resolution \tag{8.4}$$

The desired function of the gimbal was to ensure the camera was perpendicular to the ground during flight, the error in the gimbal's motion could be translated from an angular accuracy ($e_{CA}$), represented by the gimbal's control accuracy, to a linear instrument uncertainty based on the aircraft's altitude ($alt$). This uncertainty was represented by the horizontal displacement the gimbal's angular accuracy would represent at altitude. Equation 8.5 defined this gimbal instrument uncertainty.

$$u_c = alt \tan(e_{CA}) \tag{8.5}$$

With the gimbal's uncertainty determined, the camera's zero-order uncertainty could be determined. From Table 8.3, the camera's resolution was defined to be in terms of pixels, but this resolution was not an indication of statistical resolution which was defined to be the smallest detectable change in a measured value. The statistical resolution of the camera was the pixel size and with the gimbal's instrument uncertainty being converted into metres based on the given flight altitude, the resolution of the camera was converted to metres. This conversion of the camera's resolution was also required to incorporate the amount of ground plane area the pixel represented at a given altitude. The area covered by a single pixel at altitude was defined by Equation 8.6.

$$Resolution_{pixel} = \frac{alt \tan\left(\frac{FOV}{2}\right)}{Resolution_{camera}} \tag{8.6}$$

The zero-order uncertainty of the camera was then determined with the use of Equation 8.4. The final design-stage uncertainty of the vision system was then determined with the use of the uncertainties determined and Equation 8.3. The design-stage of the vision system was defined by Equation 8.7 and Equation 8.8, where two equations, horizontal ($y$ axis) and vertical ($x$ axis), were necessary due to the resolution and FOV of the camera being defined in two dimensions.

$$u_{d_x} = \pm 5.70 \times 10^{-4} alt\ m\ (95\%) \tag{8.7}$$

$$u_{d_y} = \pm 8.35 \times 10^{-4} alt\ m\ (95\%) \tag{8.8}$$

With the design-stage uncertainty of the vision system known, the accuracy and repeatability of the vision system could be deduced from the results depicted in Figure 8.21. From the results gathered and with the use of Equation 8.1 and Equation 8.2, the average accuracy of the results gathered in Figure 8.21 were depicted in Table 8.5 for each testing point coordinate defined in Table 8.2.

*Table 8.5: Vision system accuracy testing results*

| *Testing Point* | *Average Error (m)* | | *Average Error Accuracy (%)* | |
|---|---|---|---|---|
| | *X Coordinate* | *Y Coordinate* | *X Coordinate* | *Y Coordinate* |
| 1 | $0.21 \times 10^{-3}$ | $3.10 \times 10^{-3}$ | 0.07 | 1.55 |
| 2 | $2.26 \times 10^{-3}$ | $0.06 \times 10^{-3}$ | 0.71 | 0.03 |
| 3 | $1.00 \times 10^{-3}$ | $1.89 \times 10^{-3}$ | 0.31 | 0.95 |
| 4 | $0.39 \times 10^{-3}$ | $0.63 \times 10^{-3}$ | 0.12 | 0.31 |
| 5 | $1.90 \times 10^{-3}$ | $0.91 \times 10^{-3}$ | 0.60 | 0.46 |
| 6 | $0.29 \times 10^{-3}$ | $0.62 \times 10^{-3}$ | 0.09 | 0.31 |
| 7 | $0.93 \times 10^{-3}$ | $0.65 \times 10^{-3}$ | 0.29 | 0.32 |
| 8 | $1.86 \times 10^{-3}$ | $1.45 \times 10^{-3}$ | 0.59 | 0.73 |
| 9 | $1.09 \times 10^{-3}$ | $0.20 \times 10^{-3}$ | 0.34 | 0.10 |
| 10 | $0.98 \times 10^{-3}$ | $1.04 \times 10^{-3}$ | 0.31 | 0.52 |
| *Overall Average* | $1.09 \times 10^{-3}$ | $1.06 \times 10^{-3}$ | 0.34 | 0.53 |

From the results gather in Table 8.5, the average error of the vision system was deduced to be within approximately 1 mm, both horizontally and vertically, about the defined test points. This error resulted in an average measurement accuracy of 0.34% in the horizontal axis and 0.53% in the vertical axis with respect to their given test point coordinates. The theorised design-state uncertainty for the given altitude of 0.275 m during testing was assessed and an error of $\pm 0.16 \times 10^{-3}$ m was expected in the horizontal axis and an error of $\pm 0.23 \times 10^{-3}$ m was expected in the vertical axis. The discrepancy in design-state uncertainty and the physical results was attributed to the fact that the red circle used for testing was hand-placed during each measurement. Human error and the possibility of the test surface not lying parallel with the camera would have resulted in impaired vision system accuracy.

With the accuracy of the vision system under the given testing parameters deduced, the repeatability of the vision system was deduced. Repeatability was defined to be the ability of a measurement device to produce the same output value during random testing of the same initial testing conditions. Equation 8.9 defined the mean value ($\bar{x}$) attained during testing [18].

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{8.9}$$

Where, $N$ represented the number of datapoints captured and $x_i$ represented the $i^{th}$ datapoint value being assessed. With the mean value for each datapoint known, the standard deviation ($s_x$) of each datapoint, with respect to the mean, was determined and was defined by Equation 8.10 [18].

$$s_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{8.10}$$

From this standard deviation, the variance ($s_x^2$) was determined, where the variance was an indication of the probable measure of the variation found in the data set and was defined by Equation 8.11 [18].

$$s_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{8.11}$$

With these statistical values of the data now defined, the precision interval in which a value at the calculated mean value was predictable within a 95% accuracy was determined with the use of Equation 8.12.

$$x_i = \bar{x} \pm t_{v,P} s_x \ (95\%) \tag{8.12}$$

Where,

$$v = N - 1 \tag{8.13}$$

Where, $v$ represented the degrees of freedom and $t_{v,P}$ represented the t-distribution at a value of $v$ and probability $P$. As such, the value of $t_{v,P}$ was found to be 2.77 as defined by Figliola et al. in *Table 4.4* [18]. As several datapoints were tested for accuracy and repeatability, the results for each datapoint can be seen in Table 8.6 and Table 8.7.

*Table 8.6: Repeatability testing statistics results (mean, standard deviation, variance)*

| *Testing Point* | *Mean Coordinate Value (m)* | | *Standard Deviation (m)* | | *Variance (m)* | |
|---|---|---|---|---|---|---|
| | *X* | *Y* | *X* | *Y* | *X* | *Y* |
| 1 | 29.52 x10$^{-3}$ | 99.10 x10$^{-3}$ | 0.35 x10$^{-3}$ | 0.4 x10$^{-3}$ | 1.23x10$^{-7}$ | 1.57 x10$^{-7}$ |
| 2 | 69.26 x10$^{-3}$ | 38.44 x10$^{-3}$ | 0.27 x10$^{-3}$ | 0.00 x10$^{-3}$ | 0.72 x10$^{-7}$ | 0.00 |
| 3 | -39.20 x10$^{-3}$ | 49.61 x10$^{-3}$ | 0.66 x10$^{-3}$ | 0.26 x10$^{-3}$ | 4.38 x10$^{-7}$ | 0.66 x10$^{-7}$ |
| 4 | -77.25 x10$^{-3}$ | 0.63 x10$^{-3}$ | 0.54 x10$^{-3}$ | 0.21 x10$^{-3}$ | 2.93 x10$^{-7}$ | 0.44 x10$^{-7}$ |
| 5 | -44.90 x10$^{-3}$ | -80.59 x10$^{-3}$ | 0.00 x10$^{-3}$ | 0.21 x10$^{-3}$ | 0.00 | 0.44 x10$^{-7}$ |
| 6 | -7.22 x10$^{-3}$ | -1.88 x10$^{-3}$ | 0.22 x10$^{-3}$ | 0.52 x10$^{-3}$ | 0.49 x10$^{-7}$ | 2.71 x10$^{-7}$ |
| 7 | 15.38 x10$^{-3}$ | -48.65 x10$^{-3}$ | 0.70 x10$^{-3}$ | 0.00 x10$^{-3}$ | 4.93 x10$^{-7}$ | 0.00 |
| 8 | -65.64 x10$^{-3}$ | 102.45 x10$^{-3}$ | 0.55 x10$^{-3}$ | 0.43 x10$^{-3}$ | 3 x10$^{-7}$ | 1.81 x10$^{-7}$ |
| 9 | 1.09 x10$^{-3}$ | 15.09 x10$^{-3}$ | 0.22 x10$^{-3}$ | 0.21 x10$^{-3}$ | 0.5 x10$^{-7}$ | 0.44 x10$^{-7}$ |
| 10 | 63.98 x10$^{-3}$ | -102.04 x10$^{-3}$ | 0.22 x10$^{-3}$ | 0.00 x10$^{-3}$ | 0.48 x10$^{-7}$ | 0.00 |

*Table 8.7: Repeatability testing statistics results (precision interval)*

| *Testing Point* | $x_i = \bar{x} \pm t_{v,P} s_x$ (95%) | |
|---|---|---|
| | *X Coordinate (m)* | *Y Coordinate (m)* |
| 1 | 29.52 ± 0.97 x10$^{-3}$ | 99.10 ± 1.10 x10$^{-3}$ |
| 2 | 69.26 ± 0.74 x10$^{-3}$ | 38.44 ± 0.00 x10$^{-3}$ |
| 3 | -39.20 ± 1.83 x10$^{-3}$ | 49.61 ± 0.71 x10$^{-3}$ |
| 4 | -77.25 ± 1.50 x10$^{-3}$ | 0.63 ± 0.58 x10$^{-3}$ |
| 5 | -44.90 ± 0.00 x10$^{-3}$ | -80.59 ± 0.58 x10$^{-3}$ |
| 6 | -7.22 ± 0.61 x10$^{-3}$ | -1.88 ± 1.44 x10$^{-3}$ |
| 7 | 15.38 ± 1.94 x10$^{-3}$ | -48.65 ± 0.00 x10$^{-3}$ |
| 8 | -65.64 ± 1.52 x10$^{-3}$ | 102.45 ± 1.18 x10$^{-3}$ |
| 9 | 1.09 ± 0.62 x10$^{-3}$ | 15.09 ± 0.58 x10$^{-3}$ |
| 10 | 63.98 ± 0.61 x10$^{-3}$ | -102.04 ± 0.00 x10$^{-3}$ |

From the results gathered in Table 8.6 and Table 8.7, the average precision interval for measurements about the two coordinate axes were defined. For the x-axis, the average precision interval was found to be ±1.03x$10^{-3}$ m and for the y-axis, the precision interval was found to be ±0.62x$10^{-3}$ m. Although the y-axis appeared to possess a more repeatable measurement, a comparison of the precision intervals over the camera's visible area showed that the x-axis precision interval represented 0.32% of the visible area's x-axis dimension. The precision interval of the y-axis represented 0.31% of the visible area's y-axis dimension and as such, the repeatability of both axes was found to be marginally different.

From the results gathered during the vision system's accuracy and repeatability testing, the design-stage uncertainty and precision interval could be extrapolated to determine the performance of the vision system over the camera's functional range. Assuming a linear trend in the calculated vision system performance, the extrapolated vision system's precision intervals can be seen in Figure 8.22. As design-stage uncertainty was defined as a function of altitude, these results could be plotted without the need for extrapolation and can be seen Figure 8.23.

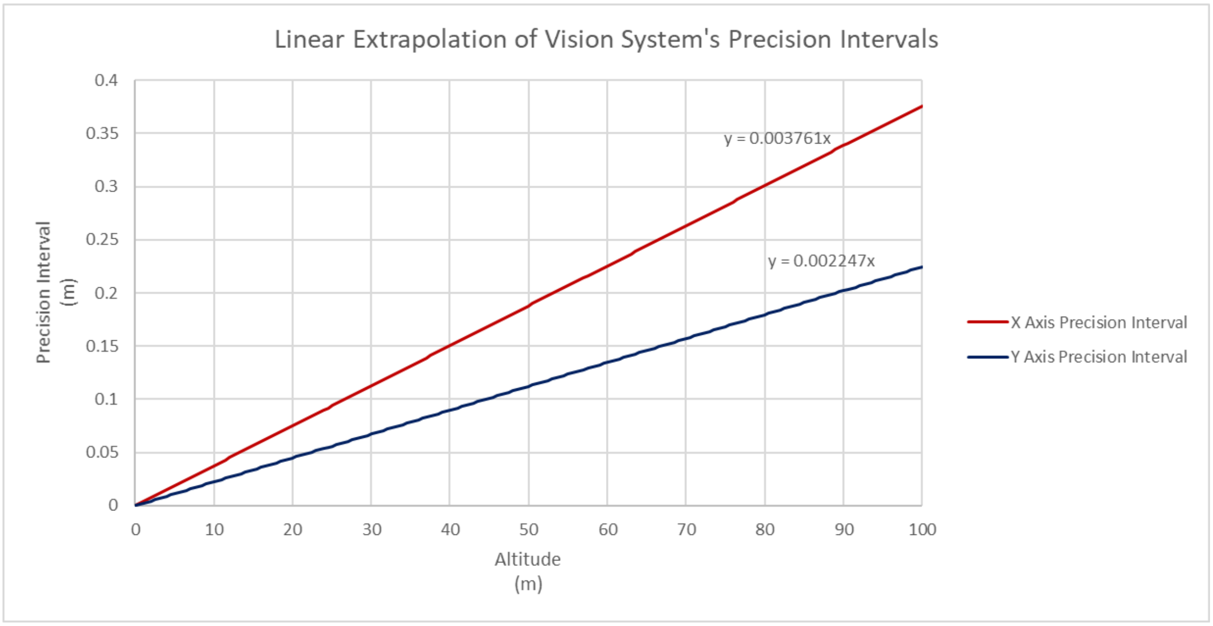


*Figure 8.22: Linear extrapolation of the vision system's precision intervals*

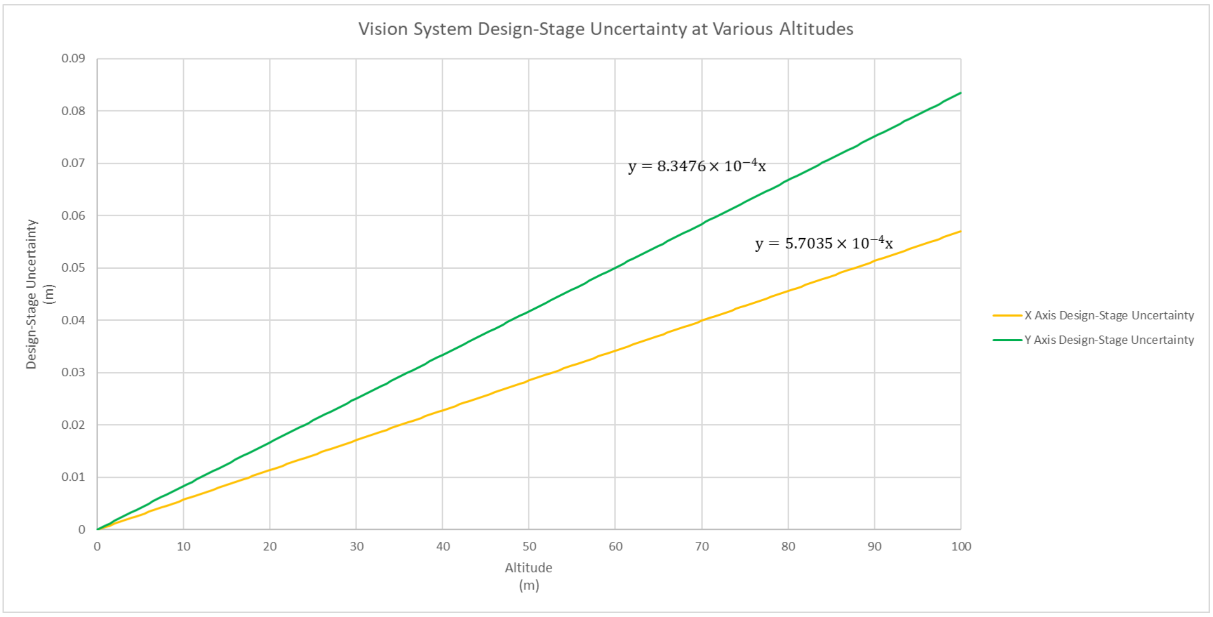
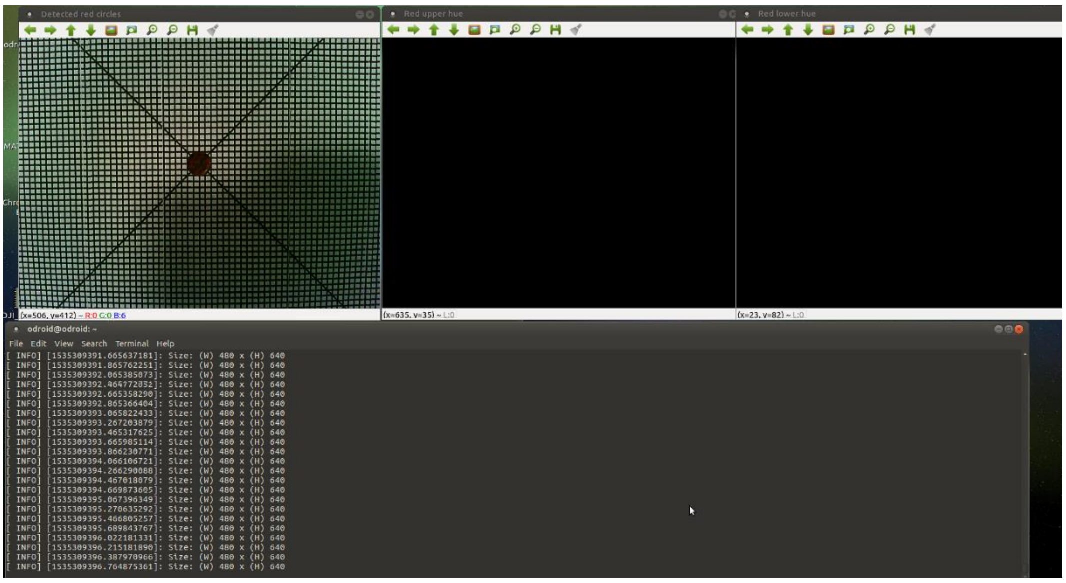


*Figure 8.23: Vision system design-stage uncertainty at various altitudes*

From Figure 8.22 and Figure 8.23, the theorised performance of the vision system could be seen. From Figure 8.22, it was noted that the precision interval of the vision system was found to increase to ±0.38 m about the x-axis and ±0.23 m about the y-axis at an altitude of 100 m, the upper limit of the CMOS camera's functional range. The increase in precision intervals, at the upper limit of the CMOS camera's range, were still within the defined acceptable area of the drop-zone, which was defined to be within a 5 m radius of the drop-zone identifier. Looking at the design-stage uncertainty over the CMOS camera's function range, the linear increase in the design-stage uncertainty was noted, as defined by Equation 8.7 and Equation 8.8. This increased design-stage uncertainty resulted in a maximum uncertainty of ±0.06 m in the x-axis and ±0.08 m in the y-axis. Combining these accuracy and repeatability results, it was concluded that the vision system was fully suited for its desired application in terms of achieving results which were both accurate and repeatable. This deduction about the accuracy and repeatability of the vision system concluded the first live footage testing of the vision system.

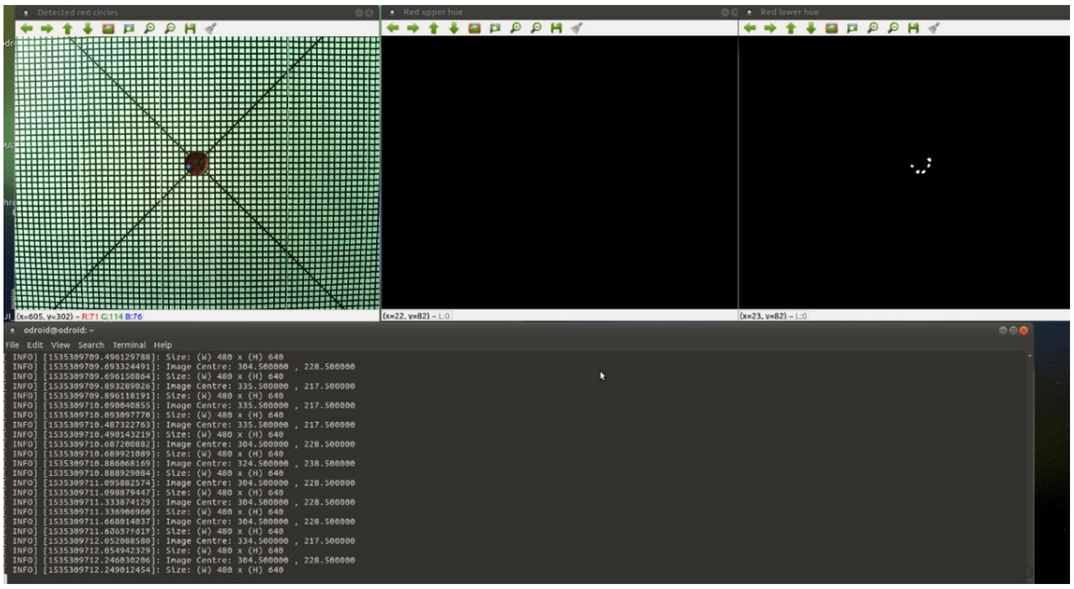
The second live footage testing configuration allowed for the testing of differing lighting effects on the camera, which subsequently further tested the hue range tolerance of the camera. As per the basic shape and colour testing of the vision system, a four-windowed resulting figure described the results of the second live video testing. The upper left window depicted the streamed video footage with the resulting object recognition result superimposed, in the upper middle window the upper red hue range could be seen, the upper middle window depicted the upper red hue range of the live footage, the upper left window depicted the lower red hue range of the live footage and the lower window depicted the algorithm feedback. The image size and the centre coordinates of any detected red circles were defined. The testing environment for this second live footage test was not fully illuminated by natural light and as such, artificial light was utilised in these tests.



```
[ INFO] [1535309391.665637181]: Size: (W) 480 x (H) 640
[ INFO] [1535309391.865762251]: Size: (W) 480 x (H) 640
[ INFO] [1535309392.065385073]: Size: (W) 480 x (H) 640
[ INFO] [1535309392.464772052]: Size: (W) 480 x (H) 640
[ INFO] [1535309392.665358290]: Size: (W) 480 x (H) 640
[ INFO] [1535309392.865366404]: Size: (W) 480 x (H) 640
[ INFO] [1535309393.065822433]: Size: (W) 480 x (H) 640
[ INFO] [1535309393.267203879]: Size: (W) 480 x (H) 640
[ INFO] [1535309393.465317625]: Size: (W) 480 x (H) 640
[ INFO] [1535309393.665985114]: Size: (W) 480 x (H) 640
[ INFO] [1535309393.866230771]: Size: (W) 480 x (H) 640
[ INFO] [1535309394.066106721]: Size: (W) 480 x (H) 640
[ INFO] [1535309394.266290088]: Size: (W) 480 x (H) 640
[ INFO] [1535309394.467018079]: Size: (W) 480 x (H) 640
[ INFO] [1535309394.669873605]: Size: (W) 480 x (H) 640
[ INFO] [1535309395.067396349]: Size: (W) 480 x (H) 640
[ INFO] [1535309395.270635292]: Size: (W) 480 x (H) 640
[ INFO] [1535309395.466805257]: Size: (W) 480 x (H) 640
[ INFO] [1535309395.689843767]: Size: (W) 480 x (H) 640
[ INFO] [1535309396.022181331]: Size: (W) 480 x (H) 640
[ INFO] [1535309396.215181890]: Size: (W) 480 x (H) 640
[ INFO] [1535309396.387970966]: Size: (W) 480 x (H) 640
[ INFO] [1535309396.764875361]: Size: (W) 480 x (H) 640
```

*Figure 8.24: Vision system test 9 – Live video feed of red circle with low lighting*

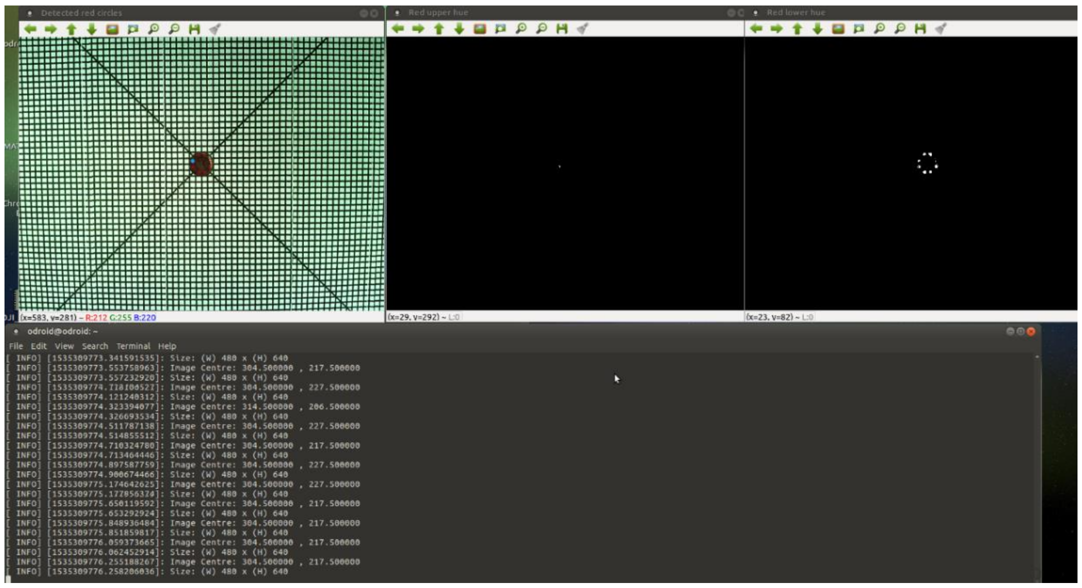
From Figure 8.22, it was noted that when only ambient artificial light was present, the image processing algorithms were unable to detect the red circle on the test template. The ability of the image processing algorithms to detect the red circle during low light conditions was compromised, indicating that application of the delivery system would be dependent on available light.



```
[ INFO] [1535309709.496129788]: Size: (W) 480 x (H) 640
[ INFO] [1535309709.693324491]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309709.696150864]: Size: (W) 480 x (H) 640
[ INFO] [1535309709.893289026]: Image Centre: 335.500000 , 217.500000
[ INFO] [1535309709.896118191]: Size: (W) 480 x (H) 640
[ INFO] [1535309710.090046855]: Image Centre: 335.500000 , 217.500000
[ INFO] [1535309710.093097770]: Size: (W) 480 x (H) 640
[ INFO] [1535309710.487322763]: Image Centre: 335.500000 , 217.500000
[ INFO] [1535309710.490143219]: Size: (W) 480 x (H) 640
[ INFO] [1535309710.687200882]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309710.689921089]: Size: (W) 480 x (H) 640
[ INFO] [1535309710.886068169]: Image Centre: 324.500000 , 238.500000
[ INFO] [1535309710.888929084]: Size: (W) 480 x (H) 640
[ INFO] [1535309711.095882574]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309711.098879447]: Size: (W) 480 x (H) 640
[ INFO] [1535309711.333874129]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309711.336906960]: Size: (W) 480 x (H) 640
[ INFO] [1535309711.668014037]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309711.6[illegible]]: Size: (W) 480 x (H) 640
[ INFO] [1535309712.052088580]: Image Centre: 334.500000 , 217.500000
[ INFO] [1535309712.054942329]: Size: (W) 480 x (H) 640
[ INFO] [1535309712.246030206]: Image Centre: 304.500000 , 228.500000
[ INFO] [1535309712.249012454]: Size: (W) 480 x (H) 640
```

*Figure 8.25: Vision system test 10 – Live video feed of red circle with lighting from a single side*

The same test as seen in Figure 8.24 was repeated with the addition of a single artificial light source illuminating the testing template from the side. The results of this test were noted in Figure 8.25, where it was found that the vision system was able to detect half of the red circle on the testing template. Additionally, it was noted that the introduction of the grid on the test template resulted in the division of the red circle into the respective grid squares they fell within. This separation of circle portions was an indication that the hue filter of the image processing algorithm was not able to detect the darker portions of the circle. The coordinates of each circle detected were noted in the lower window of Figure 8.25.



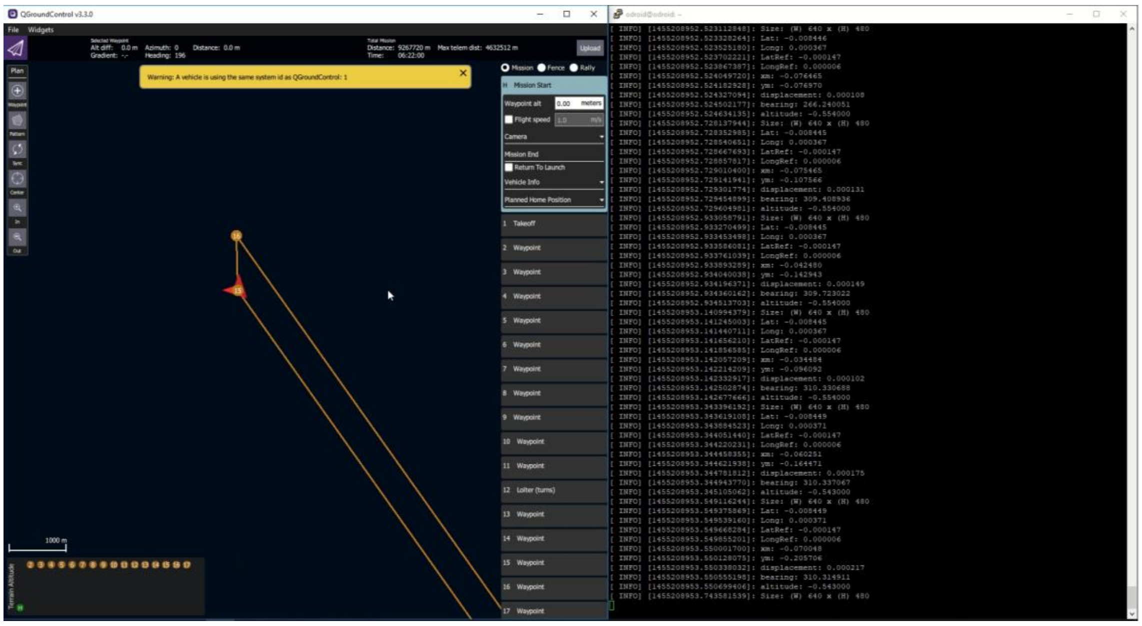
```
[ INFO] [1535309773.341591535]: Size: (W) 480 x (H) 640
[ INFO] [1535309773.553758963]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309773.557232920]: Size: (W) 480 x (H) 640
[ INFO] [1535309774.[illegible]]: Image Centre: 304.500000 , 227.500000
[ INFO] [1535309774.121240312]: Size: (W) 480 x (H) 640
[ INFO] [1535309774.323394077]: Image Centre: 314.500000 , 206.500000
[ INFO] [1535309774.326693534]: Size: (W) 480 x (H) 640
[ INFO] [1535309774.511787138]: Image Centre: 304.500000 , 227.500000
[ INFO] [1535309774.514855512]: Size: (W) 480 x (H) 640
[ INFO] [1535309774.710324780]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309774.713464446]: Size: (W) 480 x (H) 640
[ INFO] [1535309774.897587759]: Image Centre: 304.500000 , 227.500000
[ INFO] [1535309774.900674466]: Size: (W) 480 x (H) 640
[ INFO] [1535309775.174642625]: Image Centre: 304.500000 , 227.500000
[ INFO] [1535309775.177856320]: Size: (W) 480 x (H) 640
[ INFO] [1535309775.650119592]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309775.653292924]: Size: (W) 480 x (H) 640
[ INFO] [1535309775.848936484]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309775.851859817]: Size: (W) 480 x (H) 640
[ INFO] [1535309776.059373665]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309776.062452914]: Size: (W) 480 x (H) 640
[ INFO] [1535309776.255188267]: Image Centre: 304.500000 , 217.500000
[ INFO] [1535309776.258206036]: Size: (W) 480 x (H) 640
```

*Figure 8.26: Vision system test 11 – Live video feed of red circle with full lighting*

From Figure 8.26, the final illumination results were noted where full artificial lighting was introduced. Portioning of the red circle in the grid was still present, indicating that the undetected portions were not within the image processing algorithms' hue ranges. The detected portions of the circle's coordinates were also noted in the lower window of Figure 8.26.

From these live footage illumination tests, it was clear that the vision system required sufficient lighting to detect the drop-zone identifier, further iterating the time-frame of operation of the delivery system. Methods to mitigate these illumination issues were possible, such as multispectral cameras to detect different trace signatures of the drop-zone identifier or the illumination of the drop-zone identifier. With these possible alterations noted, the development of the delivery system with the current system was perused as daytime testing of the delivery system utilised and conceptual proof was the desired outcome of this research. Looking at the CMOS camera's parameters, it was defined that the CMOS camera sensor required 70% illumination for optimal use.

With the camera calibrated for the correct lighting and hue ranges, a final offboard test was conducted to determine whether the image recognition and object tracking algorithms could detect and locate the physical drop-zone identifier. The final offboard vision system test also allowed for the initial calibration of the image tracking algorithm's ability to define the location of the drop-zone identifier in terms of the global frame. The aircraft was carried over the drop-zone identifier to simulate the flight of the aircraft. The results from these tests were displayed as the relocation of the drop-zone identifier's waypoint relative to the simulated aircraft's reference GPS location. The test results were displayed as a screenshot of the GCS map waypoints on the left and a real-time update of the displacement of the drop-zone identifier relative to the reference GPS location and camera frame was displayed on the right, as seen in Figure 8.27. This test represented the first integrated test between vision system and the FCU and made use of the drop-zone identifier which would be used for the application of the integrated system. This final drop-zone identifier was a red circle, 0.4 m in diameter.



```
[ INFO] [1455208952.523112848]: Size: (W) 640 x (H) 480
[ INFO] [1455208952.523328264]: Lat: -0.008446
[ INFO] [1455208952.523525180]: Long: 0.000367
[ INFO] [1455208952.523702221]: LatRef: -0.000147
[ INFO] [1455208952.523867387]: LongRef: 0.000006
[ INFO] [1455208952.524049720]: xm: -0.076465
[ INFO] [1455208952.524182928]: ym: -0.076970
[ INFO] [1455208952.524327094]: displacement: 0.000108
[ INFO] [1455208952.524502177]: bearing: 266.240051
[ INFO] [1455208952.524634135]: altitude: -0.554000
[ INFO] [1455208952.728137944]: Size: (W) 640 x (H) 480
[ INFO] [1455208952.728352985]: Lat: -0.008445
[ INFO] [1455208952.728540651]: Long: 0.000367
[ INFO] [1455208952.728667693]: LatRef: -0.000147
[ INFO] [1455208952.728857817]: LongRef: 0.000006
[ INFO] [1455208952.729010400]: xm: -0.075465
[ INFO] [1455208952.729141941]: ym: -0.107566
[ INFO] [1455208952.729301774]: displacement: 0.000131
[ INFO] [1455208952.729454899]: bearing: 309.408936
[ INFO] [1455208952.729604981]: altitude: -0.554000
[ INFO] [1455208952.933058791]: Size: (W) 640 x (H) 480
[ INFO] [1455208952.933270499]: Lat: -0.008445
[ INFO] [1455208952.933453498]: Long: 0.000367
[ INFO] [1455208952.933586081]: LatRef: -0.000147
[ INFO] [1455208952.933761039]: LongRef: 0.000006
[ INFO] [1455208952.933893289]: xm: -0.042480
[ INFO] [1455208952.934040038]: ym: -0.142943
[ INFO] [1455208952.934196371]: displacement: 0.000149
[ INFO] [1455208952.934360162]: bearing: 309.723022
[ INFO] [1455208952.934513703]: altitude: -0.554000
[ INFO] [1455208953.140994379]: Size: (W) 640 x (H) 480
[ INFO] [1455208953.141245003]: Lat: -0.008445
[ INFO] [1455208953.141440711]: Long: 0.000367
[ INFO] [1455208953.141656210]: LatRef: -0.000147
[ INFO] [1455208953.141856585]: LongRef: 0.000006
[ INFO] [1455208953.142057209]: xm: -0.034484
[ INFO] [1455208953.142214209]: ym: -0.096092
[ INFO] [1455208953.142332917]: displacement: 0.000102
[ INFO] [1455208953.142502874]: bearing: 310.330688
[ INFO] [1455208953.142677666]: altitude: -0.554000
[ INFO] [1455208953.343396192]: Size: (W) 640 x (H) 480
[ INFO] [1455208953.343619108]: Lat: -0.008449
[ INFO] [1455208953.343884523]: Long: 0.000371
[ INFO] [1455208953.344051440]: LatRef: -0.000147
[ INFO] [1455208953.344220231]: LongRef: 0.000006
[ INFO] [1455208953.344458355]: xm: -0.060251
[ INFO] [1455208953.344621938]: ym: -0.164471
[ INFO] [1455208953.344781812]: displacement: 0.000175
[ INFO] [1455208953.344943770]: bearing: 310.337067
[ INFO] [1455208953.345105062]: altitude: -0.543000
[ INFO] [1455208953.549116244]: Size: (W) 640 x (H) 480
[ INFO] [1455208953.549375869]: Lat: -0.008449
[ INFO] [1455208953.549539160]: Long: 0.000371
[ INFO] [1455208953.549668284]: LatRef: -0.000147
[ INFO] [1455208953.549855201]: LongRef: 0.000006
[ INFO] [1455208953.550001700]: xm: -0.070048
[ INFO] [1455208953.550128075]: ym: -0.205706
[ INFO] [1455208953.550338032]: displacement: 0.000217
[ INFO] [1455208953.550555198]: bearing: 310.314911
[ INFO] [1455208953.550699406]: altitude: -0.543000
[ INFO] [1455208953.743581539]: Size: (W) 640 x (H) 480
```

*Figure 8.27: Vision system test 12 – Live video feed testing of the location allocation of the drop-zone identifier*

From Figure 8.27, the successful identification of drop-zone identifier and the allocation of drop-zone location waypoints were noted. In the left window of Figure 8.27, the designation of the drop-zone's location, waypoint 16, relative to the reference GPS location, waypoint 15, was noted to be above and to the left of the aircraft. Looking at the real-time results depicted in the right window of Figure 8.27 confirmed these results, as the reference location, displacement and subsequent calculated location of the drop-zone identifier were noted.

Confirmation of the ability of the vision system to deduce the location of the drop-zone identifier based on the aircraft's GPS location implied that the integrated vision and flight control system was ready for a flight test. As such, this vision system test represented the final offboard test.

## 8.3.2 Onboard Testing

With the vision system and image processing algorithm calibrated, the refinement of the algorithms' integration with the onboard hardware could be completed. This refinement included the determination of the drop-zone identifier's location relative to the global frame with the assistance of the Pixhawk's GPS and compass during flight. To test the integrated vision system and autonomous flight control system, the aircraft was taken to Port Elizabeth Radio Flyers (PERF) airfield for testing and a survey mission was planned over the airfield. The payload release indicator was placed within the survey area and the aircraft's MAVROS node, autonomous flight node, image capture node and the image processing node were executed. Successful detection of the drop-zone identifier was indicated by the vision system's ability to allocate the drop-zone waypoint (waypoint 15) accordingly. The results of the onboard test were depicted in Figure 8.28, where in the left window, the initial survey waypoint allocation could be noted and in the right window, the vision system allocated waypoints were noted after a mission update request.

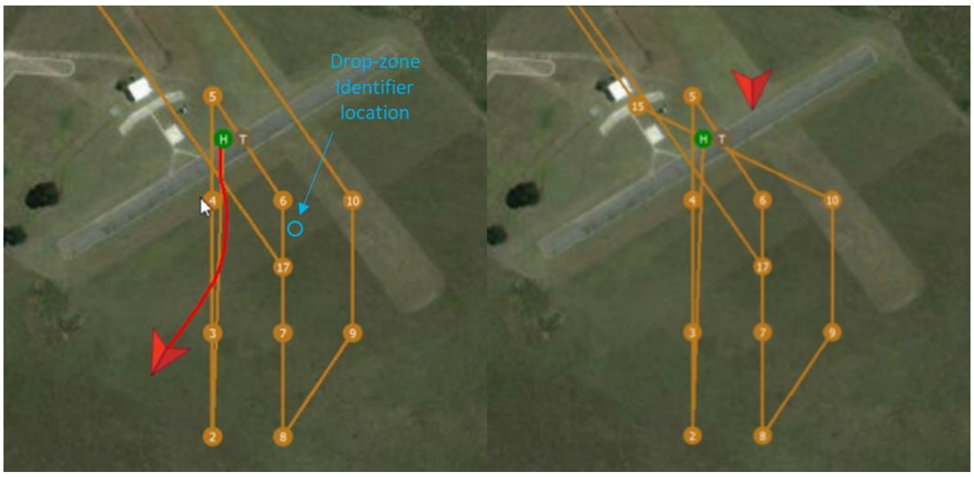


*Figure 8.28: GCS mission map view of the first onboard integrated testing of the vision system and autonomous flight control system*

Comparing the two window views in Figure 8.28, it was noted that the allocation of the drop-zone location waypoint was present after requesting a mission waypoint update. The drop-zone location waypoint, waypoint 15, can be seen in the top left-hand corner of the right window in Figure 8.28. Prior to any deductions being made based on the result obtained, it should be noted that the drop-zone identifier was placed in region between waypoint 6 and waypoint 17. It was noted that the designated drop-zone location waypoint was placed in the wrong location. Upon assessment of the airfield, it was found that an orange coloured windsock was present at the airfield's clubhouse and was the location specified by the vision system to be the drop-zone. From this result, several deductions were made, the first being that the vision system had successfully detected an object meeting its required drop-zone classification parameters and had successfully allocated a waypoint indicating said drop-zone's location. The remaining deductions were that the image processing algorithm's parameters were in need of refinement as the aircraft did fly over the correct drop-zone indicator without detection, indicating incorrect hue range filtering. Additionally, the detection of the windsock indicated that the *HoughCircles* function's *minRadius* and *maxRadius* parameters required refinement, as the windsock's geometry was detected. Although not the desired result of the first onboard test of the integrated vision system and autonomous flight control, the result displayed in Figure 8.28 was a clear indicator that the integrated systems were functioning correctly and were relaying information between one another.

After the correction of the necessary algorithm parameters, the onboard test was ready to be repeated. Two alterations to the physical mission were changed during this second test, namely: the mission was changed from the traditional survey to an accumulation of waypoints not within the vicinity of the drop-zone indicator and the drop-zone indicator was placed further away from the airfield clubhouse. The reason for the change in mission waypoints was due to the autonomous flight control of the aircraft in executing a mission survey had already been defined and the desired result was to test the vision system's integration with FCU. The movement of the drop-zone indicator away from the clubhouse was to avoid the detection of any other object's other than the drop-zone identifier. The aircraft was then flown over the newly placed and recorded location of the drop-zone identifier and the success of the vision system's ability to define the drop-zone's location was assessed. The positioning of the mission waypoints and placement location of the drop-zone identifier prior to the second test can be seen in the left window of Figure 8.29. The vision system's reallocation of the drop-zone location waypoint can be seen in right window of Figure 8.29. Reallocation of the mission waypoints required the new designation of the drop-zone location waypoint due to the structure of the autonomous flight node. Therefore, the drop-zone location was defined by waypoint 17. Additionally, with the mission waypoints reallocated, the location of the headwind loiter, and initial payload release approach waypoint were also reallocated upon detection of the drop-zone identifier and these waypoints were defined to be waypoint 15 and waypoint 16. The aircraft was flown at an average altitude of 75 m and an average airspeed of 12 m/s.

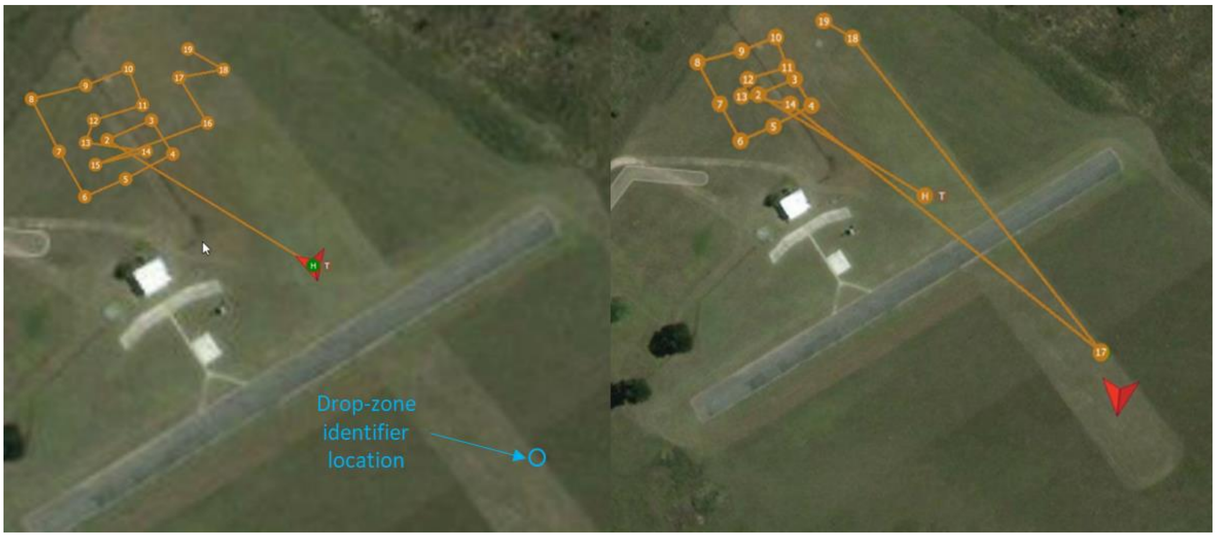


*Figure 8.29: GCS mission map view of the second onboard integrated testing of the vision system and autonomous flight control system*

During the second onboard integrated test, the aircraft was flown over the approximate drop-zone identifier several times and from Figure 8.29, it was noted that upon mission waypoint update, the location of the drop-zone had been redefined. Visual comparison of both the right and left windows of Figure 8.29, indicated that the vision system was successfully able to identify and define the drop-zone location based on the aircraft's GPS location. To confirm the visual comparison's results, the vision system's image processing node logs were gathered for the computation of statistical results. As discussed in Chapter 7, the use of the ROS node development configuration prevented any form of node feedback to the operator and as such, the recorded locations defined by the vision system were assessed post-mission. Additionally, the vision system could record and transmit the location of the drop-zone identifier several times in a single fly over, as the refresh rate of the CMOS camera presented the image processing algorithm with new images every few milliseconds. As such, accuracy and repeatability tests were conducted. The vision system defined drop-zone location information retrieved from the image processing node can be seen in Table 8.8, Figure 8.30 and Figure 8.31.

Table 8.8: Vision system defined coordinates of the drop-zone identifier

| | Latitude | Longitude | Displacement (m) |
|---|---|---|---|
| Drop-Zone Identifier True Coordinates | -33.8761795 | 25.3518254 | - |
| | Vision System Determined Coordinates | | |
| 1 | -33.8761765 | 25.3518152 | 1.17 |
| 2 | -33.8761795 | 25.3518492 | 1.07 |
| 3 | -33.8761743 | 25.3518397 | 1.34 |
| 4 | -33.8761782 | 25.3518191 | 0.63 |
| 5 | -33.8761664 | 25.3518191 | 1.58 |
| 6 | -33.8761966 | 25.3518365 | 2.3 |
| 7 | -33.8761927 | 25.3518223 | 1.49 |
| 8 | -33.8761882 | 25.3518094 | 1.99 |
| 9 | -33.8761861 | 25.3518333 | 1.01 |
| 10 | -33.8761585 | 25.3518207 | 2.49 |
| 11 | -33.8761795 | 25.3518223 | 0.37 |
| 12 | -33.8761795 | 25.3518286 | 0.34 |
| 13 | -33.8761822 | 25.3518318 | 0.55 |
| 14 | -33.8761808 | 25.3518333 | 0.7 |
| 15 | -33.8761835 | 25.3518397 | 1.31 |
| 16 | -33.8761848 | 25.3518159 | 0.99 |
| 17 | -33.8761927 | 25.3518238 | 1.48 |
| 18 | -33.8761611 | 25.3518096 | 2.59 |
| 19 | -33.8761848 | 25.3518508 | 2.34 |
| 20 | -33.8761785 | 25.3518397 | 1.29 |
| 21 | -33.8761680 | 25.3518380 | 1.61 |

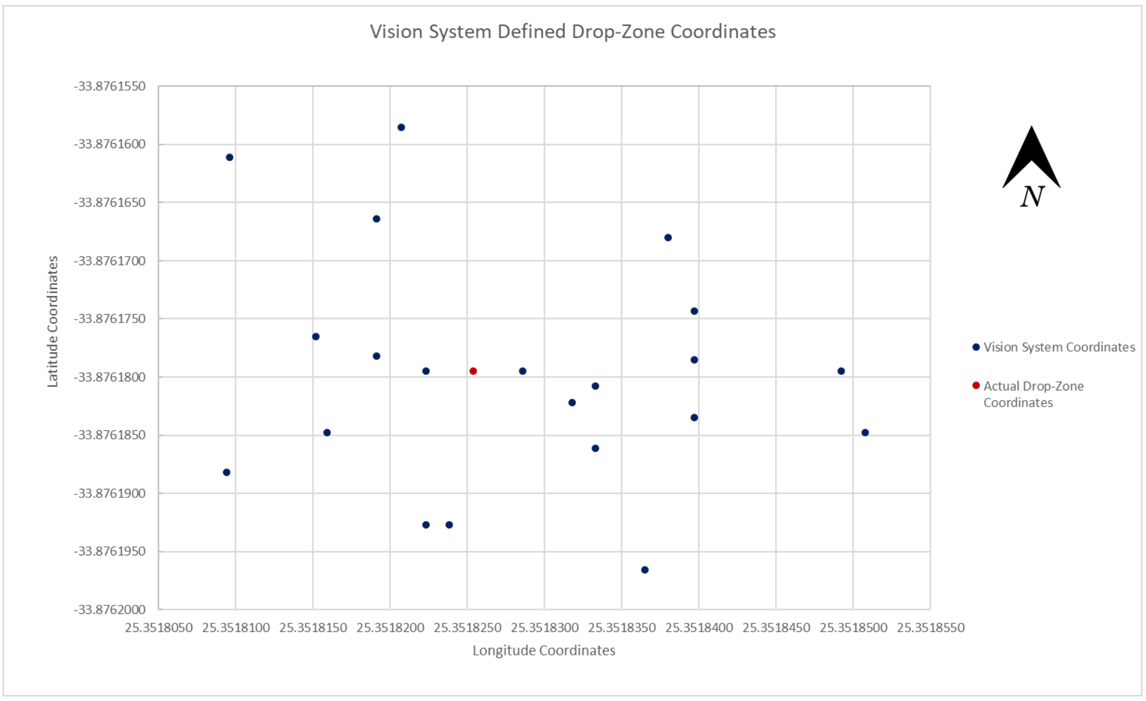


*Figure 8.30: Vision system defined drop-zone identifier locations relative to the true drop-zone location*

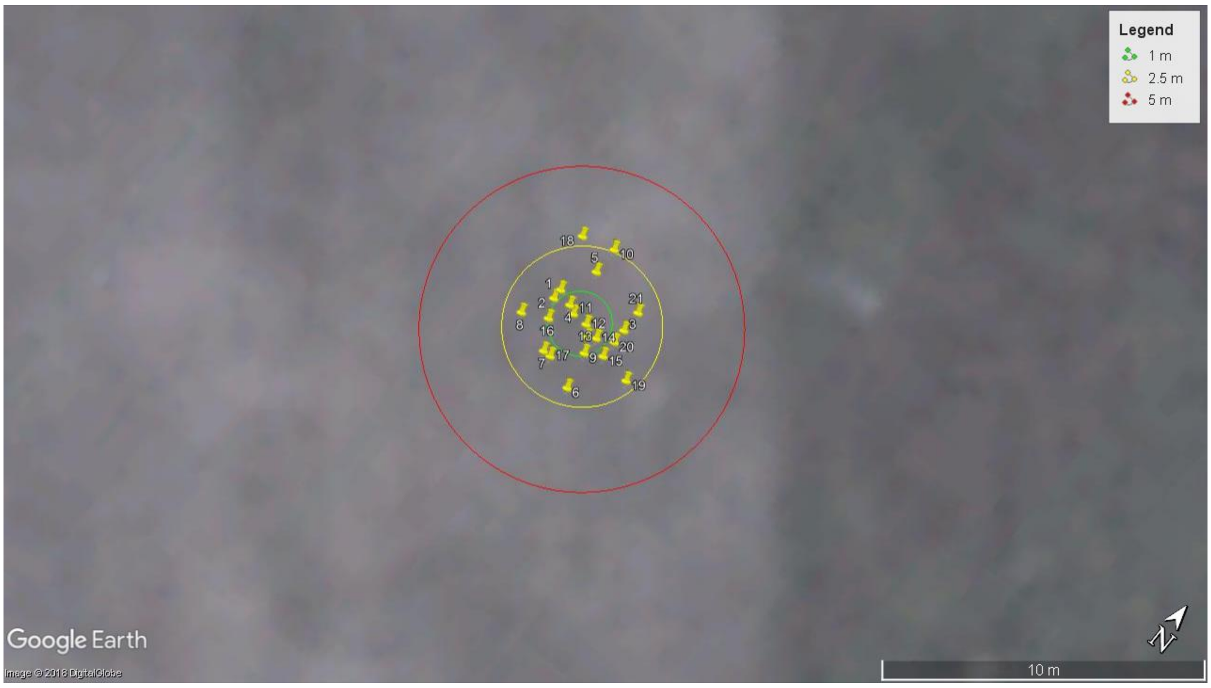


*Figure 8.31: Vision system defined drop zone identifier locations relative to acceptable drop-zone area*

From Table 8.8, it was noted that the vision system was able to determine 21 drop-zone coordinates during the second integrated test. Of these 21 locations, 6 were within a 1 m radius of the true location of the placed drop-zone identifier, with the closest determined location calculated to be within 0.34 m of the true location, location 12 in Table 8.8. Of the remaining 15 locations, 14 were determined by the vision system within a 2.5 m radius of the true location, with the farthest recorded location placed at 2.59 m from the true location, location 18 in Table 8.8. Figure 8.30 was utilised to show the distribution of the vision system-defined locations relative to the true location. Figure 8.31 was used to illustrate the location coordinates with respect to the allowable drop-zone area and was constructed with the use of Google Earth Pro.

From the results displayed in Figure 8.30 and Figure 8.31, no discernible pattern to the vision system's identification could be defined. The statistical accuracy and repeatability of the vision system was determined and through the use of Equation 8.9, Equation 8.10 and Equation 8.11, the repeatability results determined for the second onboard test can be seen in Table 8.9.

*Table 8.9: Repeatability results for second onboard integrated test (mean, standard deviation, variance)*

| | *Mean Value* | *Standard Deviation* | *Variance* |
|---|---|---|---|
| *Latitude (Degrees)* | -33.8761796 | 9.9 x10-6 | 0.98x10-10 |
| *Longitude (Degrees)* | 25.3518285 | 12.1 x10-6 | 1.45x10-10 |
| *Displacement (m)* | 1.36 | 0.68 | 0.46 |

From the data captured in Table 8.9, the precision interval of the mean was deduced, where the precision interval of the mean made use of the standard deviation of the mean as opposed to the standard deviation. The significance of the standard deviation of the mean showed an accurate representation of how well a measurement would be defined over an entire dataset when only a small portion of sample data was provided. The standard deviation of the mean was defined by Equation 8.14.

$$s_{\bar{x}} = \frac{s_x}{\sqrt{N}} \tag{8.14}$$

With the use of Equation 8.14, the possible values of the true mean were determined by Equation 8.15. The resulting values can be seen in Table 8.10 [18].

$$x' = \bar{x} \pm t_{v,P} s_{\bar{x}} \; (P\%) \tag{8.15}$$

*Table 8.10: Possible values of the true mean for the results gathered for the second integrated test*

| | *Possible values of the true mean* |
|---|---|
| *Latitude (Degrees)* | -33.88 ± 4.51 x10$^{-6}$ |
| *Longitude (Degrees)* | 25.35 ± 5.51 x10$^{-6}$ |
| *Displacement (m)* | 1.36 ± 0.31 |

From Table 8.9 and Table 8.10, the latitude and longitude results defined the repeatability of the second integrated test, as their coordinates were specific to the testing area. The defining result from the second integrated test was the repeatability of the displacement of the vision system defined drop-zone locations. The average displacement from the true drop-zone location was found to be 1.36 m with an uncertainty of ±0.31 m. From the largest predicted displacement of the vision system's measurements, as extrapolated in Figure 8.22, it was found that the vision results should have resulted in the detection of the drop-zone identifier within 0.32 m of the true location. Introducing the greatest design-stage uncertainty from Figure 8.23, for the given testing altitude, would have resulted in an additional 0.08 m of displacement from the true drop-zone location. From the results gathered in Table 8.10, the resulting average displacement introduced approximately 1 m of unpredicted deviation from the desired location. Additionally, assuming the possible value of the mean to shift towards the true location by the given precision in Table 8.10, the resulting average was still approximately 0.7 m greater than the predicted value. Several issues could have resulted in this deviation but due to the majority of the results captured in Table 8.8 occurring between the 1 m radius mark and the 2.5 m radius mark it was deduced that the uncertainty of the vision system could not be assumed to be the same linear trend defined in Figure 8.22. A new linear trend was defined for the displacement uncertainty of the vision system's measurements and built upon the results captured in Figure 8.22. This new trend can be seen in Figure 8.32.

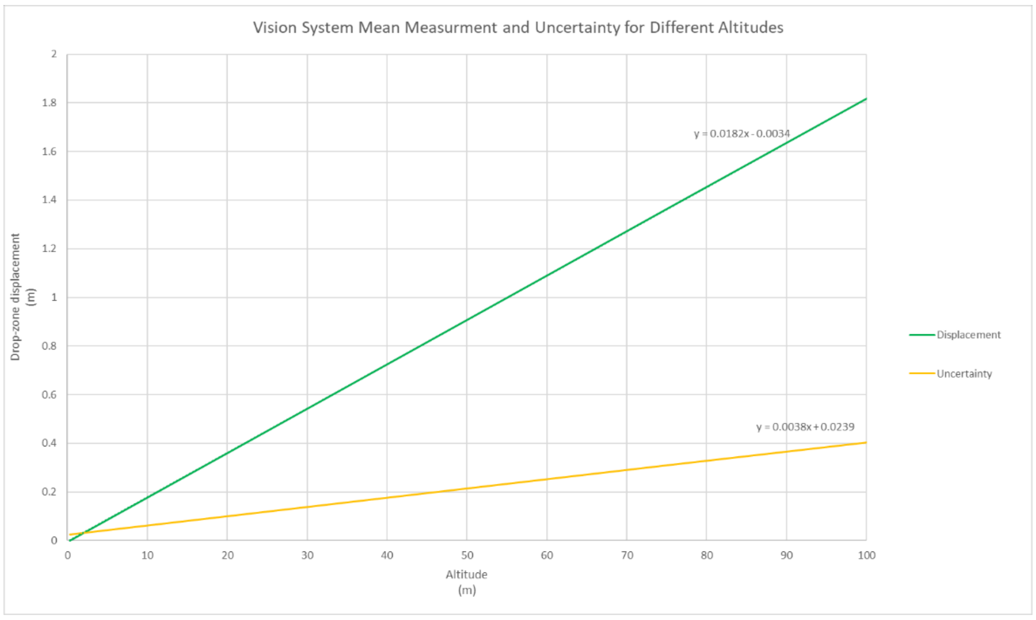


*Figure 8.32: Vision system mean measurement displacement uncertainty for different altitudes*

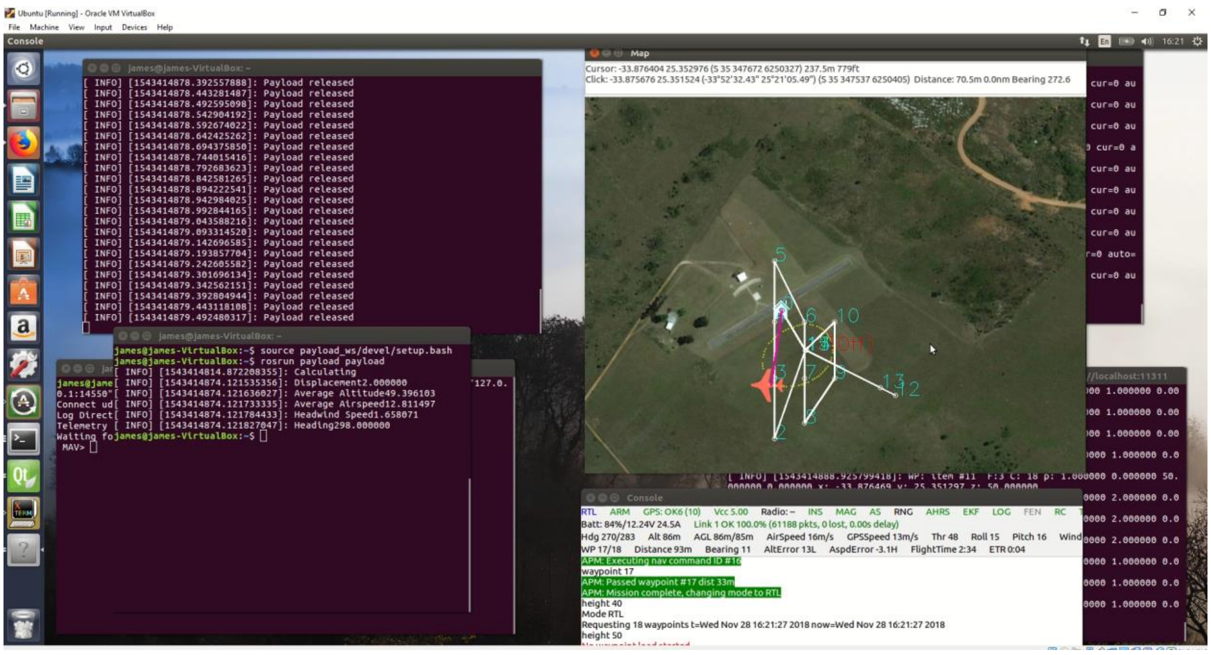
With the final mean and uncertainty in the vision system's displacement measurements from the true drop-zone location defined, it was noted that at the upper limit of the CMOS camera's operation range, the measurement uncertainty was still within the defined allowable drop-zone area. The maximum vision system mean displacement and uncertainty at 1.82 ± 0.40 m at an altitude of 100 m. The final performance of the integrated system would be defined by the additional uncertainty that the payload release trajectory would provide.

## 8.4 Payload Release Testing

Simulated and physical testing of the payload release algorithm and payload release mechanism were discussed in this section.

### 8.4.1 Simulation

To determine the baseline results expected from the payload release system, payload release trajectory simulations were conducted. The simulations made use of the lookup tables and equations developed, as these described the same computational information available to the payload release node during flight. The simulations were conducted with the use of the MAVProxy SITL and a MAVProxy HITL, where the location of the drop-zone was simulated to have been detected by the vision system. The simulation tested the payload release node algorithm's ability to determine the headwind velocity and direction, and placing of the correct payload release waypoints. The HITL aspect of the payload release simulation was the introduction of the Pixhawk and payload release servo into the simulation, where visual confirmation of payload release was obtained through the actuation of the payload release servo. The SITL simulation of the payload release node can be seen in Figure 8.33.



```
[ INFO] [1543414878.392557888]: Payload released
[ INFO] [1543414878.443281487]: Payload released
[ INFO] [1543414878.492595098]: Payload released
[ INFO] [1543414878.542904192]: Payload released
[ INFO] [1543414878.592674022]: Payload released
[ INFO] [1543414878.642425262]: Payload released
[ INFO] [1543414878.694375850]: Payload released
[ INFO] [1543414878.744015416]: Payload released
[ INFO] [1543414878.792683623]: Payload released
[ INFO] [1543414878.842581265]: Payload released
[ INFO] [1543414878.894222541]: Payload released
[ INFO] [1543414878.942984025]: Payload released
[ INFO] [1543414878.992844165]: Payload released
[ INFO] [1543414879.043588216]: Payload released
[ INFO] [1543414879.093314520]: Payload released
[ INFO] [1543414879.142696585]: Payload released
[ INFO] [1543414879.193857704]: Payload released
[ INFO] [1543414879.242605582]: Payload released
[ INFO] [1543414879.301696134]: Payload released
[ INFO] [1543414879.342562151]: Payload released
[ INFO] [1543414879.392804944]: Payload released
[ INFO] [1543414879.443118108]: Payload released
[ INFO] [1543414879.492480317]: Payload released
```

```
james@james-VirtualBox:~$ source payload_ws/devel/setup.bash
james@james-VirtualBox:~$ rosrun payload payload
[ INFO] [1543414814.872208355]: Calculating
[ INFO] [1543414874.121535356]: Displacement2.000000
[ INFO] [1543414874.121636027]: Average Altitude49.396103
[ INFO] [1543414874.121733335]: Average Airspeed12.811497
[ INFO] [1543414874.121784433]: Headwind Speed1.658071
[ INFO] [1543414874.121827047]: Heading298.000000
james@james-VirtualBox:~$
```

```
MAV>
```

```
Cursor: -33.876404 25.352976 (S 35 347672 6250327) 237.5m 779ft
Click: -33.875676 25.351524 (-33°52'32.43" 25°21'05.49") (S 35 347537 6250405)  Distance: 70.5m 0.0nm Bearing 272.6
```

```
RTL  ARM  GPS: OK6 (10)  Vcc 5.00  Radio: --  INS  MAG  AS  RNG  AHRS  EKF  LOG  FEN  RC
Batt: 84%/12.24V 24.5A  Link 1 OK 100.0% (61188 pkts, 0 lost, 0.00s delay)
Hdg 270/283  Alt 86m  AGL 86m/85m  AirSpeed 16m/s  GPSSpeed 13m/s  Thr 48  Roll 15  Pitch 16  Wind
WP 17/18  Distance 93m  Bearing 11  AltError 13L  AspdError -3.1H  FlightTime 2:34  ETR 0:04
APM: Executing nav command ID #16
waypoint 17
APM: Passed waypoint #17 dist 33m
APM: Mission complete, changing mode to RTL
height 40
Mode RTL
Requesting 18 waypoints t=Wed Nov 28 16:21:27 2018 now=Wed Nov 28 16:21:27 2018
height 50
```

Figure 8.33: SITL simulation of the payload release node

From the SITL simulation of the payload release node, successful computation of the required payload release approach and payload release waypoint was achieved. The payload release algorithm, through application of the lookup tables, was able to interpolate the simulated flight data and to determine a theorised release displacement. From the simulation it was found that the aircraft achieved: an average altitude of 49.4 m, an average airspeed of 12.81 m/s and an average headwind speed of 1.67 m/s from a heading of 298˚. With these averages, the algorithm was able to determine that a location of 2 m before the drop-zone identifier's determined location was the correct moment for payload release. Upon inspection of the lookup tables, as seen in Appendix J1 and Appendix J2, the displacement determined by the algorithm was within the calculated displacement's predicted by the mathematical model described in Chapter 6. The updated waypoint map from the simulated payload release can be seen in Figure 8.34, where the payload release location (waypoint 14) can be seen next to the determined drop-zone identifier's location (waypoint 15).

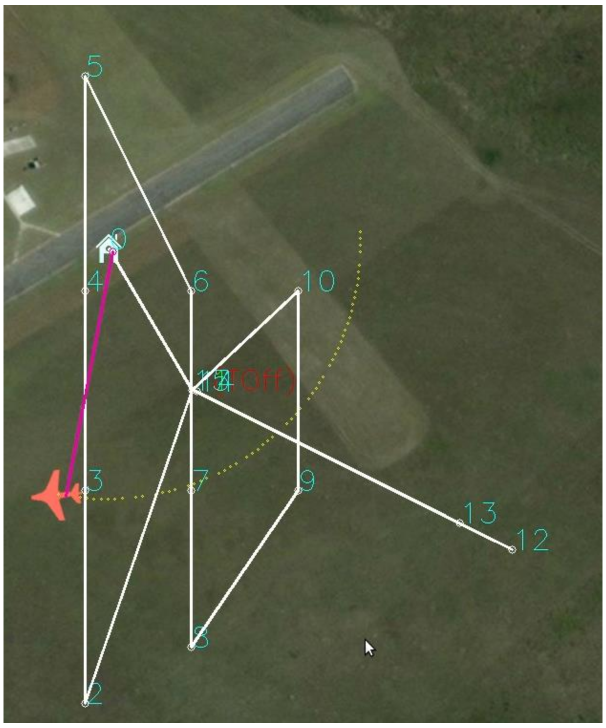


*Figure 8.34: Updated waypoint map indicating the payload release waypoints*

The simulation was conducted under several headwind velocities and headings, and the resulting placements of the payload release waypoints followed the success of the results depicted in Figure 8.33 and Figure 8.34. From these simulations, it was determined that the payload release node was functioning correctly, and physical testing was needed to provide an indication as to the success of the payload release system as a whole.

## 8.4.2 Physical Payload Release

Although the simulation provided an indication of the locations for payload release, they did not simulate the payloads fall trajectory. As such, integrated payload releases were conducted onboard the aircraft. These physical payload release tests were to conclude the accuracy of the payload release algorithm by dropping the payload during flight and determining their final displacement relative to the point of release. Aircraft parameters such as: altitude, airspeed, groundspeed and GPS location were recorded. Additionally, the payload's impact location was also recorded. With these parameters, the payload's impact location versus the simulation predicted location was deduced and the resulting error noted. The parameters recorded a payload release test can be seen in Table 8.11.

*Table 8.11: Payload release test parameters*

| *Parameter* | *Value* | *Unit* |
|---|---|---|
| Release Altitude | 76.2 | m |
| Aircraft Airspeed | 12.6 | m/s |
| Aircraft Groundspeed | 8.6 | m/s |
| Headwind Velocity | 4 | m/s |
| Aircraft GPS Location | Latitude: -34.005355<br>Longitude: 25.682660 | Degrees |
| Payload Impact Location | Latitude: -34.005369<br>Longitude: 25.682628 | Degrees |

From the recorded parameters recorded in Table 8.11, the payload was found to enter both Phase A and Phase B of the payload release trajectory described in Chapter 6. The deduction that the payload would enter both release trajectory phases was made with the use of Equation 6.19, where the equilibrium time of the payload was deduced to be 1.65 seconds. With the equilibrium time known, Equation 6.21 was used to determine the total fall time of the payload from the release altitude and was determined to be 3.94 seconds. Due to the payload possessing a larger total fall time than equilibrium time, the payload was found to fall for an additional 2.29 seconds under the effects of headwind vector addition. At the point of equilibrium, the payload was found to have displaced 1.96 m, as determined with the use of Equation 6.20. Additionally, the payload equilibrium altitude was found to be 44.37 m and the Phase B displacement was found to be 5.29 m. The resulting final theoretical payload displacement was found to be 3.33 m behind the initial release location, where forward was defined to be a heading of 63˚. Comparing the theoretical displacement of the payload with the actual displacement of the payload, it was found that the physical payload release resulted in a payload displacement of 3.47 m behind and to the initial release location. As such, the physical payload release of the payload resulted in a payload displacement error of 0.14 m from the theorised location and was within the acceptable area of the drop-zone. The payload release trajectory for the physical testing can be seen in Figure 8.35.

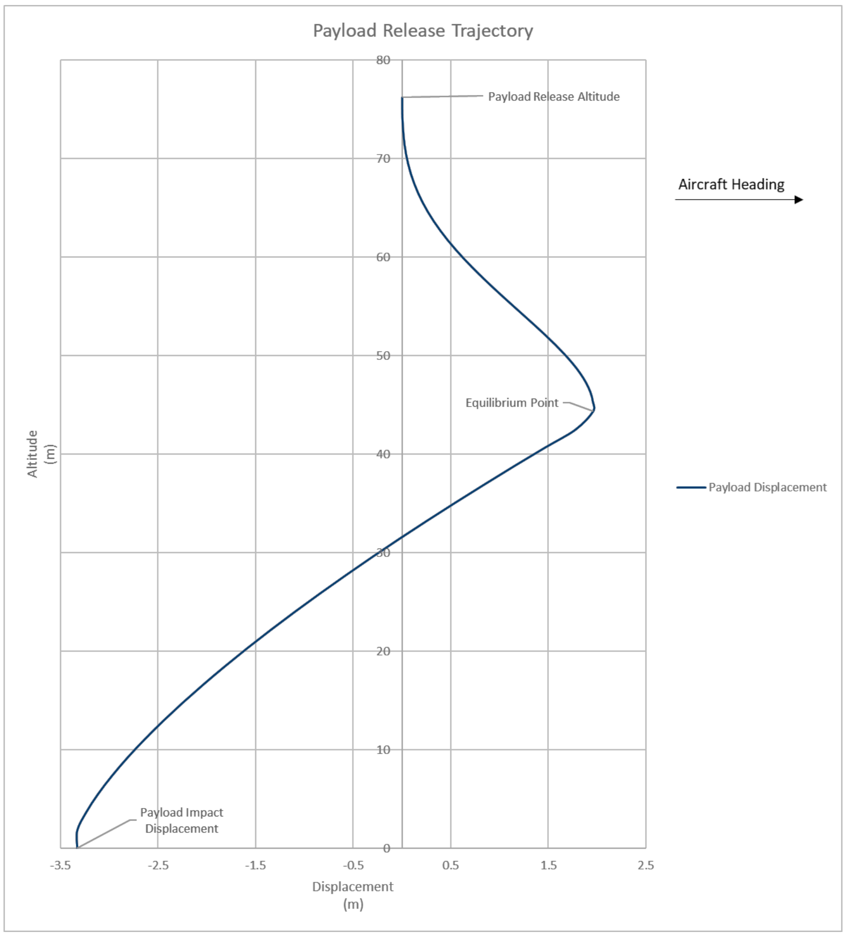


*Figure 8.35: Payload release trajectory during physical testing*

Two more payload release tests were conducted at different airspeed and altitudes, and an average payload displacement error of ±0.28 m was achieved. In addition to this average displacement error, it was noted that the payload tended to land to the right of the aircraft's heading. This tendency for the payload to land to the right of the aircraft's heading indicated that the algorithm was defining the payload release approach along a heading where the headwind would hit the aircraft on its port side. As such, the algorithm was altered to correct for this heading offset.

Images of the aircraft during the payload release testing were captured and overlaid to provide a view of the payload's trajectory. The trajectory of the payload was highlighted due to the distorted quality of the overlaid image, as seen in Figure 8.36.

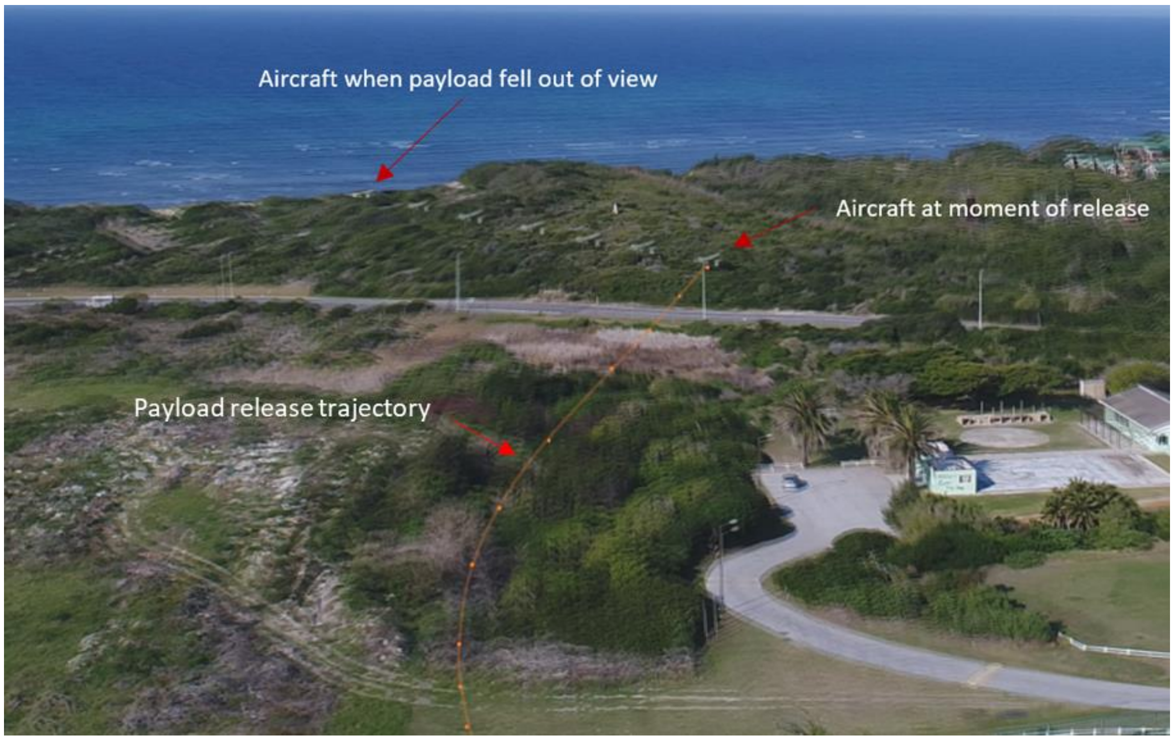

Aircraft when payload fell out of view
Aircraft at moment of release
Payload release trajectory

*Figure 8.36: Stitched image illustrating the payload release trajectory and aircraft flight progression during payload release testing*

In terms of the payload's physical capabilities when impacting the ground, it was found that the 3D printed payload design was able to withstand three to four payload release tests before physical damage occurred. During the payload release testing, the GPS module installed within the payload was fully protected for any physical damage. During the initial physical testing stages, it was noted that the payload release mechanism could be triggered, and the payload would not release. These failed drops resulted in small alterations to the payload release mechanism to remove the edges which were catching during release. However, it was noted that these failed drops only occurred on testing days with higher wind speeds and the system functioned accordingly at lower wind speeds. Furthermore, during missions where the drop-zone identifier was not detected, and the aircraft would circle back to its takeoff position, it was noted that the payload release command would be triggered due to the mission being aborted by the FCU. The release of the payload indicated that the FCU deemed all waypoints met if no new waypoints were presented to it and as such, an additional release command condition was introduced to ensure the drop location waypoint had been defined.

## 8.5 System Performance

Combing the payload release results with the remaining subsystems allowed for the system's overall performance to be evaluated. Taking the average errors achieved within each subsystem and defining the mean vision system accuracy to be the mean accuracy of the system, the integrated system's performance was defined by Equation 8.16.

$$x' = 1.82 \pm 0.49\,m \tag{8.16}$$

From Equation 8.16, the final performance of the integrated system was found to achieve an average payload displacement of 1.82 m from the true location of the designated drop-zone identifier when operating at an approximate altitude of 75 m. At this altitude, the displacement was found to possess an average error of ±0.49 m. A linear trend was noted with the system's performance errors and as such, the accuracy of the system decreased with an increase in altitude. Therefore, the system performance defined in Equation 8.16 would decrease with a decrease in altitude.

Considering the integrated performance and the defined allowable circular drop-zone area of 10 m in diameter, the integrated system was able to achieve the desired result. The performance of the integrated system was able to achieve the desired results within 13.19% of the designated drop-zone area, indicating that the remaining 86.81% of the drop-zone's designated area was available to compensate for any errors in payload release. Radially, the performance of the system was found to achieve payload release within an average range of 36.24% of the defined drop-zone radius.

## 8.6 Chapter Conclusion

Chapter 8 discussed the details pertaining to the test of the three defining subsystems within this research. Results depicting the simulation and physical results were discussed and a performance evaluation of each system was used to declare the performance of the integrated system and it was found that the system was able to achieve the desired results through implementation of the developed subsystems. Chapter 9 concludes the research by defining the overall performance of the integrated system and reiterates the research objectives achieved. Additionally, Chapter 9 defined possible future applications and alterations to the research to expand upon the results achieved.

# Chapter 9

# Conclusion and Recommendations

This chapter concludes this dissertation, highlighting the final performance of the vision-based autonomous aircraft payload delivery system developed. Additional sections discussed within this chapter included research contributions and recommendations for future research.

## 10.1 Conclusion

The aim of this research was to develop a modular vision-based autonomous aircraft payload delivery system for application in the automated delivery of a given payload to a drop-zone location. This payload delivery system was designed for use in, amongst various other applications, the service delivery sector. The desired objective was to promote the development of the service delivery sector into the era of Industry 4.0.

To achieve the aims and objectives of this research, an application specific hardware and software architecture was developed. The hardware architecture consisted of a test bench aircraft, the Skywalker 2013, integrated with the Pixhawk flight controller to enable the system with autonomous flight control. Additionally, an Odroid-XU4 companion computer was integrated onboard to provide the necessary computational power required to achieve a fully automated vision-based payload delivery system, where the vision system consisted of an ELP-USB500W04AF-A60 CMOS camera and TAROT gimbal. A Linux-based ROS node software architecture was developed, where five C++ ROS nodes were utilised for integrated control. Selection of these system architecture components was done through the research of relevant literature on similar systems.

This research found through application of SITL and HITL simulations, and physical testing, it was found that a low-input operator defined mission was successfully achieved with fully autonomous flight control of the aircraft. Comparative results captured during physical flight tests illustrated a 14.91% reduction in power consumption with the application the autonomous flight control of the aircraft as opposed to manual flight control of the aircraft.

Through application of simulated footage, both onboard and offboard the aircraft, an image processing algorithm was developed which was capable of drop-zone identification and location up to 100 m away based on the vision hardware utilised.

Physical testing of the vision system and image processing algorithm resulted in drop-zone identification and location within the defined acceptable drop-zone area. Application of the vision system at an altitude of 75 m resulted in the determination of the drop-zone location with an average displacement of 1.82 m from the true drop-zone location. Additionally, it was found that approximately 30% of the detected drop-zone locations were within a 1 m radius of the true location. The resulting accuracy of the vision system was found to improve upon the research conducted by Hinas et al. [27], where similar vision system test results were achieved at an altitude of 40 m.

Application of SITL and HITL simulations, and the mathematical model developed in Chapter 6 resulted in the successful payload displacement prediction of payloads dropped from altitude. A payload displacement prediction uncertainty error of ±0.28 m was achieved, with the most accurate payload release achieving a payload displacement error 0.14 m from the theoretical location, at an altitude of 76.2 m. The resulting automated payload release testing provided the necessary evidence to conclude that a two-phase payload release trajectory was present, as described in Chapter 6. Additionally, the payload release results indicated an improvement upon the research conducted by Boura et al. [10], where through application of an autonomous aircraft payload delivery system, an average payload displacement of 75 ft (~23 m) was achieved during testing.

As an integrated unit, the system was able to successfully fly an autonomous mission and through vision-based survey, determine the true location of the drop-zone within the acceptable tolerance. Through drop-zone identification, the system was able to update mission waypoints, execute environmental parameter survey for payload release trajectory calculations and subsequently release the payload to within the acceptable drop-zone area. As such, this research successfully met the required research aims and objectives defined and proved the proposed hypothesis.

## 10.2 Research Contributions

With the conclusion of this research, the following research contributions were achieved:

- The development of a vision system and image processing algorithm which achieved similar results in target location determination accuracy at an altitude of 75 m when compared to the apparent average accuracy of Hinas et al.'s vision system [27] at an altitude of 40 m.
- The development of a fully integrated autonomous aircraft with an integrated autonomous vision system capable of achieving the automated payload release. Where, this research built upon the research conducted by Boura et al. [10] in the development of an autonomous aircraft with payload release capabilities.
- Based on the work conducted by Boura et al. [10], where payload trajectory modelling was achieved through real-time environmental parameter monitoring, it was found that payload trajectory modelling could be furthered. As such, this research included the development of a two-dimensional two-phase payload release trajectory model for the prediction of a payload's ballistic fall trajectory, which achieved successful displacement prediction results during simulated and physical testing.
- An integrated payload delivery system capable of achieving the same results as achieved by Zipline [50] and DHL Parcelcopter [16], two commercially used systems implementing autonomous aircraft payload delivery. Additionally, the introduction of a vision system, a system the two commercial systems did not possess, increased payload delivery accuracy and security by ensuring payload drop-zone identification.
- The development of a fully automated mission flight plan based on a single user-defined waypoint. Improving upon the DHL Parcelcopter's [16] system of manual user mission designation and subsequently optimising mission waypoint designation.

## 10.3 Recommendations for Future Research

Upon completion of this research and reflection on the results gathered, several future considerations were derived to further this research. These considerations included:

- Application of the integrated system within different airframes to allow for comparative results to be drawn of the performance of the integrated system within another airframe.
- Application of a larger FOV vision system or several vision systems to allow for a larger survey area, in terms of spacing between survey waypoints.
- Future development of the image processing algorithm for use in unique drop-zone identifier locations as opposed to a generic red circle. Such an application would allow for greater payload delivery security in terms of decreasing the number of false detections by the vision system.
- Development of a controlled payload could be undertaken in order to mitigate the design-stage uncertainty errors and overall errors obtained.
- Further testing of the payload release system to allow for system performance to be determined under a greater range of testing parameters, such as higher altitude releases, greater aircraft ground speeds and greater headwind velocities.

## References

[1] Ardupilot Dev Team. Automatic Takeoff. [Internet]: ArduPilot; 2016 [cited 2018 March 10]. Available from: http://ardupilot.org/plane/docs/automatic-takeoff.html

[2] Ardupilot Dev Team. Complete Parameter List. [Internet]: ArduPilot; 2016 [cited 2018 March 10]. Available from: http://ardupilot.org/plane/docs/parameters.html

[3] ArduPilot Dev Team. Fixed-wing aircraft. [Internet]: ardupilot.org; 2016 [cited 2018 April 26]. Available from: http://ardupilot.org/plane/

[4] ArduPilot Dev Team. QuadPlane Overview. [Internet]: 2016 [cited 2018 April 26]. Available from: http://ardupilot.org/plane/docs/quadplane-overview.html

[5] ArduPilot Dev Team. What is a MultiCopter and How Does it Work? [Internet]: ardupilot.org; 2016 [cited 2018 April 26]. Available from: http://ardupilot.org/copter/docs/what-is-a-multicopter-and-how-does-it-work.html

[6] Benson T. Centre of Pressure – cp. [Internet]: NASA; 2014 [Update 2014 June 12; cited 2018 June 10]. Available from: https://www.grc.nasa.gov/www/k-12/rocket/cp.html

[7] Benson T. Determining Centre of Pressure – cp. [Internet]: NASA; 2014 [Update 2014 June 12; cited 2018 June 10]. Available from: https://www.grc.nasa.gov/www/k-12/rocket/rktcp.html

[8] Benson T. Shape Effects on Drag. [Internet]: NASA; 2015 [Updated 2015 October 22; cited 2018 June 10]. Available from: https://spaceflightsystems.grc.nasa.gov/education/rocket/shaped.html

[9] Bhutia TK. Lapse Rate. [Internet]: Encyclopaedia Britannica; 2016 [cited 2018 June 08]. Available from: https://www.britannica.com/science/lapse-rate

[10] Boura D, Hajicek D, Semke W. Automated Air Drop System for Search and Rescue Applications Utilizing Unmanned Aircraft Systems. American Institute of Aeronautics and Astronautics. 2011; 1-8.

[11] Bradski G, Kaehler A. Learning OpenCV. 1st Edition. Sebastopol: O'Reilly; 2008.

[12] Byaruhanga, J. Improving Service Delivery in Developing Countries. Academia. 2011. https://www.academia.edu/1281472/Improving_Service_delivery_in_Developing_countries. 2011 January 01. Accessed Date: 2018 April 30.

[13] Communications Security, Inc.. SSH (SECURE SHELL). [Internet]: ssh.com; 2018 [cited 2018 September 28]. Available from: https://www.ssh.com/ssh/

[14] Crawley SW, Dillon RM. Steel Buildings: Analysis and Design. 4th Edition. United States of America: John Wiley & Sons; 1993.

[15] David, M. Get lat/long given current point, distance and bearing. [Internet]: StackOverflow; 2011 [cited 2018 July 16]. Available from: https://stackoverflow.com/questions/7222382/get-lat-long-given-current-point-distance-and-bearing

[16] Deutsche Post AG. DHL Parcelcopter. [Internet]: Deutsche Post DHL Group; 2018 [cited 2018 October 15]. Available from: https://www.dpdhl.com/en/media-relations/specials/dhl-parcelcopter.html

[17] Elpcctv.com. MINI 5MP AF USB CAMERA MODULE OV5640 COLOR CMOS SENSOR. [Internet]: Ailipu Technology Co.,Ltd.; 2018 [cited 2018 June 07]. Available from: http://www.elpcctv.com/mini-5mp-af-usb-camera-module-ov5640-color-cmos-sensor-p-234.html

[18] Figliola, RS, Beasley, DE. Theory and Design for Mechanical Measurements. 6th Edition. Hoboken: John Wiley & Sons; 2015.

[19] Geissbauer, R, Vedsø, J, Schrauf, S. A Strategists Guide to Industry 4.0. [Internet]: strategy+business; 2016 [cited 2018 April 25]. Available from: https://www.strategy-business.com/article/A-Strategists-Guide-to-Industry-4.0?gko=7c4cf.

[20] Gomez, DA. Writing a Simple Image Publisher (C++). [Internet]: ROS.org; 2015-May [cited 2018 July 20]. Available from: http://wiki.ros.org/image_transport/Tutorials/PublishingImages

[21] GoPro. What is the Centre of Gravity on the HERO camera?. [Internet]: GoPro.com; 2018 [cited 2018 August 11]. Available from: https://gopro.com/help/articles/question_answer/What-is-the-Center-of-Gravity-on-the-HERO-Camera

[22] Hall N. Ballistic Flight Equations (no drag – no thrust). [Internet]: NASA; 2018 [Updated 2018 April 05; cited 2018 May 02]. Available from: https://www.grc.nasa.gov/WWW/k-12/airplane/ballflght.html.

[23] Hall N. Flight Equations with Drag (no thrust – constant mass). [Internet]: NASA; 2018 [Updated 2018 April 05; cited 2018 June 10]. Available from: https://www.grc.nasa.gov/www/k-12/airplane/flteqs.html.

[24] Hall N. Wing Geometry Definitions. [Internet]: NASA; 2018 [cited 2018 August 13]. Available from: https://www.grc.nasa.gov/www/k-12/airplane/geom.html

[25] Hardkernel Co. ODroid-XU4. [Internet]: hardkernel.com; 2018 [cited 2018 March 27]. Available from: https://www.hardkernel.com/shop/odroid-xu4/

[26] Hassanalian M, Abdelkefi A. Classifications, applications, and design challenges of drones: A review. Progress in Aerospace Sciences. 2017; 91: 99-131.

[27] Hinas, A., Roberts, J.M, Gonzalez, F. Vision-Based Target Finding and Inspection of a Ground Target Using a Multirotor UAV System. Sensors. 2017; 2929:1-17.

[28] Hochstenbach M, Notteboom C, Theys B, De Schutter J. Design and Control of an Unmanned Aerial Vehicle for Autonomous Parcel Delivery with Transition from Vertical Take-off to Forward Flight. International Journal of Micro Air Vehicles. 2015; 7(4): 395-405.

[29] International Civil Aviation Organization. Frequently Used Terms. [Internet]: International Civil Aviation Organization; 2018 [cited 2018 February 05]. Available from: https://www.icao.int/safety/UA/UASToolkit/Pages/FAQ.aspx

[30] Jewett JW Jr., Serway RA. Physics for Scientists and Engineers with Modern Physics. 8th Edition. Canada: Brooks/ Cole CENGAGE Learning; 2010.

[31] Opencv dev team. Hough Circle Transform. [Internet]: OpenCV; 2014 [cited 2018 July 30]. Available from: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html#hough-circle-transform

[32] OpenCV dev team. How the image matrix is stored in the memory?. [Internet]: OpenCV; 2014 [cited 15 October 2018]. Available from: https://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html#how-the-image-matrix-is-stored-in-the-memory

[33] OpenCV dev team. Image Filtering. [Internet]: OpenCV; 2014 [cited 2018 July 25]. Available from: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#image-filtering

[34] Osswald, S. Writing a Simple Image Subscriber (C++). [Internet]: ROS.org; 2015 May [cited 2018 July 20]. Available from: http://wiki.ros.org/image_transport/Tutorials/SubscribingToImages

[35] Pike, J. Principles of Airdrop Supply and Resupply Operations. [Internet]: GlobalSecurity.org; 2000 [cited 2018 April 10]. Available from: https://www.globalsecurity.org/military/library/policy/army/fm/10-500-1/Ch2.htm#top

[36] Prabowo YA, Trilaksono BR, Triputra FR. Hardware In-the-Loop Simulation for Visual Servoing of Fixed Wing UAV. International Conference on Electrical Engineering and Informatics (ICEEI). 2015; 247-252.

[37] PX4 Dev Team. Pixhawk 1 Flight Controller. [Internet]: Droncode.org; 2018 [cited 2018 March 18]. Available from: https://docs.px4.io/en/flight_controller/pixhawk.html

[38] Samaritan's Purse. 2011 Year in Pictures. [Internet]: Samaritanspurse.org; 2011 [cited 2018 October 15]. Available from: https://www.samaritanspurse.org/article/2011-year-in-pictures-2/

[39] Shankland, S. Zipline drone launch. [Internet]: CNET; 2018 April [cited 2018 October 15]. Available from: https://www.cnet.com/pictures/take-a-look-at-ziplines-new-drone-delivery-system/

[40] Sol. Detect red circles in an image using OpenCV. [Internet]: Solarian Programmer; 2015-May [cited 2018 July 25]. Available from: https://solarianprogrammer.com/2015/05/08/detect-red-circles-image-using-opencv/

[41] Sullivan, JM. Evolution or Revolution? The Rise of UAVs. IEEE Technology and Society Magazine. 2006; 25(3): 43-49.

[42] Sutton GP, Biblarz O. Rocket Propulsion Elements. 7th Edition. Canada: John Wiley & Sons, INC.; 2001.

[43] TSC. MAVROS/CustomModes. [Internet]: ROS.org; 2018 [cited 2018 July 07]. Available from: http://wiki.ros.org/mavros/CustomModes

[44] U.S. Department of Transport. Pilot's Handbook of Aeronautical Knowledge. FAA-H-8083-25B. Oklahoma City: United States Department of Transportation, Federal Aviation Administration, Airman Testing Standards Branch; 2016.

[45] U.S. Department of Transport. Glider Flying Handbook. FAA-H-8083-13A. Oklahoma City: United States Department of Transportation, Federal Aviation Administration, Airman Testing Standards Branch; 2013.

[46] UAVChallenge. Search and Rescue. [Internet]: uavchallenge.org; 2014 [cited 2018 March 22]. Available from: https://uavchallenge.org/search-and-rescue/

[47] Unknown. MAVLINK Common Message Set. [Internet]: MAVLINK; 2018 [cited 2018 July 07]. Available from: https://mavlink.io/en/messages/common.html

[48] Vooon. MAVROS. [Internet]: ROS.org; 2018 [cited 2018 June 12]. Available from: http://wiki.ros.org/mavros

[49] Vooon. MAVROS_MSGS. [Internet]: ROS.org; 2018 [cited 2018 June 12]. Available from: http://wiki.ros.org/mavros_msgs

[50] Zipline. Zipline. [Internet]: Zipline; 2018 [cited 2018 February 12]. Available from: http://www.flyzipline.com/

# Appendix A1 – Autonomous Flight Node Code

```
//Vision-Based Autonomous Aircraft Payload Delivery System
//James Sewell
//MEng (Mechatronics)
//Nelson Mandela University
///////////////////////////////////////////////
/////////////////FLIGHT NODE/////////////////
///////////////////////////////////////////////
///////////////////HEADERS///////////////////
//ROS and Other Libraries
#include <ros/ros.h> //ROS Header
#include <math.h> //Maths header
#include <geometry_msgs/Twist.h> //Velocity
//MAVROS Message Types
#include <mavros_msgs/OverrideRCIn.h> //Payload release
#include <mavros_msgs/SetMode.h> //Define aircraft flightmode
#include <mavros_msgs/State.h> //Current autopilot state
#include <mavros_msgs/VFR_HUD.h> //Telemetry
#include <mavros_msgs/Waypoint.h> //Waypoint header
#include <mavros_msgs/WaypointClear.h> //Clear waypoints header
#include <mavros_msgs/WaypointList.h> //List current waypoints
#include <mavros_msgs/WaypointReached.h> //Confirmation on reaching waypoint
//MAVROS Service Types
#include <mavros_msgs/CommandBool.h> //Arming statement
#include <mavros_msgs/CommandTOL.h> //Takeoff and Landing
#include <mavros_msgs/WaypointPull.h> //Request waypoints from FCU
#include <mavros_msgs/WaypointPush.h> //Send waypoints to FCU
#include <mavros_msgs/WaypointSetCurrent.h> //Define a waypoint for the
aircraft to fly to (Provide index)
//Global Constants
float pi = 3.14159265359;
//////////////////CALLBACKS//////////////////
//Current flight controller status
mavros_msgs::State FCU_State; //Assigning the FCU state variable
void state_cb(const mavros_msgs::State::ConstPtr& msg)
{
FCU_State = *msg;
}
//Current flight controller variables (airspeed, heading, etc)
mavros_msgs::VFR_HUD HV; //Assigning the FCU state variable
void vfr_cb(const mavros_msgs::VFR_HUD::ConstPtr& msg)
{
HV = *msg;
}
//Current waypoint reached status
mavros_msgs::WaypointReached WayReached;
void wayreach_cb(const mavros_msgs::WaypointReached::ConstPtr& msg)
{
WayReached = *msg;
}
//Current list of waypoints
mavros_msgs::WaypointList WayList;
void waylist_cb(const mavros_msgs::WaypointList::ConstPtr& msg)
{
WayList = *msg;
}
//////////////////MAIN CODE//////////////////
int main(int argc, char **argv)
```

```
{
/*******************Topics*******************/
ros::init(argc, argv, "auto_flight_node"); //ROS Node Initialiser
ros::NodeHandle nh;
//Subscribers (Listens for information from FCU)
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/
state", 10, state_cb); //Listens for aircraft state
ros::Subscriber vfr_sub = nh.subscribe<mavros_msgs::VFR_HUD>("mavros/
vfr_hud", 10, vfr_cb); //Listens for aircraft variables
ros::Subscriber way_reach_sub = nh.subscribe<mavros_msgs::WaypointReached>
("mavros/mission/reached", 10, wayreach_cb); // Listens for whether the
aircraft reached the desired waypoint
ros::Subscriber way_list_sub = nh.subscribe<mavros_msgs::WaypointList>
("mavros/mission/waypoints", 10, waylist_cb); // Listens for the list of
waypoints
//Publishers (Transmits information to FCU)
ros::Publisher release_mechanism_pub =
nh.advertise<mavros_msgs::OverrideRCIn>("mavros/rc/override", 10); //
Publishisher for payload release
ros::Publisher velocity_pub = nh.advertise<geometry_msgs::Twist>("mavros/
setpoint_velocity/cmd_vel_unstamped", 10); //Publisher for desired
mission airspeed
//Services and Clients (Allows for functions to be defined and called,
returning a booleean result as to the clients success at executing the
service
//Two members: request and response
ros::ServiceClient arming_client =
nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd/arming"); //
Arming client
ros::ServiceClient set_mode_client =
nh.serviceClient<mavros_msgs::SetMode>("mavros/set_mode"); //Flight mode
setting client
ros::ServiceClient wayclear_client =
nh.serviceClient<mavros_msgs::WaypointClear>("mavros/mission/clear"); //
Clear waypoints client
ros::ServiceClient waypull_client =
nh.serviceClient<mavros_msgs::WaypointPull>("mavros/mission/pull"); //
Download waypoints from FCU
ros::ServiceClient waypush_client =
nh.serviceClient<mavros_msgs::WaypointPush>("mavros/mission/push"); //
Upload waypoints to FCU
ros::ServiceClient waysetcurrent_client =
nh.serviceClient<mavros_msgs::WaypointSetCurrent>("mavros/mission/
set_current"); //Update next waypoint to fly to (use for when desired
waypoint has been passed or is far away)
//Communication Transmission Rate (>>2Hz)
ros::Rate rate(20.0); //rate defines the frequency (rate is an attribute
of the ROS::Rate topic)
//Wait for connection to be established between MAVROS and autopilot
while(ros::ok() && !FCU_State.connected)
{
ros::spinOnce(); //Ensures that subscriptions, publishers, callbacks,
etc are executed (laymans term = ROS ISR), Puases other nodes an
runs this code
rate.sleep(); //Ensures that the ROS::Rate maintrains the desired
frequency of operation
}
```

```
/******************Initialise*****************/
//Ensures the aircraft is disarmed and set to manual mode initially
ROS_INFO("Initialising...");
ros::Time last_request = ros::Time::now(); //Current flight control time
mavros_msgs::SetMode aircraft_set_mode; //Define variable for setting the
aircraft's flight mode
mavros_msgs::CommandBool arm_motor_cmd; //Define the arming variable
aircraft_set_mode.request.custom_mode = "MANUAL"; //Assign the desired
aircraft flight mode (MANUAL Mode)
arm_motor_cmd.request.value = false; //Disarm the aircraft (Service-Client
memeber)
while (FCU_State.mode != "MANUAL" || FCU_State.armed) //Disarming and
MANUAL mode transmission loop
{
if (FCU_State.mode != "MANUAL" && (ros::Time::now() - last_request >
ros::Duration(0.2))) //Delay by 'X' seconds to prevent bottleneck
{
if (set_mode_client.call(aircraft_set_mode) &&
aircraft_set_mode.response.mode_sent) //Call service client
{
ROS_INFO("Manual Flight Mode"); //Confirmation
}
last_request = ros::Time::now(); //Update timer
}
else
{
if (FCU_State.armed && (ros::Time::now() - last_request >
ros::Duration(0.2))) //Delay by 'X' seconds to prevent
bottleneck
{
if (arming_client.call(arm_motor_cmd) &&
arm_motor_cmd.response.success) //Call service client
{
ROS_INFO("Motors Disarmed"); //Confirmation
}
last_request = ros::Time::now(); //Update timer
}
}
ros::spinOnce();
rate.sleep();
}
ROS_INFO("Initialisation Complete");
//Define flight velocity
geometry_msgs::Twist velo;
velo.linear.x = 13;
velocity_pub.publish(velo);
ros::spinOnce();
rate.sleep();
/***************Mission Waypoint*************/
//Waypoint variables
mavros_msgs::WaypointClear wayclear;
mavros_msgs::WaypointPush waypush;
mavros_msgs::WaypointSetCurrent waysetcur;
mavros_msgs::Waypoint Points[18]; //Waypoint array
mavros_msgs::Waypoint wp_dud; //Dud waypoint
mavros_msgs::Waypoint wp_takeoff; //Target waypoint
mavros_msgs::Waypoint wp_landing; //Landing waypoint
```

```
mavros_msgs::Waypoint wp_initial; //Initial target waypoint
mavros_msgs::Waypoint wp_sl; //Survey waypoint lineup
mavros_msgs::Waypoint wp_s1; //Survey waypoint #1
mavros_msgs::Waypoint wp_s2; //Survey waypoint #2
mavros_msgs::Waypoint wp_s3; //Survey waypoint #3
mavros_msgs::Waypoint wp_s4; //Survey waypoint #4
mavros_msgs::Waypoint wp_s5; //Survey waypoint #5
mavros_msgs::Waypoint wp_s6; //Survey waypoint #6
mavros_msgs::Waypoint wp_s7; //Survey waypoint #7
mavros_msgs::Waypoint wp_s8; //Survey waypoint #8
mavros_msgs::Waypoint wp_hw; //Headwind waypoint #3
mavros_msgs::Waypoint wp_target; //Target waypoint
mavros_msgs::Waypoint wp_dal; //Drop approach lineup
mavros_msgs::Waypoint wp_da1; //Drop approach waypoint #1
mavros_msgs::Waypoint wp_da2; //Drop approach waypoint #2 --> Drop payload
location
//Assigning the waypoints to the array
Points[0] = wp_dud; //redundant
Points[1] = wp_takeoff;
Points[2] = wp_sl;
Points[3] = wp_s1;
Points[4] = wp_s2;
Points[5] = wp_s3;
Points[6] = wp_s4;
Points[7] = wp_s5;
Points[8] = wp_s6;
Points[9] = wp_s7;
Points[10] = wp_s8;
Points[11] = wp_hw;
Points[12] = wp_dal;
Points[13] = wp_da1;
Points[14] = wp_da2;
Points[15] = wp_target;
Points[16] = wp_landing;
Points[17] = wp_initial;
//Clear all onboard waypoints and refresh FCU
bool NoOld = false;
bool newMission = false;
last_request = ros::Time::now();
while (NoOld != true)
{
if (NoOld != true && (ros::Time::now() - last_request > ros::Duration
(0.2)))
{
if (wayclear_client.call(wayclear))
{
ROS_INFO("Waypoints Cleared %d",
wayclear.response.success); //Confirm how many waypoints
were cleared
NoOld = true;
}
last_request = ros::Time::now();
}
ros::spinOnce();
rate.sleep();
}
//Redundant waypoint (mimics takeoff) - Necessary for transmission of all
```

```
waypoints or first is ignored
Points[0].frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
Points[0].command = 22; //takeoff command
Points[0].is_current = false;
Points[0].autocontinue = true;
Points[0].param1 = 15; //pitch
Points[0].param2 = 0;
Points[0].param3 = 0;
Points[0].param4 = 0;
Points[0].x_lat = -33.876505;
Points[0].y_long = 25.351275;
Points[0].z_alt = 50.0;
//Takeoff waypoint
Points[1].frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
Points[1].command = 22; //takeoff command
Points[1].is_current = false;
Points[1].autocontinue = true;
Points[1].param1 = 15; //pitch
Points[1].param2 = 0;
Points[1].param3 = 0;
Points[1].param4 = 0;
Points[1].x_lat = -33.876505;
Points[1].y_long = 25.351275;
Points[1].z_alt = 50.0;
//Initial target waypoint
Points[17].frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
Points[17].command = 16;
Points[17].is_current = false;
Points[17].autocontinue = true;
Points[17].param1 = 0;
Points[17].param2 = 2;
Points[17].param3 = 0;
Points[17].param4 = 0;
Points[17].x_lat = -33.876505; //Approximate drop-zone latitude (centre of
survey)
Points[17].y_long = 25.351275; //Approximate drop-zone longitude (centre
of survey)
Points[17].z_alt = 50.0; //Altitude
//Generic waypoint parameters
for(int x = 2; x < 17; x++)
{
Points[x].frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
Points[x].command = 16; //waypoint command
Points[x].is_current = false;
Points[x].autocontinue = true;
Points[x].param1 = 0;
Points[x].param2 = 2; //waypoint reached precision
Points[x].param3 = 0;
Points[x].param4 = 0;
Points[x].z_alt = 50.0;
}
//Computation of survey area
/*
3
|
2----4 8
| | |
| + |
| | |
1 5----7
|
6
*/
float R = 6378.1; //Radius of the Earth (km)
float B1 = 0; //Bearing upward (0 degrees -> radians)
float B2 = pi/2; //Bearing right (90 degrees -> radians)
float B3 = pi; //Bearing down (180 degrees -> radians)
float B4 = 3*pi/2; //Bearing left (270 degrees -> radians)
float d = 0.035; //Distance in km
float lat_ref = Points[17].x_lat * (pi/180); //Initial target latitude
converted to radians
float long_ref = Points[17].y_long * (pi/180); //Initial target longitude
converted to radians
//Survey coordinate #3
Points[6].x_lat = (asin(sin(lat_ref)*cos(d/R) + cos(lat_ref)*sin(d/R)*cos
(B1))) * (180/pi);
Points[6].y_long = (long_ref + atan2(sin(B1)*sin(d/R)*cos(lat_ref), cos(d/
R) - sin(lat_ref)*sin(Points[6].x_lat))) * (180/pi);
//Survey coordinate #4
Points[7].x_lat = (asin(sin(lat_ref)*cos(d/R) + cos(lat_ref)*sin(d/R)*cos
(B3))) * (180/pi);
Points[7].y_long = (long_ref + atan2(sin(B3)*sin(d/R)*cos(lat_ref), cos(d/
R) - sin(lat_ref)*sin(Points[7].x_lat))) * (180/pi);
//Survey coordinate #1
Points[3].x_lat = Points[7].x_lat;
Points[3].y_long = (long_ref + atan2(sin(B4)*sin(d/R)*cos(lat_ref), cos(d/
R) - sin(lat_ref)*sin(Points[3].x_lat))) * (180/pi);
//Survey coordinate #2
Points[4].x_lat = Points[6].x_lat;
Points[4].y_long = Points[3].y_long;
//Survey coordinate #5
Points[9].x_lat = Points[7].x_lat;
Points[9].y_long = (long_ref + atan2(sin(B2)*sin(d/R)*cos(lat_ref), cos(d/
R) - sin(lat_ref)*sin(Points[9].x_lat))) * (180/pi);
//Survey coordinate #6
Points[10].x_lat = Points[4].x_lat;
Points[10].y_long = Points[9].y_long;
//Survey coordinate #SL
Points[2].x_lat = (asin(sin(lat_ref)*cos(0.09/R) + cos(lat_ref)*sin(0.09/
R)*cos(B3))) * (180/pi);
Points[2].y_long = Points[3].y_long;
//Survey coordinate #T1
Points[5].x_lat = (asin(sin(lat_ref)*cos(0.09/R) + cos(lat_ref)*sin(0.09/
R)*cos(B1))) * (180/pi);
Points[5].y_long = Points[3].y_long;
//Survey coordinate #T2
Points[8].x_lat = (asin(sin(lat_ref)*cos(0.09/R) + cos(lat_ref)*sin(0.09/
R)*cos(B3))) * (180/pi);
Points[8].y_long = Points[7].y_long;
//Display waypoints and their locations
for(int x = 2; x < 15; x++)
{
ROS_INFO("%i %f", x, Points[x].x_lat);
ROS_INFO("%i %f", x, Points[x].y_long);
```

```
}
waypush.request.start_index = 0; //Start new waypoint list from the
begining
last_request = ros::Time::now();
while (newMission != true) //New mission transmission loop
{
if (newMission != true && (ros::Time::now() - last_request >
ros::Duration(0.2)))
{
for(int x = 0; x < 18; x++)
{
waypush.request.waypoints.push_back(Points[x]); //send all waypoints to the FCU
}
if (waypush_client.call(waypush)) //Call waypoint upload client
{
ROS_INFO("Mission waypoints sent: %i",
waypush.response.success);
ROS_INFO("Number of waypoints sent: %i",
waypush.response.wp_transfered);
newMission = true; //Mission sent boolean (terminating
statement)
}
last_request = ros::Time::now();
}
ros::spinOnce();
rate.sleep();
}
/************Flight Mode Selection***********/
//Defines desired aircraft flight mode and arms motors
aircraft_set_mode.request.custom_mode = "AUTO"; //Assign the desired
aircraft flight mode
arm_motor_cmd.request.value = true; //Arm the aircraft (Service-Client
memeber)
last_request = ros::Time::now();
//Confirm that flight mode was successfully entered
while(FCU_State.mode != "AUTO" || !FCU_State.armed) //Arming and AUTO
Flight Mode Transmission Loop
{
if(!FCU_State.armed && (ros::Time::now() - last_request >
ros::Duration(2.0))) //Delay by 'X' seconds to prevent bottleneck
{
if(arming_client.call(arm_motor_cmd) &&
arm_motor_cmd.response.success)
{
ROS_INFO("Aircraft Armed");
}
last_request = ros::Time::now(); //Update current timer time
}
if(!FCU_State.armed && (ros::Time::now() - last_request >
ros::Duration(2.0)))
{
if(FCU_State.mode != "AUTO" && (ros::Time::now() - last_request >
ros::Duration(2.0))) //Delay by 'X' seconds to prevent
bottleneck
{
if(set_mode_client.call(aircraft_set_mode) &&
aircraft_set_mode.response.mode_sent)
{
ROS_INFO("AUTO Flight Mode");
}
last_request = ros::Time::now();
}
}
ros::spinOnce();
rate.sleep();
}
/******************Payload Release*****************/
mavros_msgs::WaypointSetCurrent WayFlyTo;
bool dropped = false;
mavros_msgs::OverrideRCIn release_cmd; //Release payload actuator
while (ros::ok)
{
if (dropped == true)
{
WayFlyTo.wp_seq = 12; //Confirmation flyover
waysetcurrent_client.call(WayFlyTo); //Confirmation flyover client
call
}
if(WayReached.wp_seq == 14 && dropped != true) //Wait till payload
waypoint has been defined and obtained
{
release_cmd.channels[6] = 2000; //Release payload servo position
dropped = true;
}
release_mechanism_pub.publish(release_cmd); //Publish release command
ros::spinOnce();
rate.sleep();
}
return 0;
}
```

## Appendix A2 – Image Capture Node Code

```
//Vision-Based Autonomous Aircraft Payload Delivery System
//James Sewell
//MEng (Mechatronics)
//Nelson Mandela University
////////////////////////////////////////////////
/////////////////IMAGE CAPTURE////////////////
////////////////////////////////////////////////
#include <ros/ros.h> //ROS header
#include <image_transport/image_transport.h> //Image transporter header for
inter-node transfer
#include <opencv2/highgui/highgui.hpp> //OPENCV Header
#include <cv_bridge/cv_bridge.h> //OPENCV Bridge Header
#include <sstream> // for converting the command line parameter to integer
int main(int argc, char** argv)
{
ros::init(argc, argv, "image_publisher");
ros::NodeHandle nh;
image_transport::ImageTransport it(nh); //Define nodehandler function
image_transport::Publisher pub = it.advertise("camera/image", 1); //Define
pusblisher file from which image will be subscribed
cv::VideoCapture cap(0); //Begin image stream from USB device 0
// Check if video device can be opened with the given index
if(!cap.isOpened()) return 1; //Ensure camera stream is present
cv::Mat frame; //Create image file variable
ros::Rate loop_rate(30); //Set transmission rate 30 FPS
while (nh.ok()) //While publisher is active
{
cap >> frame; //Capture image
msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame).toImageMsg
(); //Change format of captured image
pub.publish(msg); //Publish Image
cv::waitKey(1); //Provide transmission time
ros::spinOnce();
loop_rate.sleep();
}
}
```

## Appendix A3 – Image Processing Node Code

```
//Vision-Based Autonomous Aircraft Payload Delivery System
//James Sewell
//MEng (Mechatronics)
//Nelson Mandela University
///////////////////////////////////////////////
/////////////IMAGE PROCESSING NODE/////////////
///////////////////////////////////////////////
#include <ros/ros.h> //ROS Header
#include <std_msgs/String.h> //String Header
#include <image_transport/image_transport.h> //Image Transporter Header
#include <opencv2/highgui/highgui.hpp> //OPENCV Header
#include <cv_bridge/cv_bridge.h> //OPENCV Bridge Header
#include <mavros_msgs/Waypoint.h> //Waypoint Header
#include <mavros_msgs/WaypointPush.h> //Upload waypoints to FCU
#include <mavros_msgs/VFR_HUD.h> //Telemetry Header
#include <sensor_msgs/NavSatFix.h> //GPS Header
#include <std_msgs/Float64.h> //Altitude header (realtive to takeoff)
#include <math.h> //Maths header
//Namespaces - for neater code
using namespace cv;
using namespace ros;
using namespace std;
using namespace mavros_msgs;
using namespace sensor_msgs;
Waypoint wp_target; //Drop-zone waypoint
Waypoint wp_hw; //Headwind determination waypoint
//Variables
float BH = 0; //Drop-zone Bearing
float H_cur = 0; //Heading
float Alt_cur = 0; //Altitude
float x = 0; //Pixel x coordinate
float y = 0; //Pixel y coordinate
float xm = 0; //x axis pixel displacement
float ym = 0; //y axis pixel displacement
float lat_ref = 0; //Reference latitude
float long_ref = 0; //Reference longitude
float d = 0; //Ground plane displacement
float R = 0; //Earth's radius
float B = 0; //Fixed bearing
float pi = 3.14159265359;
//Current flight controller variables (airspeed, heading, etc)
VFR_HUD HV;
void vfr_cb(const mavros_msgs::VFR_HUD::ConstPtr& msg)
{
HV = *msg;
}
//Obtain current GPS location
NavSatFix GPS;
void nsf_cb(const sensor_msgs::NavSatFix::ConstPtr& msg)
{
GPS = *msg;
}
//Obtain relative altitude
std_msgs::Float64 Alt;
void alt_cb(const std_msgs::Float64::ConstPtr& msg)
{
Alt = *msg;
```

```
}
//Image callback - Begin object recognition and tracking
void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
try
{
NodeHandle n;
Subscriber vfr_sub = n.subscribe<mavros_msgs::VFR_HUD>("mavros/vfr_hud",
10, vfr_cb); //VFR_HUD Subscriber
Subscriber nsf_sub = n.subscribe<sensor_msgs::NavSatFix>("mavros/
global_position/global", 10, nsf_cb); //GPS info Subscriber
Subscriber alt_sub = n.subscribe<std_msgs::Float64>("mavros/
global_position/rel_alt", 10, alt_cb); //Altitude subscriber
ServiceClient waypush_client = n.serviceClient<mavros_msgs::WaypointPush>
("mavros/mission/push"); //Waypoint upload header
lat_ref = GPS.latitude * (pi/180); //Reference latitude (current latitude
of aircraft)
long_ref = GPS.longitude * (pi/180); //Reference longitude (current
longitude of aircraft)
H_cur = HV.heading; //Current heading
Alt_cur = Alt.data; //Current altitude
Rate rate(30.0); //Transmission rate of imagery
Mat BGR = cv_bridge::toCvShare(msg, "bgr8")->image; //Convert image from
msg to BGR format
Mat orig_image = BGR.clone(); //Make copy of image for processing
medianBlur(BGR, BGR, 7); //Introduce blur for image flitering and
converting
// Convert input image to HSV
Mat HSV;
cvtColor(BGR, HSV, COLOR_BGR2HSV); //Convert BGR image to HSV image
// Threshold the HSV image, keep only the red pixels
Mat lower_hue;
Mat upper_hue;
inRange(HSV, Scalar(0, 100, 100), Scalar(10, 255, 255), lower_hue); //
Lower hue threshold range
inRange(HSV, Scalar(165, 100, 100), Scalar(179, 255, 255), upper_hue); //
Upper hue threshold range
// Combine the above two images
Mat hue_image;
addWeighted(lower_hue, 1.0, upper_hue, 1.0, 0.0, hue_image); //Combine hue
images for object recognition
// Introduce interference
GaussianBlur(hue_image, hue_image, Size(9, 9), 2, 2); //Introduce noise
// Hough transform to detect circles
vector<Vec3f> circles; //Detected circles array
HoughCircles(hue_image, circles, CV_HOUGH_GRADIENT, 1, hue_image.rows/8,
100, 25, 0, 0); //HOUGH CIRCLE TRANSFORNATION
// Highlight detected object
for(size_t cur = 0; cur < circles.size(); ++cur)
{
Point center(circles[cur][0], circles[cur][1]); //Define centre point
of detected circle
int radius = circles[cur][2]; //Define radius of detected circle
circle(orig_image, center, radius, Scalar(239, 152, 38), 2); //Overlay
detected cricle outline onto origional image
ROS_INFO("Image Centre: %f , %f", circles[cur][0], circles[cur]
[1]); //Display circle centre coordinates
```

```
}
imshow("Lower hue", lower_hue); //Display lower hue image
imshow("Upper hue", upper_hue); //Display upper hue image
imshow("Vision Output", orig_image); //Display circle image overlay
ROS_INFO("Size: (W) %i x (H) %i", orig_image.cols, orig_image.rows); //
Define image size
waitKey(10); //allow for display of image for given milliseconds (Image
overlay refreshrate)
//Target Locating
if(circles.size() != 0) //Check if any circles were detected
{
WaypointPush waypush;
//Target bearing
BH = 0;
x = circles[0][0] - (orig_image.cols / 2); //x pixel coordinate
y = circles[0][1] - (orig_image.rows / 2); //y pixel coordinate
xm = (abs(x)/(orig_image.cols/2)) * (Alt_cur*tan(pi/6)); //x
displacement in meters
ym = (abs(y)/(orig_image.rows/2)) * (Alt_cur*tan(pi/8)); //y
displacement in meters
/* Region designation - (0,0) in top-left corner of 2
2|1
---
3|4
*/
//Determine bearing of target based on image segment
if(circles[0][0] >= 0 && circles[0][0] <= (orig_image.cols / 2)) //
region 2 and 3
{
if(circles[0][1] >= 0 && circles[0][1] <= (orig_image.rows /
2)) //region 2
{
BH = H_cur + atan(abs(x)/abs(y));
}
else //region 3
{
BH = H_cur + (180 - atan(abs(x)/abs(y)));
}
}
else //region 1 and 4
{
if(circles[0][1] > (orig_image.rows / 2)) //region 4
{
BH = H_cur + 180 + atan(abs(x)/abs(y));
}
else //region 1
{
BH = H_cur - atan(abs(x)/abs(y));
}
}
//Bearing regression (0 - 359)
if(BH >= 360)
{
BH = BH - 360;
}
else if(BH < 0)
{
```

```
BH = BH + 360;
}
GPS.position_covariance_type = 3; //Type of GPS frame reference (3 -
Covariance known)
d = sqrt(pow(abs(xm), 2) + pow(abs(ym), 2)) / 1000; //magnitude of
vector to target location in km
R = 6378.1; //Radius of the Earth (km)
B = BH * pi/180; //Bearing upward (0 degrees -> radians)
wp_target.frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
wp_target.command = 16; //MAV_CMD Waypoint type
wp_target.is_current = false;
wp_target.autocontinue = true;
wp_target.param1 = 0;
wp_target.param2 = 2;
wp_target.param3 = 0;
wp_target.param4 = 0;
wp_target.z_alt = 50.0;
//Deduce target's location based on current location
wp_target.x_lat = (asin(sin(lat_ref)*cos(d/R) + cos(lat_ref)*sin(d/R)
*cos(B))) * (180/pi); //Target Latitude
wp_target.y_long = (long_ref + atan2(sin(B)*sin(d/R)*cos(lat_ref), cos
(d/R) - sin(lat_ref)*sin(wp_target.x_lat))) * (180/pi); //Tag=rget
Longitude
//Display target details deduced
ROS_INFO("Lat: %f", wp_target.x_lat); //Target latitude
ROS_INFO("Long: %f", wp_target.y_long); //Target Longitude
ROS_INFO("LatRef: %f", lat_ref); //Reference latitude
ROS_INFO("LongRef: %f", long_ref); //Reference longitude
ROS_INFO("xm: %f", xm); //X axis displacement (km)
ROS_INFO("ym: %f", ym); //Y axis displacment (km)
ROS_INFO("displacement: %f", d); //Total displacment (km)
ROS_INFO("bearing: %f", BH); //Bearing
ROS_INFO("altitude: %f", Alt_cur);
//Update target waypoint parameters
wp_hw.frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
wp_hw.command = 18;
wp_hw.is_current = false;
wp_hw.autocontinue = true;
wp_hw.param1 = 3;
wp_hw.param2 = 0;
wp_hw.param3 = 30;
wp_hw.param4 = 1;
wp_hw.z_alt = 50.0;
wp_hw.x_lat = wp_target.x_lat;
wp_hw.y_long = wp_target.y_long;
//Transmit updated target location parameters to FCU
bool targetSent = false;
waypush.request.start_index = 15; //Update waypoint 15
ros::Time last_request = ros::Time::now();
while (targetSent != true)
{
if (targetSent != true && (ros::Time::now() - last_request >
ros::Duration(0.2)))
{
waypush.request.waypoints.push_back(wp_target); //send target
waypoint to the FCU
if (waypush_client.call(waypush))
{
targetSent = true;
}
last_request = ros::Time::now();
}
ros::spinOnce();
rate.sleep();
}
//Transmit updated headwind determination location parameters to FCU
targetSent = false;
waypush.request.start_index = 11; //Update waypoint 11
last_request = ros::Time::now();
while (targetSent != true)
{
if (targetSent != true && (ros::Time::now() - last_request >ros::Duration(0.2)))
{
waypush.request.waypoints.push_back(wp_hw); //send target
waypoint to the FCU
if (waypush_client.call(waypush))
{
targetSent = true;
}
last_request = ros::Time::now();
}
ros::spinOnce();
rate.sleep();
}
}
}//End "if circles detected" loop - End target location identification
//Throw error if image publisher fails
catch (cv_bridge::Exception& e)
{
ROS_ERROR("Could not convert from '%s' to 'bgr8'.", msg->encoding.c_str
());
}
}
int main(int argc, char **argv) //Initial loop
{
init(argc, argv, "image_listener"); //Define node name
NodeHandle nh;
namedWindow("Vision Output", WINDOW_AUTOSIZE); //Create viewable window -
Image overlay output
namedWindow("Lower hue", WINDOW_AUTOSIZE); //Create viewable window -
Lower hue range
namedWindow("Upper hue", WINDOW_AUTOSIZE); //Create viewable window -
Upper hue range
startWindowThread(); //Begin window view and image display
image_transport::ImageTransport it(nh); //Define source of image
image_transport::Subscriber sub = it.subscribe("camera/image", 1,
imageCallback); //Image subscriber
spin();
}
```

## Appendix A4 – Payload Release Node Code

```
//Vision-Based Autonomous Aircraft Payload Delivery System
//James Sewell
//MEng (Mechatronics)
//Nelson Mandela University
///////////////////////////////////////////////
/////////////////PAYLOAD NODE/////////////////
///////////////////////////////////////////////
///////////////////HEADERS///////////////////
#include <ros/ros.h> //ROS Header
#include <math.h> //Maths header
#include <mavros_msgs/VFR_HUD.h> //Telemetry
#include <mavros_msgs/Waypoint.h>
#include <mavros_msgs/WaypointList.h>
#include <mavros_msgs/WaypointReached.h> //Confirmation on reaching waypoint
#include <mavros_msgs/WaypointPush.h> //Send waypoints to device (Device =
Pixhawk)
//Global Constants
float pi = 3.14159265359;
using namespace ros;
using namespace std;
using namespace mavros_msgs;
//Current flight controller variables (airspeed, heading, etc)
VFR_HUD HV;
void vfr_cb(const mavros_msgs::VFR_HUD::ConstPtr& msg)
{
HV = *msg;
}
//Current waypoint reached status
WaypointReached WayReached;
void wayreach_cb(const mavros_msgs::WaypointReached::ConstPtr& msg)
{
WayReached = *msg;
}
//Current waypoints on FCU
WaypointList WayList;
void waylist_cb(const mavros_msgs::WaypointList::ConstPtr& msg)
{
WayList = *msg;
}
////////////////////////////////
//PAYLOAD RELEASE LOOPKUP TABLES//
////////////////////////////////
//EQUILIBRIUM TIMES
//Equilibrium time lookup table for 45m
float lookup45_pv10[16] = { 3.1, 3.1, 2.5243, 1.5483, 1.089, 0.8272, 0.6607,
0.5459, 0.4623, 0.3996, 0.3505, 0.3111, 0.2789, 0.2528, 0.2299, 0.2111 };
float lookup45_pv11[16] = { 3.1, 3.1, 2.8707, 1.7434, 1.2238, 0.9301, 0.7432,
0.6149, 0.5217, 0.4514, 0.3965, 0.3527, 0.3169, 0.2872, 0.2621, 0.2408 };
float lookup45_pv12[16] = { 3.1, 3.1, 3.1, 1.9436, 1.3655, 1.0385, 0.8303,
0.6872, 0.5834, 0.5049, 0.4436, 0.3947, 0.3547, 0.3215, 0.2936, 0.2698 };
float lookup45_pv13[16] = { 3.1, 3.1, 3.1, 2.1449, 1.5087, 1.1484, 0.9189,
0.761, 0.6463, 0.5596, 0.492, 0.4378, 0.3936, 0.3569, 0.326, 0.2996 };
float lookup45_pv14[16] = { 3.1, 3.1, 3.1, 2.3455, 1.6517, 1.2584, 1.0076,
0.835, 0.7095, 0.6146, 0.5406, 0.4813, 0.4328, 0.3926, 0.3587, 0.3298 };
float lookup45_pv15[16] = { 3.1, 3.1, 3.1, 2.5477, 1.7967, 1.3704, 1.0983,
0.9108, 0.7745, 0.6713, 0.5907, 0.5262, 0.4734, 0.4296, 0.3926, 0.3611 };
float lookup45_pv16[16] = { 3.1, 3.1, 3.1, 2.7463, 1.939, 1.4802, 1.1872,
0.9852, 0.8382, 0.7268, 0.6398, 0.5701, 0.5132, 0.4658, 0.4259, 0.3917 };
float lookup45_pv17[16] = { 3.1, 3.1, 3.1, 2.9464, 2.0833, 1.5921, 1.2781,
1.0614, 0.9037, 0.7841, 0.6906, 0.6157, 0.5544, 0.5034, 0.4604, 0.4237 };
float lookup45_pv18[16] = { 3.1, 3.1, 3.1, 3.143, 2.2242, 1.7009, 1.3662,
1.1351, 0.9668, 0.8392, 0.7394, 0.6593, 0.5938, 0.5394, 0.4934, 0.4541 };
//Equilibrium time lookup table for 50m
float lookup50_pv10[16] = { 3.1928, 3.1928, 2.6542, 1.6293, 1.1469, 0.8716,
0.6959, 0.5748, 0.4872, 0.4208, 0.369, 0.3276, 0.2942, 0.266, 0.2427,
0.2222 };
float lookup50_pv11[16] = { 3.1928, 3.1928, 2.959, 1.8242, 1.2864, 0.9786,
0.7818, 0.6465, 0.548, 0.4733, 0.4152, 0.3688, 0.3308, 0.3, 0.2729, 0.251 };
float lookup50_pv12[16] = { 3.1928, 3.1928, 3.1928, 2.0201, 1.4274, 1.0875,
0.8694, 0.7189, 0.6097, 0.5269, 0.4624, 0.4108, 0.3686, 0.3337, 0.3045,
0.2791 };
float lookup50_pv13[16] = { 3.1928, 3.1928, 3.1928, 2.2167, 1.5698, 1.1973,
0.9580, 0.7927, 0.6724, 0.5813, 0.5104, 0.4536, 0.4072, 0.3684, 0.3361,
0.3085 };
float lookup50_pv14[16] = { 3.1928, 3.1928, 3.1928, 2.4125, 1.7125, 1.3077,
1.0474, 0.8673, 0.7359, 0.6366, 0.5589, 0.497, 0.4462, 0.404, 0.3682,
0.3380 };
float lookup50_pv15[16] = { 3.1928, 3.1928, 3.1928, 2.6069, 1.8553, 1.4189,
1.1375, 0.9426, 0.8004, 0.6924, 0.6082, 0.5407, 0.4854, 0.4397, 0.4013,
0.368 };
float lookup50_pv16[16] = { 3.1928, 3.1928, 3.1928, 2.8006, 1.9983, 1.5304,
1.2282, 1.0184, 0.865, 0.7489, 0.6579, 0.5849, 0.5253, 0.4757, 0.4341,
0.3989 };
float lookup50_pv17[16] = { 3.1928, 3.1928, 3.1928, 2.992, 2.1407, 1.6423,
1.3193, 1.0947, 0.9303, 0.8057, 0.7081, 0.6297, 0.5657, 0.5125, 0.4676,
0.4293 };
float lookup50_pv18[16] = { 3.1928, 3.1928, 3.1928, 3.1812, 2.2824, 1.7543,
1.4106, 1.1712, 0.9962, 0.863, 0.7368, 0.6724, 0.6065, 0.5498, 0.5018,
0.4606 };
//Equilibrium time lookup table for 55m
float lookup55_pv10[16] = { 3.3486, 3.3486, 2.8617, 1.7281, 1.2082, 0.9154,
0.7296, 0.6023, 0.5101, 0.4406, 0.3865, 0.3433, 0.308, 0.2788, 0.2543,
0.2334 };
float lookup55_pv11[16] = { 3.3486, 3.3486, 3.182, 1.9246, 1.3472, 1.0215,
0.8148, 0.6731, 0.5704, 0.4928, 0.4325, 0.3843, 0.345, 0.3124, 0.285,
0.2616 };
float lookup55_pv12[16] = { 3.3486, 3.3486, 3.3486, 2.1233, 1.4875, 1.1287,
0.9008, 0.7444, 0.6311, 0.5455, 0.4789, 0.4256, 0.3822, 0.3461, 0.3158,
0.29 };
float lookup55_pv13[16] = { 3.3486, 3.3486, 3.3486, 2.3233, 1.6295, 1.2375,
0.9884, 0.8173, 0.6932, 0.5995, 0.5265, 0.4681, 0.4205, 0.381, 0.3477,
0.3194 };
float lookup55_pv14[16] = { 3.3486, 3.3486, 3.3486, 2.5226, 1.7713, 1.3465,
1.0762, 0.8904, 0.7557, 0.6538, 0.5744, 0.5109, 0.4591, 0.4161, 0.3799,
0.349 };
float lookup55_pv15[16] = { 3.3486, 3.3486, 3.3486, 2.7237, 1.9153, 1.4575,
1.166, 0.9655, 0.8199, 0.7098, 0.624, 0.5553, 0.4992, 0.4526, 0.4133,
0.3799 };
float lookup55_pv16[16] = { 3.3486, 3.3486, 3.3486, 2.921, 2.0564, 1.5663,
1.254, 1.039, 0.8828, 0.7646, 0.6724, 0.5986, 0.5383, 0.4882, 0.446, 0.41 };
float lookup55_pv17[16] = { 3.3486, 3.3486, 3.3486, 3.1201, 2.1997, 1.6773,
1.344, 1.1145, 0.9476, 0.8212, 0.7225, 0.6435, 0.579, 0.5253, 0.4801,
0.4415 };
float lookup55_pv18[16] = { 3.3486, 3.3486, 3.3486, 3.3152, 2.3393, 1.785,
1.4311, 1.1872, 1.0098, 0.8755, 0.7705, 0.6865, 0.6178, 0.5606, 0.5125,
0.4714 };
//DISPLACEMENT
//Displacment lookup table for 45m
float lookup45d_pv10[16] = { 1.7779, 1.5511, 0.9298, 0.5696, 0.4091, 0.3165,
0.2567, 0.215, 0.1844, 0.161, 0.1426, 0.1278, 0.1157, 0.1055, 0.0969,
0.0895 };
float lookup45d_pv11[16] = { 2.1438, 1.917, 1.2504, 0.7721, 0.5551, 0.4297,
0.3486, 0.2921, 0.2506, 0.219, 0.194, 0.174, 0.1574, 0.1436, 0.1319,
0.1219 };
float lookup45d_pv12[16] = { 2.5424, 2.3156, 1.6352, 1.0364, 0.743, 0.5739,
0.4648, 0.3888, 0.3332, 0.2907, 0.2574, 0.2305, 0.2084, 0.19, 0.1744,
0.161 };
float lookup45d_pv13[16] = { 2.9735, 2.7467, 2.0663, 1.3403, 0.9606, 0.7418,
0.6006, 0.5024, 0.4305, 0.3756, 0.3324, 0.2977, 0.2692, 0.2453, 0.2252,
0.2079 };
float lookup45d_pv14[16] = { 3.4366, 3.2098, 2.5294, 1.6956, 1.2152, 0.9385,
0.7599, 0.6356, 0.5446, 0.4751, 0.4206, 0.3766, 0.3405, 0.3104, 0.2849,
0.263 };
float lookup45d_pv15[16] = { 3.9315, 3.7047, 3.0243, 2.1069, 1.5108, 1.1673,
0.9455, 0.7912, 0.678, 0.5917, 0.5238, 0.4692, 0.4243, 0.3868, 0.355,
0.3278 };
float lookup45d_pv16[16] = { 4.4579, 4.2311, 3.5507, 2.5718, 1.8463, 1.4278,
1.1573, 0.969, 0.8308, 0.7254, 0.6425, 0.5757, 0.5208, 0.4749, 0.436,
0.4027 };
float lookup45d_pv17[16] = { 5.0154, 4.7886, 4.1082, 3.0856, 2.2196, 1.7192,
1.3953, 1.1695, 1.0037, 0.8771, 0.7774, 0.697, 0.6309, 0.5757, 0.5288,
0.4887 };
float lookup45d_pv18[16] = { 5.6037, 5.3769, 4.6965, 3.6557, 2.6358, 2.0451,
1.6622, 1.395, 1.1985, 1.0483, 0.9299, 0.8344, 0.7558, 0.6901, 0.6343,
0.5865 };
//Displacment lookup table for 50m
float lookup50d_pv10[16] = { 1.8607, 1.6246, 0.9764, 0.6072, 0.4323, 0.3326,
0.2685, 0.2249, 0.1928, 0.1688, 0.15, 0.135, 0.1222, 0.1125, 0.1035,
0.0969 };
float lookup50d_pv11[16] = { 2.2433, 2.0071, 1.3125, 0.8192, 0.5841, 0.4494,
0.363, 0.303, 0.2596, 0.2274, 0.2018, 0.1814, 0.1647, 0.1498, 0.1394,
0.1282 };
float lookup50d_pv12[16] = { 2.66, 2.4238, 1.7154, 1.0755, 0.7682, 0.5911,
0.4775, 0.399, 0.3416, 0.2986, 0.2646, 0.2375, 0.2158, 0.1974, 0.1812,
0.169 };
float lookup50d_pv13[16] = { 3.1105, 2.8743, 2.1658, 1.3799, 0.9874, 0.7602,
0.6141, 0.5129, 0.4396, 0.3837, 0.3398, 0.3045, 0.2762, 0.253, 0.2329,
0.2156 };
float lookup50d_pv14[16] = { 3.5944, 3.3582, 2.6498, 1.7357, 1.2448, 0.9595,
0.7751, 0.6479, 0.5546, 0.4839, 0.4285, 0.3838, 0.3477, 0.3176, 0.2933,
0.2713 };
float lookup50d_pv15[16] = { 4.1114, 3.8752, 3.1668, 2.1461, 1.5429, 1.191,
0.9629, 0.8043, 0.6882, 0.6007, 0.5319, 0.4769, 0.4321, 0.3944, 0.362,
0.3365 };
float lookup50d_pv16[16] = { 4.6611, 4.4249, 3.7165, 2.6139, 1.8844, 1.4567,
1.1787, 0.9852, 0.8435, 0.7351, 0.6512, 0.5839, 0.5289, 0.4829, 0.444,
0.4094 };
float lookup50d_pv17[16] = { 5.2431, 5.007, 4.2985, 3.1421, 2.272, 1.7587,
1.4245, 1.1907, 1.0201, 0.8894, 0.7875, 0.7061, 0.639, 0.5831, 0.5364,
0.4963 };
float lookup50d_pv18[16] = { 5.8572, 5.621, 4.9126, 3.7327, 2.7076, 2.0993,
1.7019, 1.424, 1.2192, 1.0642, 0.9419, 0.8444, 0.7634, 0.6957, 0.6395,
0.5924 };
//Displacment lookup table for 55m
float lookup55d_pv10[16] = { 1.9433, 1.6968, 1.0207, 0.635, 0.4521, 0.3478,
0.281, 0.2346, 0.2014, 0.176, 0.1564, 0.1403, 0.1276, 0.1173, 0.1077,
0.1006 };
float lookup55d_pv11[16] = { 2.3424, 2.0959, 1.3711, 0.8398, 0.6048, 0.4688,
0.3808, 0.3194, 0.2742, 0.2398, 0.2126, 0.1907, 0.1727, 0.1576, 0.1448,
0.1339 };
float lookup55d_pv12[16] = { 2.777, 2.5306, 1.7911, 1.1051, 0.7936, 0.6139,
0.4977, 0.4168, 0.3574, 0.3121, 0.2764, 0.2477, 0.2241, 0.2044, 0.1877,
0.1733 };
float lookup55d_pv13[16] = { 3.2468, 3.0004, 2.2609, 1.411, 1.013, 0.7834,
0.635, 0.5316, 0.4558, 0.398, 0.3525, 0.3159, 0.2857, 0.2606, 0.2392,
0.221 };
float lookup55d_pv14[16] = { 3.7514, 3.5049, 2.7654, 1.7687, 1.2698, 0.9819,
0.7959, 0.6664, 0.5714, 0.4989, 0.4419, 0.3959, 0.3582, 0.3266, 0.2999,
0.277 };
float lookup55d_pv15[16] = { 4.2903, 4.0438, 3.3043, 2.1828, 1.568, 1.2131,
0.9836, 0.8239, 0.7066, 0.6171, 0.5467, 0.4899, 0.4433, 0.4043, 0.3712,
0.3429 };
float lookup55d_pv16[16] = { 4.8631, 4.6166, 3.8771, 2.651, 1.9065, 1.4763,
1.1979, 1.004, 0.8615, 0.7527, 0.6671, 0.5981, 0.5413, 0.4939, 0.4537,
0.4192 };
float lookup55d_pv17[16] = { 5.4695, 5.223, 4.4836, 3.1687, 2.2834, 1.7709,
1.4388, 1.2072, 1.0368, 0.9067, 0.8041, 0.7214, 0.6533, 0.5964, 0.5481,
0.5067 };
float lookup55d_pv18[16] = { 6.1092, 5.8627, 5.1232, 3.7431, 2.7035, 2.1005,
1.7091, 1.4357, 1.2344, 1.0805, 0.9591, 0.8611, 0.7804, 0.7128, 0.6555,
0.6063 };
int main(int argc, char **argv)
{
init(argc, argv, "payload_node");
NodeHandle nh;
Subscriber vfr_sub = nh.subscribe<VFR_HUD>("mavros/vfr_hud", 10,
vfr_cb); //Telemetry subscriber
Subscriber way_reach_sub = nh.subscribe<WaypointReached>("mavros/mission/
reached", 10, wayreach_cb); //Waypoint reached subscriber
Subscriber way_list_sub = nh.subscribe<WaypointList>("mavros/mission/
waypoints", 10, waylist_cb); //Waypoint list
ServiceClient waypush_client = nh.serviceClient<WaypointPush>("mavros/
mission/push"); //Upload waypoints to FCU
Rate rate(20.0);
float h_max = 0; //Headwind heading
float g_min = 100000; //Minimum ground speed
float aspd = 0; //Airspeed variable
float ave_aspd = 0; //Average airspeed
float gspd = 0; //Groundspeed variable
float head = 0; //Heading variable
float ave_wspd = 0; //Average Wind Speed
int wspd_cnt = 0; //Wind speed measurement counter
float alt = 0; //Altitude variable
float ave_alt = 0; //Average altitude
float dra = 0; //Displacemnt for phase A release
float drb = 0; //Displacemnt for phase B release
float drf = 0; //Final displacment
//Wait till in the headwind determination loiter
```

```
while(WayReached.wp_seq < 10)
{
spinOnce();
rate.sleep();
}
//Determine headwind direction and speed
ROS_INFO("Calculating");
while(WayReached.wp_seq == 10)
{
head = HV.heading; //Current aircraft heading
aspd = HV.airspeed; //Current Airspeed
gspd = HV.groundspeed; //Current groundspeed
ave_wspd = ave_wspd + abs(aspd - gspd); //Cummulative windspeed
ave_aspd = ave_aspd + aspd; //Cummulative airspeed
alt = alt + HV.altitude; //Cummulative altitude
wspd_cnt = wspd_cnt + 1;
if(head <= g_min)
{
h_max = head;
}
spinOnce();
rate.sleep();
}
//Average headwind speed, altitude and airspeed
ave_wspd = ave_wspd / wspd_cnt;
ave_alt = alt / wspd_cnt;
ave_aspd = ave_aspd / wspd_cnt;
//Temp variables for lookup table search
int a = 0;
int b = 0;
int c = 0;
int g = round(ave_wspd);
//Phase A displacment
a = ceil(ave_aspd);
if (ave_alt >= 45 && ave_alt <= 50) //For altitudes between 45m and 50m
{
switch (a) //Airspeed switch statement
{
case 10:
b = lookup45d_pv10[g];
c = lookup50d_pv10[g];
break;
case 11:
b = lookup45d_pv11[g];
c = lookup50d_pv11[g];
break;
case 12:
b = lookup45d_pv12[g];
c = lookup50d_pv12[g];
break;
case 13:
b = lookup45d_pv13[g];
c = lookup50d_pv13[g];
break;
case 14:
b = lookup45d_pv14[g];
c = lookup50d_pv14[g];
```

```
break;
case 15:
b = lookup45d_pv15[g];
c = lookup50d_pv15[g];
break;
case 16:
b = lookup45d_pv16[g];
c = lookup50d_pv16[g];
break;
case 17:
b = lookup45d_pv17[g];
c = lookup50d_pv17[g];
break;
case 18:
b = lookup45d_pv18[g];
c = lookup50d_pv18[g];
break;
}
dra = b + (c - b)*((ave_alt - 45) / 5);
}
else if ((ave_alt >= 50 && ave_alt <= 55) || (ave_alt == 50)) //For
altitudes between 50m and 55m
{
switch (a) //Airspeed switch statement
{
case 10:
b = lookup50d_pv10[g];
c = lookup55d_pv10[g];
break;
case 11:
b = lookup50d_pv11[g];
c = lookup55d_pv11[g];
break;
case 12:
b = lookup50d_pv12[g];
c = lookup55d_pv12[g];
break;
case 13:
b = lookup50d_pv13[g];
c = lookup55d_pv13[g];
break;
case 14:
b = lookup50d_pv14[g];
c = lookup55d_pv14[g];
break;
case 15:
b = lookup50d_pv15[g];
c = lookup55d_pv15[g];
break;
case 16:
b = lookup50d_pv16[g];
c = lookup55d_pv16[g];
break;
case 17:
b = lookup50d_pv17[g];
c = lookup55d_pv17[g];
break;
```

```
case 18:
b = lookup50d_pv18[g];
c = lookup55d_pv18[g];
break;
}
dra = b + (c - b)*((ave_alt - 50) / 5);
}
else
{
dra = 2;
}
a = 0;
b = 0;
c = 0;
float heq = 0; //Equilibrium altitude
float tl = 0; //Time left to fall
float tint = 0;
float dbx = 0;
//Phase B displacment
if (ave_wspd >= 3) //Cancel out none Phase B falls
{
a = ceil(ave_aspd);
if (ave_alt >= 45 && ave_alt <= 50) //For altitudes between 45m and
50m
{
switch (a) //Airspeed switch statement
{
case 10:
b = lookup45_pv10[g];
c = lookup50_pv10[g];
break;
case 11:
b = lookup45_pv11[g];
c = lookup50_pv11[g];
break;
case 12:
b = lookup45_pv12[g];
c = lookup50_pv12[g];
break;
case 13:
b = lookup45_pv13[g];
c = lookup50_pv13[g];
break;
case 14:
b = lookup45_pv14[g];
c = lookup50_pv14[g];
break;
case 15:
b = lookup45_pv15[g];
c = lookup50_pv15[g];
break;
case 16:
b = lookup45_pv16[g];
c = lookup50_pv16[g];
break;
case 17:
b = lookup45_pv17[g];
```

```
c = lookup50_pv17[g];
break;
case 18:
b = lookup45_pv18[g];
c = lookup50_pv18[g];
break;
}
heq = ave_alt - ave_alt * ((b + (c - b)*((ave_alt - 45) / 5))/sqrt
((2*ave_alt)/9.81)); //Altitude of euqilibrium
tl = sqrt((2 * ave_alt) / 9.81) - ((b + (c - b)*((ave_alt - 45) /
5)));
tint = tl / 10; //remaining time 1oth
for (int x = 1; x <= 10; x++)
{
dbx = dbx + tint * ave_aspd*pow(((heq / 10) /
ave_alt),0.4);
}
drb = dbx; //Phase B displacment
}
else if ((ave_alt >= 50 && ave_alt <= 55) || (ave_alt == 50)) //For
altitudes between 50m and 55m
{
switch (a) //Airspeed switch statement
{
case 10:
b = lookup50_pv10[g];
c = lookup55_pv10[g];
break;
case 11:
b = lookup50_pv11[g];
c = lookup55_pv11[g];
break;
case 12:
b = lookup50_pv12[g];
c = lookup55_pv12[g];
break;
case 13:
b = lookup50_pv13[g];
c = lookup55_pv13[g];
break;
case 14:
b = lookup50_pv14[g];
c = lookup55_pv14[g];
break;
case 15:
b = lookup50_pv15[g];
c = lookup55_pv15[g];
break;
case 16:
b = lookup50_pv16[g];
c = lookup55_pv16[g];
break;
case 17:
b = lookup50_pv17[g];
c = lookup55_pv17[g];
break;
case 18:
```

```
b = lookup50_pv18[g];
c = lookup55_pv18[g];
break;
}
heq = ave_alt - ave_alt * ((b + (c - b)*((ave_alt - 45) / 5)) /
sqrt((2 * ave_alt) / 9.81)); //Altitude of euqilibrium
tl = sqrt((2 * ave_alt) / 9.81) - ((b + (c - b)*((ave_alt - 45) /
5)));
tint = tl / 10; //remaining time 10th
for (int x = 1; x <= 10; x++)
{
dbx = dbx + tint * ave_aspd*pow(((heq / 10) /
ave_alt),0.4);
}
drb = dbx; //Phase B displacement
}
}
else
drb = 0; //Failsafe value
//Final displacment
drf = dra - drb;
//Display calculated values
ROS_INFO("Displacement%f", drf);
ROS_INFO("Average Altitude%f", ave_alt);
ROS_INFO("Average Airspeed%f", ave_aspd);
ROS_INFO("Headwind Speed%f", ave_wspd);
ROS_INFO("Heading%f", h_max);
//Waypoint variables
WaypointPush waypush;
Waypoint WpRef; //Drop-zone location detected (waytpoint 15)
Waypoint wp_dal; //Drop approach lineup
Waypoint wp_da1; //Drop approach waypoint #1
Waypoint wp_da2; //Drop payload
Waypoint Points[3];
wp_dal.frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
wp_dal.command = 16;
wp_dal.is_current = false;
wp_dal.autocontinue = true;
wp_dal.param1 = 0;
wp_dal.param2 = 2;
wp_dal.param3 = 0;
wp_dal.param4 = 0;
wp_dal.z_alt = 50.0;
wp_da1.frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
wp_da1.command = 16;
wp_da1.is_current = false;
wp_da1.autocontinue = true;
wp_da1.param1 = 0;
wp_da1.param2 = 2;
wp_da1.param3 = 0;
wp_da1.param4 = 0;
wp_da1.z_alt = 50.0;
wp_da2.frame = 3; // mavros_msgs::Waypoint::FRAME_GLOBAL;
wp_da2.command = 16;
wp_da2.is_current = false;
wp_da2.autocontinue = true;
wp_da2.param1 = 0;
```

```
wp_da2.param2 = 2;
wp_da2.param3 = 0;
wp_da2.param4 = 0;
wp_da2.z_alt = 50.0;
WpRef = WayList.waypoints[15];
float h2 = h_max; //Heading for payload release
float h1 = h2; //Heading for waypoints
float R = 6378.1; //Radius of the Earth (km)
float B = 0; //Bearing for payload
float Bw = 0; //Bearing for lineup
float d = drf/1000; //Distance in km
//Heading correction
if(d < 0)
{
h2 = h2 - 180;
}
if(h2 >= 360)
{
h2 = h2 - 360;
}
else if(h2 < 0)
{
h2 = 360 + h2;
}
B = h2 * (pi / 180);
if(h1 >= 360)
{
h1 = h1 - 360;
}
else if(h1 < 0)
{
h1 = 360 + h1;
}
Bw = h1 * (pi / 180);
d = abs(d); //Absolute displacement
float lat_ref = WpRef.x_lat * (pi/180); //Initial target latitude
converted to radians
float long_ref = WpRef.y_long * (pi/180); //Initial target longitude
converted to radians
//Drop approach 1
wp_da1.x_lat = (asin(sin(lat_ref)*cos(0.100/R) + cos(lat_ref)*sin(0.100/R)
*cos(Bw))) * (180/pi);
wp_da1.y_long = (long_ref + atan2(sin(Bw)*sin(0.100/R)*cos(lat_ref), cos
(0.100/R) - sin(lat_ref)*sin(wp_da1.x_lat))) * (180/pi);
//Drop approach lineup
wp_dal.x_lat = (asin(sin(lat_ref)*cos((0.12)/R) + cos(lat_ref)*sin((0.12)/
R)*cos(Bw))) * (180/pi);
wp_dal.y_long = (long_ref + atan2(sin(Bw)*sin((0.12)/R)*cos(lat_ref), cos
((0.12)/R) - sin(lat_ref)*sin(wp_dal.x_lat))) * (180/pi);
//Drop Payload
wp_da2.x_lat = (asin(sin(lat_ref)*cos(d/R) + cos(lat_ref)*sin(d/R)*cos
(B))) * (180/pi);
wp_da2.y_long = (long_ref + atan2(sin(B)*sin(d/R)*cos(lat_ref), cos(d/R) -
sin(lat_ref)*sin(wp_da2.x_lat))) * (180/pi);
Points[0] = wp_dal;
Points[1] = wp_da1;
Points[2] = wp_da2;
```

```
waypush.request.start_index = 12;
ros::Time last_request = Time::now();
bool newMission = false;
//Transmit new waypoints
while (newMission != true)
{
if (newMission != true && (Time::now() - last_request > Duration
(0.2)))
{
for(int x = 0; x < 3; x++)
{
waypush.request.waypoints.push_back(Points[x]); //send all
waypoints to the FCU
}
if (waypush_client.call(waypush))
{
newMission = true; //Mission sent boolean (terminating
statement)
}
last_request = Time::now();
}
spinOnce();
rate.sleep();
}
return 0;
}
```

# Appendix B – Payload Equilibrium Time MATLAB Code

```
clc;
clear;
h = input('Enter the drop altitude: '); %Release altitude
va = input('Enter the initial payload airspeed: '); %Aircraft airspeed
vw = input('Enter the wind speed: '); %Headwind velocity at altitude
p = 1.225; %kg/m^3 - air density
A = 0.00283; %m^2 - cross-sectional area
Cd = 0.096; %drag coefficient
m = 0.14; %kg - payload mass
g = 9.81; %m/s^2 - gravitational coefficient
a = 2.5; %Hellmann coefficient
Vt = sqrt((2*m*g)/(p*A*Cd));%Terminal Velocity
tf = sqrt(2*h / g); %Fall time
var = round((tf/0.05) + 1);
dxdt = zeros(var, 1); %Deceleration matrix
td = zeros(var, 1); %Time matrix
t = 0;
i = 1;
while i <= var
dxdt(i) = ((p*A*Cd)/(2*m))*(((vw*((Vt^2/2*g)*(log(Vt^2/((-Vt*tan(g*t/Vt))^2 +
Vt^2)))/h)^(1/a))^2) + ((((Vt^2)*va)/((Vt^2)+g*va*t))^2)); %Equilibrium Time Equation
td(i) = t;
t = t + 0.05;
i = i + 1;
end
j = 1;
az = 1;
bz = 0;
while az >= 0
az = dxdt(j);
j = j + 1;
end
az = dxdt(j-2);
bz = dxdt(j-1);
%Plot
plot(td, dxdt);
title('PROJECTILE PHASE 1 ACCELERATION CURVE');
legend('Projectile acceleration');
xlabel('Time (Seconds)');
ylabel('Acceleration (m/^2)');
ylim([-inf inf]);
    xlim([-inf inf]);
```

## Appendix C1 - Pixhawk Specifications

### ***Key Features***

- Main System-on-Chip: STM32F427
  - CPU: 180 MHz ARM® Cortex® M4 with single-precision FPU
  - RAM: 256 KB SRAM (L1)
- Failsafe System-on-Chip: STM32F100
  - CPU: 24 MHz ARM Cortex M3
  - RAM: 8 KB SRAM
- Wifi: ESP8266 external
- GPS: U-Blox® 7/8 (Hobbyking®) / U-Blox 6 (3D Robotics)
- Optical flow: PX4 Flow unit
- Redundant power supply inputs and automatic failover
- External safety switch
- Multicolor LED main visual indicator
- High-power, multi-tone piezo audio indicator
- microSD card for high-rate logging over extended periods of time

**Connectivity**

- 1x I2C
- 1x CAN (2x optional)
- 1x ADC
- 4x UART (2x with flow control)
- 1x Console
- 8x PWM with manual override
- 6x PWM / GPIO / PWM input
- S.BUS / PPM / Spektrum input
- S.BUS output

### ***Specifications***

**Processor**

- 32bit STM32F427 Cortex-M4F core with FPU
- 168 MHz
- 256 KB RAM
- 2 MB Flash
- 32 bit STM32F103 failsafe co-processor

**Sensors**

- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- Invensense MPU 6000 3-axis accelerometer/gyroscope
- MEAS MS5611 barometer

**Interfaces**

- 5x UART (serial ports), one high-power capable, 2x with HW flow control
- 2x CAN (one with internal 3.3V transceiver, one on expansion connector)
- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
- Futaba S.BUS® compatible input and output
- PPM sum signal input
- RSSI (PWM or voltage) input
- I2C and SPI
- 3.3 and 6.6V ADC inputs
- Internal microUSB port and external microUSB port extension

**Power System and Protection**

- Ideal diode controller with automatic failover
- Servo rail high-power (max. 10V) and high-current (10A+) ready
- All peripheral outputs over-current protected, all inputs ESD protected

**Voltage Ratings**

Pixhawk can be triple-redundant on the power supply if three power sources are supplied. The three rails are: Power module input, servo rail input, USB input.

**Normal Operation Maximum Ratings**

Under these conditions all power sources will be used in this order to power the system

- Power module input (4.8V to 5.4V)
- Servo rail input (4.8V to 5.4V) **UP TO 10V FOR MANUAL OVERRIDE, BUT AUTOPILOT PART WILL BE UNPOWERED ABOVE 5.7V IF POWER MODULE INPUT IS NOT PRESENT**
- USB power input (4.8V to 5.4V)

**Absolute Maximum Ratings**

Under these conditions the system will not draw any power (will not be operational), but will remain intact.

- Power module input (4.1V to 5.7V, 0V to 20V undamaged)
- Servo rail input (4.1V to 5.7V, 0V to 20V)
- USB power input (4.1V to 5.7V, 0V to 6V)

## Appendix C2 – Pixhawk Port Pinouts

*TELEM1, TELEM2 ports*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | TX (OUT) | +3.3V |
| 3 (blk) | RX (IN) | +3.3V |
| 4 (blk) | CTS (IN) | +3.3V |
| 5 (blk) | RTS (OUT) | +3.3V |
| 6 (blk) | GND | GND |

*GPS port*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | TX (OUT) | +3.3V |
| 3 (blk) | RX (IN) | +3.3V |
| 4 (blk) | CAN2 TX | +3.3V |
| 5 (blk) | CAN2 RX | +3.3V |
| 6 (blk) | GND | GND |

*SERIAL 4/5 port - due to space constraints two ports are on one connector.*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | TX (#4) | +3.3V |
| 3 (blk) | RX (#4) | +3.3V |
| 4 (blk) | TX (#5) | +3.3V |
| 5 (blk) | RX (#5) | +3.3V |
| 6 (blk) | GND | GND |

*ADC 6.6V*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | ADC IN | up to +6.6V |
| 3 (blk) | GND | GND |

*ADC 3.3V*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | ADC IN | up to +3.3V |
| 3 (blk) | GND | GND |
| 4 (blk) | ADC IN | up to +3.3V |
| 5 (blk) | GND | GND |

*I2C*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | SCL | +3.3 (pullups) |
| 3 (blk) | SDA | +3.3 (pullups) |
| 4 (blk) | GND | GND |

*CAN*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | CAN_H | +12V |
| 3 (blk) | CAN_L | +12V |
| 4 (blk) | GND | GND |

*SPI*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | SPI_EXT_SCK | +3.3 |
| 3 (blk) | SPI_EXT_MISO | +3.3 |
| 4 (blk) | SPI_EXT_MOSI | +3.3 |
| 5 (blk) | !SPI_EXT_NSS | +3.3 |
| 6 (blk) | !GPIO_EXT | +3.3 |
| 7 (blk) | GND | GND |

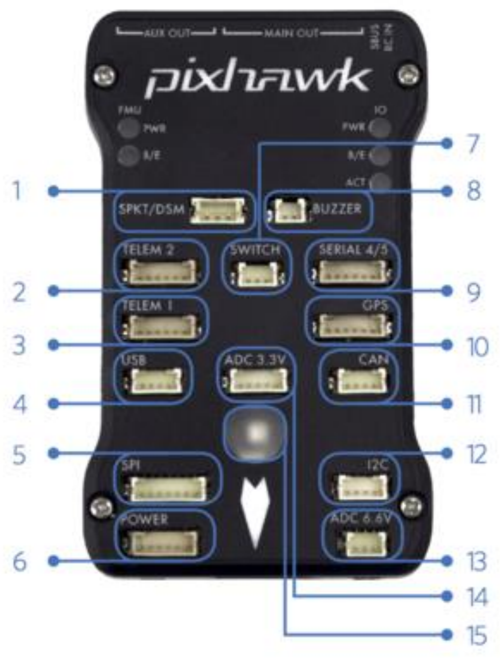
*POWER*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +5V |
| 2 (blk) | VCC | +5V |
| 3 (blk) | CURRENT | +3.3V |
| 4 (blk) | VOLTAGE | +3.3V |
| 5 (blk) | GND | GND |
| 6 (blk) | GND | GND |

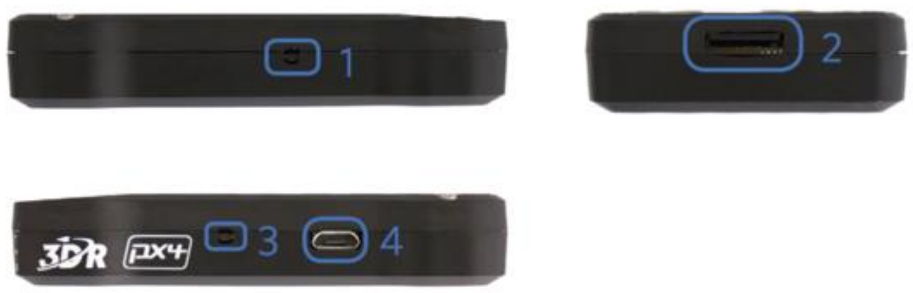
*SWITCH*

| Pin | Signal | Volt |
|---|---|---|
| 1 (red) | VCC | +3.3V |
| 2 (blk) | !IO_LED_SAFETY | GND |
| 3 (blk) | SAFETY | GND |

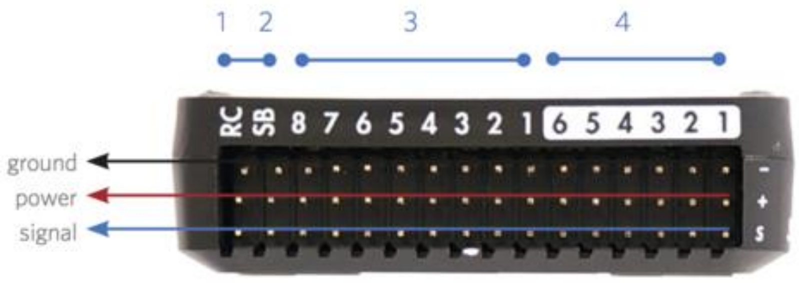
## Appendix C3 – Pixhawk Schematic



| | |
|---|---|
| 1 | Spektrum DSM receiver |
| 2 | Telemetry (radio telemetry) |
| 3 | Telemetry (on-screen display) |
| 4 | USB |
| 5 | SPI (serial peripheral interface) bus |
| 6 | Power module |
| 7 | Safety switch button |
| 8 | Buzzer |
| 9 | Serial |
| 10 | GPS module |
| 11 | CAN (controller area network) bus |
| 12 | I²C splitter or compass module |
| 13 | Analog to digital converter 6.6 V |
| 14 | Analog to digital converter 3.3 V |
| 15 | LED indicator |



1 Input/output reset button
2 SD card
3 Flight management reset button
4 Micro-USB port



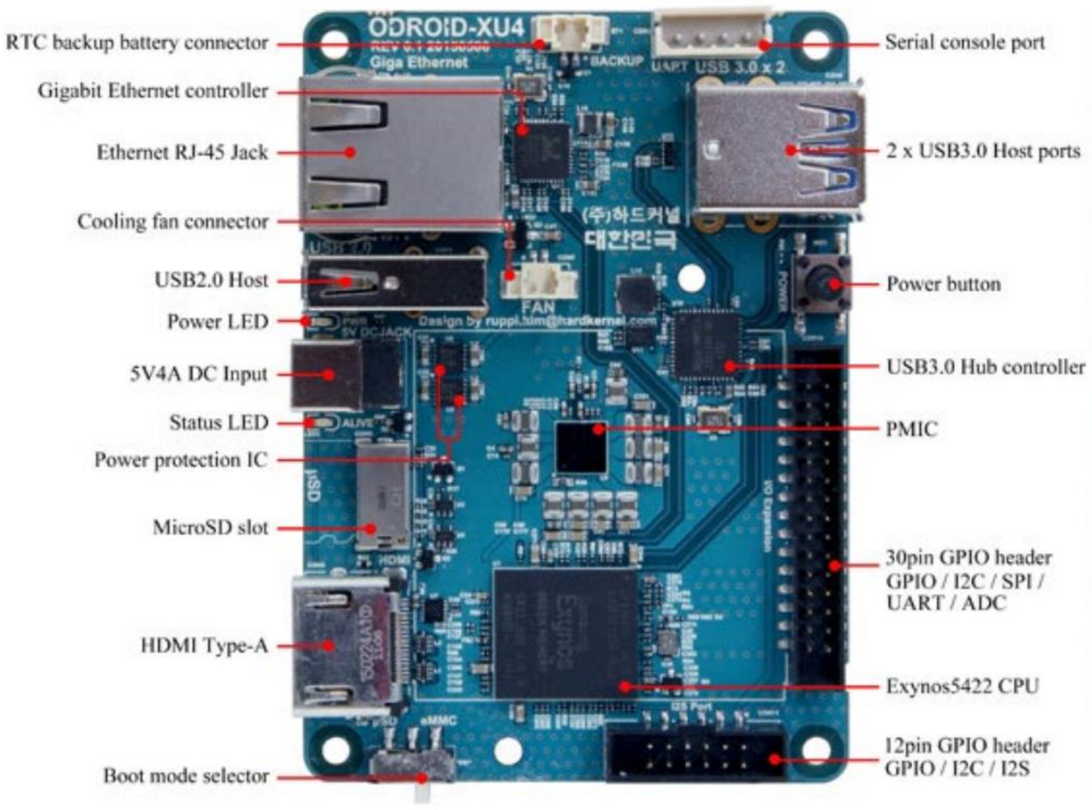
1 Radio control receiver input
2 S.Bus output
3 Main outputs
4 Auxiliary outputs

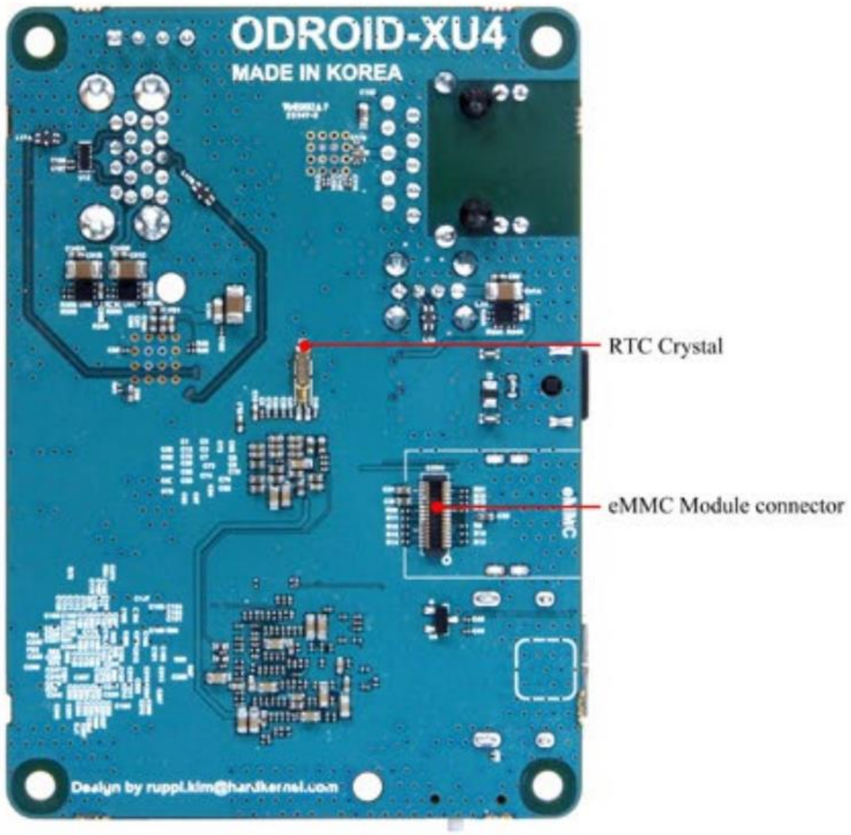
## Appendix D1 – ODroid-XU4 Specifications

*SPECIFICATIONS*

| | |
|---|---|
| Processor | Samsung Exynos5422 ARM® Cortex™-A15 Quad 2.0GHz/Cortex™-A7 Quad 1.4GHz |
| Memory | 2Gbyte LPDDR3 RAM PoP (750Mhz, 12GB/s memory bandwidth, 2x32bit bus) |
| 3D Accelerator | Mali™-T628 MP6 OpenGL ES 3.1 / 3.0 / 2.0 / 1.1 and OpenCL 1.2 Full profile |
| Audio | HDMI Digital audio output. Optional USB sound card |
| USB3.0 Host | SuperSpeed USB standard A type connector x 2 port |
| USB2.0 Host | HighSpeed USB standard A type connector x 1 port |
| Display | HDMI 1.4a with a Type-A connector |
| Storage (Option) | eMMC module socket : eMMC 5.0 Flash Storage (up to 64GByte)<br>MicroSD Card Slot (up to 128GByte) |
| Fast Ethernet LAN | 10/100/1000Mbps Ethernet with RJ-45 Jack ( Auto-MDIX support) |
| WiFi (Option) | USB IEEE 802.11 ac/b/g/n 1T1R WLAN with Antenna (External USB adapter) |
| HDD/SSD SATA interface (Optional) | SuperSpeed USB (USB 3.0) to Serial ATA3 adapter for 2.5"/3.5" HDD and SSD storage |
| Power Input | 4.8Volt~5.2Volt (5V/4A Power supply is recommended) |
| System Software | Ubuntu 16.04 + OpenGL ES + OpenCL on Linux Kernel 4.14 LTS<br>Android 4.4.2 on Kernel LTS 3.10<br>Android 7.1 is available as a community driven OS development.<br>Full source code is accessible via our Github. |
| Size | 83 x 58 x 20 mm (weight: 38gram) without cooler approx. |

## Appendix D2 – ODroid-XU4 Schematic

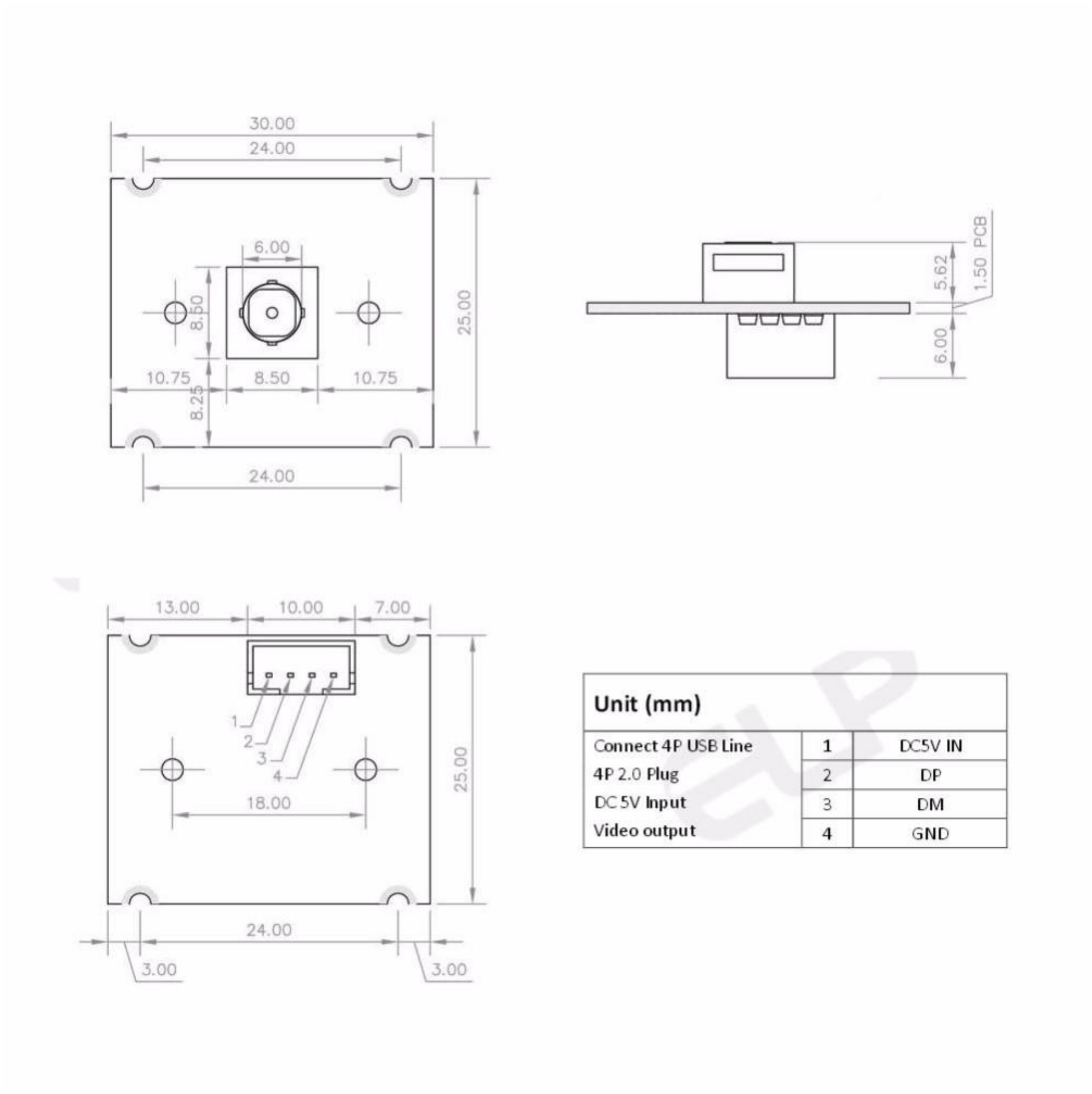




## Appendix E1 - ELP-USB500W04AF-A60 Camera Specifications

| | |
|---|---|
| Module No. | ELP-USB500W04AF-A60 |
| Sensor | OV5640 (1/4” ) |
| Max. Resolution | 2592 (H) *1944 (V) |
| Sensitivity | 600mV/Lux-sec |
| Pixel Size | 1.4µm x1.4µm |
| Image area | 3673.6µmx2738.4µm |
| Maximum Image Transfer Rate | 2592x1944@ 15fps MJPEG / 2048x1536@ 15fps MJPEG<br>1600x1200@ 15fps MJPEG / 1920x1080@ 15fps MJPEG<br>1280x1024@15fps MJPEG/ 1280x720@ 30fps MJPEG<br>1024 x 768@ 30fps MJPEG/ 800 x 600@ 30fps MJPEG<br>640x480@ 30fps MJPEG / 320x240@ 30fps MJPEG |
| Camera Board Assembly technique | SMT (ROSH) |
| Focus | Auto |
| Object distance | 5CM-100M |
| Resolution | 600LW/PH (Center) |
| interface | USB 2.0 High Speed |
| protocol | USB Video Class（UVC） |
| S/N Ratio | 36dB |
| Dynamic Range | 68dB |
| Shutter | Rolling shutter/ frame exposure |
| Package | CSP, Bare Die |
| AGC/AEC/AWB/ABF | Auto |
| Output Formats | YUYV/MJPEG |
| Adjustable parameter | Brightness Contrast Hue Saturation Sharpness Gamma White Balance Exposure Focus |
| Package | LQFN-40pin |
| Lens Parameter | Size: 1/4, Iris: F2.8, Focus: 3.2mm, FOV: 65Degree<br>Relative Illumination (Sensor): 70%<br>IR Filter: 650±10nm |
| LED board power connector | Support 2P-2.0mm socket |
| Power supply | USB BUS POWER 4P-2.0mm socket |
| Operating Voltage | DC5V |
| Power consumption | 150mW （VGA）; 200mW （QSXGA）; |
| Module Size | 32X25MM |
| Temperature (Operation) | -20°C to 70°C |
| Temperature (Stable Image) | 0°C to 60°C |
| USB Cable | 1M(2M/3M/5M Optional) |
| Operating system request | Win XP/Vista/Win7/Win8 / Linux with UVC（above linux-2.6.26）<br>MAC-OS X 10.4.8 or later/Android 4.0 or above with UVC |
| Certifications | FCC and CE |

## Appendix E2 - ELP-USB500W04AF-A60 Camera Schematic



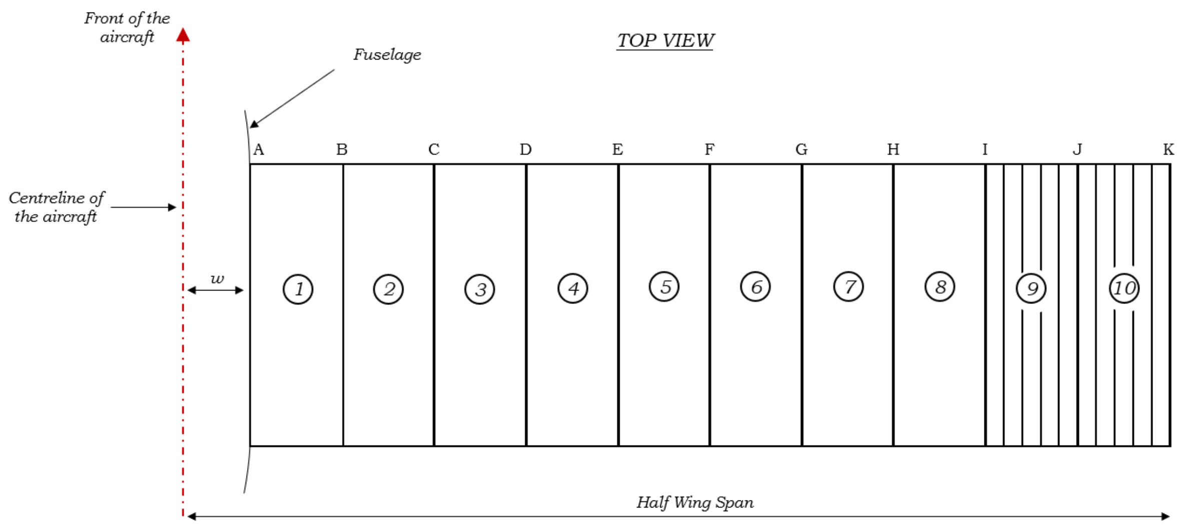
| Unit (mm) | | |
|---|---|---|
| Connect 4P USB Line | 1 | DC5V IN |
| 4P 2.0 Plug | 2 | DP |
| DC 5V Input | 3 | DM |
| Video output | 4 | GND |

# Appendix F - TAROT GoPro 3D V metal 3-axis gimbal Specifications

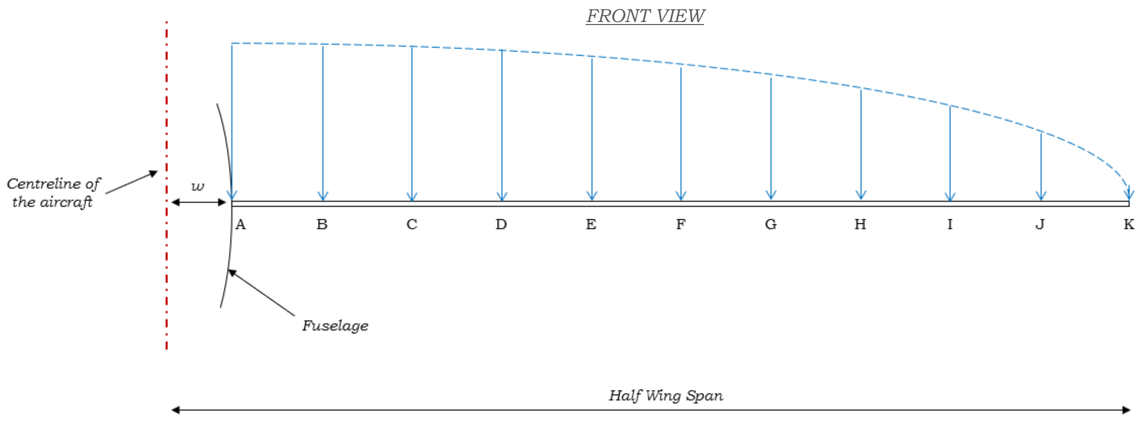
| | |
|---|---|
| Input power | 3S-6S Li(11V-26V) |
| Operating current | 30mA (@ 25V)/50mA (@ 12V) |
| Stall current | 350mA (@ 25V)/700mA (@ 12V) |
| Attitude control accuracy | ±0.02 degrees |
| *Maximum controllable speed* | |
| Rotation direction (PAN) | ±200°/s |
| Tilt direction (TILT) | ±200°/s |
| Rolling direction (ROLL) | ±200°/s |
| *Controllable rotation range* | |
| Rotation direction (PAN) | ±330° |
| Tilt direction (TILT) | -135° to +45° |
| Rolling direction (ROLL) | ±48° |
| | |
| S-Bus/PPM/DSM receiver support | |
| Working temperature | -20℃ ~ +50℃ |
| Maximum dimensions | 60x75x100mm |
| *Adjust the software installation requirements* | |
| Windows XP; Windows VISTA; Windows 7; Windows 8 (32 or 64 bit) | |

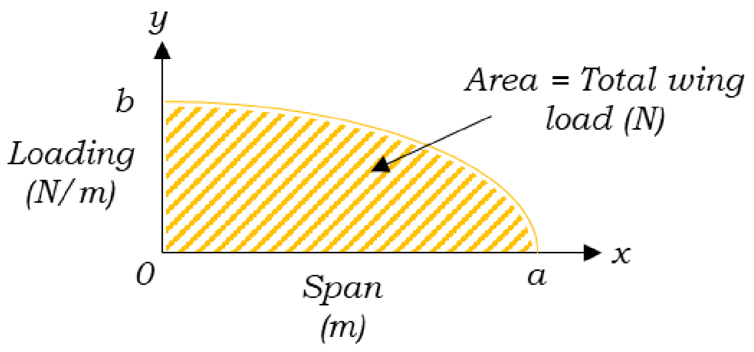
## Appendix G - Wing Design Calculation

The Clark Y aerofoil was selected as the desired new wing aerofoil profile. Two EPS Wing division for strength analysis



Elliptical wing loading function





*Maximum Wing Load*

n = 4 – Wing loading factor

$m_a$ = 4 kg – Aircraft mass

$m_w$ = 0.2 kg – Single wing mass

The maximum expected wing load was defined to be,

$$F_{WL} = \frac{gn(m_a - 2m_w)}{2}$$

$$F_{WL} = \frac{9.81(4)(5-2(0.2))}{2}$$

$$F_{WL} = 90.252\,N$$

*Introducing a factor of safety, $F_{WL} = 100\,N$

Knowing the maximum expected wing load, the maximum bending moment, $b$, and the loaded span length, $a$, could be deduced.

$$a = \frac{Span}{2} - w$$

$$a = \frac{1.975}{2} - 0.055$$

$$a = 0.9325\,m$$

$$A_{ellipse} = \frac{\pi ab}{4} = F_{WL}$$

$$\frac{\pi(0.9325)b}{4} = 100$$

$$b = 136.54043375176\,N/m$$

With the value for $b$ now known, the elliptical load function could be determined with respect to the displacement

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$$y = \sqrt{b^2\left(1 - \frac{x^2}{a^2}\right)}$$

$$y = 136.54043375176\sqrt{1 - \frac{x^2}{0.86955625}}$$

For each wing segment, numbered 1 – 10, the net force exerted over the wing area could be determined. The net force per segment was represented by the area of the elliptical wing loading graph, which was initially determined through application of a Left Riemann Sum of the elliptical load function. Subdivisions for the Riemann Sum were taken to be the wing segment divisions. The resulting wing loading results incurred an overshoot due to the application of the Left Riemann Sum and as such, a finite integral was used to determine a less exaggerated wing loading force.

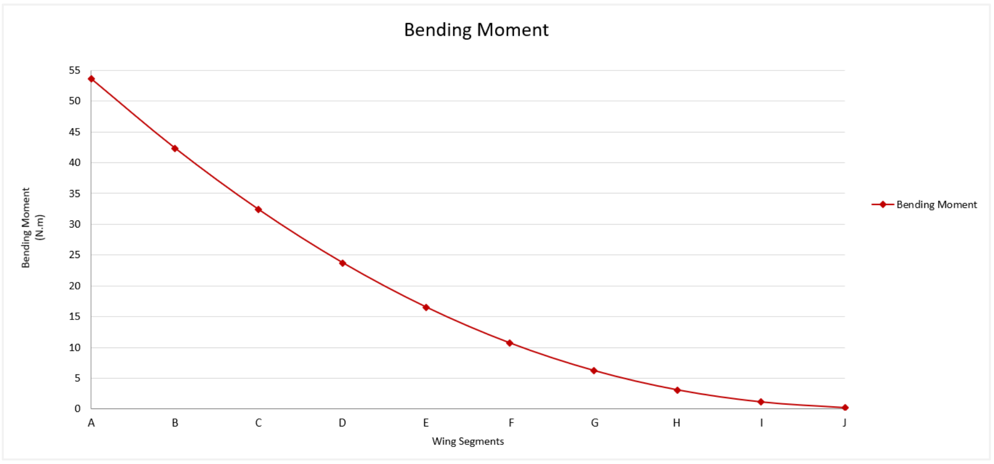
$$F_{net} = \int_{d0}^{d1}\left(136.5404\sqrt{1 - \frac{x^2}{0.86955625}}\right)dx$$

*Maximum Bending Moment and Shear Force*

From the net force, the bending moment of the wing was then determined. The maximum bending moment was found to be 53.584 N.m at wing point A.

$M_{Seg} = \sum_{n=K}^{Seg} F_{net_n} d_n$

Where, $d_n$ represented the bending moment test point displacement from the reference wing point. The bending moment curve of the wing was found to be as follows,

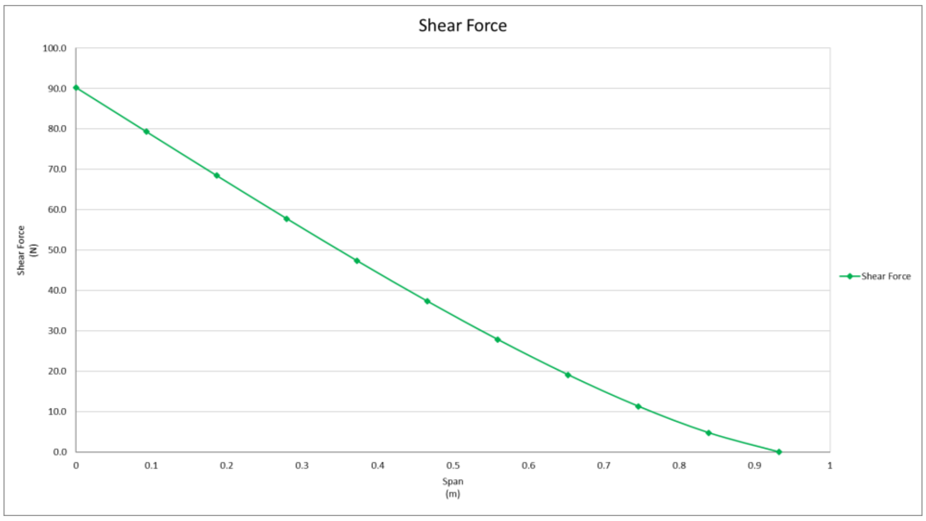


The maximum bending moment represented the required minimum bending moment the wing's spar caps were required to be able to endure.

The required shear strength of the wing was represented by the shear strength of the structural material between the spar caps and the wing surface. In the case of this wing, this shear strength was the strength of the EPS foam. Required shear strength was determined by determining the shear stress exerted on the wing by the shear force. Where, shear stress was defined to be the shear force applied to a given area.

$\tau = \frac{V}{A}$

The shear force curve for the wing was found to



The required shear strength of the wing was determined to be 407.068 kPa and was within the allowable shear strength of the EPS foam which was found to be 482.633 kPa. This provided a shear strength factor of safety of 1.186.

*Spar Cap Moment of Area*

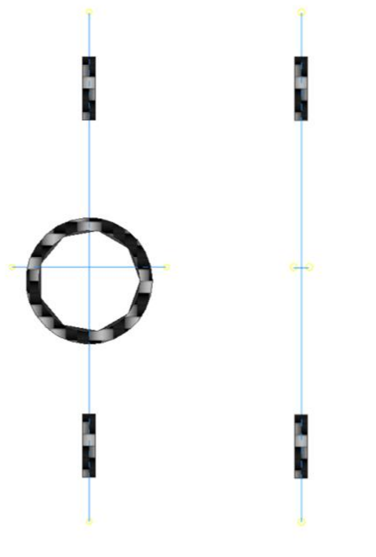
With the required bending moment of the wing's spar caps known, the require moment of inertia could be determined. The desired spar cap material was selected to be pultruded Carbon Fibre, with yield strength of 1.6 GPa.

$$I = \frac{My}{\sigma}$$

The required spar cap moments of inertia were found to be as follows;

| *Wing Segment* | *Moment of Inertia ($m^4$)* |
|---|---|
| 1 | $8.372 \times 10^{-12}$ |
| 2 | $6.613 \times 10^{-12}$ |
| 3 | $5.055 \times 10^{-12}$ |
| 4 | $3.708 \times 10^{-12}$ |
| 5 | $2.579 \times 10^{-12}$ |
| 6 | $1.669 \times 10^{-12}$ |
| 7 | $0.9719 \times 10^{-12}$ |
| 8 | $0.4791 \times 10^{-12}$ |
| 9 | $0.173 \times 10^{-12}$ |
| 10 | $0.0272 \times 10^{-12}$ |

Due to the high yield strength of the pultruded carbon fibre and diminishing moment of inertia required for each wing segment, a two-stage spar cap design was executed. The spar cap designs can be seen below.



*First Spar Cap* *Second Spar Cap*

Where,

First Spar Cap: $I = 2.30515 \times 10^{-9}\ m^4$

Second Spar Cap: $I = 2.006537 \times 10^{-9}\ m^4$

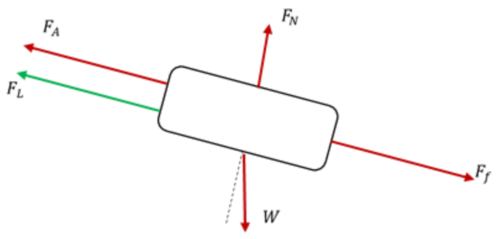
| Wing Point | | Net Force | Shear Stress | Bending Moment | Moment of Inertia |
|---|---|---|---|---|---|
| A | 1 | 12.7111397 | 1.185313777 | 53.58356654 | 8.37243E-12 |
| B | 2 | 12.58284624 | 1.173350412 | 42.32308565 | 6.61298E-12 |
| C | 3 | 12.32223829 | 1.149048721 | 32.34960715 | 5.05463E-12 |
| D | 4 | 11.92060598 | 1.111596508 | 23.73174175 | 3.70808E-12 |
| E | 5 | 11.36293284 | 1.059593487 | 16.50636445 | 2.57912E-12 |
| F | 6 | 10.62452128 | 0.990736609 | 10.67860027 | 1.66853E-12 |
| G | 7 | 9.66365516 | 0.901135844 | 6.220285526 | 9.7192E-13 |
| H | 8 | 8.403234494 | 0.783601617 | 3.066310073 | 4.79111E-13 |
| I | 9 | 6.670194871 | 0.621995672 | 1.107306032 | 1.73017E-13 |
| J | 10 | 3.738606423 | 0.348625049 | 0.174312524 | 2.72363E-14 |

## Appendix H1 – Catapult Launcher Calculations

Launch angle: 15° ($\alpha$)

Launch velocity: 18 knots ~ 9.26 m/s ($V_L$)

Catapult Length: 2 m ($L$)



From the force diagram,

*Normal force*

$$F_N = W\cos(\alpha) = mg\cos(\alpha)$$

$$F_N = 3(9.81)\cos(15) = 28.42719707\ N$$

*Frictional Force*

$$F_f = \mu F_N + W\sin(\alpha)$$

$$F_f = (0.4)27.42719707 = \ 10.97083883\ N$$

*Applied Force*

$$F_A = F_f + W\sin(\alpha)$$

$$F_A = 10.97083883 + (3)(9.81)\sin(15) = 18.5878833\ N$$

*Launch Force*

*Launch time*

$$t = \frac{2L}{V_L + V_i}$$

$$t = \frac{2(2)}{9.26+0} = 0.432\ s$$

*Launch Acceleration*

$$a = \frac{dV_L}{dt}$$

$$a = 9.26/0.432 = 21.4352\ m/s^2$$

$F_L = ma$

$F_L = 3(21.4352) = 64.3056$ N

Introduction of launch trolley, new launcher displacement is now 1.8 m. Additional applied force provided from aircraft's motor, reduced preload required.

*Elastic constant*

$k = \frac{F_L}{x}$

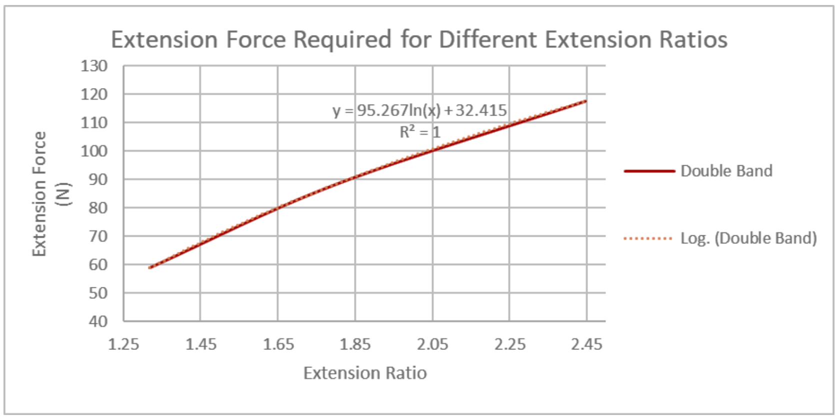
$k = \frac{64.3056}{1.8} = 35.7253 \cong 36$

*Rubber extension testing*

| *Pieces of Rubber Tubing* | *Extension Distance (m)* | *Extension Ratio* | *Required Force for Extension (N)* |
|---|---|---|---|
| Single Band (Initial length – 4.7 m) | 7 | 1.4894 | 34.34 |
| | 10 | 2.1277 | 49.05 |
| | 13.4 | 2.8511 | 68.67 |
| Double Band (Initial length – 2.35 m) | 3.1 | 1.3191 | 58.86 |
| | 4.23 | 1.8 | 88.29 |
| | 5.75 | 2.4468 | 117.72 |

Double band rubber tubing selected as desired launch system.

*Required extension ratio*



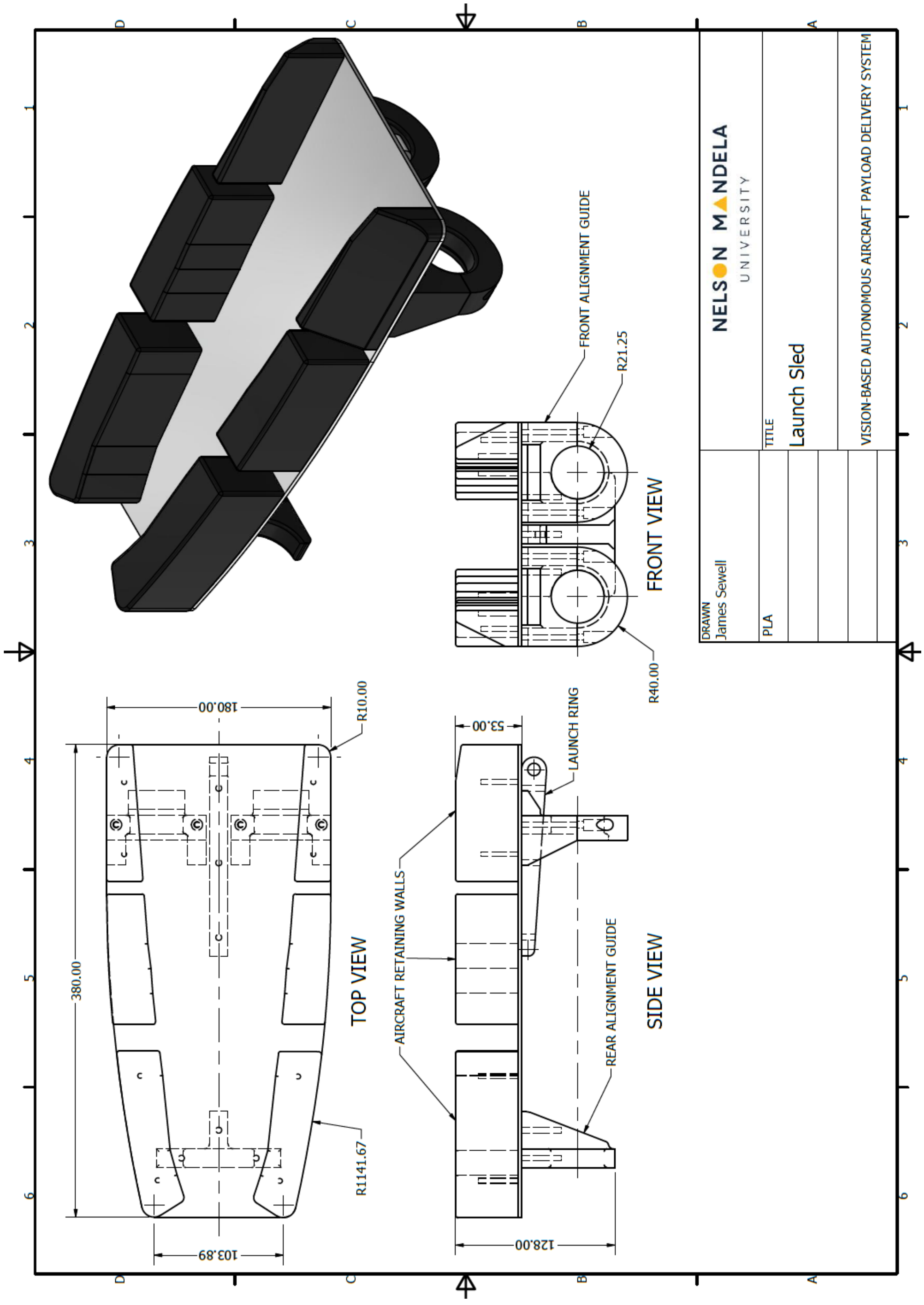
$y = 95.267\ln(x) + 32.415 = F_L$

$64.3056 = 95.267\ln(x) + 32.415$

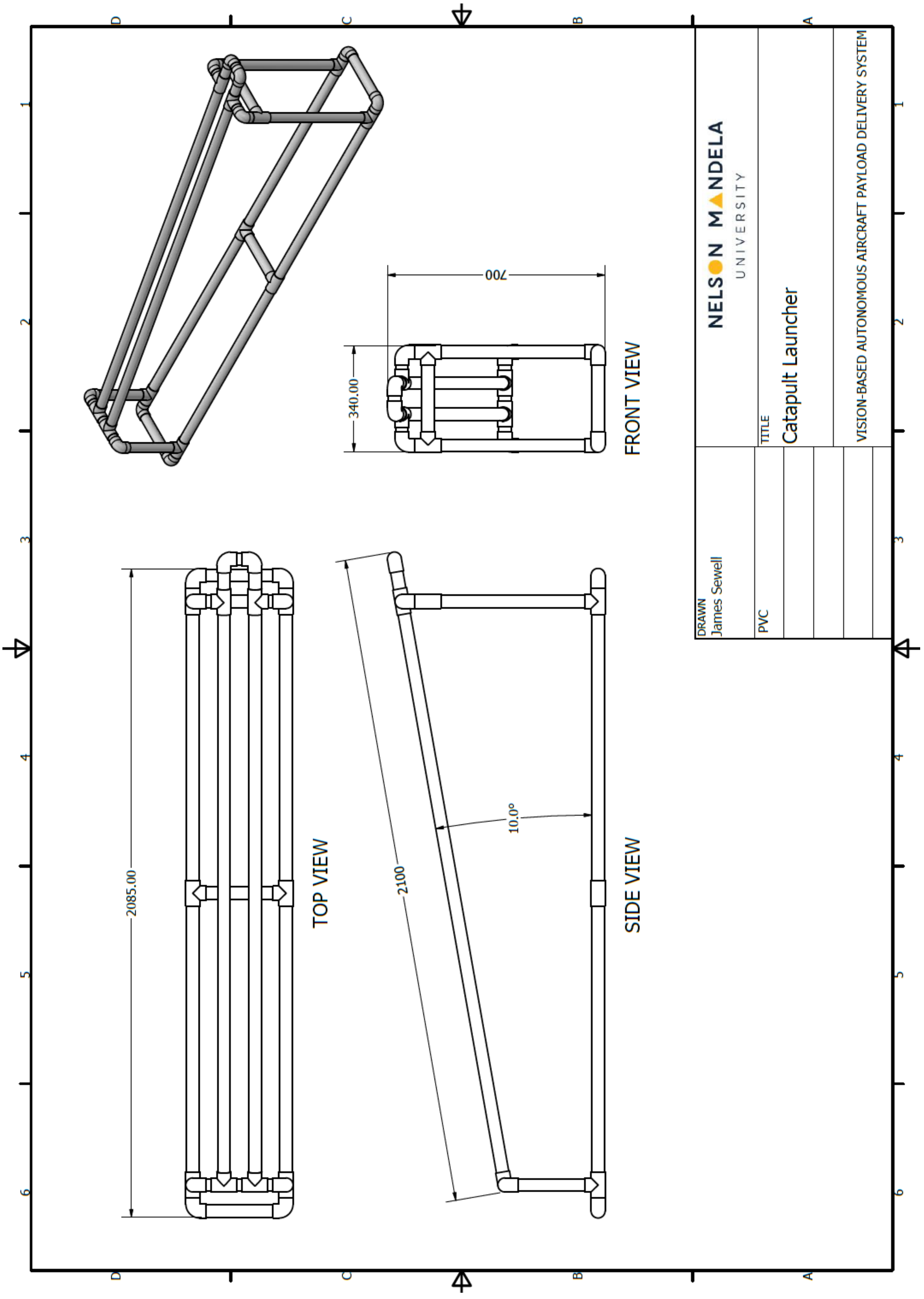
Extension ratio (x) = 1.398

The final extension ratio was changed to 1.8 to allow for additional preload of ~30 N.

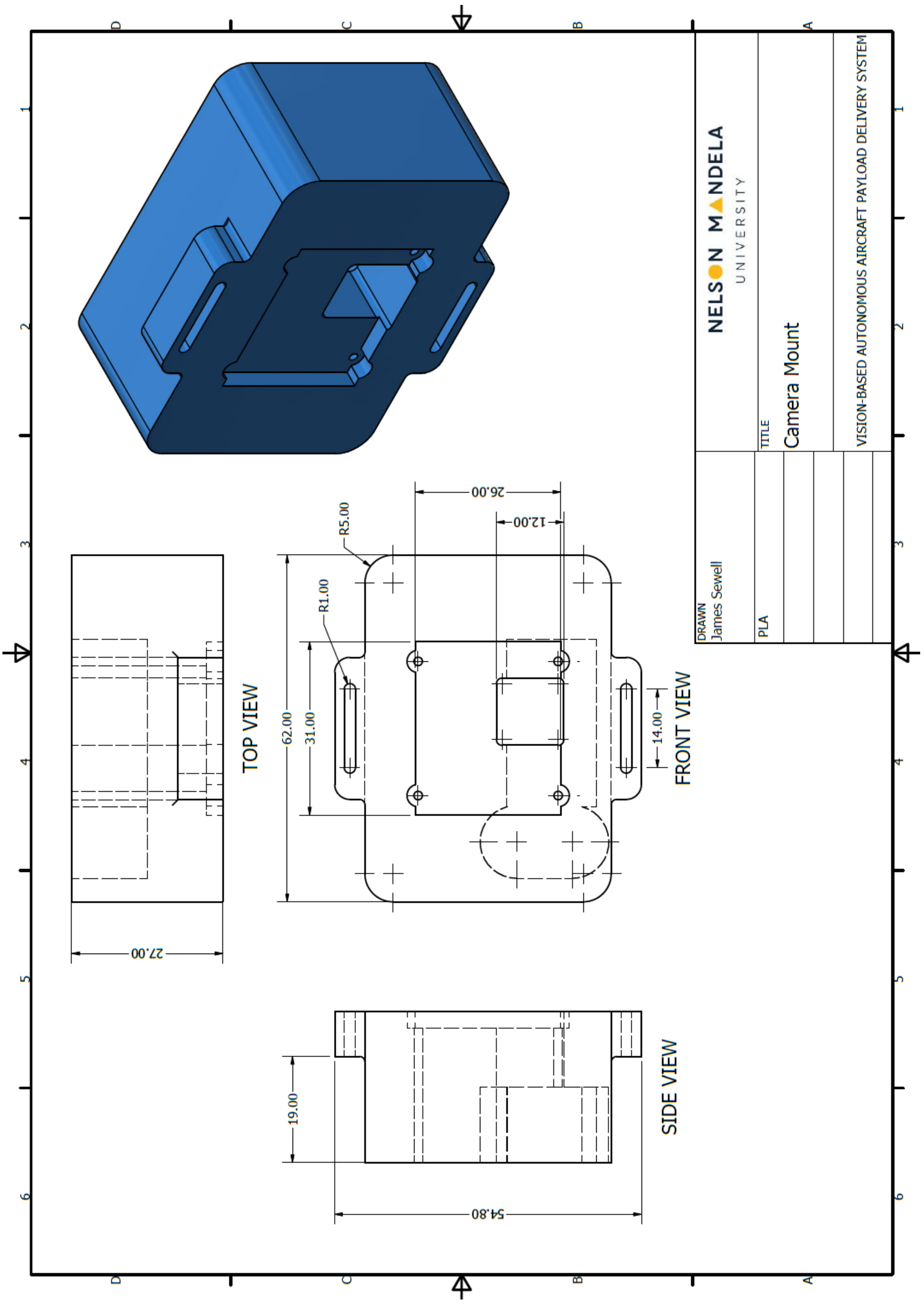
## Appendix H2 - Launcher Sled and Catapult CAD Design



TOP VIEW
380.00
180.00
103.89
R10.00
R1141.67
SIDE VIEW
53.00
128.00
AIRCRAFT RETAINING WALLS
LAUNCH RING
REAR ALIGNMENT GUIDE
FRONT VIEW
FRONT ALIGNMENT GUIDE
R21.25
R40.00
NELSON MANDELA UNIVERSITY
DRAWN
James Sewell
PLA
TITLE
Launch Sled
VISION-BASED AUTONOMOUS AIRCRAFT PAYLOAD DELIVERY SYSTEM

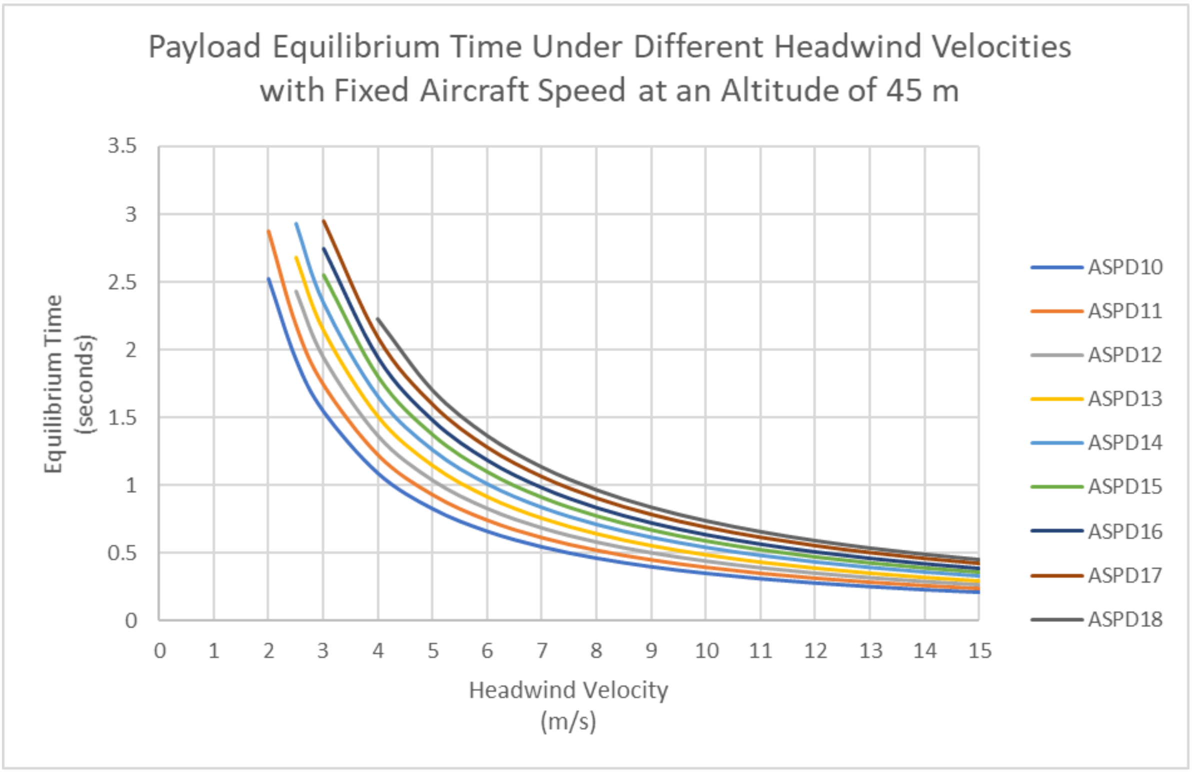

TOP VIEW
2085.00
SIDE VIEW
2100
10.0°
FRONT VIEW
700
340.00
NELSON MANDELA UNIVERSITY
DRAWN
James Sewell
PVC
TITLE
Catapult Launcher
VISION-BASED AUTONOMOUS AIRCRAFT PAYLOAD DELIVERY SYSTEM
A
B
C
D
1
2
3
4
5
6

## Appendix I – CMOS Camera Mount CAD Design


TOP VIEW
FRONT VIEW
SIDE VIEW
27.00
62.00
31.00
R1.00
R5.00
26.00
12.00
14.00
19.00
54.80
NELSON MANDELA UNIVERSITY
TITLE
Camera Mount
VISION-BASED AUTONOMOUS AIRCRAFT PAYLOAD DELIVERY SYSTEM
DRAWN
James Sewell
PLA

## Appendix J1 – Payload Release Equilibrium Time Lookup Tables

Altitude of 45 m



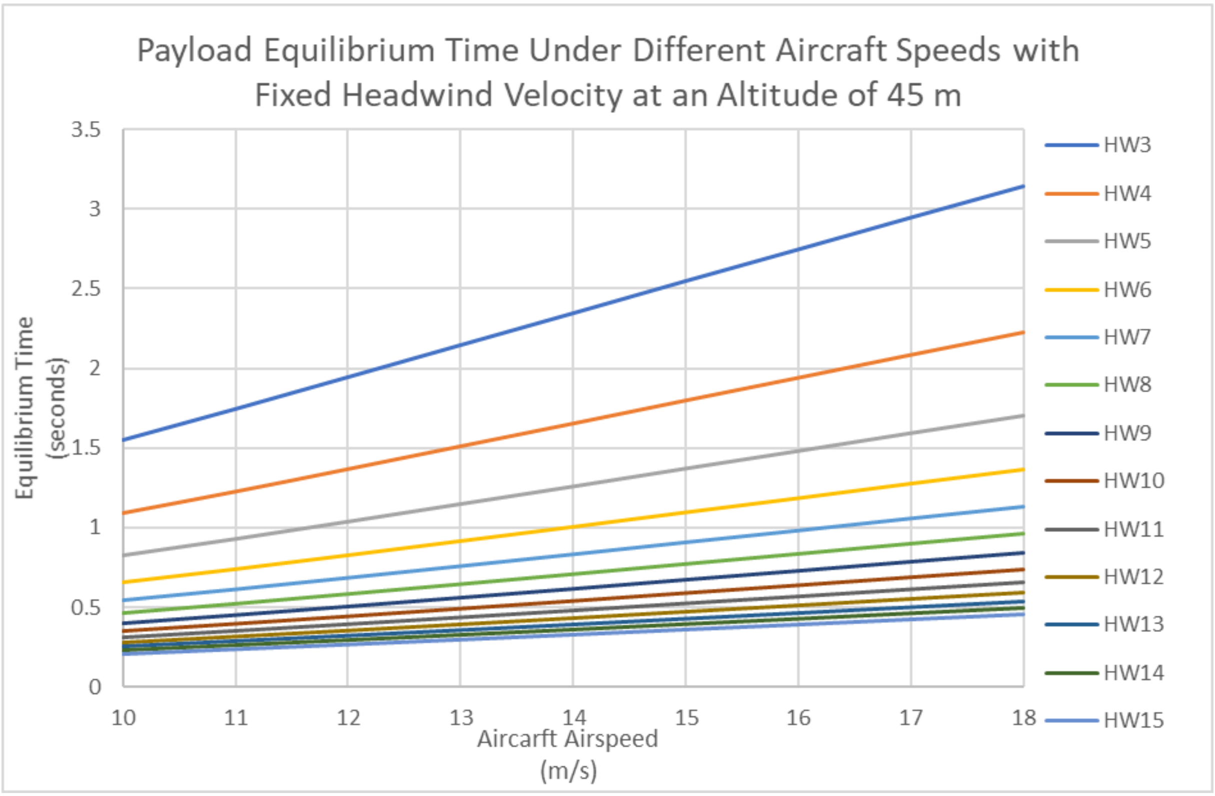
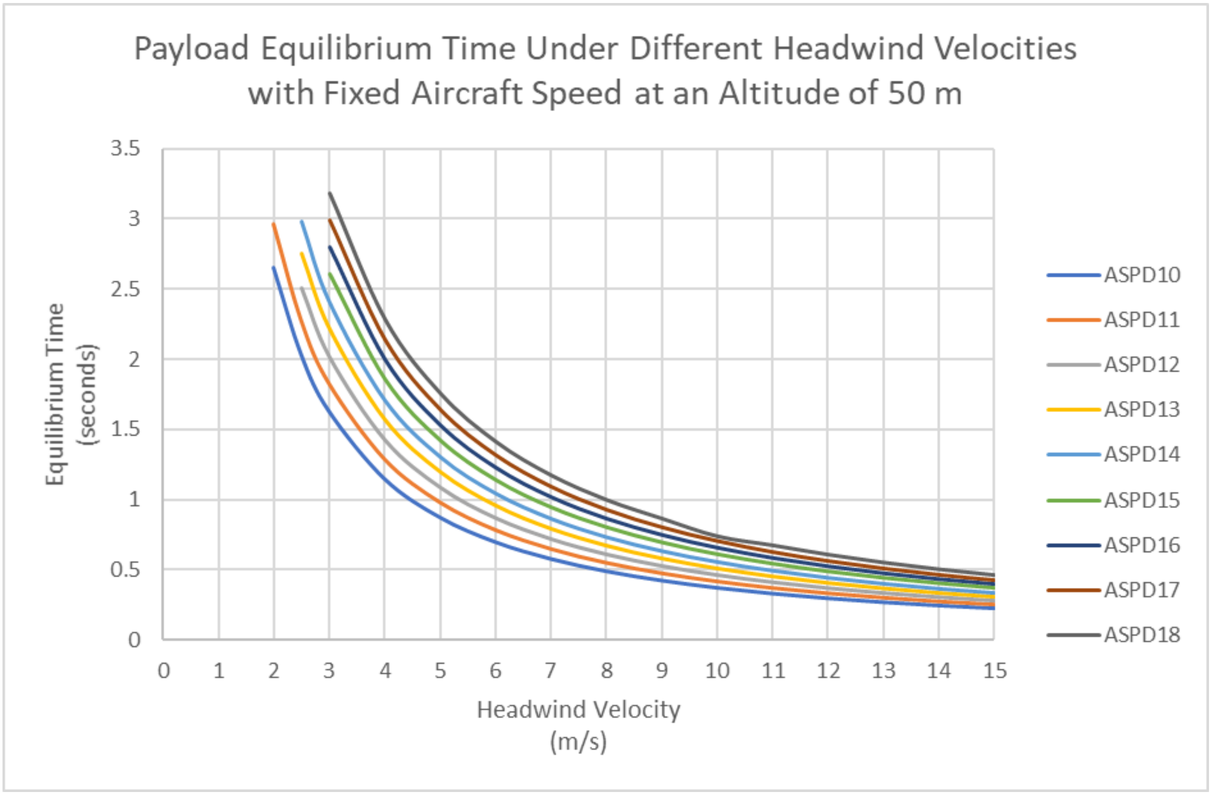


*Table K1.12: Payload Release Equilibrium Time Lookup Tables for an altitude of 45 m*

| Aircraft Speed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | - | - | - | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | - |
| 2 | 2.524324 | 2.870736 | - | - | - | - | - | - | - |
| 2.5 | 1.93125 | 2.181695 | 2.430843 | 2.680707 | 2.929263 | 3.17894 | 3.424242 | 3.670357 | 3.913149 |
| 3 | 1.548333 | 1.743416 | 1.943576 | 2.144918 | 2.345506 | 2.547747 | 2.746345 | 2.946422 | 3.143044 |
| 4 | 1.088953 | 1.223845 | 1.365531 | 1.508727 | 1.651719 | 1.796721 | 1.939006 | 2.083261 | 2.224202 |
| 5 | 0.827193 | 0.930095 | 1.038468 | 1.148391 | 1.258354 | 1.370351 | 1.480192 | 1.592089 | 1.700937 |
| 6 | 0.66069 | 0.743249 | 0.830305 | 0.918863 | 1.007583 | 1.098262 | 1.187159 | 1.278068 | 1.366194 |
| 7 | 0.545858 | 0.614879 | 0.687217 | 0.760983 | 0.834974 | 0.91082 | 0.985152 | 1.061409 | 1.135122 |
| 8 | 0.46226 | 0.521746 | 0.583362 | 0.646325 | 0.709546 | 0.774516 | 0.838172 | 0.903655 | 0.966799 |
| 9 | 0.399563 | 0.45138 | 0.504863 | 0.559618 | 0.614647 | 0.671322 | 0.726839 | 0.784086 | 0.839172 |
| 10 | 0.350534 | 0.396516 | 0.443639 | 0.49196 | 0.540564 | 0.59072 | 0.639841 | 0.690599 | 0.73935 |
| 11 | 0.311058 | 0.352653 | 0.394676 | 0.437831 | 0.481271 | 0.526176 | 0.570147 | 0.61567 | 0.65932 |
| 12 | 0.278869 | 0.31686 | 0.35471 | 0.393633 | 0.432838 | 0.47343 | 0.513171 | 0.554387 | 0.593847 |
| 13 | 0.252792 | 0.287151 | 0.321529 | 0.356925 | 0.3926 | 0.42959 | 0.465801 | 0.503413 | 0.539374 |
| 14 | 0.229877 | 0.262134 | 0.293583 | 0.325998 | 0.358688 | 0.392629 | 0.42585 | 0.460407 | 0.493406 |
| 15 | 0.211075 | 0.240806 | 0.269752 | 0.29962 | 0.329756 | 0.361083 | 0.391743 | 0.423678 | 0.454139 |

Altitude of 50 m



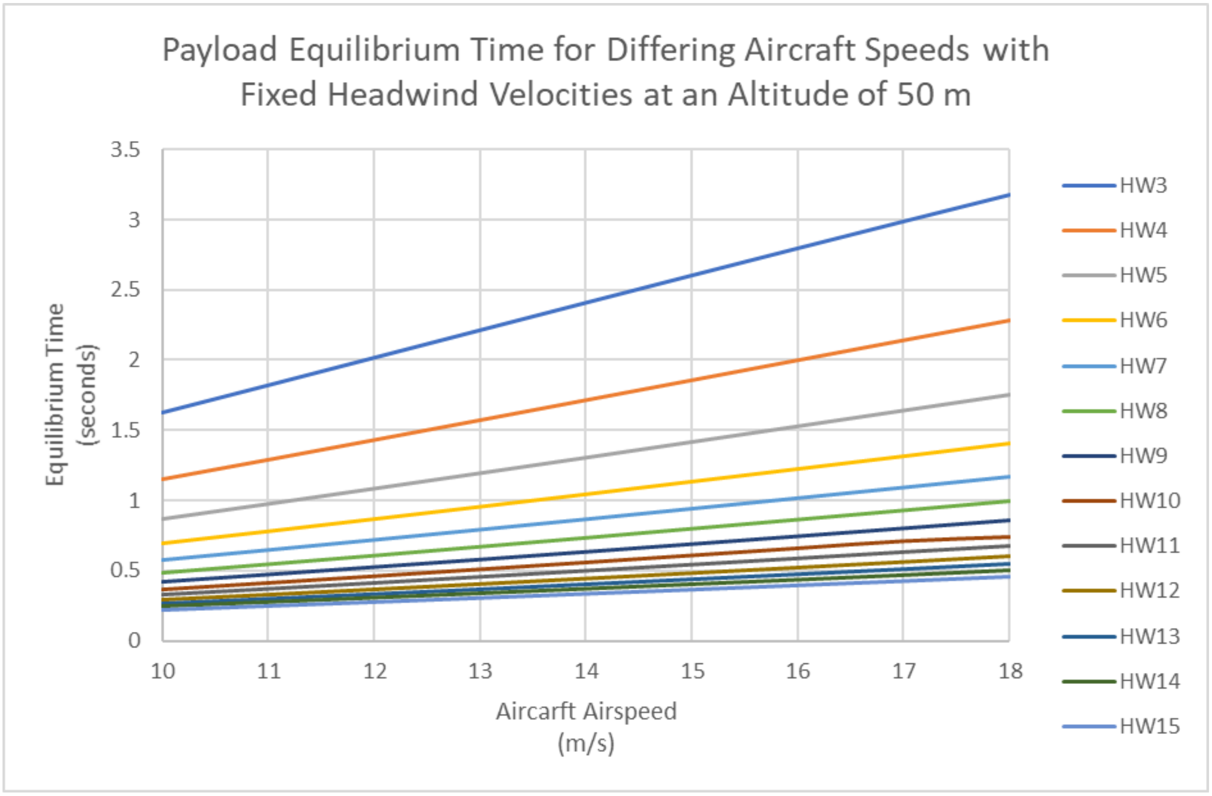
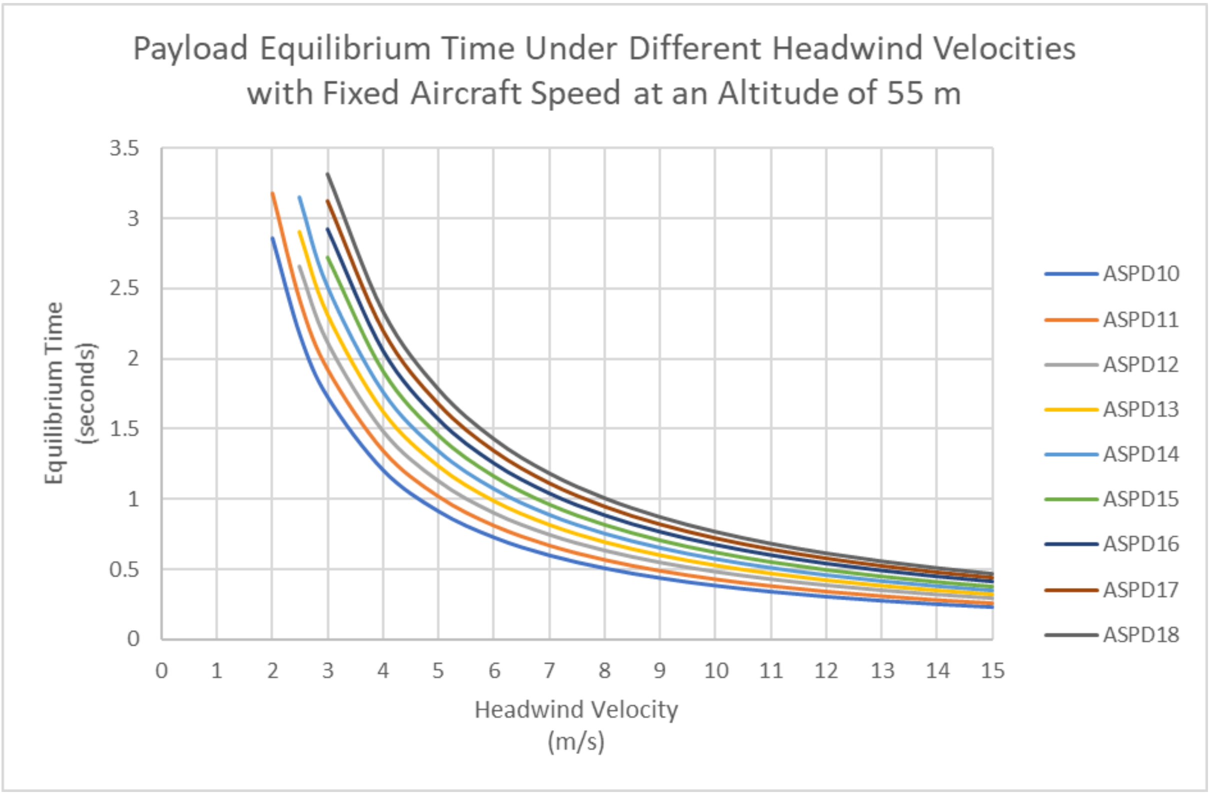


*Table K1.13: Payload Release Equilibrium Time Lookup Tables for an altitude of 50 m*

| Aircraft Speed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | - | - | - | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | - |
| 2 | 2.654 | 2.959 | - | - | - | - | - | - | - |
| 2.5 | 2.032 | 2.271 | 2.51 | 2.749 | 2.986 | - | - | - | - |
| 3 | 1.629 | 1.824 | 2.02 | 2.217 | 2.413 | 2.607 | 2.801 | 2.992 | 3.181 |
| 4 | 1.147 | 1.286 | 1.427 | 1.57 | 1.713 | 1.855 | 1.998 | 2.141 | 2.282 |
| 5 | 0.872 | 0.979 | 1.088 | 1.197 | 1.308 | 1.419 | 1.53 | 1.642 | 1.754 |
| 6 | 0.696 | 0.782 | 0.869 | 0.958 | 1.047 | 1.138 | 1.228 | 1.319 | 1.411 |
| 7 | 0.575 | 0.647 | 0.719 | 0.793 | 0.867 | 0.943 | 1.018 | 1.095 | 1.171 |
| 8 | 0.487 | 0.548 | 0.61 | 0.672 | 0.736 | 0.8 | 0.865 | 0.93 | 0.996 |
| 9 | 0.421 | 0.473 | 0.527 | 0.581 | 0.637 | 0.692 | 0.749 | 0.806 | 0.863 |
| 10 | 0.369 | 0.415 | 0.462 | 0.51 | 0.559 | 0.608 | 0.658 | 0.708 | 0.737 |
| 11 | 0.328 | 0.369 | 0.411 | 0.454 | 0.497 | 0.541 | 0.585 | 0.63 | 0.672 |
| 12 | 0.294 | 0.331 | 0.369 | 0.407 | 0.446 | 0.485 | 0.525 | 0.566 | 0.607 |
| 13 | 0.266 | 0.3 | 0.334 | 0.368 | 0.404 | 0.44 | 0.476 | 0.513 | 0.55 |
| 14 | 0.243 | 0.273 | 0.305 | 0.336 | 0.368 | 0.401 | 0.434 | 0.468 | 0.502 |
| 15 | 0.222 | 0.251 | 0.279 | 0.309 | 0.338 | 0.368 | 0.399 | 0.429 | 0.461 |

Altitude of 55 m



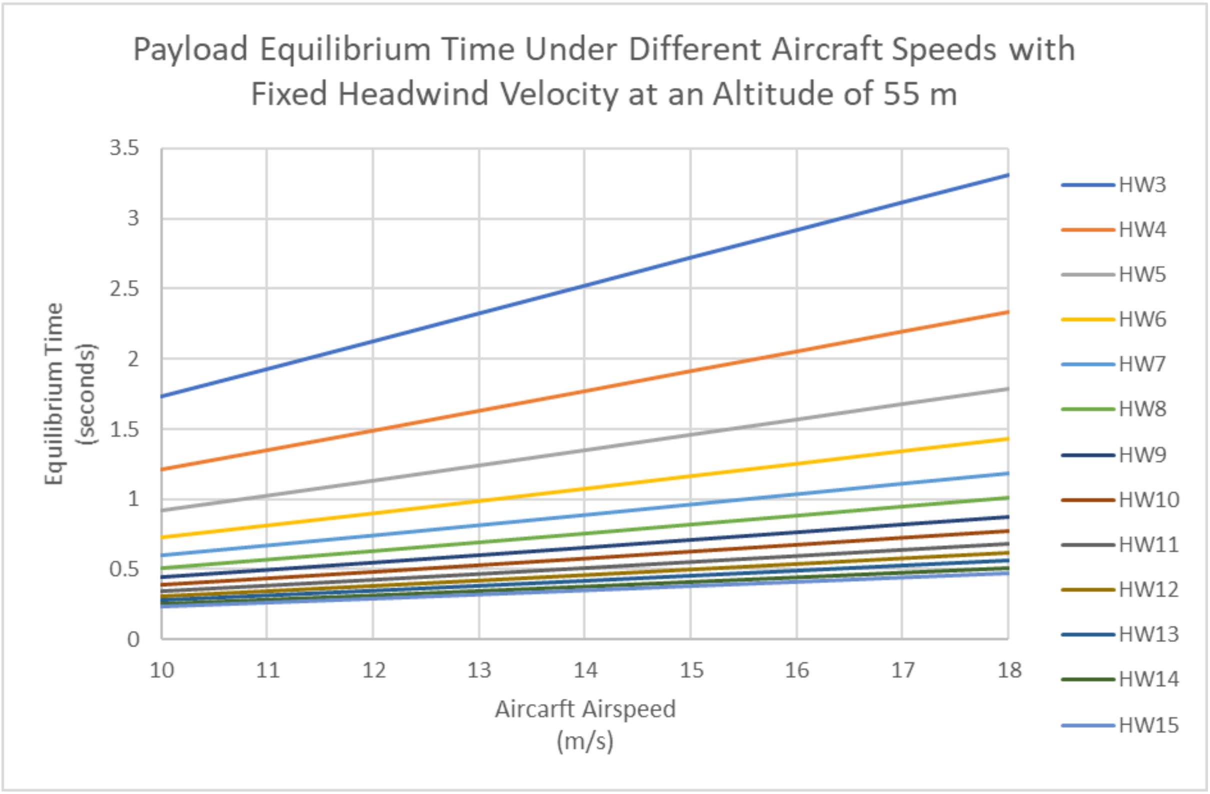
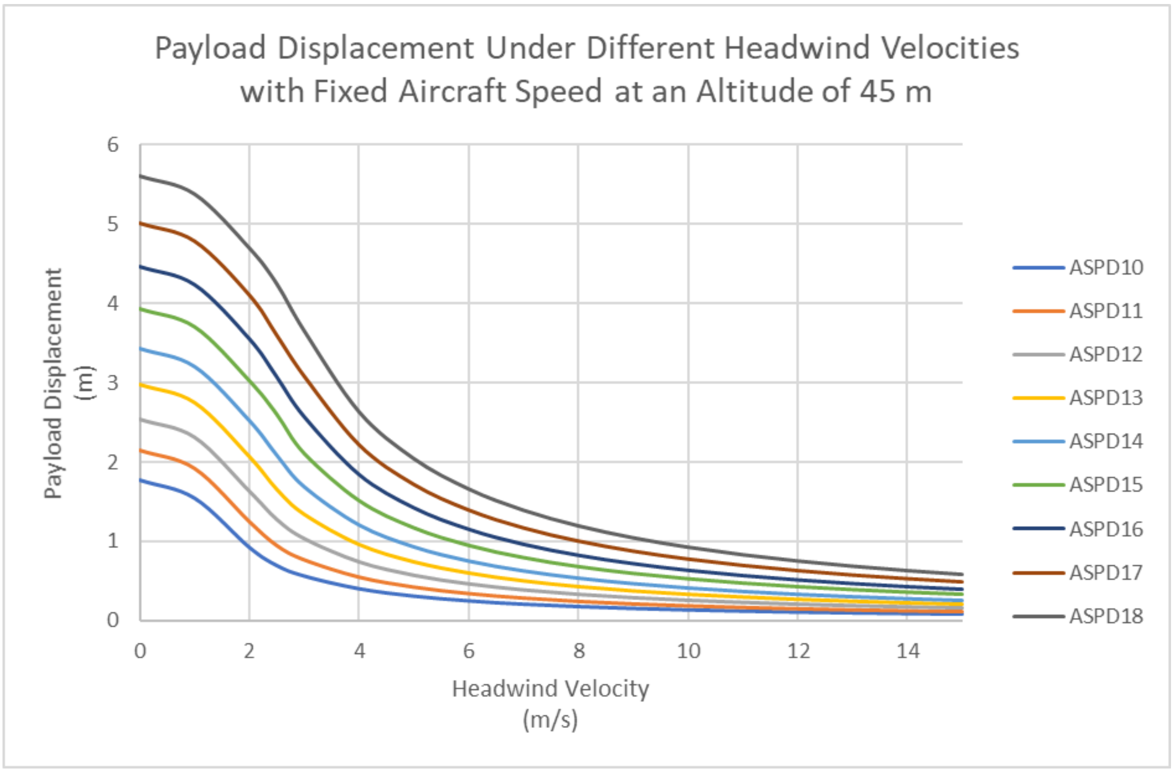


*Table K1.3: Payload Release Equilibrium Time Lookup Tables for an altitude of 55 m*

| Aircraft Airspeed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | - | - | - | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | - |
| 2 | 2.861717 | 3.181984 | - | - | - | - | - | - | - |
| 2.5 | 2.168057 | 2.412846 | 2.660413 | 2.908922 | 3.156139 | - | - | - | - |
| 3 | 1.728101 | 1.924619 | 2.123253 | 2.32328 | 2.522565 | 2.723714 | 2.921039 | 3.120058 | 3.315211 |
| 4 | 1.208218 | 1.347165 | 1.487485 | 1.629492 | 1.771302 | 1.915298 | 2.056421 | 2.199693 | 2.339297 |
| 5 | 0.915354 | 1.021533 | 1.128691 | 1.237548 | 1.34645 | 1.457534 | 1.566325 | 1.677322 | 1.784968 |
| 6 | 0.729605 | 0.814831 | 0.900799 | 0.988397 | 1.076159 | 1.166006 | 1.253952 | 1.344037 | 1.431076 |
| 7 | 0.60229 | 0.67306 | 0.744414 | 0.817309 | 0.890428 | 0.965512 | 1.038976 | 1.114476 | 1.187198 |
| 8 | 0.51011 | 0.570353 | 0.631071 | 0.693238 | 0.755661 | 0.819928 | 0.882786 | 0.947568 | 1.009804 |
| 9 | 0.440586 | 0.49285 | 0.54551 | 0.59953 | 0.653823 | 0.709847 | 0.764627 | 0.821221 | 0.875468 |
| 10 | 0.386463 | 0.432489 | 0.478851 | 0.526492 | 0.574413 | 0.623962 | 0.672396 | 0.722545 | 0.770517 |
| 11 | 0.343254 | 0.38428 | 0.425597 | 0.468118 | 0.51092 | 0.555256 | 0.598586 | 0.643536 | 0.686458 |
| 12 | 0.30804 | 0.344977 | 0.382167 | 0.420496 | 0.459104 | 0.49916 | 0.5383 | 0.578975 | 0.617752 |
| 13 | 0.278845 | 0.312382 | 0.346141 | 0.380978 | 0.416091 | 0.452576 | 0.488219 | 0.52532 | 0.560638 |
| 14 | 0.254287 | 0.284955 | 0.315821 | 0.347709 | 0.379868 | 0.41333 | 0.446015 | 0.480086 | 0.512478 |
| 15 | 0.233373 | 0.261591 | 0.289985 | 0.319353 | 0.348986 | 0.379859 | 0.41001 | 0.441483 | 0.471367 |

## Appendix J2 – Payload Release Phase A Displacement Look Up Tables

<u>Altitude of 45 m</u>



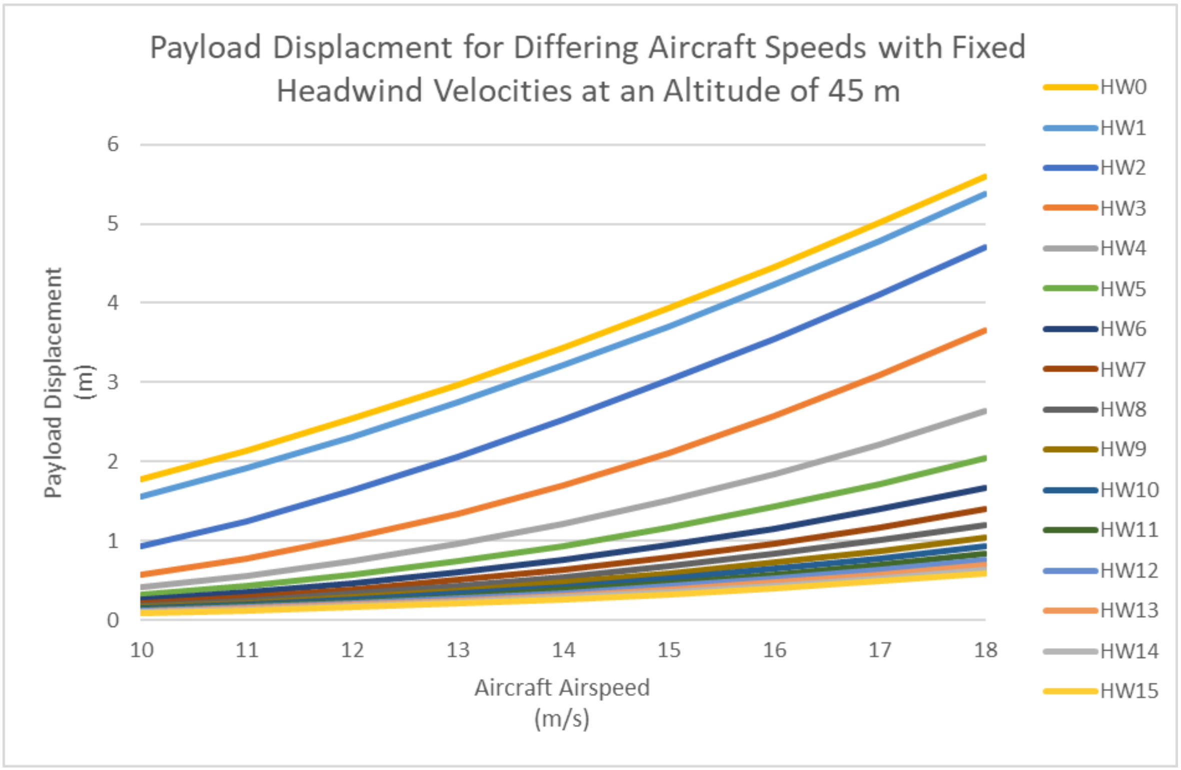
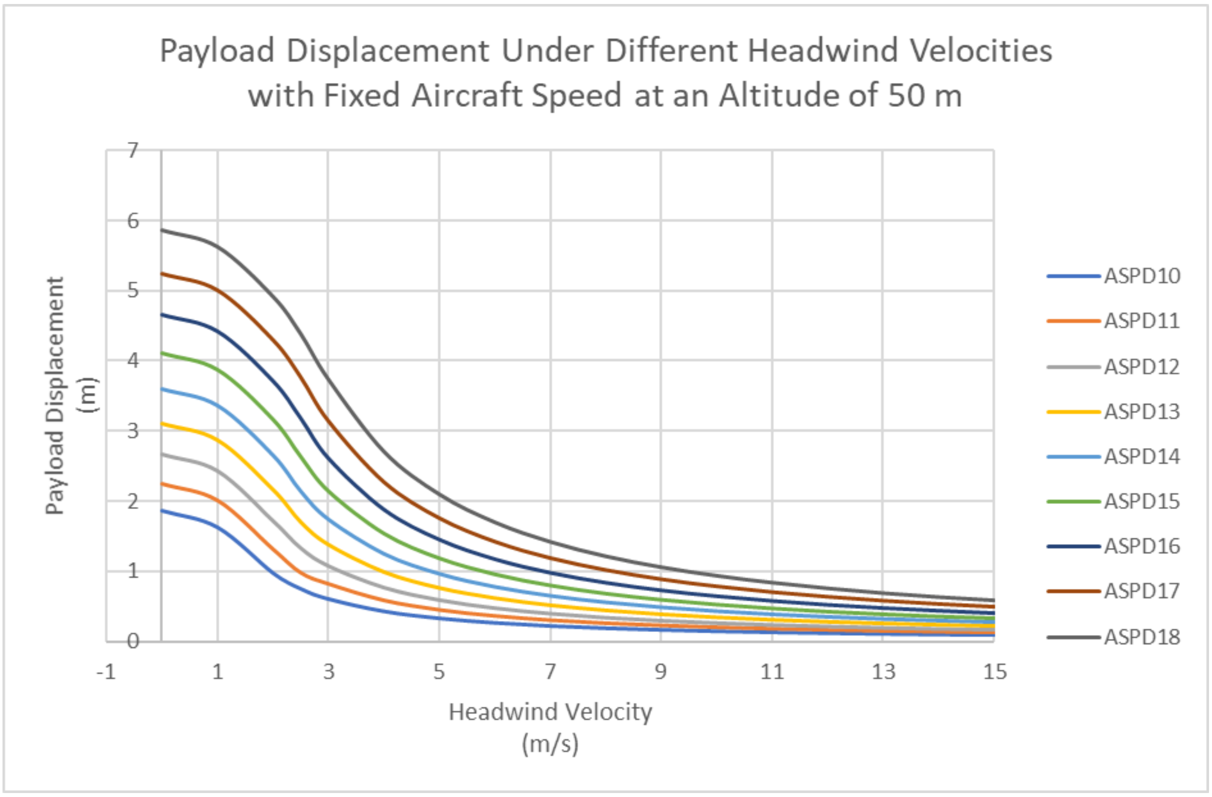


*Table K2.14: Payload Release Phase A Displacement Look Up Table at an Altitude of 45 m*

| Aircraft Speed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | 1.777896 | 2.143759 | 2.542394 | 2.973459 | 3.43662 | 3.931546 | 4.457912 | 5.015396 | 5.603681 |
| 1 | 1.551097 | 1.91696 | 2.315594 | 2.746659 | 3.20982 | 3.704747 | 4.231113 | 4.788597 | 5.376882 |
| 2 | 0.929783 | 1.250363 | 1.635195 | 2.06626 | 2.529422 | 3.024348 | 3.550714 | 4.108198 | 4.696483 |
| 2.5 | 0.702427 | 0.951627 | 1.279756 | 1.655369 | 2.094208 | 2.601247 | 3.072845 | 3.601899 | 4.247741 |
| 3 | 0.569564 | 0.772051 | 1.03637 | 1.340303 | 1.695618 | 2.10692 | 2.57177 | 3.085588 | 3.655657 |
| 4 | 0.409132 | 0.555062 | 0.742952 | 0.960559 | 1.215203 | 1.510841 | 1.846302 | 2.219643 | 2.635786 |
| 5 | 0.316531 | 0.42972 | 0.573899 | 0.741826 | 0.938484 | 1.167323 | 1.427783 | 1.719179 | 2.045142 |
| 6 | 0.25666 | 0.34863 | 0.464754 | 0.600635 | 0.759863 | 0.945491 | 1.157299 | 1.395271 | 1.662242 |
| 7 | 0.214966 | 0.29213 | 0.388835 | 0.502442 | 0.635639 | 0.791165 | 0.968998 | 1.169512 | 1.395005 |
| 8 | 0.184365 | 0.250645 | 0.333172 | 0.430458 | 0.544573 | 0.677998 | 0.830838 | 1.0037 | 1.198503 |
| 9 | 0.16101 | 0.218972 | 0.290727 | 0.375575 | 0.47514 | 0.591692 | 0.725418 | 0.87707 | 1.048283 |
| 10 | 0.142637 | 0.194046 | 0.257362 | 0.332437 | 0.420566 | 0.523842 | 0.642504 | 0.777395 | 0.929934 |
| 11 | 0.127829 | 0.173951 | 0.23049 | 0.297699 | 0.376618 | 0.469192 | 0.575694 | 0.697023 | 0.834428 |
| 12 | 0.115657 | 0.157428 | 0.208416 | 0.269164 | 0.34052 | 0.424294 | 0.520786 | 0.630927 | 0.755828 |
| 13 | 0.105487 | 0.143619 | 0.189982 | 0.245337 | 0.310376 | 0.386796 | 0.474912 | 0.575674 | 0.690079 |
| 14 | 0.096869 | 0.131915 | 0.174371 | 0.225161 | 0.284851 | 0.355039 | 0.43605 | 0.528841 | 0.634315 |
| 15 | 0.08948 | 0.121879 | 0.160993 | 0.207872 | 0.262979 | 0.327822 | 0.402734 | 0.488672 | 0.586457 |

Altitude of 50 m



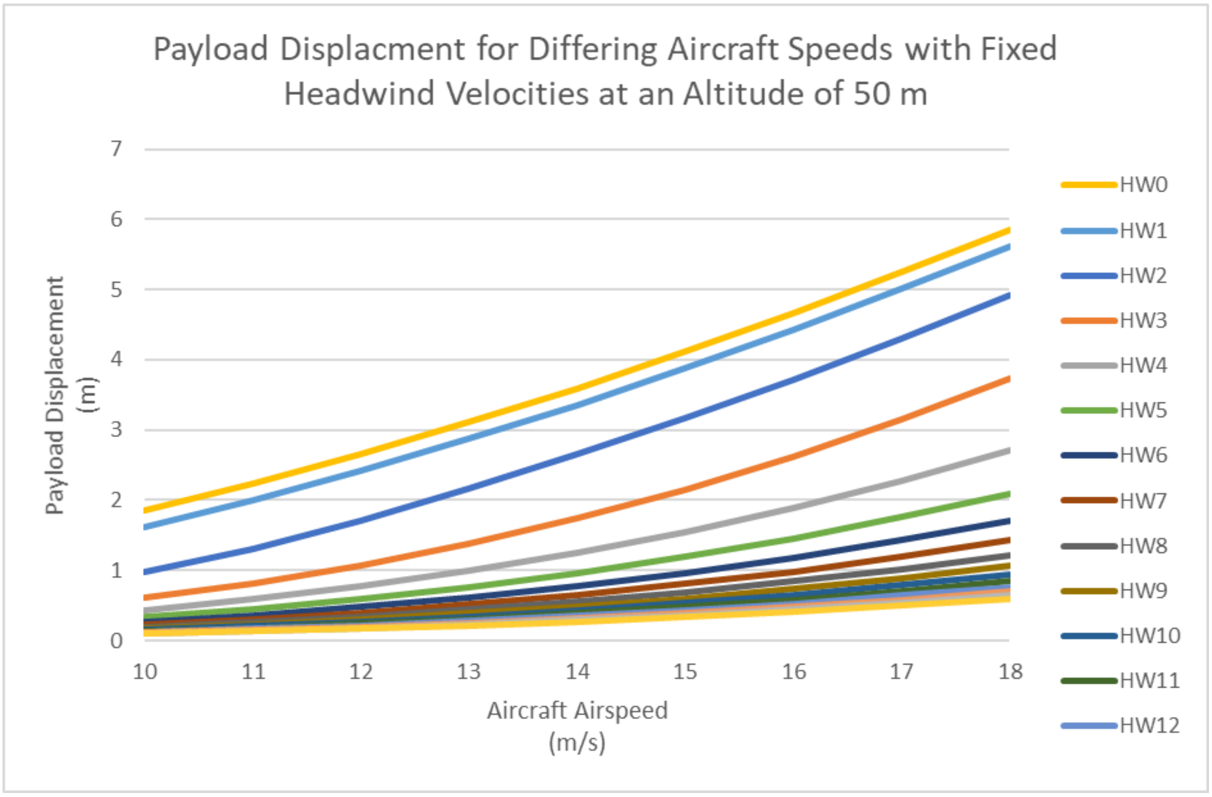
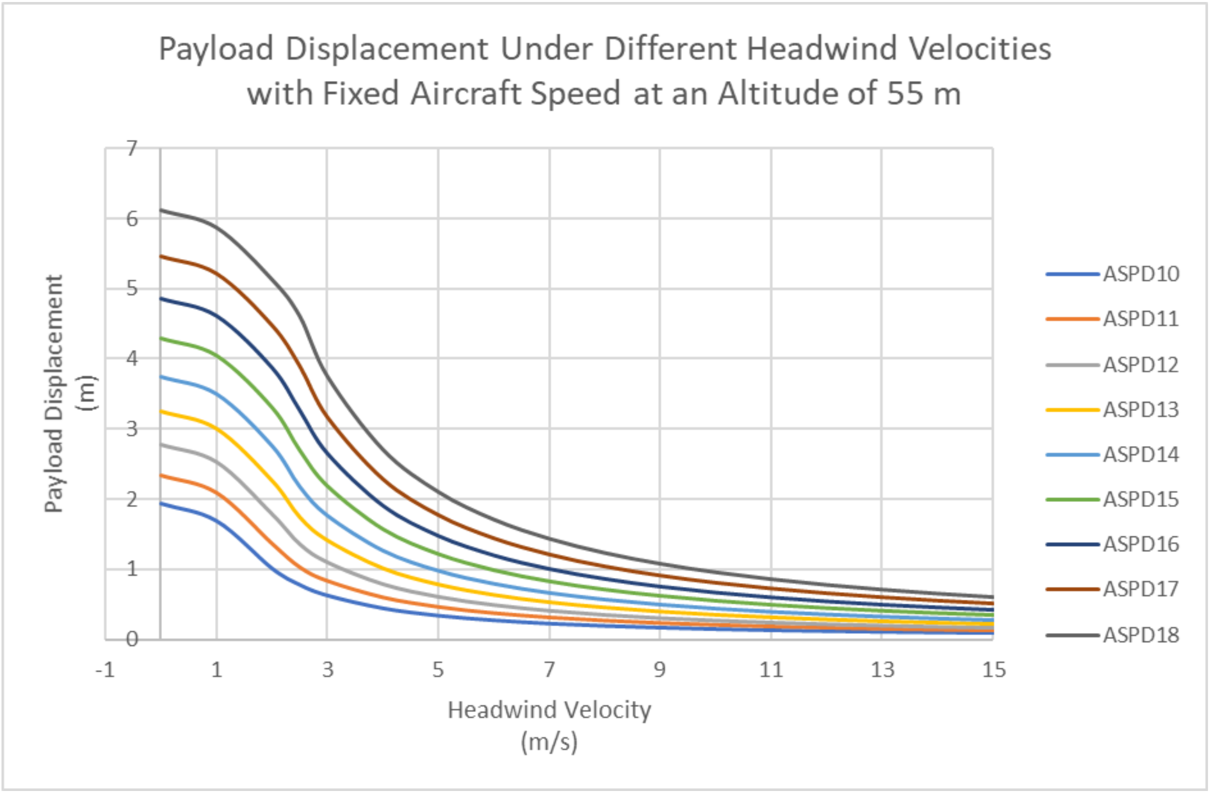


*Table K15.2: Payload Release Phase A Displacement Look Up Table at an Altitude of 50 m*

| Aircraft Speed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | 1.860723 | 2.24326 | 2.659961 | 3.110453 | 3.594373 | 4.111359 | 4.661056 | 5.243115 | 5.857191 |
| 1 | 1.624572 | 2.007109 | 2.423809 | 2.874302 | 3.358221 | 3.875207 | 4.424905 | 5.006964 | 5.621039 |
| 2 | 0.97642 | 1.312456 | 1.715355 | 2.165848 | 2.649767 | 3.166753 | 3.716451 | 4.298509 | 4.912585 |
| 2.5 | 0.752403 | 0.977832 | 1.328795 | 1.701552 | 2.136095 | 2.635413 | 3.18511 | 3.767169 | 4.381244 |
| 3 | 0.607164 | 0.819209 | 1.075544 | 1.379882 | 1.735672 | 2.146074 | 2.613938 | 3.142148 | 3.732704 |
| 4 | 0.432278 | 0.584112 | 0.768162 | 0.987422 | 1.244756 | 1.542875 | 1.884426 | 2.27204 | 2.707605 |
| 5 | 0.332647 | 0.449366 | 0.591076 | 0.760153 | 0.959463 | 1.191002 | 1.456736 | 1.758723 | 2.099291 |
| 6 | 0.268536 | 0.362978 | 0.47749 | 0.614121 | 0.775107 | 0.962902 | 1.178743 | 1.424472 | 1.701852 |
| 7 | 0.224891 | 0.303042 | 0.398968 | 0.512924 | 0.647855 | 0.804344 | 0.98519 | 1.190684 | 1.424039 |
| 8 | 0.192822 | 0.259597 | 0.341604 | 0.43956 | 0.554624 | 0.688183 | 0.843495 | 1.020098 | 1.219246 |
| 9 | 0.168842 | 0.22741 | 0.298594 | 0.38368 | 0.483948 | 0.600661 | 0.735068 | 0.88938 | 1.064168 |
| 10 | 0.150024 | 0.201756 | 0.264626 | 0.339822 | 0.428532 | 0.531933 | 0.651193 | 0.787467 | 0.94189 |
| 11 | 0.135027 | 0.18141 | 0.237532 | 0.304461 | 0.383831 | 0.476852 | 0.583925 | 0.706103 | 0.844434 |
| 12 | 0.122184 | 0.164733 | 0.215752 | 0.276163 | 0.347681 | 0.432087 | 0.528878 | 0.638993 | 0.76337 |
| 13 | 0.112534 | 0.149807 | 0.197356 | 0.253043 | 0.317641 | 0.394403 | 0.482936 | 0.583092 | 0.695735 |
| 14 | 0.103473 | 0.13938 | 0.181236 | 0.232892 | 0.293279 | 0.36196 | 0.443958 | 0.536366 | 0.639481 |
| 15 | 0.096852 | 0.128187 | 0.168984 | 0.215579 | 0.271337 | 0.336453 | 0.409439 | 0.496338 | 0.59237 |

Altitude of 55 m



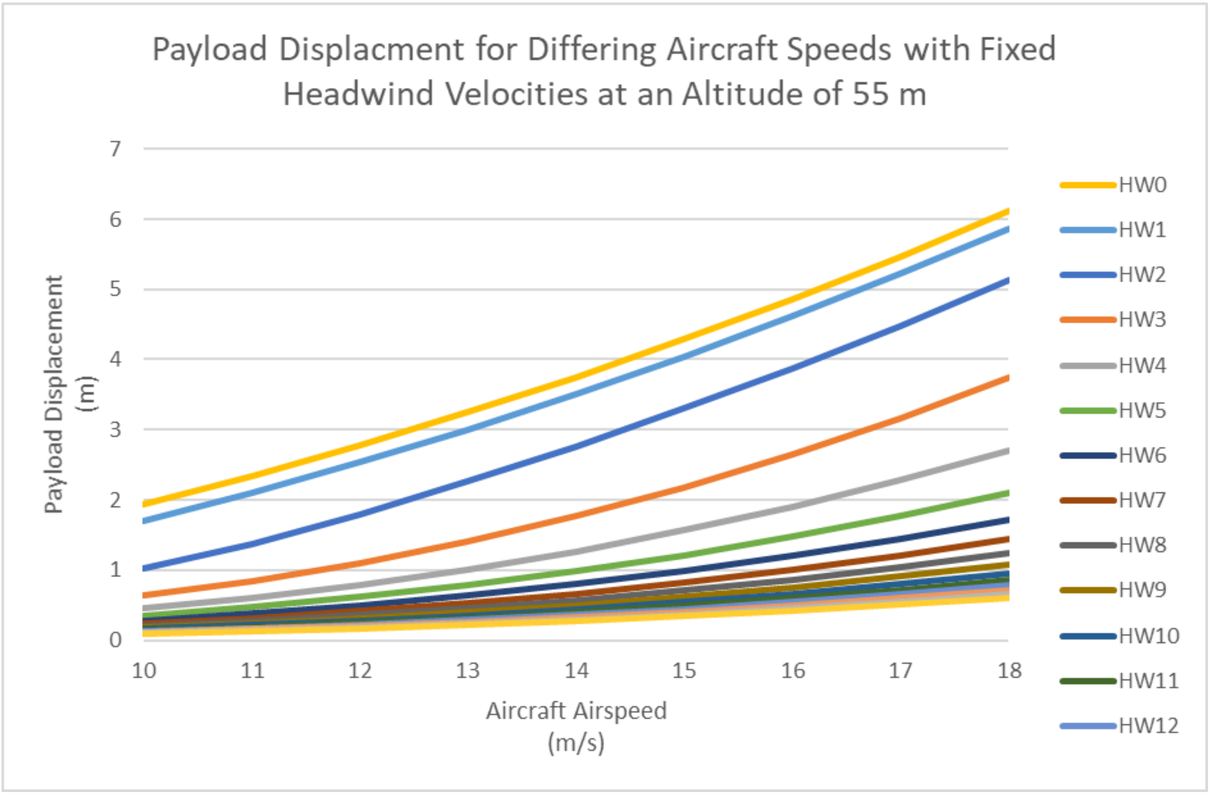
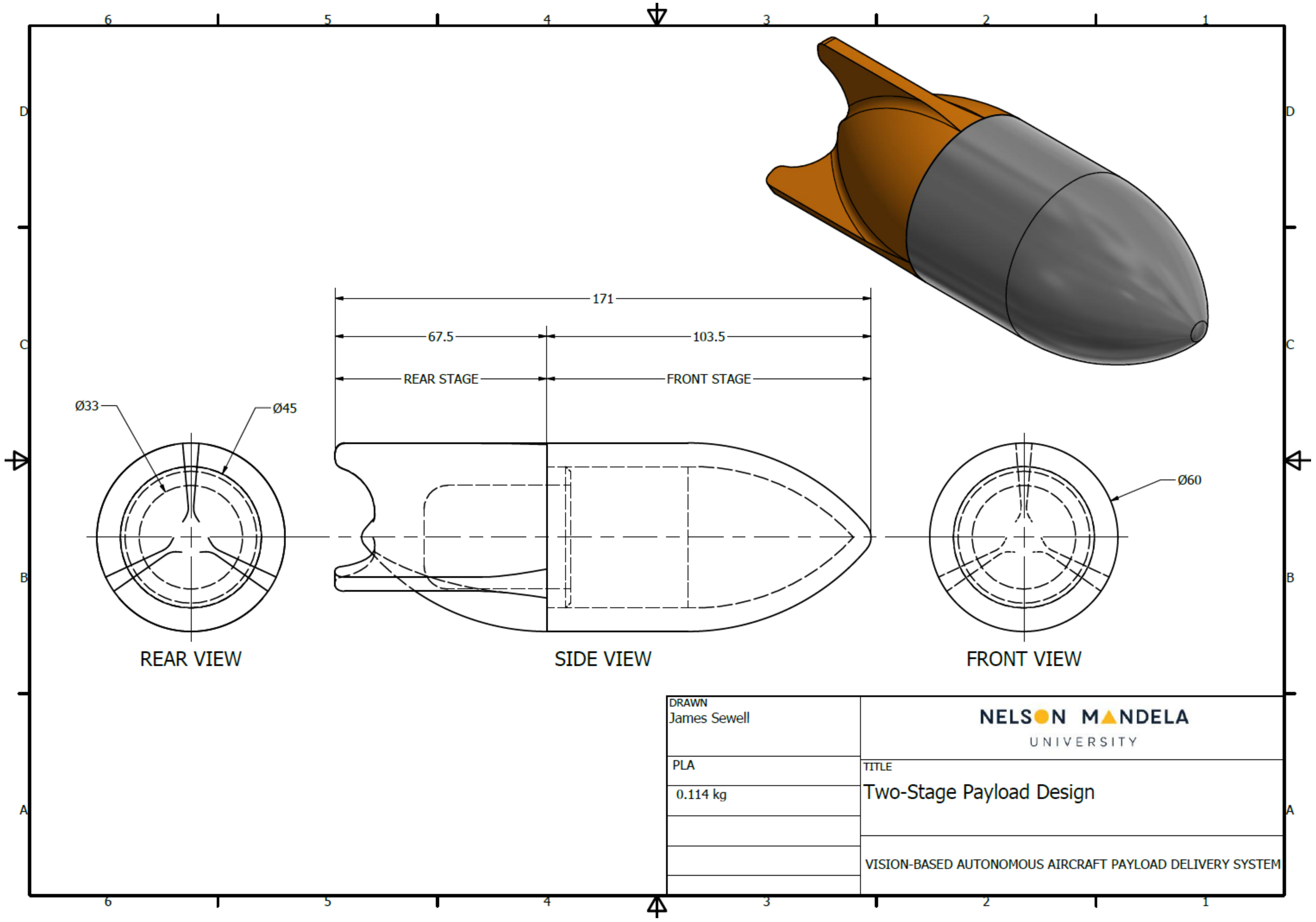


*Table K2.3: Payload Release Phase A Displacement Look Up Table at an Altitude of 55 m*

| Aircraft Speed | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| Headwind Velocity | | | | | | | | | |
| 0 | 1.943266 | 2.342387 | 2.777048 | 3.246845 | 3.75138 | 4.290264 | 4.863111 | 5.469541 | 6.109182 |
| 1 | 1.696773 | 2.095895 | 2.530555 | 3.000352 | 3.504888 | 4.043771 | 4.616618 | 5.223049 | 5.862689 |
| 2 | 1.020674 | 1.371111 | 1.791077 | 2.260874 | 2.76541 | 3.304293 | 3.87714 | 4.483571 | 5.123212 |
| 2.5 | 0.786818 | 1.033962 | 1.363197 | 1.740811 | 2.182068 | 2.692043 | 3.267073 | 3.900088 | 4.600335 |
| 3 | 0.634973 | 0.839767 | 1.105149 | 1.411025 | 1.768689 | 2.182848 | 2.651045 | 3.168742 | 3.743135 |
| 4 | 0.452147 | 0.604789 | 0.793627 | 1.01299 | 1.269761 | 1.567992 | 1.906502 | 2.283399 | 2.703521 |
| 5 | 0.347804 | 0.468845 | 0.613865 | 0.783366 | 0.981933 | 1.213102 | 1.476313 | 1.770929 | 2.100509 |
| 6 | 0.281026 | 0.380788 | 0.497663 | 0.634963 | 0.795912 | 0.983646 | 1.197945 | 1.438844 | 1.709112 |
| 7 | 0.234623 | 0.319372 | 0.416754 | 0.53165 | 0.666411 | 0.823854 | 1.003958 | 1.20715 | 1.435667 |
| 8 | 0.201382 | 0.274238 | 0.357381 | 0.455847 | 0.571394 | 0.706577 | 0.861503 | 1.036832 | 1.234425 |
| 9 | 0.175987 | 0.239753 | 0.312072 | 0.398007 | 0.498894 | 0.617069 | 0.752725 | 0.906663 | 1.080467 |
| 10 | 0.156389 | 0.212596 | 0.276432 | 0.352516 | 0.441871 | 0.546654 | 0.667111 | 0.804133 | 0.959091 |
| 11 | 0.140331 | 0.190689 | 0.247711 | 0.31586 | 0.395923 | 0.489904 | 0.598084 | 0.721409 | 0.861082 |
| 12 | 0.127607 | 0.172667 | 0.224104 | 0.285734 | 0.358161 | 0.443256 | 0.541323 | 0.653342 | 0.780379 |
| 13 | 0.117323 | 0.157596 | 0.20438 | 0.260565 | 0.326612 | 0.404276 | 0.493877 | 0.596412 | 0.712836 |
| 14 | 0.107667 | 0.144818 | 0.18767 | 0.239243 | 0.299886 | 0.371249 | 0.453665 | 0.548136 | 0.655524 |
| 15 | 0.100623 | 0.133855 | 0.173343 | 0.220964 | 0.276973 | 0.342932 | 0.419177 | 0.506711 | 0.606318 |

## Appendix K – Payload CAD Design



| | |
|---|---|
| DRAWN James Sewell | NELSON MANDELA UNIVERSITY |
| PLA | TITLE Two-Stage Payload Design |
| 0.114 kg | |
| | VISION-BASED AUTONOMOUS AIRCRAFT PAYLOAD DELIVERY SYSTEM |