Localization and Control of a Quadcopter Universal Payload System

by

Mark James Sherstan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

Abstract

There is a growing trend of using unmanned aerial vehicles (also known as UAVs, uncrewed aerial vehicles, or drones) to manipulate and interact with their surroundings. The algorithms and tools used are typically unique to the different tasks performed by UAVs; however, the fundamental UAV system largely remains consistent. This thesis offers a general quadrotor UAV system capable of performing multiple tasks dependent on available payloads. The system is empirically developed and performs localization using ArUco fiducial markers, an inertial measurement unit, and a Kalman filter. Navigation is achieved using a proportional integral derivative controller, and interfacing with various payloads is accomplished with a custom-developed universal payload manipulator. The system is described in detail and demonstrated with real-world experimentation, including the deployment of an example payload for removing twist caps off sump lubricant reservoirs. Empirical results show the developed system as an initial functional prototype in ideal conditions, and further work is required to increase system reliability and functionality for non-ideal scenarios.

Acknowledgements

I would like to express my appreciation to all the people who helped and made it possible for me to perform this work. Additionally, I gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC) for its financial support.

I would like to express a special thank-you to my supervisor Dr. Michael Lipsett. Dr. Lipsett provided unparalleled technical insight and valuable wisdom that went beyond this research. I appreciate the time Dr. Lipsett made for me and the career path on which he has guided me.

The team at Pegasus Imagery deserves special thanks for the unique industry experience they provided and for supporting me as I finished this academic journey.

I would like to thank Eric Wells for all of the side projects we completed and for providing insight into problems when I was stuck. I would also like to thank my lab mates Rijesh Augustine and Nicolas Olmedo for their technical support and assistance with experiments.

I also want to express my appreciation to Dave Klein for providing a fresh set of eyes in editing this thesis. Additionally, I would like to thank Dr. Martin Barczyk, who provided valuable recommendations into the early experimental system.

To my amazing parents and sister, I would like to express my gratitude and appreciation for supporting me from the very beginning. I especially want to thank my family for providing support and encouragement during the countless hours of using their garage as a testing facility, even if there are still permanent remnants of failed experiments.

I also thank my in-laws for their continuous support and my two grandfathers, who sparked my interest in taking things apart and showing me what true hard work is.

I cannot express enough thanks to my incredible wife Cynthia, who has made uncountable sacrifices for me and is my best friend. I love you.

Lastly, I would like to thank God, who has given me everything.

Contents

Acknowledgements **iii**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Background 1
1.2 Motivation 4
1.3 Literature Review 5
1.3.1 Localization 6
1.3.2 Control 8
1.3.3 Payloads 10
1.3.4 Aerial Manipulation 11
1.4 Research Objectives 13
1.5 Thesis Outline 13

2 Localization and Control **15**

2.1 Localization 15
2.1.1 ArUco Library 15
2.1.2 Camera Models and Calibration 17
2.1.3 Image Processing and Pose Extraction 20
2.1.4 Transformation of Coordinate Frames 24
2.1.5 Kalman Filter 27
2.2 Control 29
2.2.1 Proportional Integral Derivative (PID) Controller 29
2.2.2 Trajectory Generation 30

3 System Overview **32**
3.1 UAV System Overview . . . 32
3.1.1 General Configuration . . . 32
3.1.2 Autopilot Selection and Integration . . . 35
3.1.3 Single Board Computer Selection . . . 36
3.1.4 Camera Selection . . . 40
3.1.5 Software Development . . . 42
3.2 Universal Payload Adapter: The Quick Connect . . . 45
3.2.1 Mechanical Design . . . 46
3.2.2 Electrical Design . . . 50
3.2.3 Software Design . . . 53
3.2.4 Payload Docking Station . . . 55

4 Payload Development Example **57**
4.1 Conceptual Design . . . 57
4.1.1 Design Space . . . 57
4.1.2 Cap Manipulation . . . 58
4.1.3 Torque Generation . . . 60
4.2 Final Design . . . 64
4.2.1 Mechanical Design . . . 64
4.2.2 Electrical Design . . . 66
4.2.3 Software Design . . . 71
4.2.4 Control Design . . . 74

5 Experimental Trials and System Performance **80**
5.1 Introductory Experimental Testing . . . 80
5.1.1 Software in the Loop (SITL) Simulation . . . 80
5.1.2 Online Testing with Safety Tether . . . 82
5.2 Computational Performance . . . 83
5.3 Localization . . . 88
5.3.1 ArUco Pose Validation . . . 88
5.3.2 Kalman Filter Validation . . . 91
5.4 PID Tuning . . . 95
5.5 Visual Servoing Landing . . . 99
5.6 Aerial Manipulation . . . 101
5.6.1 Payload Swapping . . . 101
5.6.2 Cap Removal . . . 102

6 Conclusions and Future Work **105**
6.1 Conclusions 105
6.2 Future Work 106
6.2.1 Localization 106
6.2.2 Control 107
6.2.3 General System and Universal Payload Design 108

References **109**

List of Tables

3.1 UAV Manufacturer Components and Part Numbers 34
3.2 Used Python Packages (*Non Standard Package, **Compiled from Source Code) . . 43
3.3 Quick Connect Major PCB Components 51

4.1 Cap Manipulator Major PCB Components and Peripherals 68
4.2 Experimental Yaw Torque Summary Table 79

5.1 Raw ArUco Pose Difference Statistics Between the Two Intel RealSense T265 Cameras 91
5.2 PID Gain, Bounding, and Filtering Summary Table 98

List of Figures

1.1 General Atomics MQ-1 Predator UAV firing a Hellfire Missile 2
1.2 Quadrotor UAV Autonomous Valve Interfacing Task 4
1.3 Distribution of Various Control Strategies For: (A) Quadrotor, (B) Multirotor 9
1.4 Four AEROARMS platforms: (A) multidirectional-thrust hexarotor with a rigidly attached end effector, (B) multidirectional-thrust octorotor with inspection arm, (C) aerial dual-arm manipulator with stiff joints, and (D) aerial dual-arm manipulator with compliant joints 12

2.1 (A) ArUco Marker ID 0 and (B) Occluded ArUco Board, both with Axis Projections and ID Labelling 16
2.2 (A) Raw Fisheye Image, (B) Pinhole Equivalent Image 18
2.3 Pinhole Camera Model 19
2.4 Example Calibration Images with Labelled IDs 20
2.5 ArUco Marker ID 0 with Highlighted Bits 21
2.6 Detection and Identification Pipeline of ArUco Markers: (A) Original Image, (B) Thresholded Image, (C) Contour Extraction, (D) Filtered Contours, (E) Single Marker Extraction, and (F) Single Marker Binarization 22
2.7 ArUco, Camera, and Body Coordinate Systems 24
2.8 UAV North, East, Down (NED) Coordinate System 26
2.9 ArUco Yaw Calculation Based On Orientation: (A) Downwards Facing Camera vs (B) Forward Facing Camera 27

3.1 UAV Top View with Labelled Components 33
3.2 UAV Bottom View with Labelled Components 33
3.3 UAV Electrical and Signal Diagram 35
3.4 Single Board Computers Considered: (A) Raspberry Pi 4B, (B) Odroid C2, (C) NVIDIA Jetson Nano, (D) Raspberry Pi 3B+, (E) Odroid C4, (F) Odroid XU4 . . . 38
3.5 Capture Rate per Second vs Camera Resolution 39

3.6 Pose Calculation per Second vs Camera Resolution 39
3.7 Rolling vs Global Shutter Comparison Example 41
3.8 T265 with Labeled Coordinate Frames 41
3.9 (A) Multithreading vs (B) Multiprocessing 44
3.10 Processor and Thread Allocation 45
3.11 Quick Connect Sub Assemblies Exploded: (A) Actuator, (B) Male Quick Connect, and (C) Female Quick Connect 47
3.12 Quick Connect States: (A) Disengaged State, (B) Engaged State 48
3.13 Quick Connect Free Body Diagram 49
3.14 Quick Connect Electrical and Signal Diagram 50
3.15 Quick Connect PCB: (Left) Assembled Board, (Right) Board Layout with Hidden Fill Zones 51
3.16 Quick Connect Microcontroller Program Flow Chart 53
3.17 Quick Connect SBC Function Flow Charts 54
3.18 Quick Connect Payload Communication Flow Chart 55
3.19 UAV Payload Docking Station with Approaching UAV 56

4.1 Prototyped Cap Manipulators: (A) Adaptive Gripper, (B) Static Gripper, (C) Rack and Pinion Gripper, and (D) Lead Screw Gripper 58
4.2 Free Body Diagram of a Quadcopter While Hovering 60
4.3 Quadcopter Executing Positive Roll ϕ for a Counterclockwise Torque 62
4.4 Over-Actuated Quadcopter Concept 63
4.5 Cap Manipulator Top View with Labelled Components 65
4.6 Cap Manipulator Bottom View with Labelled Components 65
4.7 Cap Manipulator Electrical and Signal Diagram 67
4.8 Cap Manipulator PCB Board: (Left) Assembled Board with Wiring, (Right) Board Layout with Hidden Fill Zones 68
4.9 Raw and Filtered Current Readings with Dynamic Jaw Moving From Open to Close 71
4.10 Main Cap Manipulator Code Flow Chart 72
4.11 Open Case Switch Flow Chart (Continued From Figure 4.10) 73
4.12 Close Case Switch Flow Chart (Continued From Figure 4.10) 73
4.13 Clamping Sequence Current and Force Sensor Readings 74
4.14 Clockwise Ramp Input: Torque as a Function of Time 76
4.15 Counterclockwise Ramp Input: Torque as a Function of Time (Torque Converted to Positive) 76
4.16 Clockwise Step Input: Torque as a Function of Time 78

4.17 Counterclockwise Step Input: Torque as a Function of Time (Torque Converted to Positive) 78

5.1 Gazebo Simulation Environment 81
5.2 North East Down Position Controller Simulation Response 81
5.3 Yaw Position Controller Simulation Response 82
5.4 UAV Tethering System: (A) System Diagram, (B) Tethered UAV During Flight . . . 83
5.5 I/O and Logging Call Graph with Highlighted Regions of Interest 84
5.6 I/O and Logging Detail Figure 5.5-A Call Graph 85
5.7 I/O and Logging Detail Figure 5.5-B Call Graph 85
5.8 Vision Process Main Thread Call Graph 86
5.9 Vision Process Spawned Threads Call Graph 87
5.10 (A) Frequency Rate and (B) Actual Sleep Time Versus Requested Sleep Time 88
5.11 UAV Vision with Adaptive Threshold Responses at an Altitude of 75 cm 89
5.12 UAV Vision with Adaptive Threshold Responses at an Altitude of 11 cm (landing) . 89
5.13 Raw ArUco Pose Scatter Plot Comparing the Two Intel RealSense T265 Cameras . . 91
5.14 Yaw Kalman Filter Validation Using Autopilot as Ground Truth with Highlighted Regions of Interest 93
5.15 North Kalman Filter Validation Using Autopilot Controller Input and Pitch Output for Comparison 95
5.16 Three North 30 cm Step Responses During Single Flight 97
5.17 UAV Trajectory Tracking and Successful Landing Plot 99
5.18 Landing Locations Scatter Plot and Failed Landing for Scale 100
5.19 Payload Retrieval Flight Sequence: (A) Payload to be retrieved, (B) UAV approaching payload docking station, (C) Engaging Quick Connect and verifying connection, (D) Takeoff and begin mission 101
5.20 Cap Removal: (A) Small Diameter 3.9 cm, (B) Medium Diameter 5.4 cm, (C) Large Diameter 6.6 cm, and (D) Extra Large Diameter 9.1 cm 103

Chapter 1

Introduction

1.1 Background

Unmanned aerial vehicles are aircraft without a human pilot onboard. The first recorded use of UAVs dates back to 1849, when Austrian soldiers rigged approximately 200 balloons with fuses and explosives to bomb Venice. Due to changing winds, the operation had only limited success [1, 2]; however, the start of UAV technology had begun.

With the first powered flight of the Wright brothers in 1903 [3] and the start of World War I (WWI) in 1914, UAVs experienced significant technological advancements. The first major development was an automatic control system developed by Elmer Sperry and Peter Hewitt in 1916, and in 1917 the United States (US) Navy provided funding to use the technology for a flying bomb [4]. Although the program was cancelled one year later, the project paved the way for the Kettering Bug. The Kettering Bug worked by calculating the number of required engine revolutions to reach a target. At the target, the UAV would dive and deliver 82 kg of explosives [5]. However, even with an automatic stabilization system, testing proved unreliable (it had only 22% accuracy) [5], and the aircraft was not used for fear of carrying explosives over Allied troops [6]. Both the United Kingdom and Germany experimented with similar systems at the time [4].

After WWI, the aviation industry began developing rapidly. The first non-stop transatlantic flight was achieved in 1919 by Alcock and Brown [7], and in 1929 Doolittle achieved the first instruments-only flight [8]. UAVs also made advancements, such as the first remote-controlled (RC) aircraft achieving all phases of flight (taking off, maneuvering, and landing) in 1924 [9]. In 1935, the Royal Navy successfully used RC technology to develop the Queen Bee UAV for anti-aircraft gunnery target training, leading to the production of nearly 400 of the unmanned aircraft [10].

During World War II, UAV technology continued to develop in concert with other aerospace technologies such as jet engines and more complex auto-stabilization systems. One such example was Germany's V-1 "buzz bomb", which was capable of speeds up to 640 km/h and a range of up to 250 km [4]. Although still a UAV, this class of vehicle was recategorized as a cruise missile. The main difference between UAVs and missiles is that a missile is a one-way weapons system, while a UAV is intended to be reusable (or at least recoverable).

The modern era of UAVs began in 1960, when a U-2 photographic reconnaissance aircraft flown by US pilot Gary Powers was shot down over the Soviet Union. In the wake of the diplomatic incident created by the capture of the pilot, the US created a classified unmanned reconnaissance program named "Red Wagon", which involved retrofitting a target drone for remote controlled photographic surveillance missions [11]. Seven years later, Israeli intelligence successfully acquired photos using a reconnaissance UAV during the 1967 Six-Day War; this provided a strategic advantage with no risk to their pilots [12]. In 1973, the US officially confirmed its use of UAVs in Vietnam [11]. Between 1964 and 1972, the US flew an estimated 3435 UAV missions; notably, 554 UAVs were lost [13].

As world conflicts continued in the twentieth century, so did the development of UAV systems. Israel created the first real-time surveillance UAVs and decoy drones in the 1973 Yom Kippur War to reduce the number of manned aircraft shot down by surface-to-air missiles [15]. In 1994, the US debuted a new endurance aerial reconnaissance UAV, the MQ-1 Predator. In 2001, the Predator was modified so that it could carry two AGM-114 Hellfire surface-to-air missiles, thereby becoming the world's first "stalk and kill" UAV [14]. The Predator, pictured in Figure 1.1, played a significant role in Operation Enduring Freedom and paved the way for today's cutting-edge UAVs. Because most UAV technological developments have been based on military applications, many of these

Figure 1.1: General Atomics MQ-1 Predator UAV firing a Hellfire Missile [14]

technologies are shrouded in secrecy and remain classified. However, modern UAV technologies, when they have been shared or otherwise become publicly known, are still evidently pushing the boundaries of modern technology. For example, the Defense Advanced Research Projects Agency (DARPA) AlphaDogfight Trials have demonstrated the capabilities of implementing artificial intelligence (AI) control algorithms into an F-16 fighter jet, effectively turning a previously piloted aircraft into an imposing UAV for aerial dogfights [16].

Multirotors are one of the most widely used UAV platforms by consumers and researchers due to unique qualities such as their vertical takeoff and landing (VTOL) capability, cost efficiency, and ability to hover [17]. One of the most common types of multirotors is a quadcopter, whose development can be traced back to the Bréguet-Richet Gyroplane in 1907. Gyroplane No. 1 was human-piloted and was able to lift approximately 0.5 metres off the ground; however, it was highly unstable, and required support on all four corners from a ground crew [18]. In 1922, the de Bothezat helicopter, also known as the Jerome-de Bothezat Flying Octopus, made its first flight under contract to the US Army Air Service; however, the program was cancelled in 1924 due to its complexity [19]. In 1924, Oehmichen set a world record in France for his quadcopter design flying 360 metres in a straight line [20]. In 1939, the world's first practical helicopter (designed by Sikorsky) took flight, using a primary rotor for lift and an anti-torque rotor on the tail [21]; however, the quadcopter trend continued into 1956, when the Convertawings Model A Quadrotor prototype was developed; this prototype demonstrated the ability to control roll, pitch, and yaw by varying the propellers' speed [22].

Other quadcopter projects were developed by various militaries in the latter half of the twentieth century; however, it was not until 1991 that the first consumer quadcopter UAV became readily available to the public with the release of the Keyence Gyrosaucer II E-570 in Japan [23]. This breakthrough turned quadcopters into UAVs for the first time, using previously developed military technologies such as auto-stabilization and RC control. Eight years later, Draganfly released a consumer quadrotor UAV, followed by the first multirotor UAV featuring an integrated camera system in 2001 [24]. The quadcopter became increasingly popular with researchers, in part due to increasing integration with other complementary technologies; at the 2010 Consumer Electronics Show (CES) in Las Vegas, Parrot unveiled its AR.Drone, which allowed users to control the quadcopter UAV from their smartphones [25]. Three years later, DJI released the Phantom quadcopter UAV, which exploded in popularity. As of October 2019, 77% of all registered UAVs in the US were DJI products; the next closest competitor was Intel, with 4% [26]. As of September 2020, approximately 1.2 million UAVs are registered for recreational use in the US, and another half-million are registered for commercial purposes [27].

With unmanned multirotors making their first appearance in the research world only 20 years ago, the field is still fairly young [24]. UAV multirotor research was minimal during the initial years, and

it was not until 2011 that rapid interest started to take hold [28]. Multirotor UAVs are now often used for visual inspections of hard-to-reach or dangerous areas such as railway infrastructure [29], bridges [30], and high voltage power lines [31]. Visual inspections, surveillance, and other passive tasks are now just a subset of what UAVs can do and multirotor UAVs have been developed to perform active tasks such as manipulating their surrounding environment [32].

Using a UAV to physically interact with an environment is termed "aerial manipulation". Among nearly 8500 published papers relating to UAVs between 2008 and September 2017, only 1.7% of them related to aerial manipulation [28]. However, using the same search terms and database as [28], interest in aerial manipulation was found to have more than doubled (to 4.6%) as of November 2020, and research relating to UAVs is increasing exponentially. Researchers have demonstrated multirotor UAV capabilities such as the ability to open drawers [33], open doors [34], unscrew light bulbs [35], interface and rotate valves as in Figure 1.2 [36], acquire water samples [37], have two UAVs work together to overcome payload restrictions [38], and acquire volcanic rock samples [39], which shows that aerial robotics is indeed in its golden period [32], or perhaps just entering it.

Figure 1.2: Quadrotor UAV Autonomous Valve Interfacing Task [36]

1.2 Motivation

There is a growing trend to use UAVs for inspection of industrial facilities and infrastructure. These types of inspections commonly entail remote sensing using cameras for qualitative assessment [40], photogrammetry [41], and temperature monitoring using thermal imaging cameras [42]. The benefits of using drones are to reduce risk to inspection personnel near operating equipment, reduce inspection costs, and improve auditability of archived data [43].

With aging infrastructure, there is an increased need for condition information. Condition information provides valuable insight into the current state of an asset and allows for a corresponding maintenance plan to be implemented [44]. Despite increasing use of UAVs and ground robots to gather data and samples [45], infrastructure condition information is still commonly gathered manually [46]. Access to equipment to obtain samples can be challenging, with equipment located in

confined spaces, at remote locations, and at elevated heights. The use of UAVs to acquire samples from machinery and infrastructure not only reduces risk to workers, but also allows for condition-based maintenance and frees inspectors to conduct further value-added activities [43].

The main diagnostic approaches for detecting impending machinery failure are elevated temperature trends, vibration spectral features, and anomalous components detected in lubricant analysis [47]. To gather the necessary samples for detecting machinery failure, multiple different sensors and sampling mechanisms are required. Our research group's previous work demonstrated remote sensing, monitoring, and data collection using a fixed-wing UAV, but the UAV lacked physical manipulation of the environment [48]. A multirotor system for the deployment of sensors for condition monitoring was later conceptualized [49], and the requirements for a UAV to collect vibration data by deploying a vibration sensor was further expanded on [50]. The main diagnostic approaches for detecting impending machinery failure and corresponding payloads were addressed with primarily results in [51]. Various other sensors and samplers have been conceptualized and prototyped [52, 53]; however, creating a reliable platform to deploy the sensors and payloads has yet to be developed.

Due to the requirement of acquiring multiple different samples for machinery diagnostics, it is challenging to have a single payload accurately and reliably retrieve the various samples. Suppose a single UAV could swap between payloads and their corresponding sampling procedure; in that case, a greater focus can be put into payload development and validation, as the foundation for deployment is already in place. Similarly, with the previous examples of aerial manipulation presented [33, 34, 35, 36, 37, 38, 39], each system is capable of a unique specific task. Although the tasks are vastly different, all systems can be fundamentally broken into mechanical design, localization, and control. Using the same UAV to unscrew a light bulb or retrieve a volcanic rock sample by varying the payload and its dedicated firmware turns a niche system into an extremely versatile one. To develop a versatile aerial payload platform, the system must first localize itself in a given environment. Once localized, the UAV should be able to maneuver using some control method. Finally, the UAV will need to interact with payloads and execute the corresponding task the payload was developed for. The following literature review provides relevant details for the system's localization, control, and payload development.

1.3 Literature Review

The following literature review is a general overview of the systems and methods used for localization, control, and payload manipulation. Each following chapter includes additional supporting literature directly related to the specific methods and technologies used in the research.

1.3.1 Localization

The first challenge in creating a system capable of interfacing with various payloads is localizing the UAV in a given environment. Once the UAV is localized, it can navigate to retrieve, manipulate, and return a payload. Localizing in an outdoor environment is commonly achieved using a Global Navigation Satellite System (GNSS) [54]. GNSS is a collective term for different types of satellite navigation systems. One of the most common GNSS is the Navstar Global Positioning System, commonly termed GPS, which is owned by the United States government. GPS services are a space-based radio navigation system which provides positioning, navigation, and timing services [55]. Localization with a GPS requires at least four satellites to be electronically visible from the receiver. A GPS satellite transmits a radio signal with the current time and position; using the known speed of radio waves and the delay time between the signal transmission and retrieval, the distance between the transmitter and receiver can be calculated. Using multiple satellites and localization algorithms, the precise location of the receiver can be found [56].

GPS is limited by the system's accuracy and the line of sight between satellites and the receiver. A well-designed GPS receiver is reported to have a horizontal accuracy of three metres or better 95% of the time [55]. Optimized GPS receivers have reported increased performance, with systems accurate up to 30 centimetres [57]. Such accuracies are sufficient for tasks that do not require high precision (such as road vehicle navigation), but are insufficient for many robotic applications [58]. A second limitation of a GPS is noisy or weak signals in obstructed areas. For example, a study found GPS accuracy to decrease by 29% in young forest conditions (and 50% under closed forest canopies) compared to clear open skies [59].

To mitigate some of the limitations of GPS, a real-time kinematic (RTK) system can be used. Using a traditional GPS signal, a second data stream is acquired, with a correction value allowing for centimetre accuracy [60]. Correction values are received from a fixed base station, which must be in the receiver's general area; the closer the receiver and base station, the more accurate the values. Some RTK implementations have achieved millimetre accuracy; however, even low-cost RTK units cost hundreds of United States dollars (USD) [61]. Although an RTK system can provide the desired accuracy, it still struggles in GPS-denied environments.

A more common method for increasing the accuracy of a GPS is to apply signal processing and state estimation techniques on the raw position data. A popular method is to apply a Kalman filter [62] especially when fused with additional data streams [63]. One sensor that is commonly fused with GPS data is a magnetometer. A GPS receiver fundamentally outputs position, and when at rest, the orientation or heading is unknown. Once the receiver is moving, the heading is found with an estimator, allowing for a more accurate position estimate. To address the shortcoming of only receiving an accurate heading with motion, a magnetometer can be used. A magnetometer

measures the direction and strength of a magnetic field [64]. Measuring the Earth's magnetic field, applying calibration corrections, and computing some trigonometric equations will yield a heading, even when the system is at rest. Because the Earth's magnetic field is feeble, magnetometers must be carefully calibrated for reliable data [65]. Earth's weak magnetic field is also problematic when operating close to objects containing ferromagnetic materials or generating electromagnetic fields; the two most common corrections are hard and soft iron distortion [66]. Hard iron distortion compensates for any permanent magnetic fields from ferromagnetic materials such as a screw or bolt near the sensor. Soft iron compensation corrects for distortion caused by a material that does not necessarily generate its own magnetic field.

Many other sensor fusion methods and estimation techniques exist for increasing the reliability and accuracy of GPS positioning. Proven examples include: laser imaging detection and ranging (lidar) GPS fusion and estimation using an unscented Kalman filter to aid UAV position estimates in urban environments [67], an odometer GPS inertial navigation system (INS) using a Kalman filter to localize robots [68], and an RTK GPS INS wheel odometry vision system using an extended Kalman filter also to localize a ground robot [69]. However, a GPS is optimized for outdoor use; even when combined with other sensors, it is not feasible for localizing a UAV for precision tasks in GPS limited or denied environments. GPS-denied environments are of particular interest, as many of the previous payloads mentioned are used indoors. Additionally, machinery diagnostics will require access to hard-to-reach areas or will be surrounded by extensive infrastructure (especially in Alberta), which will degrade both the accuracy of a GPS signal and magnetometer readings. A GPS may still be used for certain applications; however, alternative localization techniques must be considered.

Methods for localizing in GPS-denied environments with high accuracy and reliability can be categorized into two main groups: ranging and non-ranging [70]. Ranging refers to acquiring distance information (similar to a GPS), from any number of sensors, including ultrasonic range finders, lidar sensors, and Bluetooth or Wi-Fi signals. Using the acquired data and known or unknown location information, a system can be localized with various technologies and methodologies [71]. The acquisition and processing of the signals are fundamental in what is known as simultaneous localization and mapping (SLAM).

The second type of method for localizing in GPS-denied environments are non-ranging methods, which primarily use image navigation techniques [70]. Images acquired by a camera contain large amounts of data. Methods such as optical flow, stereo vision, and detection of fiducial markers are just a few techniques to acquire localization data. As with ranging techniques, the data and algorithms used are common in VSLAM (visual simultaneous localization and mapping).

Other alternatives to ranging and non-ranging techniques do exist. For example, an INS system

does not require external references. Using inertial measurement units (IMUs) to sense motion, the addition of equations of motion, and dead reckoning, the sensor's position, orientation, and velocity can be tracked. The system can be highly accurate in the short term; however, due to dead reckoning, errors can compound, especially when a model of the environment's influences is lacking due to insufficient direct displacement measurements [72].

Each of the methods discussed has advantages and disadvantages, and rarely is one method used by itself. Data is often acquired from many different sensors and fused with algorithms specific to an application. A common trend is to use one or more data inputs coupled with an INS. Ranging examples include using an INS and ultrasonic sensors to successfully localize a micro air vehicle [73], development of a SLAM algorithm using lidar and an IMU for indoor UAV localization and mapping [74], and fusion of an ultra-wideband wireless sensor network and IMU to localize a mobile agent in an indoor setting [75]. Non-ranging examples include the development of an innovative filter for stereo vision and INS fusion [76], a camera-based colour blob detection algorithm and IMU fusion algorithm to aid in autonomous UAV quadrotor landings [77], and the comparison of using optical flow data and state of the art estimation techniques with and without IMU compensation for UAV flights [78]. Each paper uses a different fusion technique; however, the standard Kalman filter and its variants-such as the extended Kalman filter (EKF) and unscented Kalman filter (UKF)-are most common. Lastly, a motion capture system, such as Vicon, is not considered in the present work, as the technology requires extensive external infrastructure and does not provide a general-use solution.

1.3.2 Control

Once the UAV is localized within its given environment, a control scheme must be implemented so the system can be maneuvered to execute a given task. Every task will have different disturbances, especially when interacting with a given environment. To mitigate an unstable control system and increase system robustness due to assumptions from the modelling process, various control schemes and techniques have been proven [79].

A study published in October 2020 considered the Web of Science and Scopus databases and calculated the percentages related to various control strategies of quadcopter and multirotor UAVs found in literature over the past five years [79]. Figure 1.3 summarizes the results, and it was noted in the paper that there is more research with quadrotors than multirotors. In terms of quadrotors, the proportional integral derivative (PID) controller is the most commonly implemented linear controller, while the sliding mode and adaptive controller are the most popular nonlinear techniques. In the realm of multirotors, neural network and fuzzy-based controllers are the most popular control techniques, although the PID controller remains among the top three most frequently implemented controllers.

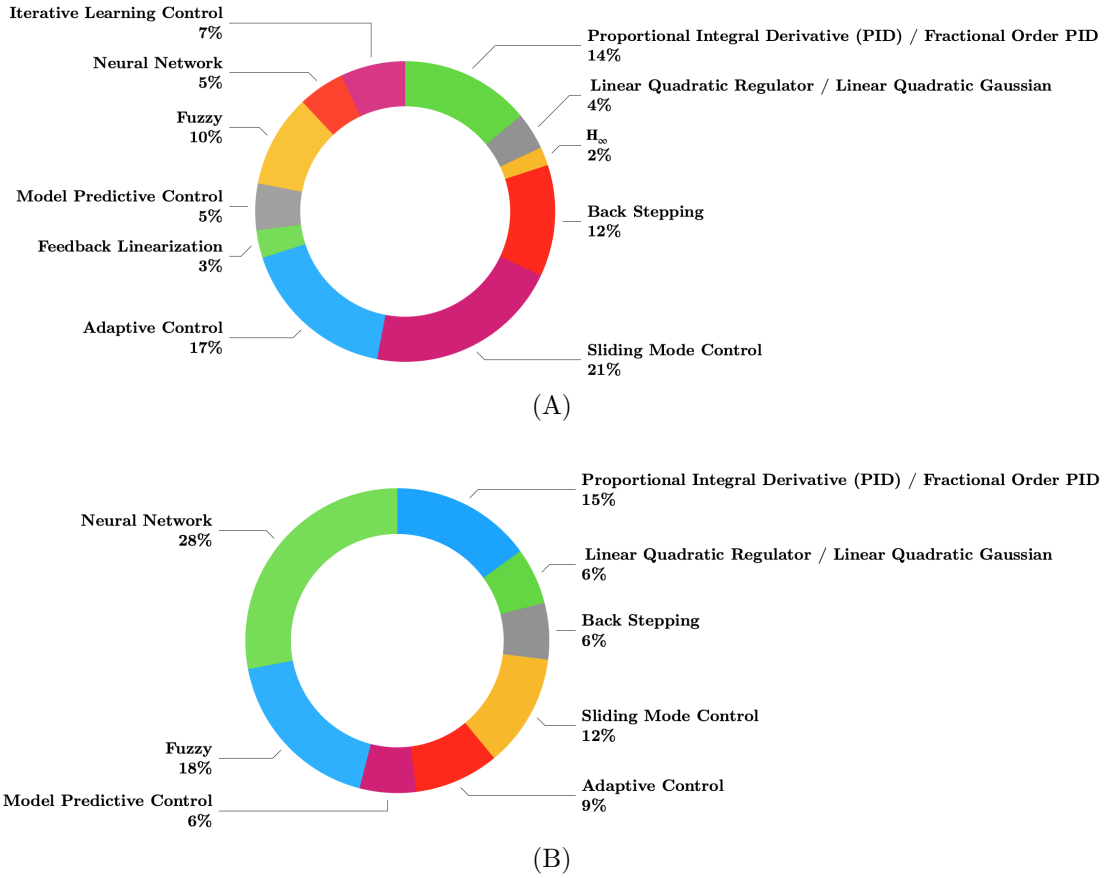


Figure 1.3: Distribution of Various Control Strategies For: (A) Quadrotor, (B) Multirotor [79]

With the PID controller being a popular implemented linear controller for quadrotor and multirotor UAVs, many studies exist in literature [36, 80, 81, 82, 83, 84]. New methods of tuning and optimizing gains for the PID controller are becoming increasingly popular; such methods include genetic algorithms [85], particle swarm optimization [86], and artificial neural networks [87]. The PID controller was also commonly found to be combined with other controllers, such as creating a fuzzy iterative learning control PID controller [88]. Additional linear control techniques exist such as the linear quadratic regulator (LQR) controller [89, 90] and the H_∞ controller [91, 92]. Due to the nonlinear nature of multirotors, nonlinear controllers were developed to handle situations where a linear controller would fail [79].

Examples of popular nonlinear controllers include sliding mode and adaptive controllers, as mentioned previously. Specific examples include implementing a quaternion-based third-order sliding mode controller, which was capable of achieving robust nonlinear performance in numerical simula-

tions [93]. Another sliding mode controller was implemented with a nonlinear disturbance observer to overcome parametric uncertainties and exogenous disturbances [94]. Sliding mode controllers and adaptive controllers have been combined to yield adaptive sliding mode controllers capable of controlling UAVs with two degree of freedom robotic arms for picking up and delivering objects [95]. Specific implementations of adaptive controllers include stabilizing a quadrotor system carrying a payload with a varying center of gravity [96], and the design of an observer-based adaptive output feedback controller for a quadrotor system with bounded uncertainties [97].

Many other controllers exist, and additional examples and implementations include the development of an auto disturbance rejection controller for a transmission power line inspection UAV [98], a model predictive controller used on a multirotor UAV with a manipulator for pick and place tasks [99], a fuzzy fault-tolerant tracking control system capable of handling nonlinear aerodynamic disturbances and actuator faults [100], and the use of model-based reinforcement learning to implement low-level control schemes based on experimental training data [101].

1.3.3 Payloads

Interfacing with a given payload requires the transmission of mechanical energy for physically locating a payload, electrical energy for powering active components of the payload, and a data communication stream for passing information between the payload and UAV. There are many modular payload or tool changer designs presented in both industry and literature for ground-based systems. Computer numerical control (CNC) machines use a variety of automatic tool changers (ATCs) leading to nearly fully automated workflows with little downtime [102, 103, 104]. ATI Industrial Automation developed the ATI Robotic Tool Changer for multi-degree of freedom (DOF) manipulators. The Robotic Tool Changer has more than 30 models, capable of loads ranging from 1.4 kg to 4000 kg even in high-heat or high-vibration environments [105]. A novel wedged lock latch payload adapter was designed and tested as an alternative to the common taper and ball approach used in ATI's design. Experimental results proved the system was capable of handling small service robotic applications [106]. Payload adapters are also found in space applications. For example, Jet Propulsion Laboratory (JPL) integrated payload adapters into its all-terrain hex-limbed extra-terrestrial explorer, allowing any of its legs to select tools from a holster, turning the leg into a 6 DOF manipulator [107].

The ground-based modular payload systems presented may be transferable to a UAV; however, ground-based manipulators tend to have access to more power, lower mass constraints, and finer positional accuracy than a UAV such as a multirotor. Therefore, integrating a ground developed system with a flying system may prove challenging. From a detailed analysis of existing literature and patents, modular aerial payload designs have been of interest; however, no autonomous working system was found. A modular payload system for small mobile robots, including UAVs, is conceptu-

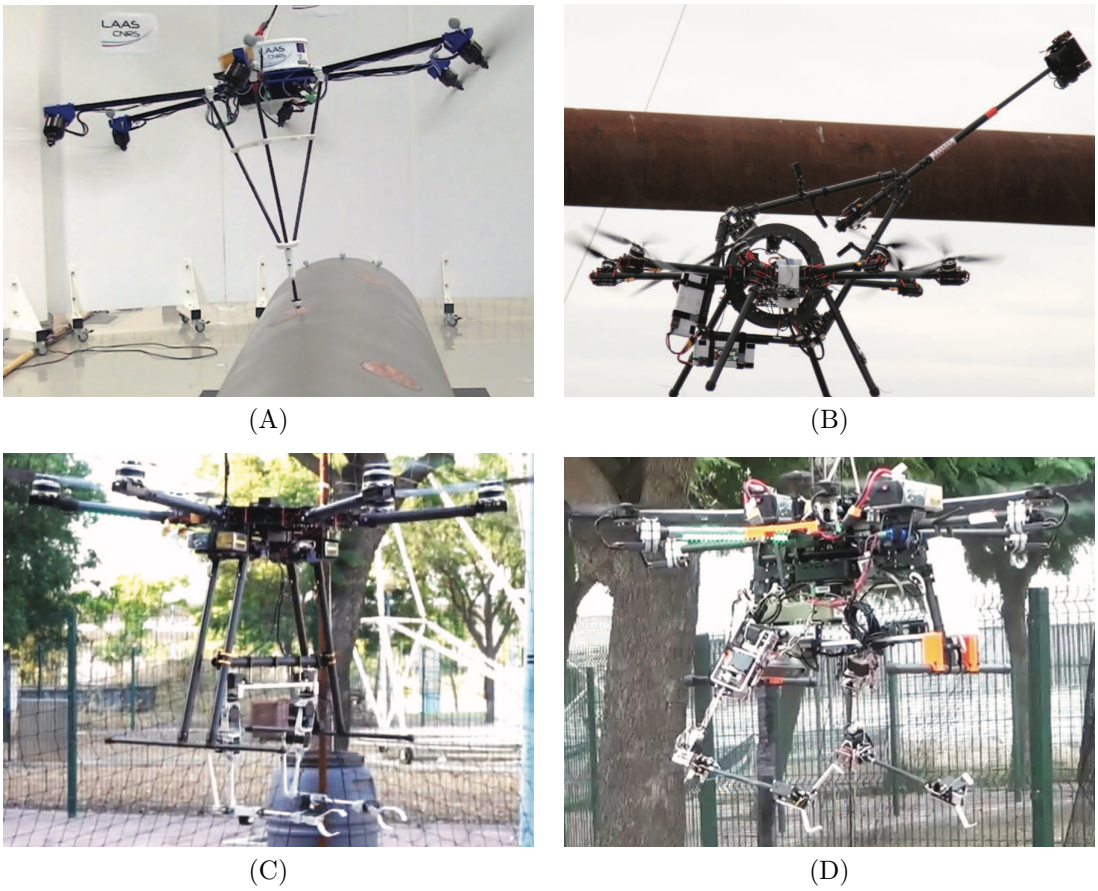
alized in [108]. Only the mechanical shell was shown to be prototyped, with no evidence of electrical or software prototyping. Additionally, experimental results were non-existent. A universal payload interface for UAVs focusing on electrical and software design was presented in [109]; however, the system does not allow for autonomous payload pick-up and drop-off without human intervention. Northrop Grumman Systems Corp has a patent for a layered architecture for customer payload systems which uses a modular design of different payloads; however, patent information suggests payloads are for fixed-wing UAVs and not autonomously swappable [110]. Although only partially applicable to interfacing with different payloads, systems for automated UAV docking and battery charging or swapping have been researched with successful results [111, 112, 113, 114].

1.3.4 Aerial Manipulation

Examples of aerial manipulation tasks have already been presented [33, 34, 35, 36, 37, 38, 39]; however, the fundamental challenges and approaches have yet to be discussed. One of the many challenges with aerial manipulation is controlling the UAV due to interactions with its environment [32]. UAVs interacting with the environment can be simplified into three main categories [36]: *Momentary coupling* (interacting with objects of finite mass in the air that are not fixed to the surrounding environment); *loose coupling* (interacting with objects attached to the environment without securing to the environment, in tasks such as assembling, inserting, pushing, or pulling); and *strong coupling* (securing or perching onto fixed objects in the environment and becoming part of it).

The challenge of executing any of the three coupling types is compensating for the reaction dynamics. When a UAV has coupled itself with the surrounding environment, the forces the UAV experiences interfere with the aerodynamics and associated model of free flight, creating a risk of the UAV becoming unstable and failing its assigned task [32]. "An analogy is to imagine a scuba diver turning an underwater valve. The diver contorts his body and aggressively kicks his legs in order to throw his body weight into turning the valve with his arms. Likewise, a UAV must 'swim' through air and may have to vector its body in order for the arms to dexterously manipulate an object" [115].

To address the issue of reaction dynamics and maintain a stable system, different approaches can be taken. Firstly, a well-designed mechanical system can significantly reduce the influence of reaction dynamics [28]. Approaches such as soft contact dynamics and smooth actuation of a manipulator can significantly improve the system's capability. The planning and modelling of the system is critical; there are two main approaches, a centralized approach and independent approach [32]. A centralized approach considers both the UAV and payload to be one system, and the controller treats everything as such. In an independent approach, the UAV and payload are treated separately, and the effects of the payload on the UAV (and vice versa) are treated as disturbances. A centralized



(A) (B)

(C) (D)

Figure 1.4: Four AEROARMS platforms [116]: (A) multidirectional-thrust hexarotor with a rigidly attached end effector, (B) multidirectional-thrust octorotor with inspection arm, (C) aerial dual-arm manipulator with stiff joints, and (D) aerial dual-arm manipulator with compliant joints

approach is simpler to implement than an independent approach; however, an independent approach tends to yield a more accurate and responsive system [28].

Other challenges exist with aerial manipulation systems, such as localization and control, which were previously discussed. The single greatest challenge of aerial manipulation is that most designs are complex mechatronic systems in a relatively new field [28]. Aerial manipulation requires developing many different subsystems that must all work together for reliable and robust performance. Entire teams often work together for years to develop an end product such as the AEROARMS Project, which aims to develop aerial robots with advanced manipulation capabilities for various inspections and maintenance [116]. Examples of four different AEROARMS platforms can be viewed in Figure 1.4.

1.4 Research Objectives

The use of UAVs is rapidly increasing in commercial applications and the research community [28]. One branch of UAV research is aerial manipulation, where researchers have demonstrated using UAVs to physically interact with an environment and achieve tasks such as opening drawers [33] and acquiring water samples [37]. Reviewing aerial manipulation literature, it was observed that developed systems were designed for a limited subset of applications. For example, the UAV, which could acquire water samples, could not open a drawer, although both systems were based on similar quadcopter specifications. The main difference between systems is the software for localizing and controlling the UAV, and the UAV's manipulator. Software can be developed to share technologies such as localizing and control; however, manipulators are typically rigidly fixed to the UAV and are not as cross-compatible.

Further review of aerial manipulation literature revealed no developed system capable of localization, control, and autonomous payload swapping. Therefore, the presented research aims to develop a UAV system that can interface with multiple payloads and provide fundamental localization and control. Developing such a system will allow researchers to use the platform for specific areas of research while minimizing others. For example, optimized payloads can be developed with limited knowledge of localization and control theory, or a focus can be placed solely on developing a novel control system capable of handling future developed payloads. The main benefit of the proposed system is that the UAV can achieve virtually any task for which it has a payload. On one flight, the UAV can be conducting a contact inspection; on the next flight, it can be transporting samples across a job site.

The presented thesis will be empirically completed. Conceptual simulations and models will be used for early system validation; however, the final presented system will be measured by real-world results. An empirical thesis was selected because aerial manipulation is a complex system composed of many parts and fields of study. Without conducting real-world experimentation, confidence in the proposed system would be low. The development process will begin by selecting a UAV platform. The localization and control methods will be developed in tandem until a universal payload system comes to fruition and the development streams are integrated into a final solution. Upon completing the development, the system's localization, control, and payload system will be individually validated for overall performance and further tested as a fully integrated system.

1.5 Thesis Outline

Chapter 1 provided background and motivation for developing an aerial manipulation multirotor UAV system capable of localization, control, and ability to interface with various payloads. Existing

literature was reviewed to highlight current technologies and show gaps for the development of the proposed UAV system.

Chapter 2 supplies the relevant theory for localizing and controlling a UAV system based on ArUco fiducial markers. The steps of extracting pose information from an image and the required transformation of coordinate frames are outlined. The theory of fusing inertial data to vision data using a Kalman filter is then discussed. With a localized system, a PID controller with trajectory generation is conceptualized for maneuvering the UAV in a given environment.

Chapter 3 describes the UAV system and justifies component selection, such as the autopilot, single board computer, and camera. An outline of the software architecture is provided based on the theory from Chapter 2. The design of a universal payload is presented, along with a detailed description of the mechanical, electrical, and software design. The addition of a payload docking station is discussed.

Chapter 4 outlines the conceptual design and design space of a twist cap manipulator used for accessing sump lubricant samples. Methods for removing a twist cap are explored with preliminary models and early experimental results. A final design of the manipulator with integration to the UAV is then presented, with a focus on mechanical, electrical, and software design.

Chapter 5 presents the experimental results of the UAV system and twist cap payload. Preliminary testing of simulations and a tethered system is discussed, followed by the developed software's computational performance, real-world validation of the localization system, results of tuning the PID controller, and attempts of precisely landing the UAV. Finally, the complete system is demonstrated, including retrieving a payload and opening a twist cap.

Chapter 6 summarizes the research completed and the resulting findings. Possible future work is also presented.

Chapter 2

Localization and Control

2.1 Localization

The first challenge of the proposed system is localizing the UAV in a given environment so that it can navigate and perform tasks. With endless combinations of sensors and techniques available to localize a system in a GPS limited or denied environment, a ranging or non-ranging technique had to be selected. Ultimately, a non-ranging localization technique using fiducial markers was selected. Fiducial markers were of interest because they allow for a structured environment that does not rely solely on naturally occurring features. Additionally, the tasks performed by the UAV can be in proximity of fiducial markers, and other techniques such as GPS or manual control can be used to navigate the UAV in proximity to the markers. Alternatively, a large area can be mapped with fiducial markers. The use of fiducial markers is also of interest because it only requires a lightweight, inexpensive camera looking for standardized markers, and was found to be a successful strategy gaining popularity with UAV research [80, 81, 82, 117, 118]. Additionally, previous work had explored the initial framework of such a system [52].

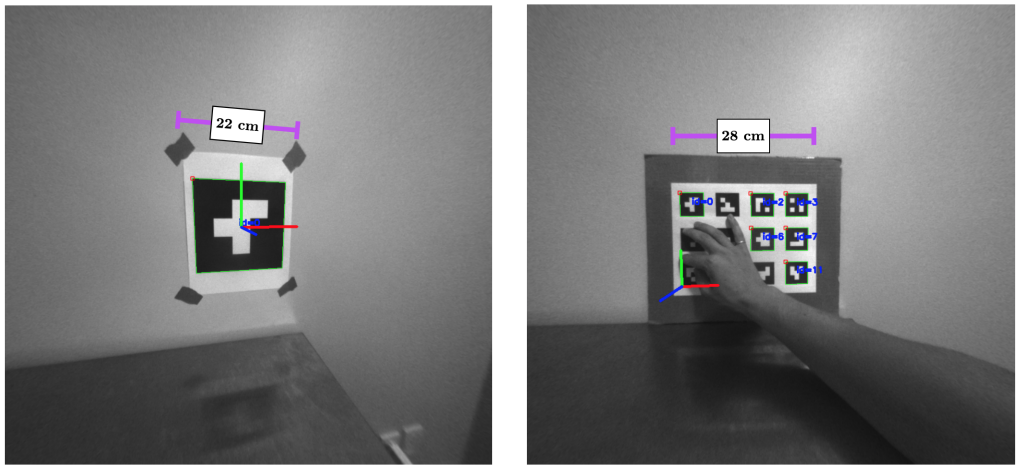
2.1.1 ArUco Library

The Augmented Reality University of Cordoba (ArUco) library [119, 120] was first released in 2014 as a fiducial marker system for camera pose estimation applications such as robot localization and augmented reality. The estimation of a pose is a fundamental component of localization and provides the 3D position and orientation of a body in space. The Open Source Computer Vision (OpenCV) [121] library, which specializes in classic and state-of-the-art real-time computer vision applications, quickly adopted the original ArUco library as a sub-module, and the two were integrated to make

use of each other's developed functions. The integration of the two libraries provided a foundation of tools to develop a localization system and was selected for two major reasons. Firstly, ArUco had highly optimized performance compared to other fiducial markers [119]; and secondly, OpenCV's advanced algorithms and open-source community provided documentation, examples, and support, thus reducing development time.

Standard ArUco markers feature an external black border with an inner matrix that encodes a binary pattern for unique identification. The black border simplifies the marker's detection in an image, and the number of bits is governed by the dictionary to which the ArUco marker belongs. An example of an identified marker can be viewed in Figure 2.1-A.

A limitation of a single ArUco marker is that if any part of the marker is corrupted in an image due to obstructions, lighting, or otherwise, the detection will fail. Additionally, a single marker will have a minimum operating distance; should the camera move inside the minimum distance, the marker will not be detected. Reducing the marker's size can decrease the minimum operating distance; however, the larger the marker in the image, the more accurate the pose estimation will be. To address this shortcoming, an ArUco board can be constructed. An ArUco board is comprised of multiple ArUco markers where only one marker is required for pose estimation. Any additional detected markers will further refine the pose estimation, increasing the accuracy. Therefore, an ArUco board allows for occlusions and is more robust to noise. An example of an ArUco board under occlusion can be found in Figure 2.1-B. Due to the increased pose estimation accuracy and reliability, the ArUco board will be used for the described system.



(A) (B)

Figure 2.1: (A) ArUco Marker ID 0 and (B) Occluded ArUco Board, both with Axis Projections and ID Labelling

2.1.2 Camera Models and Calibration

To calculate an accurate pose from ArUco markers, it is essential to have a well-calibrated camera. The ArUco algorithms make use of a pinhole camera model; however, when a camera with a large field of view (FOV) is used, the pinhole camera model breaks down and no longer holds. To convert from a large FOV to a pinhole camera model, OpenCV provides a fisheye camera model. The fisheye camera model is based on [122] and is simplified as follows.

Considering some point m in 3D space, Equation 2.1 describes the mapping of the point from the world frame to the camera frame. The coordinates of the point in the world reference frame are stored in r_w, and coordinates in the camera reference frame are stored in r_c. The rotation matrix R_{cw} and translation vector P_{cw} expresses the point transform from the world frame to the camera frame.

$$r_c = R_{cw} r_w + P_{cw} \tag{2.1}$$

Extracting the elements of r_c, the coordinates can be called x, y, and z as in Equation 2.2.

$$r_c = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.2}$$

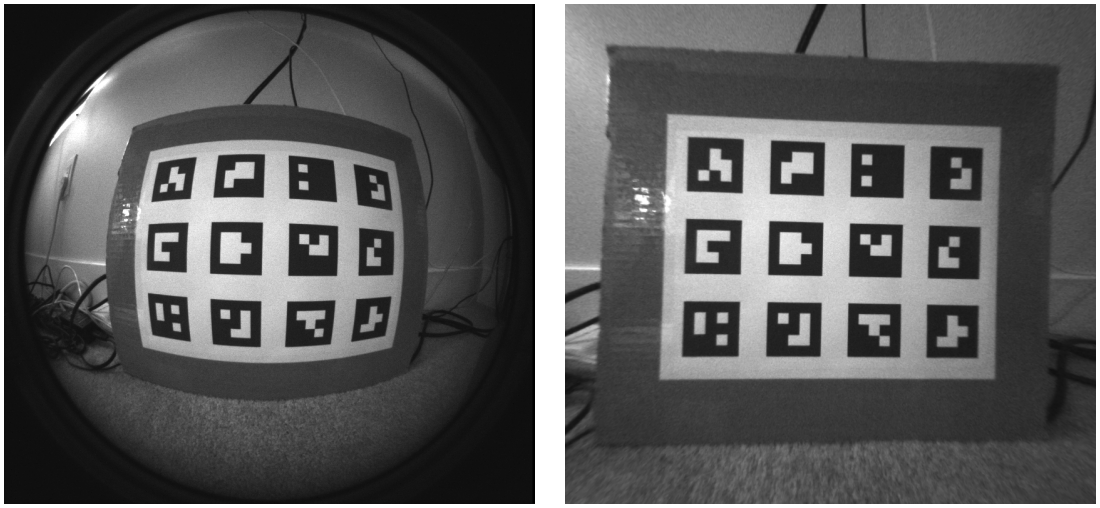
The pinhole projection coordinates a and b of point m makes use of the relations denoted in Equation 2.3, where r is the distance between the image point and the principal point, and θ is the angle between the principal axis and the incoming ray.

$$\begin{aligned} a &= \tfrac{x}{z} \\ b &= \tfrac{y}{z} \\ r &= \sqrt{a^2 + b^2} \\ \theta &= \arctan(r) \end{aligned} \tag{2.3}$$

The fisheye distortion, θ_d, is modelled in Equation 2.4, and the coefficients $k_{1..4}$ are intrinsic parameters specific to the camera and lens combination, which are empirically derived.

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{2.4}$$

The distorted point coordinates x' and y' are then a function of the fisheye distortion and pinhole projection points as in Equation 2.5.



(A) (B)

Figure 2.2: (A) Raw Fisheye Image, (B) Pinhole Equivalent Image

$$\begin{aligned} x' &= \frac{\theta_d}{r} a \\ y' &= \frac{\theta_d}{r} b \end{aligned} \tag{2.5}$$

Lastly, the distorted point coordinates can be converted into pixel coordinates u and v using Equation 2.6 and additional intrinsic camera parameters including focal lengths f_x and f_y, optical centers c_x and c_y, and the angle α between the horizontal zero axis and point m [123].

$$\begin{aligned} u &= f_x(x' + \alpha y') + c_x \\ v &= f_y y' + c_y \end{aligned} \tag{2.6}$$

Using the described model, a fisheye image can be converted to an equivalent pinhole model. The implementation can be observed in Figure 2.2, where the model successfully undistorted the fisheye image (Figure 2.2-A) to appear flat (Figure 2.2-B). The cost of the undistortion is a reduction in FOV. In Figure 2.2, the raw photo had a diagonal FOV of 163°, while the processed image was reduced to 128°.

A visual representation of the pinhole camera model is found in Figure 2.3. The model shares many similarities to the previously derived fisheye model and is explained in [124]. The distortion-free projective transformation, which is the basis of the pinhole camera model, is summarized in Equation 2.7, where P_w is a 3D point in the world frame, and p is the corresponding 2D pixel

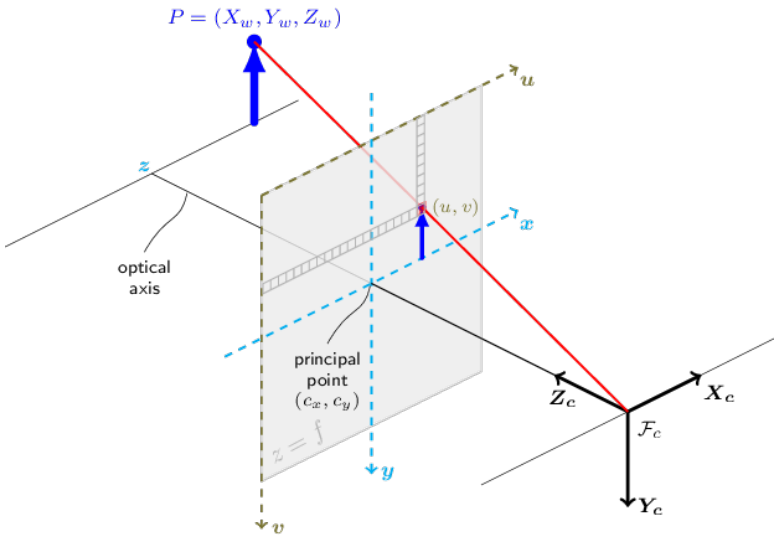


Figure 2.3: Pinhole Camera Model [124]

mapped by a perspective transformation in the image plane. The camera intrinsic parameters A are stored in a 3x3 matrix, R and t are the rotation and translation describing the transformation from the world frame to the camera frame, and s is the projective transformation's arbitrary scaling and not technically part of the camera model.

$$s\ p = A \left[R|t\right] P_w, \tag{2.7}$$

Both the fisheye and pinhole camera models are highly dependent on the camera's intrinsic parameters. These parameters can be supplied by the manufacturer or are experimentally derived. The values returned from a calibration are the camera matrix A, as in Equation 2.8, which holds the focal lengths f_x and f_y and optical centers c_x and c_y. The calibration also returns the distortion coefficients d, which are slightly different depending on if a fisheye, Equation 2.9, or pinhole model, Equation 2.10, is used. The distortion coefficient array is comprised of empirically derived constants k and p.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$d_{fish} = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix} \tag{2.9}$$

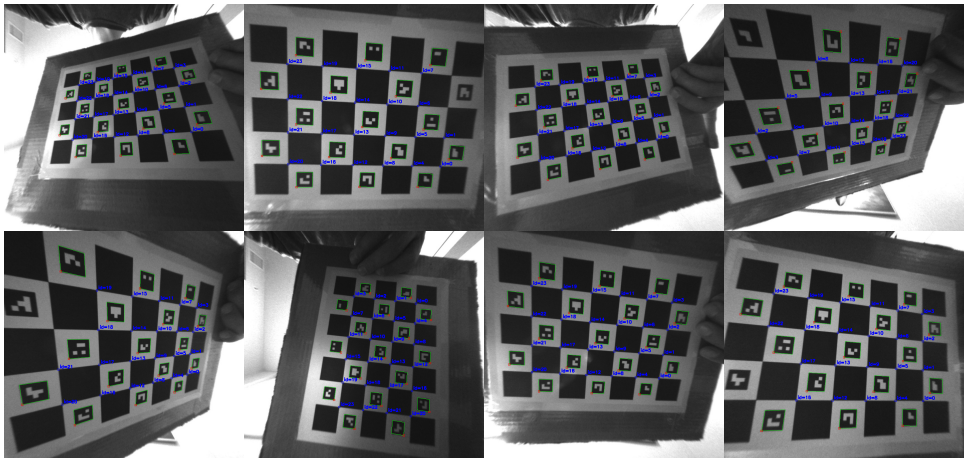


Figure 2.4: Example Calibration Images with Labelled IDs

$$d_{pin} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{2.10}$$

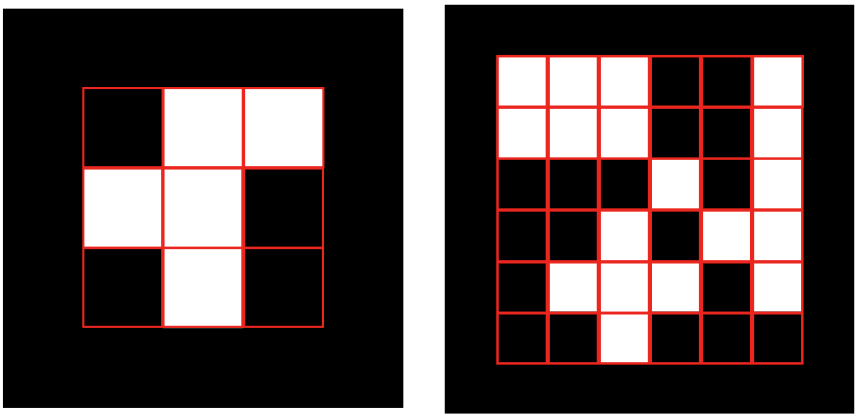
To retrieve the camera matrix and distortion values, an empirical calibration is conducted. The calibration is initiated by taking photos of a CharUco board, Figure 2.4, in multiple different orientations with the board filling a majority of the frame. A CharUco board was used over a traditional chessboard for calibration as the CharUco board is much more versatile, allowing for partial views.

The calibration algorithms are directly integrated into OpenCV and are based on the work completed in [125, 126]. A chessboard is the basis of the calibration, as the location where two black squares touch can be accurately detected in an image. Additionally, the size of the squares is known, and the squares follow a standard pattern. Using the derived camera models, the known dimensions of the CharUco board, and some additional algorithms outlined in the previous citations, the intrinsic parameters of the camera can be calculated.

For the work performed in this thesis, 100 photos were captured for one camera calibration. The image processing uses two sets of intrinsic parameters: a set of parameters provided by the manufacturer mapping from fisheye to pinhole, and a set of experimentally determined parameters further refining from the pinhole model to the ArUco library functions.

2.1.3 Image Processing and Pose Extraction

Before any image processing is conducted, an ArUco dictionary must be established. An ArUco dictionary is a binary list of markers to be considered in a specific application, and is composed



3x3 Bits 6x6 Bits

Figure 2.5: ArUco Marker ID 0 with Highlighted Bits

of two import parameters: the total number of markers in the dictionary, and the total number of bits used to create an ID. Using these two parameters, further detection and correction techniques can be implemented. Limiting the total number of markers in the dictionary and the number of bits increases the inter-marker distances, yielding better accuracy and reducing computation complexity. The cost of fewer bits is less total ID possibilities, so a balance must be considered for a specific application. A 3x3 bit marker and a 6x6 bit marker with the same ID are shown in Figure 2.5.

For the work completed in this thesis, a dictionary of 17 markers using 3x3 bits was determined to be the optimal configuration. Having 17 possible markers allowed for creating a CharUco board and left room for future marker IDs. A 3x3 bit dictionary was the smallest allowable matrix; a 2x2 system was quickly exhausted due to its limited ID range.

With an established dictionary, the pipeline for detection and identification of an ArUco marker is displayed in Figure 2.6. The original frame, Figure 2.6-A, is equivalent to an undistorted image such as the corrected fisheye image in Figure 2.2-B.

Using the original frame, Figure 2.6-A, the image is thresholded as in Figure 2.6-B. Due to the ArUco markers having a black border, the border can be identified using segmentation. The ArUco library uses an adaptive method, where each pixel's intensity is calculated using a predefined window size. If the intensity is higher than some constant value, the pixel is set to zero; otherwise, it is set to a value of one. Although the method is not computationally optimal, it is very robust in a variety of operating conditions.

Using the thresholded image, contours are obtained, as viewed in Figure 2.6-C, using the Suzuki

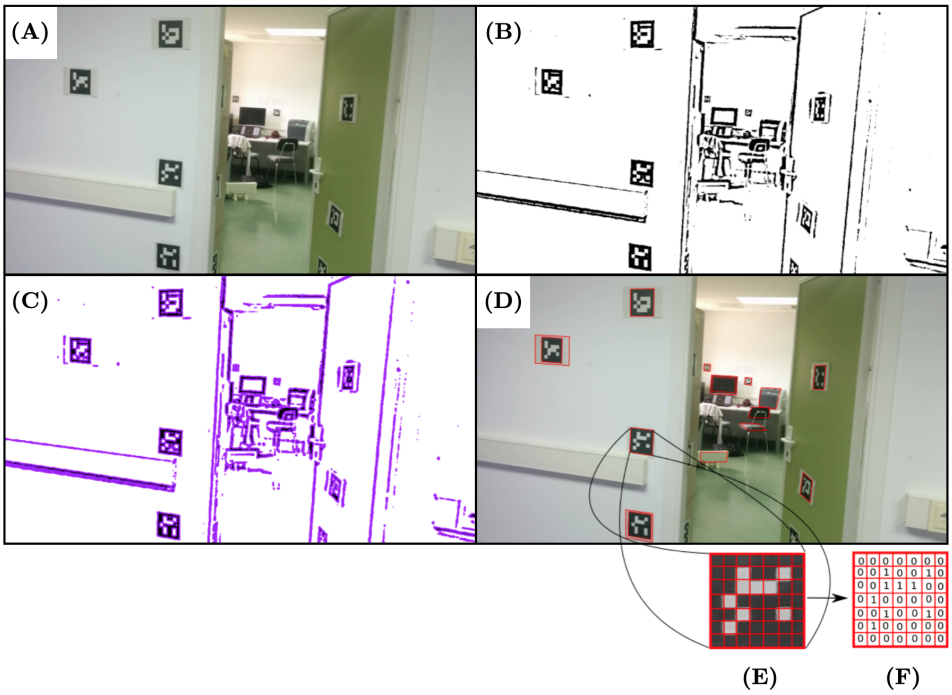


Figure 2.6: Detection and Identification Pipeline of ArUco Markers: (A) Original Image, (B) Thresholded Image, (C) Contour Extraction, (D) Filtered Contours, (E) Single Marker Extraction, and (F) Single Marker Binarization [130]

and Abe algorithms [127]. A filter then removes small contours associated with irrelevant background noise. The remaining contours are filtered using the Douglas and Peucker algorithm [128], which only keeps contours that approximate a four-corner convex polygon, as viewed in Figure 2.6-D.

The final remaining steps are marker extraction and binarization. The contour of interest is first refined using the pinhole camera model and perspective projection model as seen in Figure 2.6-E. The contour is then thresholded using Otsu's method [129]. The binarized image, as seen in Figure 2.6-F, is then sectioned into grids, and the values are compared to the defined dictionary. If the binarized contour matches the dictionary, the marker is valid and accepted.

With many of the algorithms used, there are customizable filter windows, coefficients, and parameters. It was found that a majority of the defaults were sufficient, and only the "Minimum Marker Perimeter Rate", "Corner Refinement Method", and "Adaptive Threshold Window" parameters required modifications.

The "Minimum Marker Perimeter Rate" determines the minimum perimeter of the marker. If the

parameter is too small, the computational load significantly increases down the pipeline as more contours have to be considered. However, if the parameter is too large, there is a risk that a marker contour will be removed before it is accurately identified in a later step. The argument passed to the algorithm is a ratio of the smallest perimeter of a marker in pixels to the largest pixel dimension of the input image. The default value is 0.03; however, it was found that a value of 0.05 decreased computation time without losing detection of markers.

The "Corner Refinement Method" argument is disabled by default, as it consumes computational resources and is only required for accurate pose estimation. As accurate pose estimation is required; this parameter was enabled. The function iterates over a region of interest, providing accurate sub-pixel corner locations previously discretized by the resolution of the camera.

The "Adaptive Threshold Window" consists of multiple additional parameters for setting the number of windows and their respective size. The default parameters yield three window sizes of 3, 13, and 23; however, the values were adjusted to have only two window sizes of 9 and 23. Performing the adaptive thresholding was extremely computationally intensive, and performing the thresholding with three different windows caused the program to become central processing unit (CPU) bound. Two windows lowered the computation requirements and unbounded the CPU process. The window sizes were determined empirically by testing a range of window values in different flight conditions and recording the number of accurately detected markers. A mid-sized and large-sized window vastly outperformed any combination with a small-sized window. The finals values of 9 and 23 yielded the best overall results; however, adjusting the window size by 20% in either direction provided comparable results.

With a processed image and the markers corner pixels' location known, the Perspective-n-Point (PnP) problem is solved to estimate the pose of the camera. The PnP method uses a set of n ($n \geq 3$) 3D reference points from the world frame and the 2D projections on the image plane to calculate a pose. The flag for an iterative method based on a Levenberg-Marquardt optimization was selected in OpenCV's PnP solving equation. The function with a Levenberg-Marquardt optimization minimizes reprojection error, which is the sum of squared distances between the observed projections image points and the projected object points [124]. The method provided reliable results without the need for high computational requirements.

The accuracy of the estimated pose is highly dependent on the size of the marker in an image. The larger the marker is in an image, the more pixels the marker uses, and the easier it is for the algorithms to detect corner points and contours. Accurate detection of corner points allows for precise pose estimation and correct identification of the markers' IDs. Increasing the camera's resolution also increases the estimation's accuracy because the marker will be comprised of more pixels; however, there is more data to be processed, which slows the calculations.

2.1.4 Transformation of Coordinate Frames

The resulting pose from OpenCV's PnP equation yields a transformation from the marker coordinate system into the camera coordinate system. The derivation and resulting transformation matrices follow the coordinate systems displayed in Figure 2.7.

Points X_a, Y_a, and Z_a from the ArUco marker expressed in the ArUco frame $\{a\}$ are projected into the image plane (u, v) using a perspective projection model Π, the camera intrinsic parameters A, and the transformation matrix T_{ca} as described in Section 2.1.2. The transformation matrix T_{ca} expresses points from the ArUco frame with respect to the camera frame $\{c\}$. A simplified form of the equation can be found in Equation 2.11.

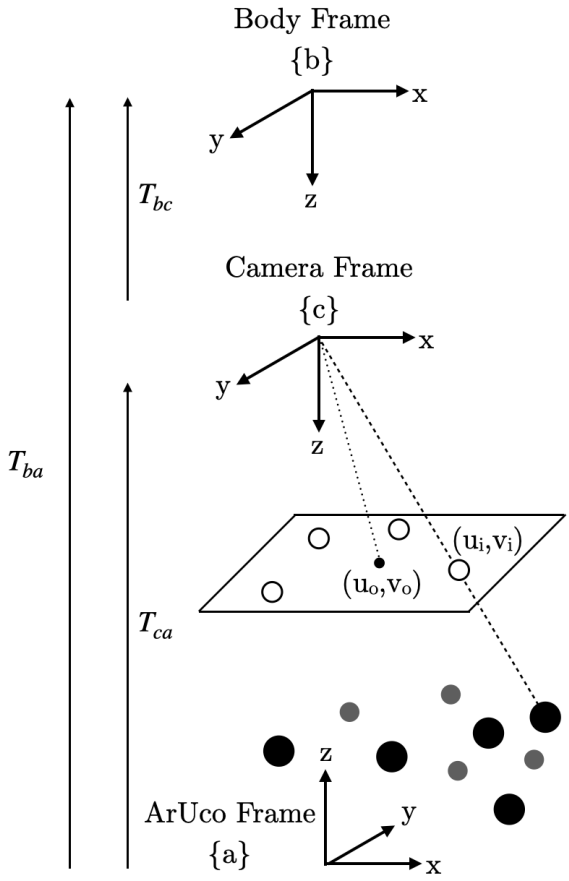


Figure 2.7: ArUco, Camera, and Body Coordinate Systems

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A\Pi T_{ca} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} \tag{2.11}$$

The expanded form of Equation 2.11 can be found in Equation 2.12 where f_x and f_y are the focal lengths, and c_x and c_y are the optical centers of the camera. The rotation matrix is comprised of r_{ij} where i and j denote the rotation matrix index and t_x, t_y, and t_z are the elements of the translation vector mapping the two coordinate frames.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} \tag{2.12}$$

Thus, the estimated pose is a combination of the rotation matrix and translation vector, which transforms a 3D point expressed from the ArUco frame into the camera frame as in Equation 2.13.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} = \begin{bmatrix} R_{ca} & t_{ca} \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} = T_{ca} \begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} \tag{2.13}$$

The UAV's body frame $\{b\}$ is located at the center of gravity (CG) of the UAV. For simplicity, the body coordinate system is assigned the same orientation as the camera frame. A translation from the CG to the center of the camera sensor is applied as in Equation 2.14. The variables d_x, d_y, and d_z are the distances from the CG of the UAV to the center of the camera optical lens in the x, y, and z directions defined in Figure 2.7.

$$T_{ba} = T_{bc}T_{ca} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} T_{ca} \tag{2.14}$$

Taking the inverse of the pose T_{ba} will yield the UAV's position in the ArUco frame. The inverse is desirable, otherwise small movements in the camera are amplified, making the UAV challenging to control. The process of inversing reference frames is found in Equation 2.15.

$$T_{ab} = T_{ba}^{-1} = \begin{bmatrix} R_{ba} & t_{ba} \\ 0_{1x3} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R_{ba}^{T} & -R_{ba}^{T} t_{ba} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.15}$$

To retrieve useful information from the UAV pose with respect to the ArUco marker, the transformation matrix T_{ab} is broken into its rotation matrix R and translation vector t as previously shown in Equation 2.13. The translation vectors are then mapped to match the coordinate system of Figure 2.8.

Extracting the yaw of the UAV requires processing the rotation matrix. The formalism of the rotation matrix provided by OpenCV is a ZYX Tait–Bryan representation which was found through numerical validation. The rotation matrix in its symbolic form can be found in Equation 2.16.

$$R_{ab} = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \cos\theta\sin\psi & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & \cos\ \sin\psi\sin\theta - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.16}$$

To retrieve yaw from the ZYX matrix, Equation 2.17 is used. The atan2 function is used in place of arctan to allocate the resultant angle to the correct quadrant.

$$\psi = \text{atan2}\left(r_{21}, r_{11}\right) \tag{2.17}$$

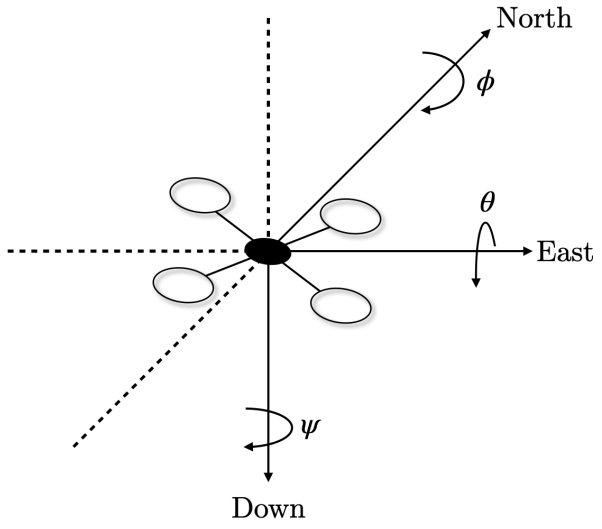


Figure 2.8: UAV North, East, Down (NED) Coordinate System

Lastly, the use of a downward-facing camera instead of a forward-facing camera was due to the calculation of yaw, as illustrated in Figure 2.9. With the configuration of a downward-facing camera, yaw is easily calculated, as there is an evident rotation of the four corners of the ArUco marker. Using a forward-facing camera, yaw is much more challenging to detect because the corner points' change will be minimal and more susceptible to error.

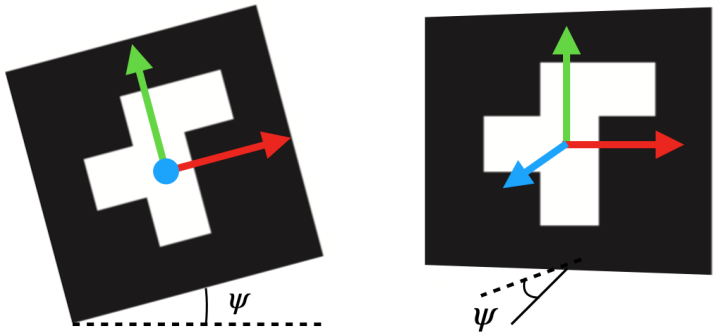


Figure 2.9: ArUco Yaw Calculation Based On Orientation: (A) Downwards Facing Camera vs (B) Forward Facing Camera

2.1.5 Kalman Filter

To filter the vision data and provide the controller with a reliable data stream, a Kalman filter [131] was implemented. Low-pass finite impulse response filters, such as a moving average filter and Olympic filter, were initially explored; however, the benefit of fusing multiple data streams to increase the pose estimation accuracy outweighed the cost of the increased integration complexity. Additionally, the Kalman filter allows for inertial navigation should the camera lose sight of an ArUco marker.

Assuming the UAV pose data stream is linear prevented the requirement of implementing a more complex Kalman filter derivative such as an extended or unscented Kalman filter. Modelling the system as linear does introduce some errors; however, it is a close approximation, especially if the control of the UAV minimizes non-linear movements. The derivation of the model and implementation of the Kalman filter follows the notation described in [132]. The dynamic and measurement model are shown in Equations 2.18 and 2.19 respectively. The system's hidden state and the observation are denoted as x and y, respectively, where k is some time step, A is the state transition matrix, and H is the observation model. Q and R are positive-definite noise covariance matrices of the measurement noise q and process noise r.

$$x_k = A_{k-1}x_{k-1} + q_{k-1}, \qquad q_{k-1} \sim N(0, Q_{k-1}) \tag{2.18}$$

$$y_k = Hx_k + r_k, \qquad r_k \sim N(0, R_{k-1}) \tag{2.19}$$

The state transition matrix for filtering yaw is formulated with basic angular kinematics, where it is assumed there is constant angular acceleration over some time interval. The variables θ, ω, and $\widetilde{\alpha}$ represent an angle, angular velocity, and angular acceleration, respectively. The time step remains k, and dt is the time difference since the last algorithm update.

$$x_k = \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} \theta_{k-1} + \omega_{k-1}dt + \frac{1}{2}\tilde{\alpha}_{k-1}dt^2 \\ \omega_{k-1} + \tilde{\alpha}_{k-1}dt \end{bmatrix} \tag{2.20}$$

Equation 2.20 can be rearranged and put into the form of Equation 2.21.

$$x_k = \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix} \tilde{\alpha}_{k-1} \tag{2.21}$$

The process noise comes from the angular acceleration, Equation 2.22, where e_{k-1} is the sensor noise and $\widetilde{\alpha}_{k-1}$ is some unknown controller input of angular acceleration applied to the body.

$$\alpha_{k-1} = \tilde{\alpha}_{k-1} + e_{k-1} \tag{2.22}$$

Using the relationship in Equation 2.23 [133], the covariance matrix of the process noise can be found as in Equation 2.24, where B is the coefficient of $\widetilde{\alpha}_{k-1}$ from Equation 2.21, I is an identity matrix, and σ_e is the standard deviation of the sensor noise.

$$e_{k-1} \sim N(0, I_{2x2}\sigma_e^2) \tag{2.23}$$

$$Q = B\Sigma B^T = \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix} I_{2x2}\sigma_e^2 \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix}^T = \begin{bmatrix} \frac{1}{4}dt^4 & 0 \\ 0 & dt^2 \end{bmatrix} \sigma_e^2 \tag{2.24}$$

The measurement model, Equation 2.25, is trivial to derive where the true values are corrupted by some measurement noise r_θ and r_ω.

$$y_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} r_\theta \\ r_\omega \end{bmatrix} \tag{2.25}$$

The measurement covariance matrix is comprised of the squared standard deviations, σ, of the noise for the respective sensor θ and ω, as in Equation 2.26.

$$R = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \tag{2.26}$$

The same approach can be used to filter the positional vision data. The main change occurs in the formulation of the state transition matrix, which changes from an angular derivation to a linear derivation. The change is shown in Equation 2.27 which makes use of the variables s, v, and $\widetilde{a}$ corresponding to linear position, velocity, and acceleration, respectively. The remaining model derivation follows the same process and is simply a change of variable notation.

$$x_k = \begin{bmatrix} s_k \\ v_k \end{bmatrix} = \begin{bmatrix} s_{k-1} + v_{k-1}dt + \frac{1}{2}\tilde{a}_{k-1}dt^2 \\ v_{k-1} + \tilde{a}_{k-1}dt \end{bmatrix} \tag{2.27}$$

2.2 Control

2.2.1 Proportional Integral Derivative (PID) Controller

The presented research focuses on designing a system capable of interacting with various payloads, not designing a novel controller capable of handling any payload. Therefore, a PID controller was selected for positioning the UAV; the implementation cost is low, and the design is well-researched. Additionally, a literature review showed that PID positional controllers are commonly used with success for quadrotor UAVs [36, 80, 81, 82, 83, 84, 134]. When the UAV is using a payload and the gains of the PID controller cannot be tuned to yield a stable system, the payload design will require a separate controller specific to its application.

The PID controller's basic form can be observed in Equation 2.28, where u is the controller output, and K_p, K_i, and K_d are gain coefficients for the respective proportional, integral, and derivative terms, which are a function of error e and time t.

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt + K_d \frac{de(t)}{dt} \tag{2.28}$$

With the goal of accurately positioning the UAV to execute some task, the UAV must be able to

navigate to a set point. To achieve navigation, the UAV uses the coordinate system previously shown in Figure 2.8. To navigate north, the UAV is given a pitch set point; to navigate east, it is given a roll set point. The down axis is controlled by a normalized thrust value, where the midpoint corresponds to a hover. Lastly, the UAV's yaw is controlled so that the UAV can be orientated in a configuration that may be required by a payload. Because no magnetometer is used and the gyroscope drifts, the UAV is given a yaw rate to achieve the desired orientation instead of a hard set value.

Additional control processing was required in tandem with the PID implementation. The controller outputs were constrained in order to limit overshoots, increase the controllability of the system, and attempt to keep a linear system. In addition to constraining the controller outputs, the integral term was bounded and reset at every mission to prevent integral windup. Similarly, the derivative term was bounded to limit amplification of noise due to the real-world operating conditions of the system. To further limit artifacts of noise the derivative term was low-pass filtered.

The final step of implementing the controller was mixing the roll and pitch based on the UAV's yaw. For example, if the UAV were orientated 90 degrees relative to the ArUco marker, roll and pitch control would need to be swapped. Additionally, if the UAV approaches a region of interest with some non-zero yaw value, the roll and pitch can work together for a smoother response. Mixing the roll and pitch is achieved by implementing a 2D coordinate transform as in Equation 2.29, where θ and ϕ is the new pitch and roll control values, ψ is yaw, and θ' and ϕ' are the original pitch and control values.

$$\begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \theta' \\ \phi' \end{bmatrix} \tag{2.29}$$

Due to the thesis's empirical approach, no transfer function or model of the system was developed. A centralized approach is assumed, since the UAV and any payload will be treated as one object, and the tuning of the controller will be conducted using the actual hardware.

2.2.2 Trajectory Generation

To smooth the positional control of the UAV and limit overshoots due to large positional errors, trajectory generation was performed. The trajectory generation was constructed from a 5th order time series polynomial as in Equation 2.30, which models position, s, as a function of time, t, with six unknown coefficients $A..F$.

$$s(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{2.30}$$

Taking the first and second derivatives of Equation 2.30 yields velocity, $\dot{s}$, and acceleration, $\ddot{s}$, profiles as in Equations 2.31 and 2.32 respectively.

$$\dot{s}(t) = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \tag{2.31}$$

$$\ddot{s}(t) = 20At^3 + 12Bt^2 + 6Ct + 2D \tag{2.32}$$

Substituting start and end times of 0 and T respectively, a 6x6 matrix can be constructed as in Equation 2.33.

$$\begin{bmatrix} s_0 \\ s_T \\ \dot{s_0} \\ \dot{s_T} \\ \ddot{s_0} \\ \ddot{s_T} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} \tag{2.33}$$

By selecting a time period and initial and end conditions for the position, velocity, and acceleration, the previously unknown coefficients can be solved and plugged back into Equations 2.30, 2.31, and 2.32. Using Equation 2.30, a list of time-varying positional set points can be calculated, which are also a function of the desired velocity and acceleration profiles. Using the trajectory as the desired set points instead of only the endpoints will minimize error in the controller and allow for stable control of the UAV's position.

With the relevant theory for localizing and controlling the UAV outlined, a UAV and universal payload system is developed in the following chapter so that the system can undergo empirical validation.

Chapter 3

System Overview

3.1 UAV System Overview

3.1.1 General Configuration

The UAV system, as observed in Figures 3.1 and 3.2, is designed around a 500 mm wheelbase quadrotor MWC X-Mode Alien frame from Afunta. A quadcopter was selected due to its mechanical simplicity, ability to hover, and maneuverability. The frame selected is optimized for imaging applications minimizing obstructions to the camera and providing ample room for mounting sensors, batteries, and other electronics. Four Emax XA2212 980 KV brushless motors, each paired with 10" x 4.5" MR propellers from HQProp, provide 8.6 N of thrust per motor. The motors receive three phase power from 20 A Racerstar RS20A V2 electronic speed controllers (ESCs) powered by a 5000 mAh Goldbat 10042126 three-cell LiPo battery. The battery's power is distributed to the ESCs using a PDB-XT60 power distribution board (PDB) from Matek Systems, which includes a battery elimination circuit (BEC) and DC/DC synchronous buck regulator for powering the flight controller. A separate Pololu D24V50F5 step-down voltage regulator is used to power the NVIDIA Jetson Nano single-board computer (SBC). A Pixhawk 4 flight controller from Holybro was selected as the primary autopilot hardware for the UAV system and includes numerous sensors and input/output (I/O) ports to interface with ESCs and other electronics. A FrSky X8R remote control (RC) receiver receives commands from a FrSky Taranis X9D Plus transmitter, facilitating manual and fail-safe control. Lastly, an NVIDIA Jetson Nano and Intel RealSense T265 tracking camera is used to process and acquire visual data, and provide the flight controller with additional controller inputs. A list of all the major components with their respective manufacturer and part numbers is summarized in Table 3.1.

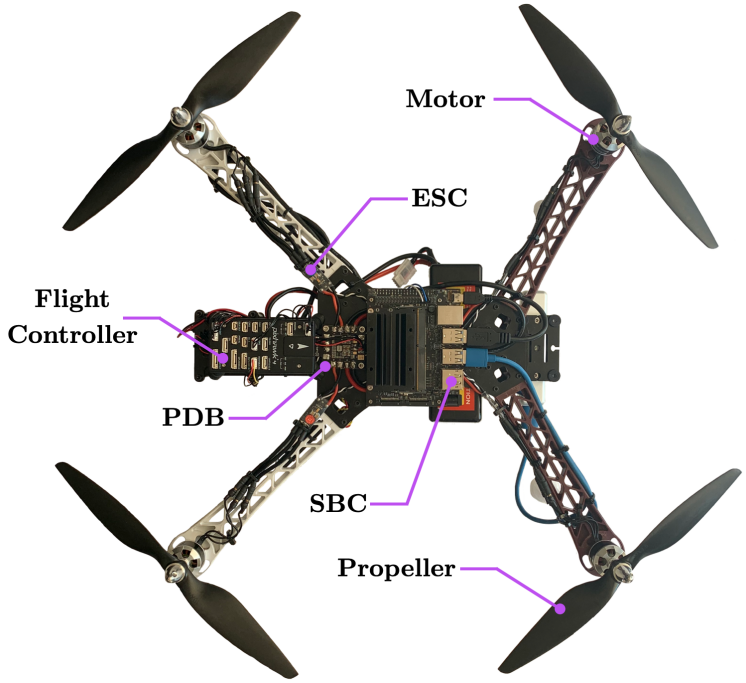


Figure 3.1: UAV Top View with Labelled Components

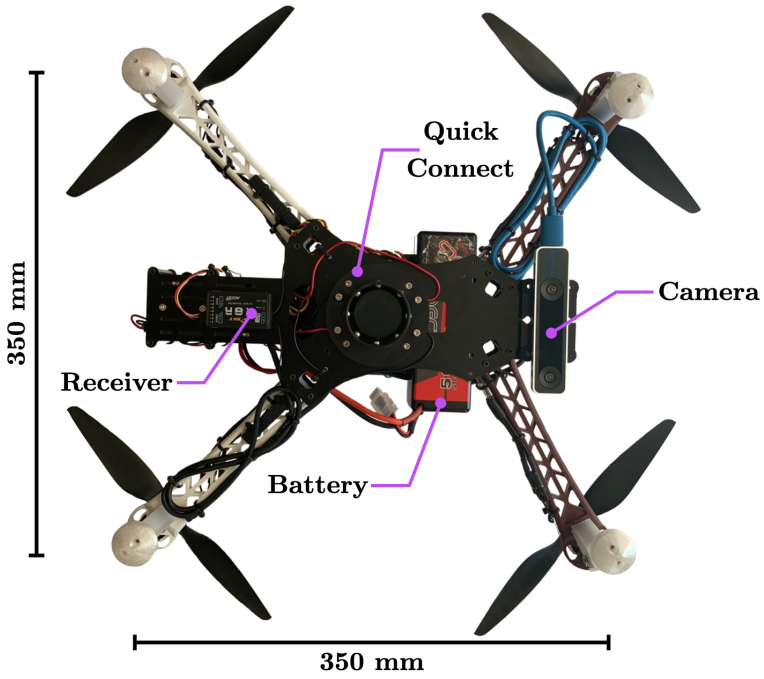


Figure 3.2: UAV Bottom View with Labelled Components

Table 3.1: UAV Manufacturer Components and Part Numbers

General Component Name	Manufacturer	Manufacturer Part Number
Frame	Afunta	HJframe-RB
Battery	Goldbat	10042126
Power Distribution Board	Matek Systems	PDB-XT60
ESCs	Racerstar	RS20A V2
Brushless Motors	EMAX	XA2212 980KV
Propellers	HQProp	MR 10x4.5 CW/CCW
Flight Controller	Holybro	Pixhawk 4
Receiver	FrSky	X8R
Voltage Regulator	Pololu	D24V50F5
Single Board Computer	NVIDIA	Jetson Nano Developer Kit (B01)
Camera	Intel	RealSense T265

A majority of the components are standard off-the-shelf parts; however, special consideration was required to mount and integrate all the parts into one system. Passively dampening the flight controller from vibrations reduced noisy sensor readings and provided more reliable controller outputs. To dampen vibrations, the mount from [135] was modified to match the mounting pattern of the UAV frame, and 3D printed using polylactic acid (PLA) filament. The camera was also dampened from vibrations using rubber grommets and plates supplied with the frame specifically designed for mounting cameras. To further reduce vibrations, all wires were cut to lengths that minimized tension on components and reduced the assembly's total mass. Vibrations in the system were characterized using data logs recorded by the autopilot firmware. The layout of the UAV components was arranged to keep the center of gravity (CG) about the center of the four propellers. Heavier components such as the battery and SBC were placed close to the CG and lighter components were placed further away in order to limit the moment that different components would introduce. Although the flight controller was not centered about the CG, offsets in the firmware allowed for corrections in the control algorithms. Custom parts were required for the UAV, including mounting of the SBC and landing gear extensions. The unpowered system weighs 1.4 kg and typically uses a 5000 mAh battery, which weighs an additional 0.4 kg. The battery is interchangeable depending on the mission to be conducted.

Even with a majority of parts being standard, the electrical system required custom wiring. A flow chart of the system can be observed in Figure 3.3. The battery voltage is distributed to the entire system from the power distribution board. The regulator in the PDB powers the flight controller, while the SBC uses its own dedicated voltage regulator. The two regulators are required because the PDB does not have the capacity to power both the flight controller and SBC. Both the flight controller and SBC route input power to their peripherals along with the required signal lines. The flight controller sends motor commands to ESCs, which convert the battery power

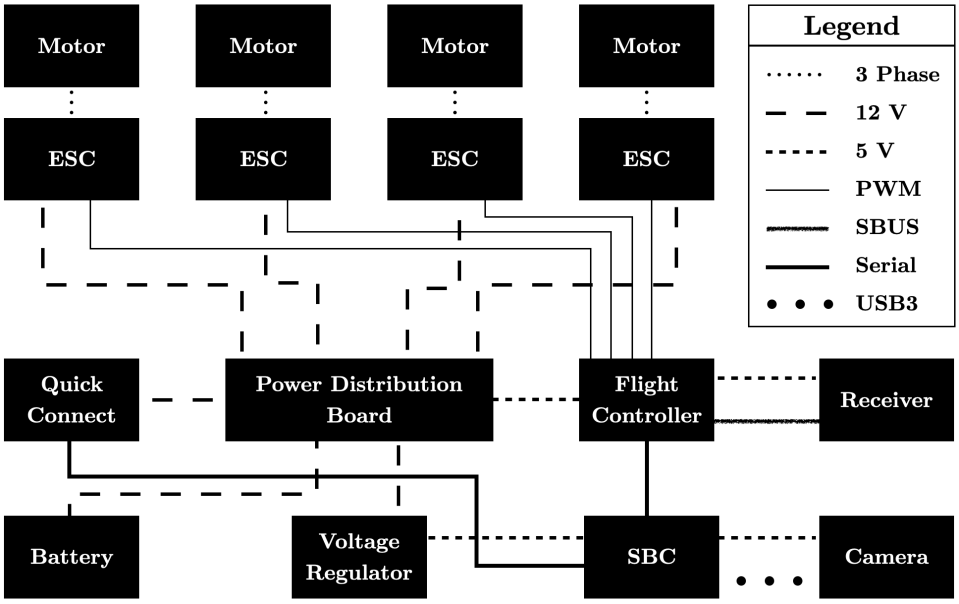


Figure 3.3: UAV Electrical and Signal Diagram

into three-phase electrical power to rotate the motors. Power and signal lines are routed to the quick connect device, which is further processed in a system of its own. To minimize signal noise, wires such as ESC signal and ground, as well as servos power, ground, and signal, are twisted together to improve electromagnetic compatibility. Additionally, all components share a common ground to prevent ground loops. Connections between the motors and ESCs used gold-plated bullet connectors to limit resistance and prevent oxidation on the contacts. Instead of semi-permanent solder connections, connectors were used to allow quick repairs if a motor required replacement or needed to have its direction reversed. The remaining connections made use of soldered connections or pre-crimped leads in housings based on the termination point. For example, all PDB connections were soldered, while connections to the autopilot used JST-GH series connections. The wire gauge was sized to minimize voltage drops and the total mass of the wiring. Lastly, electronics sensitive to electromagnetic interference (EMI) such as the microelectromechanical systems (MEMS) sensors were situated away from the high-current carrying wires (such as ESC power lines), in case any electromagnetic radiation were to escape the outer wire shielding.

3.1.2 Autopilot Selection and Integration

The Pixhawk 4 flight controller was selected, as the Pixhawk family of boards has been developed, tested, and iterated since 2013. The Pixhawk 4 is the newest release and features recent processor and sensor technology from STMicroelectronics, technologyfromBosch, and InvenSense.

Additionally, the board is designed for reliably controlling autonomous vehicles and only weighs 16 grams.

The Pixhawk 4 is optimized to run PX4 firmware; however, ArduPilot is a similar alternative, with both systems using Micro Air Vehicle Communication Protocol (MAVLink) messages to stream telemetry data and execute given commands. MAVLink is a very lightweight messaging protocol for communicating with drones and communicating between onboard drone components [136]. The main differences between the two firmware options are open-source licensing, underlying control schemes and estimators, and extra functionality outside of standard flight.

To simplify communication and control with PX4 or ArduPilot firmware, the MAVSDK software development kit (SDK) or DroneKit application programming interface (API) is used. Since both PX4 and ArduPilot use MAVLink commands, the SDK or API are technically interchangeable; however, the two firmwares use a different number of MAVLink messages, sometimes leading to cross-compatibility errors. The respective SDK and API filter the messages for the correct firmware where MAVSDK is optimized for PX4 and DroneKit for ArduPilot. The Python wrapper for MAVSDK was released in June 2019 and, when tested, was found to have an unreliable performance for real-time control. MAVSDK's reliability was found to decrease as data and command rates increased. Rates of approximately 10 Hz were found to be the fastest acceptable rate without sacrificing reliability; faster rates caused the SDK to overflow and then crash. Additionally, not all of the C/C++ functionality had been ported to the Python wrapper, leaving limited development solutions. Lastly, ArduPilot had more support for advanced control methods than PX4 (such as ground effect compensation [137]), leading to ArduPilot and DroneKit being the firmware and API of choice.

Using the vision estimation data and PID control loops, the UAV can be maneuvered by providing the autopilot with desired inputs. The inputs are encoded in the SET_ATTITUDE_TARGET MAVLink message, where desired roll and pitch are embedded in a quaternion to control the north and east axes. Because the magnetometer is deemed unreliable and disabled, yaw cannot be set directly; instead, yaw rate is used. Finally, to control the down axis, a value between 0 and 1 is computed, where 0.5 is a hover and values above or below correspond to a percentage of a configurable climb or descent rate, respectively. The setpoints are provided to the flight controller at 30 Hz, while an internal attitude control loop runs at 400 Hz to keep the UAV level. The described system makes use of ArduPilot-ArduCopter V4.0.4 firmware.

3.1.3 Single Board Computer Selection

There are two main methods for performing the required computations for controlling the UAV. The first method is to establish wireless communication between the UAV and a ground station.

Controller input data is pushed to the ground station, and the required calculations are performed. The controller output is then transmitted back to the UAV to be executed. The benefit is that such a system can use computers with ample processing power that will likely not bottleneck the system. The drawback is that a wireless system is more prone to data loss compared to a wired system. Additionally, as the UAV and ground station distance increases, larger antennas and power-hungry amplifiers will be needed to ensure reliable communication and minimal latency. Wireless methods have proven successful as in [81, 134].

The second method is to perform all the required calculations and processes onboard the UAV. The main challenge is finding hardware that is physically capable of integrating with the UAV, in terms of size, mass, and power requirements, while still providing sufficient sensing accuracy and computation power. The benefit of an onboard processing system is it allows for a standalone platform that can be deployed by simply powering on the UAV. Onboard, processing has been validated even with high-speed cameras as in [84].

The method for onboard processing was selected over a ground station for two main reasons. Firstly, single board computers (SBCs) are capable of functioning as media servers, machine learning platforms, and standalone computers [138, 139], and are relatively inexpensive compared to the cost of setting up a reliable wireless system and purchasing a ground station computer. Secondly, having a standalone system allows the UAV to be quickly deployed in virtually any environment and reduces the number of connections that need to be reliably established.

To meet the onboard processing requirements, six different SBCs were compared, as pictured in Figure 3.4. The boards selected were the Raspberry Pi 3B+ and 4B; the Odroid C2, C4, and XU4; and the NVIDIA Jetson Nano. The Raspberry Pi 3B+ and 4B and Odroid C2 and C4 were selected due to their large user community and affordable price [140, 141]. The Raspberry Pi 4 and Odroid C4 were newly released at the time of analysis, and their previous versions, the Raspberry Pi 3B+ and Odroid C2, had been used in initial system prototyping. The old boards were considered in the comparison to quantify the performance increases with the updated hardware. The Odroid XU4 was added as a new board consideration because it displayed some of the most promising benchmarks [139]. Lastly, the NVIDIA Jetson Nano was selected for the analysis because it boasted superior performance and is optimized for computer vision tasks [138].

The most computationally intensive task is working with image data. Therefore, to compare the boards' performance, an ArUco marker was placed on an empty illuminated wall. A Logitech C920 camera was placed at distance intervals of 50, 100, 150, 200, and 250 cm away from the ArUco marker, and data was then acquired for 10 seconds at each distance increment. The analysis was repeated for three different camera resolutions of 1920x1080, 1280x720, and 640x480. The SBC's dedicated Linux-based operating system was accessed through a headless connection to

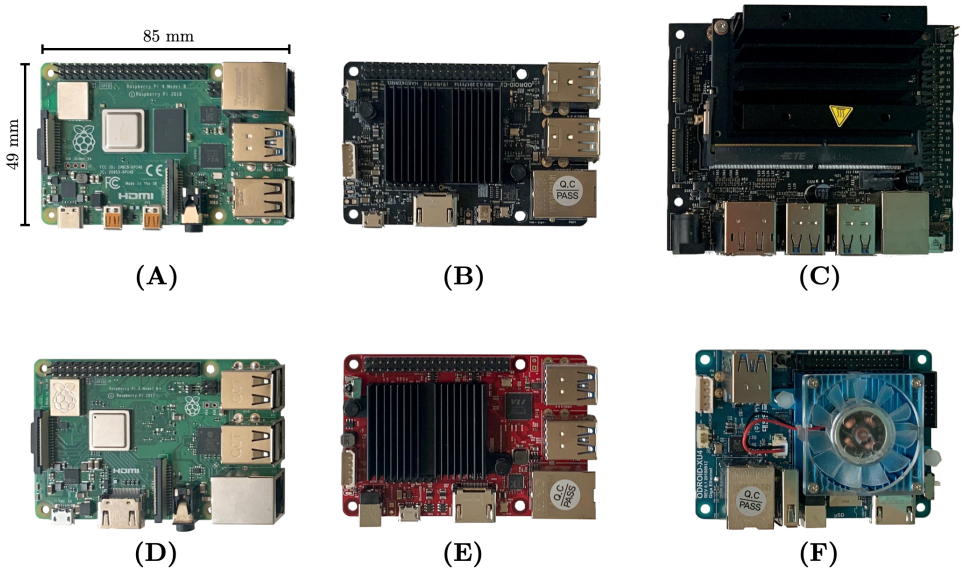


Figure 3.4: Single Board Computers Considered: (A) Raspberry Pi 4B, (B) Odroid C2, (C) NVIDIA Jetson Nano, (D) Raspberry Pi 3B+, (E) Odroid C4, (F) Odroid XU4

reduce the overhead of rendering a graphical user interface (GUI). The benchmark code was kept constant between all trails and used one thread for image capturing and the primary thread for pose estimation. Each board was powered by its recommended power supply. Upon completing the 10 second data acquisition period, the average and standard deviation of the frame rate and pose rate were calculated. The study results are observed in Figures 3.5 and 3.6 for the image frame rate and pose calculation rate, respectively. It should be noted that the comparison of the SBCs is a direct comparison of the boards in their entirety, and not their individual elements such as the operating system (OS), the amount of random access memory (RAM), and the type of data storage.

The maximum capture rate of the Logitech C920 camera is 30 frames per second (FPS); therefore, none of the SBCs' maximum frame rates will exceed 30 FPS (as observed in Figure 3.5). Most of the boards captured the full 30 FPS at low resolutions; however, as resolution increased, the capture rate trended downwards. This was expected, as there is more data to be unpacked. None of the boards could capture 30 FPS at a resolution of 1920x1080, although the Jetson Nano proved the most capable at 25.8±0.6 FPS, which is approximately 24% faster than the next best performing board, the Odroid C4. The difference in performance is primarily attributed to the USB hardware and the processing capability of the SBC of extracting data from the USB port into a usable image matrix. From the initial benchmark, it is clear that the RPi 3B+ and Odroid C2 are the slowest boards, which was expected.

The pose calculation, Figure 3.6, provides a better comparison of computation power. The frame capture and pose calculations are threaded, meaning that if one calculation has extra clock cycles available, e.g. waiting for a new frame, other calculations can be performed, and the code does not

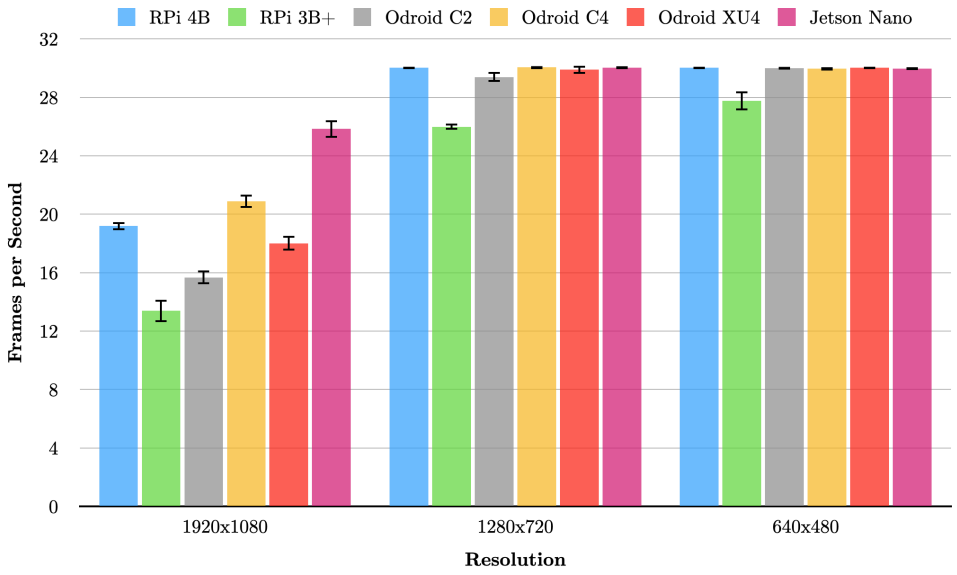


Figure 3.5: Capture Rate per Second vs Camera Resolution

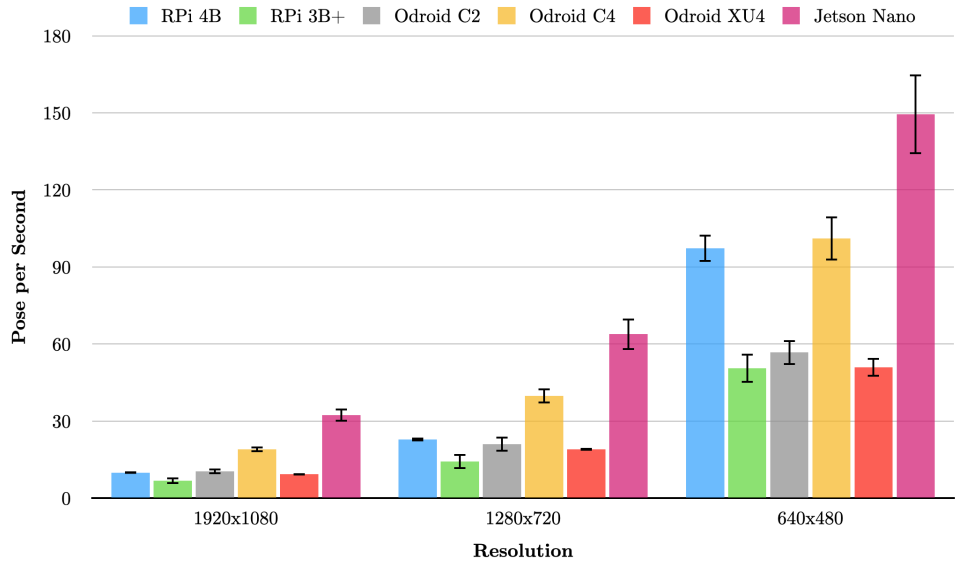


Figure 3.6: Pose Calculation per Second vs Camera Resolution

have to execute sequentially line by line. Therefore, one frame can be analyzed multiple times before a new frame is captured, explaining how rates above 30 were achieved. Calculating the same pose multiple times from the same image is useful as the system will be bottlenecked by the camera rate, which is consistent, and not by the pose estimation, which is highly variable. Additionally, a camera with a larger field of view will have more data to process, increasing the required computation power required by the adaptive thresholding algorithms. The Jetson Nano showed superior performance above all SBCs, even being able to process pose data from 1920x1080 frames at more than 30 times per second. The Odroid C4 proved to be the closest rival to the NVIDIA Jetson Nano, although the Jetson Nano's average pose calculation was approximately 60% faster across resolutions. The trend of decreasing resolution and faster processing time also becomes more evident across the boards.

Considering the FOV of the C920 camera, any SBC appears sufficient at a resolution of 640x480. However, the NVIDIA Jetson Nano and Odroid C4 can achieve approximately the same or better performance with 200% more pixels, which contributes to potentially a more accurate pose. A resolution of 1920x1080 is nearly feasible with the Jetson Nano; however, the capture and processing rate does not leave options for a wider field of view and provides minimal additional computational overhead. As anticipated, the newer models of both the Raspberry Pi and Odroid showed increased performance over their predecessors, which can be correlated to some initial performance issues during early development. Ultimately, the NVIDIA Jetson Nano was selected over the Odroid C4 due to its superior performance at the cost of an 85% larger footprint and 137% weight increase.

3.1.4 Camera Selection

The Logitech C920 HD Pro Webcam was initially selected as the camera of choice, as it was recommended for image processing applications due to its native encoding of images using the H.264 format [142]. Encoding in H.264 allows for efficient decoding of captured frames, and the camera was widely used in the OpenCV community. Additionally, the C920 outperformed its predecessor, the Logitech C270, in terms of supported resolutions, quality, and capture rate.

The C920 had two limitations that required selecting a new camera to increase the system's accuracy and reliability. The first limitation was that the C920 uses a rolling shutter. A rolling shutter scans across a scene quickly in some predefined pattern to capture the image in sections, which are subsequently pieced together. The alternative is a global shutter, where the sensor captures the entire scene at once. A visual comparison example can be found in Figure 3.7, where the filled squares signify a captured pixel.

The issue with a rolling shutter is this: if during the capturing of an image, the UAV is moving or vibrating faster than the camera capturing a section of data, the marker corners can belong to

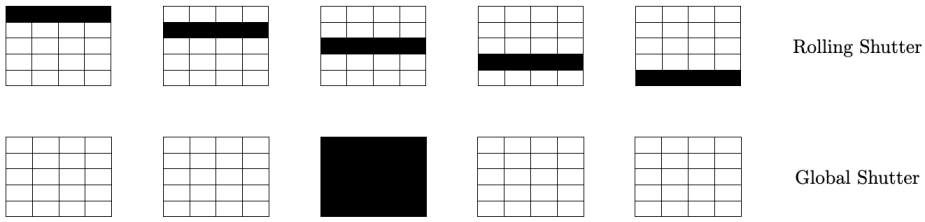


Figure 3.7: Rolling vs Global Shutter Comparison Example

different time series. Although the corner points belong to different time series, the pose calculation treats them as one time step, leading to noisy and unreliable data. The phenomenon relates to motion blur, something that global shutter cameras do not experience.

The second limitation was the field of view of the camera. The C920 has a diagonal FOV of 78°. At a one metre altitude above ground level, the UAV will only see a 1.4 m x 0.8 m area, which significantly inhibits the length of UAV trajectories and increases the likelihood of losing the target.

To address the shortcomings of the C920, the Intel RealSense Tracking Camera T265 was adopted. The T265 is designed by Intel specifically for robotic applications, including drones, and is much more than a camera. It includes two fisheye lens sensors, an IMU, and an optimized onboard visual processing unit (VPU). The VPU runs visual simultaneous localization and mapping (VSLAM) algorithms. The results are accessible through an SDK and can be fused directly with the Kalman filter and control algorithms developed. The T265 addresses the two shortcomings of the C920, as it features two global shutter cameras and an undistorted 128° FOV (a 64% increase over the C920). The T265 also has dedicated mounting points and natively records in grayscale, reducing

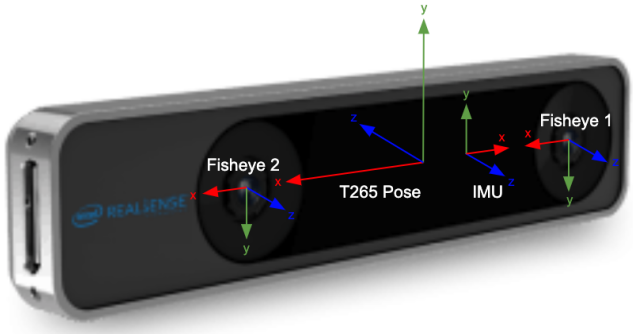


Figure 3.8: T265 with Labeled Coordinate Frames [143]

the need for custom mounting solutions or a colour conversion processing step. The camera, with its various coordinate frames, can be observed in Figure 3.8.

The one drawback of the T265 camera is that the maximum supported resolution is 848x800. The higher the frame's resolution, the more accurate the pose calculation will be as there are more pixels to analyze. Having two cameras and averaging the pose results allows for an increase in accuracy but requires twice the calculations. With the camera's resolution and the previous analysis for selecting a SBC, approximately 100 pose calculations per second was estimated for one lens. With a broad field of view and running all pose estimation calculations twice, the pose estimation actually ran at approximately 40 pose calculations per second. The computation power required to achieve these rates further validates the selection of the NVIDIA Jetson Nano.

3.1.5 Software Development

The SBCs operating system is the foundation of all the custom-developed software. The system is fundamentally limited by the capabilities of the OS, such as the scheduler and the overall resources consumed by the OS. NVIDIA provides an optimized Linux Ubuntu OS for its hardware, centered around Ubuntu 18.04.5 LTS. Linux is the preferred OS for real-time robotic applications because it commonly supports ARM and x86 based processors, has low latency, and the overall system reliability [144].

With an optimized and reliable foundation selected, a programming language was chosen. Many languages are used in real-time robotic applications, such as MATLAB, C/C++, Python, and Java. To select one language over another, a balance of performance and ease of development was considered. For example, C/C++ is a low-level language requiring compiling and careful memory management when compared to Python. The benefit of the increased complexity is an increase in performance and control over the developmental process. Ultimately, Python was selected, as initial testing suggested Python had the best ratio of performance to development complexity. Additionally, many libraries (such as OpenCV) use a wrapper around the C/C++ code, which supports simplified development without sacrificing performance. Python 3.6.9 was used because it was natively installed with NVIDIA's operating system.

Python requires importing packages, which contain classes consisting of functions and variables before they are used. A majority of the packages used in the developed code came with the standard install of Python. Non-standard packages were installed with pip, a package installer for Python. Lastly, some packages do not have a compatible precompiled binary. To install such a package, the package had to be compiled directly from the source code. Some packages call on other packages; however, a list of the directly used packages and their version numbers can be found in Table 3.2.

Table 3.2: Used Python Packages (*Non Standard Package, **Compiled from Source Code)

Package	Version	Package	Version
datetime	3.6.9	pymavlink*	2.4.9
DroneKit*	2.9.2	pyrealsense2**	2.36.0
math	3.6.9	pySerial*	3.4
Matplotlib*	3.3.1	simdkalman*	1.0.1
multiprocessing	3.6.9	statistics	3.6.9
NumPy*	1.18.4	struct	3.6.9
openCV**	4.3.0	threading	3.6.9
pandas*	1.1.0	time	3.6.9

The datetime package was used for recording timestamps when logging data. DroneKit is a dedicated Python API for interfacing with MAVLink based autopilots. Math provides access to mathematical functions, and Matplotlib is a plotting library. Multiprocessing is used to spawn processes. NumPy handles multi-dimensional arrays, matrices, and the mathematical functions associated with these forms. OpenCV is a computer vision library, and pandas is used for data manipulation and analysis. Pymavlink handles the streaming of MAVLink data streams. Pyrealsense2 is a Python wrapper for accessing the T265 camera's data stream. PySerial allows for interfacing with serial ports, and simdkalman is an optimized Kalman filter library. Statistics provides statistic-specific functions not included in the math module. Struct is used for dealing with bytes in the serial port. Threading is used to spawn threads, and time is used for tracking elapsed time and executing delays.

The decision to use the previously described packages and write custom scripts was made over the use of Robot Operating System (ROS) for three main reasons. Firstly, previous work had been conducted with OpenCV's ArUco library and DroneKit. The packages were functioning as expected, and switching to a ROS implementation did not provide any significant advantage. Secondly, running all software on an SBC, the available computational resources were known to be limited. Creating custom scripts may increase development time; however, having full control of development allows for lower latency and more streamlined code. Lastly, there are only three primary data sources; the autopilot, camera, and quick connect system. There are supported ROS nodes for interfacing with most of these primary elements, such as ArduPilot, ArUco, and the T265 camera; however, the system is relatively small and easily managed. The addition of new data sources such as lidar or external IMUs would warrant consideration on how to organize all the data streams.

Python's default is to run all code on a single process and thread, meaning code will run one line at a time. With most code exhibiting blocking tendencies, programs can be significantly slowed when performing computationally intensive tasks such as manipulating large image matrices or

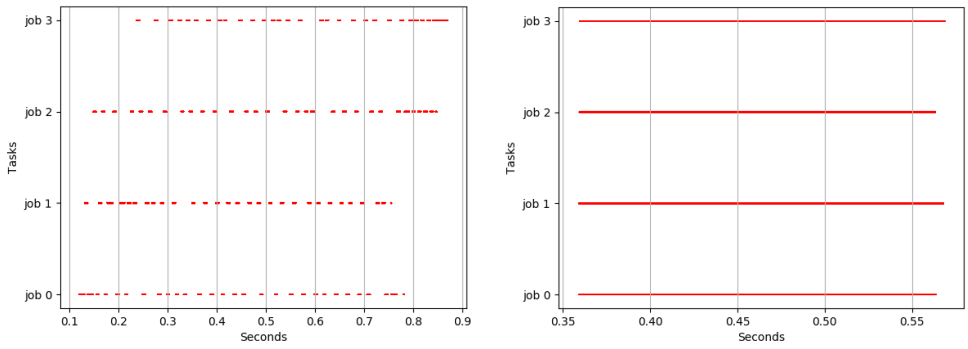


Figure 3.9: (A) Multithreading vs (B) Multiprocessing [145]

waiting for I/O data streams. To address these shortcomings, the multiprocessing and threading packages were utilized. Multiprocessing allocates processes to a physical computational core and a separate memory heap, while threads run on cores in the same unique memory heap and are virtual. Typically, multiprocessing is used for CPU-bound tasks, as an entire core can be dedicated to a calculation. In contrast, threading is used for I/O bound tasks requiring near-simultaneous execution. One exception with Python is that threads are not truly concurrent. Python makes use of a global interpreter lock (GIL), which synchronizes the execution of threads so that only one thread will execute at a time, thereby ensuring memory is managed safely.

To compare the difference between Python's multiprocessing and threading packages, a simulation of a computationally intensive task was executed using both process and threads. The results of the simulation are found in Figure 3.9. The lines show when a task was being executed, with gaps showing inactivity. Due to GIL, it can be observed that threads indeed do not run concurrently, and to complete the four tasks, it took multiprocessing approximately 44% less time than threads for the given simulation. The processing package is not four times faster than the threading library, as the relationship is not linear. To share data between processes and memory heaps, a queue is used. The passing of information in queues adds computational overhead. Additionally, not every clock cycle is used when executing a computation.

The Jetson Nano uses a quadcore ARM A57 processor. With a quadcore processor, the processor can spawn at most four processes. Threads can be spawned until the SBC runs out of memory (at 4 GB) or until the tasks start to bottleneck and affect performance. It was found that spawning two processes, each with four threads, was an optimal configuration through profiling various configurations. The structure of the main program is displayed in Figure 3.10. The OS natively runs across all four cores, is minimal due to its optimized nature, and is only accessed through the command

line, mitigating the need for a GUI. The vision core is responsible for all of the pose estimation and spawns a thread for reading image and sensor data from the T265 camera. Two additional threads are used to run the pose estimation algorithms for the two cameras. Running each of these calculations on its own process would speed up results; however, passing large image matrices in a queue was slower than using two threads. The main thread filters the data with a Kalman filter and sends the results to the I/O and logging core. The I/O and logging core main thread runs the PID control algorithms and logs data acquired by the other threads. The remaining threads acquire and send data such as the acquisition of values from the vision core, communicating with the autopilot and communicating with the quick connect and its respective payload.

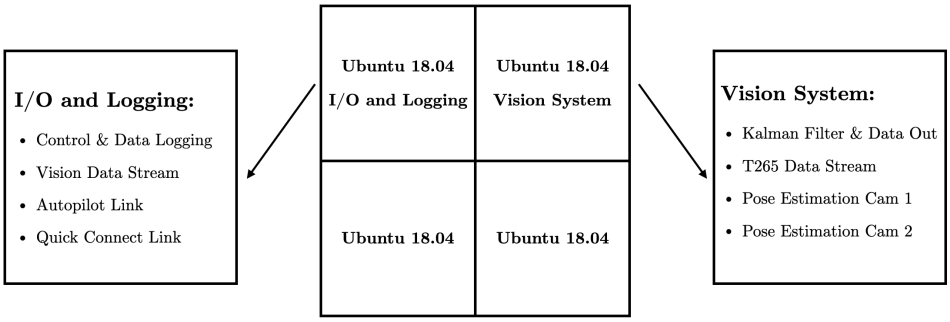


Figure 3.10: Processor and Thread Allocation

3.2 Universal Payload Adapter: The Quick Connect

The quick connect system's primary goal is to allow the UAV to interface with any payload, turning a niche system into an extremely versatile one. To interface with various payloads, the system must mechanically transmit forces such as torque and withstand dynamic loading. Additionally, most payloads require electrical power. Providing a power source can reduce the complexity and mass of developed payloads, thereby eliminating batteries and the requirements for step down or boost converters. The final requirement for a payload adapter is a communication protocol; the payload needs to be synced with the central control and state estimation algorithms so it knows when to actuate or begin logging data.

Performing a market study of currently existing technologies, ATI Industrial Automation was found as an industry leader with more than 30 years of developing state-of-the-art end-effector products and solutions [105]. ATI developed the ATI Robotic Tool Changer, which allows multi DOF robotic arms to automatically change end-effectors and other peripheral tooling. The Robotic Tool Changer has a range of models capable of lifting devices from 1.4 to 4000 kg in a range of dynamic loading scenarios, including high heat and vibrations.

The ATI Robotic Tool Changer is pneumatically actuated and comprised of two main parts. The first part is a male end effector that is fixed on some multi-degree of freedom arm. The second part is a corresponding female adapter, fixed to some tool. Aligning the male and female parts, a piston-cylinder in the male adapter is pneumatically actuated, which pushes ball bearings into a race on the female adapter, a common approach in tool changers [106]. The race is grooved to provide a secure lock due to the pneumatic actuation. Sensors regulate the piston to ensure that pressure is held, and rotational torque is transmitted through multiple dowel and pin connections previously aligned by the actuation device. To release the quick connect system, the piston's pressure is bled off, and the actuation arm can remove the male end effector and select the next payload of interest.

Due to the system's pneumatic nature, at least 550 kPa is required for a sufficient locking force. Having a compressor on the UAV capable of maintaining the required pressure and locking force would increase the overall system weight, power requirements, and take up significant space. Additionally, the cost of an industrial-grade tool changer and all the required add-ons would quickly exceed the entire UAV system's price.

To address the limitations of not using an industrial solution, a custom modified design of the Robotic Tool Changer was developed. The pneumatic piston-cylinder was swapped for a digital servo, and eight quarter-inch precision ball bearings distribute the load when engaged in the female race. Four bolts act as keyways for transmitting torque and double as contact points for transmitting 5 V and up to 3 A of electrical power to a payload. The main housings are 3D printed with PLA filament and are mounted to fit a pre-existing 64 mm diameter bolt circle in the UAV frame. The ball bearings are held in place with a stepped hole, and the modified piston is chamfered to prevent the balls from falling out in the opposing direction. A custom wireless communication protocol is established using a transceiver. The system is further described in detail in the following sections.

3.2.1 Mechanical Design

The quick connect is broken into three parts; the actuator, male quick connect, and female quick connect. Exploded renders of the three parts are found in Figure 3.11.

The actuator, as seen in Figure 3.11-A, is centered around a PLA 3D printed base. A digital Hitec HS-5645MG ultra torque servo with a standard servo horn is bolted to the 3D printed base. A slot in the base allows for the servo horn's actuation to either apply or release pressure on the piston system.

The male quick connect, as seen in Figure 3.11-B, also features a 3D printed base with a slot for the servo horn to pass through. Eight 6.35 mm chrome steel ball bearings sit in stepped holes, which are snapped into place during assembly. A chamfered plunger sits in the center (to prevent the ball

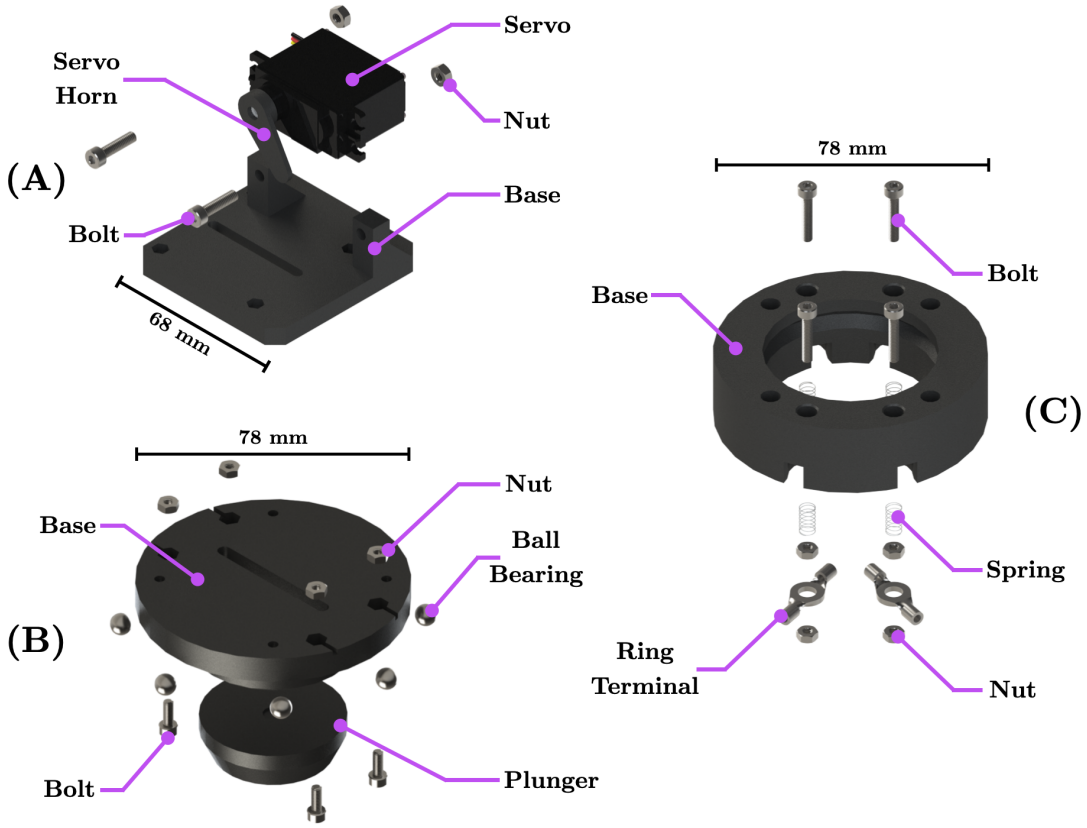


Figure 3.11: Quick Connect Sub Assemblies Exploded: (A) Actuator, (B) Male Quick Connect, and (C) Female Quick Connect

bearings from falling out the center of the quick connect) and is actuated by the servo. Four M3 stainless steel bolts have exposed heads and are responsible for transferring torque and providing power. The nut on the opposing end is soldered and epoxied into place and connected to a voltage regulator.

The female quick connect, as seen in Figure 3.11-C, 3D printed base is a negative of the male quick connect. It has a small tolerance for the male quick connect to align and has a recess for the ball bearings to sit in when actuated. To transfer electrical energy from the UAV to a payload, the contacts are placed on springs to ensure there is always pressure being applied to the electrical contact, but not so much pressure that the male and female quick connect cannot stay coupled. The contact then uses ring terminals crimped to wires to be fed to the payload.

The entire quick connect assembly made use of constant diameters, and mounting bolt patterns between subassemblies were possible. For example, the actuator and male quick connect bolt

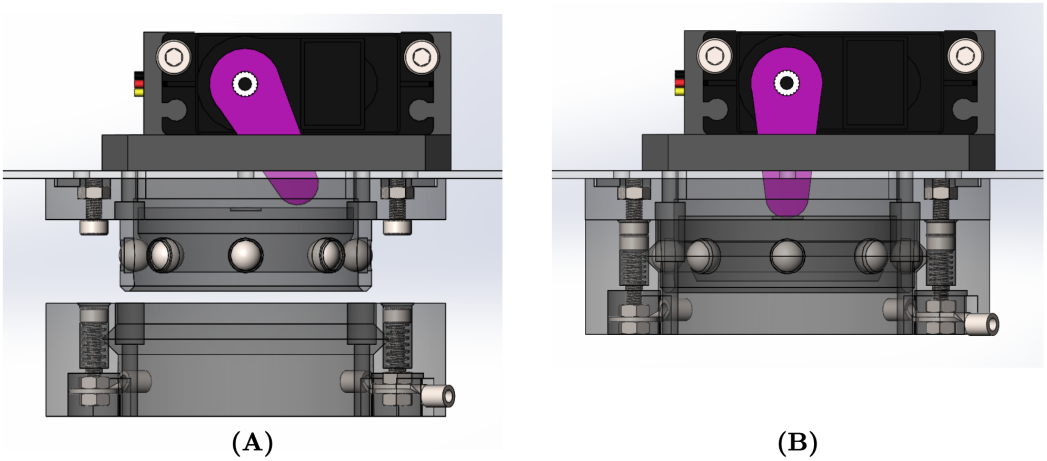


Figure 3.12: Quick Connect States: (A) Disengaged State, (B) Engaged State

through the UAV frame's bottom sharing the same bolt circle, which allows for uniform alignment. Additionally, when the male and female quick connect are engaged, they appear as one part. All parts were made with chamfers to aid in alignment, and specific tolerances were achieved with precision settings on a Prusa MK3 printer and a combination of trial and error.

Figure 3.12 shows the system in both a disengaged and engaged state. In a disengaged state, the servo arm does not pressure the plunger in the male quick connect. With the plunger not engaged, the ball bearings fall into the quick connect center, allowing the female quick connect to be mated. Additionally, the springs in the female quick connect push the contacts to their maximum height. In an engaged state, the servo arm is actuated to be parallel with the bolts in the vertical direction. Engaging the servo pushes down the plunger and forces the ball bearings into the bearing race on the female quick connect. Simultaneously, the female quick connects springs are compressed, making room for the four bolts to transmit rotational torque while also establishing an electrical contact.

To confirm that the male and female quick connects will stay engaged, and to provide an initial estimate of the withstandable forces, a free body diagram (FBD) was drawn as in Figure 3.13. The FBD is used to solve what servo input force F is required to balance the weight L of a payload. The bearing race's angle is ϕ, while the angle of the plunger is θ. The force of friction F_f is a function of the normal force F_N or L_N and the static coefficient of friction μ_s between the PLA parts and ball bearings. The normal force acting on the ball bearing in the stepped hole is denoted as N. It is assumed that all forces are evenly distributed across all eight bearings, the system is a static non-deformable body, and the force from the springs in the electrical contacts are negligible. To

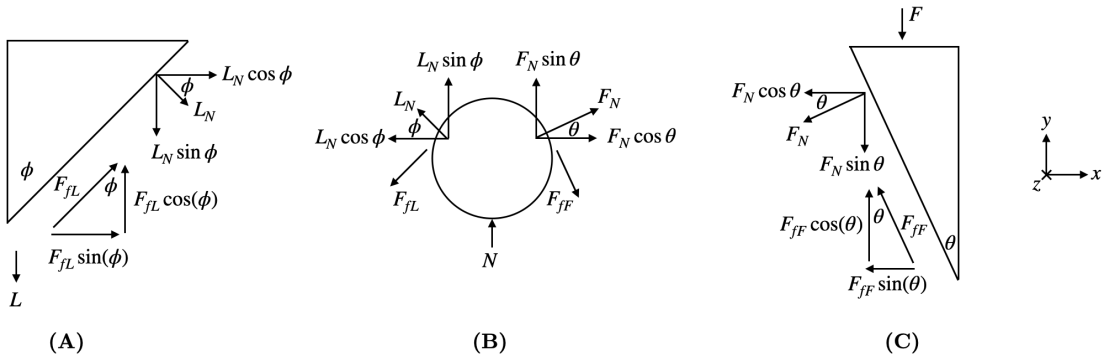


Figure 3.13: Quick Connect Free Body Diagram

solve for F in terms of L, the forces in the vertical direction must first be summed using the FBD in Figure 3.13-A and 3.13-C to yield Equations 3.1 and 3.2, respectively.

$$+\uparrow \sum = -L - L_N sin\phi + \mu_s L_N cos\phi = 0 \tag{3.1}$$

$$+\uparrow \sum = -F - F_N sin\theta + \mu_s F_N cos\theta = 0 \tag{3.2}$$

Summing forces from Figure 3.13-B in the horizontal direction yields Equation 3.3.

$$\xrightarrow{+} \sum = -L_N cos\phi - \mu_s L_N sin\phi + F_N cos\theta + \mu_s F_N sin\theta = 0 \tag{3.3}$$

Using Equations 3.1, 3.2, and 3.3, the three equations can be combined and simplified to yield Equation 3.4, the minimum required input force for the weight of some payload.

$$F = L\frac{(cos\phi + \mu_s sin\phi)(\mu_s cos\theta - sin\theta)}{(cos\theta + \mu_s sin\theta)(\mu_s cos\phi - sin\phi)} \tag{3.4}$$

Using Equation 3.4, variables can be substituted for their actual values. The angle of the plunger, θ, was set to 25°, and the bearing race angle, ϕ, was set to 45°. The angles were selected using optimal 3D printing performance parameters and by getting the system to fit together physically. A static coefficient of friction value of 0.2 was selected [146], and the result of the complete substitution is found in Equation 3.5.

$$F = 0.365L \tag{3.5}$$

For the given geometry and estimated friction, the input force must be approximately 37% that of the payload weight. An upper theoretical limit of payload mass can be calculated using the digital HS-5645MG ultra torque servo specifications and the length of the servo arm, which yields a maximum static payload of 10 kg. In reality, the quick connect was found to hold a maximum of 2.5 kg. The non-deformable body assumption broke down, and parts began to flex with increased loading, which reduced contact area and resulted in the male and female subassemblies detaching. Given the UAV's weight and the available thrust, it is highly unlikely to maximize the quick connects' capacity. Therefore, to ensure the UAV does not require more than 70% throttle to hover, payloads should not exceed 0.65 kg. This mass restriction also provides a safety factor should a payload have non-rigid elements, leading to a dynamic system.

3.2.2 Electrical Design

The quick connect electrical design consists of two main parts; a means of transferring power from the battery to a payload, and the hardware for communicating and engaging payloads. A system diagram for the electrical layout and integration with the remainder of the UAV can be viewed in Figure 3.14. Electrical energy is acquired from the battery and stepped down using a buck converter. The regulated voltage is used to power a given payload, the quick connect servo, and microcontroller. To communicate with the central control algorithms and the autopilot, a serial port is established between the SBC and microcontroller.

The electrical quick connect system is centered around a custom two-layer 45 mm x 90 mm printed circuit board (PCB). The PCB was designed to fit at the rear of the UAV, which was previously wasted space because of the long and narrow geometry. The PCB places all the required components and connectors onto one board that easily fastens to two mounting points on the UAV frame. The assembled PCB and a screenshot of the board layout can be viewed in Figure 3.15, and a table of the components, manufacturer, and part numbers can be found in Table 3.3.

When designing the PCB, a significant focus was placed on having an organized board layout and

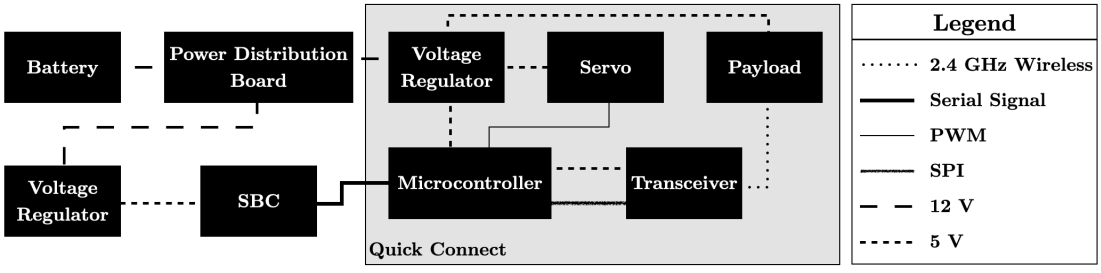


Figure 3.14: Quick Connect Electrical and Signal Diagram

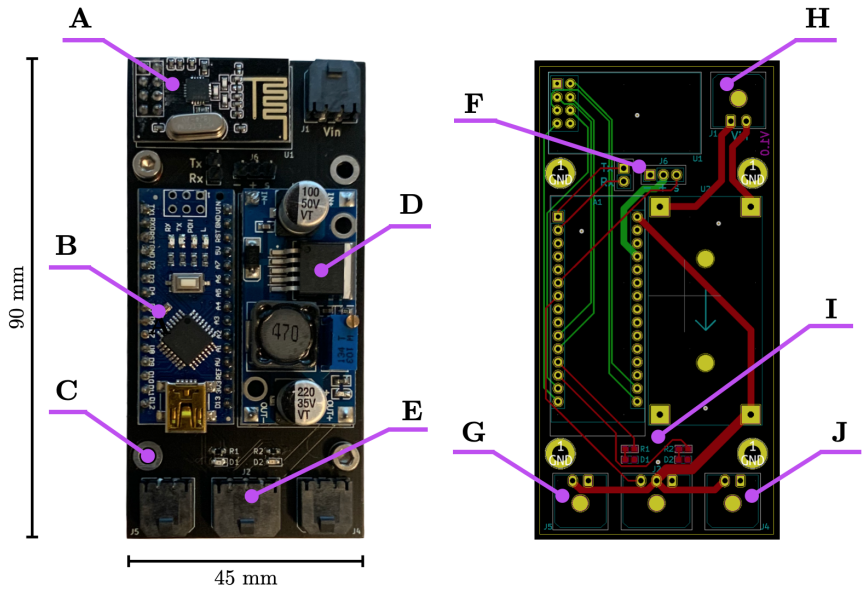


Figure 3.15: Quick Connect PCB: (Left) Assembled Board, (Right) Board Layout with Hidden Fill Zones

Table 3.3: Quick Connect Major PCB Components

ID	Part Name	Manufacturer	Manufacturer Part Number
A	Transceiver	Kuman	nRF24L01 + 2.4GHz Antenna
B	Microcontroller	Arduino	Arduino Nano v3.0 (A000005)
C	Custom PCB	JLCPCB	-
D	Buck Converter	NOUVCOO	LM2596S (NZ01)
E	Quick Connect Servo Connector	Molex	0436500300
F	Redundant Pin Access	-	-
G	Quick Connect Power Out I	Molex	0436500200
H	Power In (12 V)	Molex	0436500200
I	Indicator LEDs	-	-
J	Quick Connect Power Out II	Molex	0436500200

proper trace routing. For example, 90-degree traces were eliminated, and vias were minimized, especially when dealing with high-frequency lines such as the serial peripheral interface (SPI) used to communicate between the microcontroller and transceiver. Traces were sized according to their current draw to keep temperatures below 40°C during peak operating conditions. Additionally, a copper ground plane was used for overall thermal regulation and to provide a common ground. Lastly, the entire system and its components went through initial validation using protoboards to ensure all parts were compatible and that no additional capacitors or resistors would be required

during commissioning.

Interfacing components of the electrical system were also carefully selected. Molex Micro-Fit 3.0 connectors were used in place of standard servo connectors, as they have locking tabs, are polarized to prevent accidental mis-mating, and have reduced resistance in contacts, thus creating a more efficient system. Double-redundant power connections were made where possible in case one line failed, and parts were soldered, crimped, and/or epoxied to prevent failure due to vibrations common among flying multirotor UAVs. The final interfacing optimization is the double-redundant polarized torque and power transfer system. The contacts accepting the positive and negative power are diagonal from each other. Therefore, regardless of the UAVs orientation, the chance of a short circuit is extremely low, and there is no need for reverse polarity protection circuits. Additionally, only one positive and negative contact needs to be made to facilitate power transfer.

An Arduino Nano microcontroller was selected as the main processor because it has a small footprint and is capable of interfacing with various devices and buses. The buck converter was selected for its 3 A current capacity in a small form factor. It has an efficiency greater than 90%, making it much more appealing than linear regulators. Additionally, the LM2596S regulator has a variable output voltage, making it compatible with a variety of payloads. Although not directly part of the PCB, a digital HS-5645MG ultra torque servo was selected as the primary actuator. The servo was selected because it is digital, and thus will provide a faster response and be less susceptible to disturbances than its analog equivalent. Additionally, the servo provides 1 Nm of torque with approximately the same footprint and mass of lower-torque servos.

The decision to communicate with payloads wirelessly was made primarily from a mechanical perspective. Having to align additional contacts for transmitting and receiving lines would increase system complexity and decrease reliability. Additionally, should the contacts' resistance be too high or non-uniform, data could become corrupted and unusable. Specialized robotic mating connectors such as the ones used in the ATI system are a potential solution; however, parts were found to be expensive or custom-fabricated. Because the payload and UAV are close to each other, little consideration was required for antenna sizing or power loss calculations. Bluetooth modules such as the HC-05, Wi-Fi enabled boards such as the ESP32, and radio transceiver chips such as the nRF24L01 were all considered for establishing wireless communication. Bluetooth was found to be unreliable, and the Wi-Fi boards were more than five times the cost of an nRF24L01 transceiver and three times the cost of an HC-05. The HC-05 Bluetooth was unreliable because paired devices would often disconnect and would not reestablish connection without cycling power. With initial prototyping, the transceiver chips were reliable and selected for the final design.

3.2.3 Software Design

The software design of the quick connect establishes a wired link between the main program running on the SBC and the quick connect microcontroller. A wireless 2.4 GHz radio connection is then established between the microcontroller and a given payload. Once all the links are successfully established, a finite-state machine (FSM) facilitates payload and UAV actions.

The microcontroller requires a low-level language to be controlled; therefore, C/C++ was used for the developmental process. On the microcontroller, five libraries were used. The Arduino library provided access to the processor and its respective pinout. The Servo library was used to control the servo and generate the required modulated pulses, and the SPI library was used to communicate with the transceiver. The first three libraries are standard Arduino libraries; however, to use the nRF24L01, two open-source libraries were used [147, 148]. The libraries handle establishing communication between radios as well as encoding and decoding data with proper checksums and error handling. A flow chart of the developed system is provided in Figure 3.16.

The quick connect code begins by starting a serial port, starting the radio, and initializing the servo to be in a disengaged state. The program then runs indefinitely and checks the serial port for a state

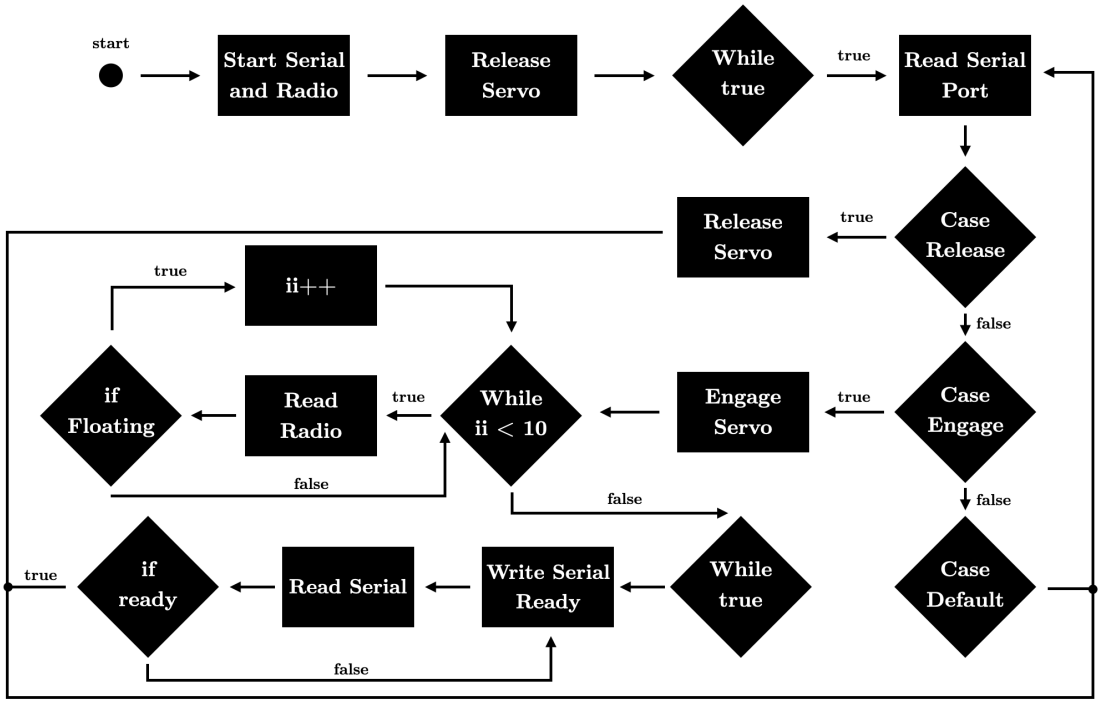


Figure 3.16: Quick Connect Microcontroller Program Flow Chart

provided by the SBC. Based on the acquired input state, a switch statement executes the correct command. Should the case be to release the quick connect, the servo is given a pulse to move to a disengaged state. Should the case be to engage a payload, the servo is engaged. A counter is initiated and will only increment should the payload make electrical contact and the two radios begin communicating. Once ten successful data packets have been received, the microcontroller notifies the SBC the payload is engaged and ready for use. The SBC returns the handshake, and a mission can begin. Should a connection not be established, the SBC can send a termination command through the serial port forcing the code to return to the main loop. Should an unknown state be received, the code defaults to clear both the radio and serial port buffer and proceeds to monitor the serial port for the next command. Commands are encoded as single hexadecimal bytes because they have low overhead and do not require unpacking for interpretation.

Opposed to the microcontroller code, which runs one line at a time, the main code on the SBC is running multiple processes and threads. Therefore, the SBC does not always maintain communication with the quick connect; it only looks to communicate when required, such as engaging or releasing a payload. The dedicated functions used on the SBC are observable in Figure 3.17. When the main code on the SBC initiates, it ensures that the quick connect is in a released position so that the initial state is always known. Otherwise, the procedure for releasing is to send the release command five times, with a small delay. There is no feedback to guarantee the payload is released; however, experimental testing proved sending the command at least five times provided a nearly

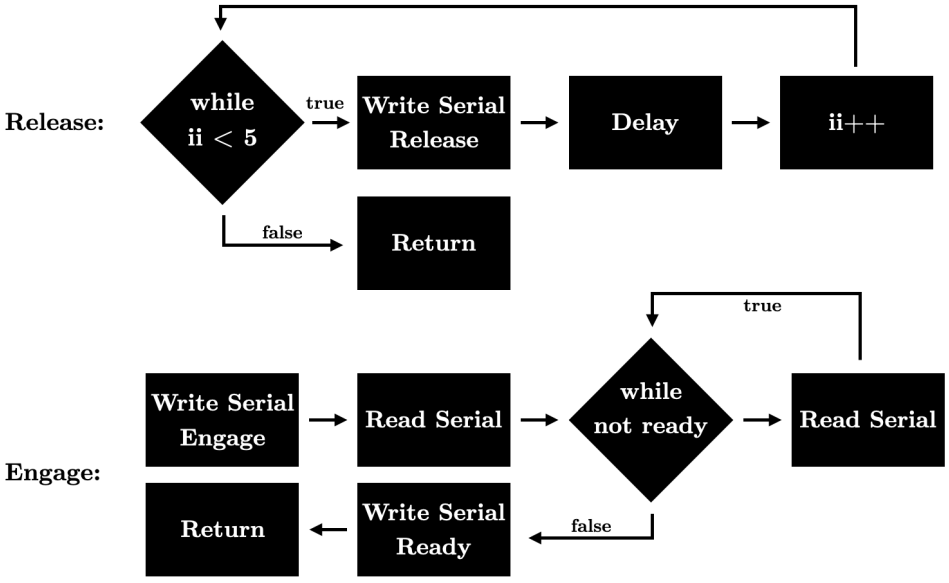


Figure 3.17: Quick Connect SBC Function Flow Charts

perfect success rate. When an engage command is sent, a loop waits for confirmation that the system is ready. Once ready, the SBC completes the handshake with a confirmation of its own. Should a specified time be exceeded, an exit command can be sent to override the system back to its main loop.

Once a payload is engaged, commands can be passed from the main program running on the SBC to the payload, as illustrated in Figure 3.18. The pass-through works by adding an additional switch case statement, which is assigned an ID. Multiple IDs can be assigned, which allows for multiple different payloads to be used. Once initiated, the microcontroller reads the serial port and sends data to the payload if it is useful; otherwise, it skips transmitting and just reads an input. The input is then sent back to the SBC over the serial port. If an exit command is sent, the loop breaks and returns to the main loop. The payload communication protocol can be summarized as a two-way bridge between payloads and the main program.

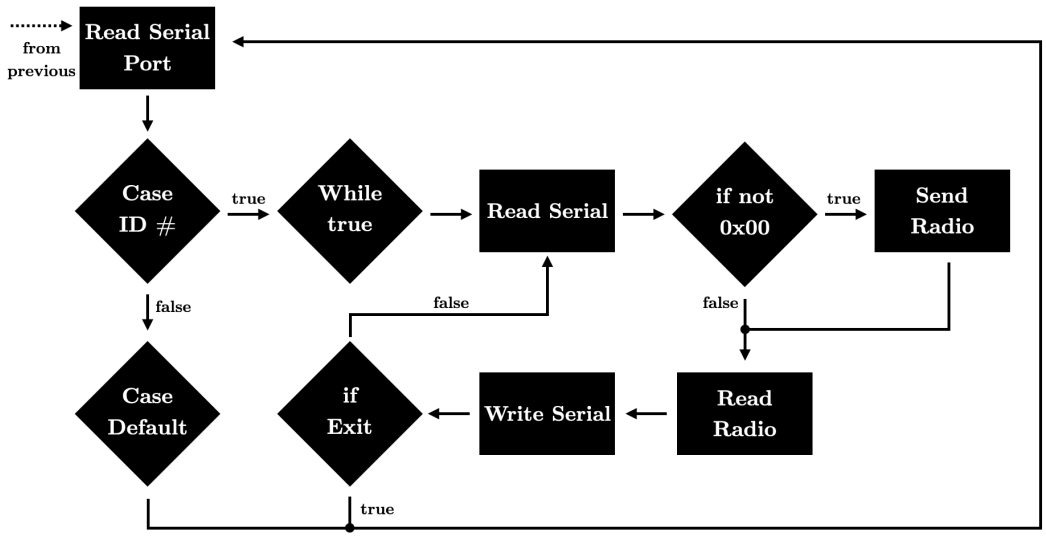


Figure 3.18: Quick Connect Payload Communication Flow Chart

3.2.4 Payload Docking Station

The UAV cannot reliably position itself with millimetre accuracy. However, should the UAV achieve centimetre accuracy, a passive system can align the UAV to achieve millimetre accuracy and allow for the retrieval of a payload using the quick connect system. The passive payload docking station features a broad white background measuring 120 cm x 90 cm to allow for clear detection of ArUco markers. An aluminum extrusion structure holds four 3D printed funnels that allow for inaccuracies of up to 7 cm in the north and east direction and 13° of yaw. The entire system is fastened to the

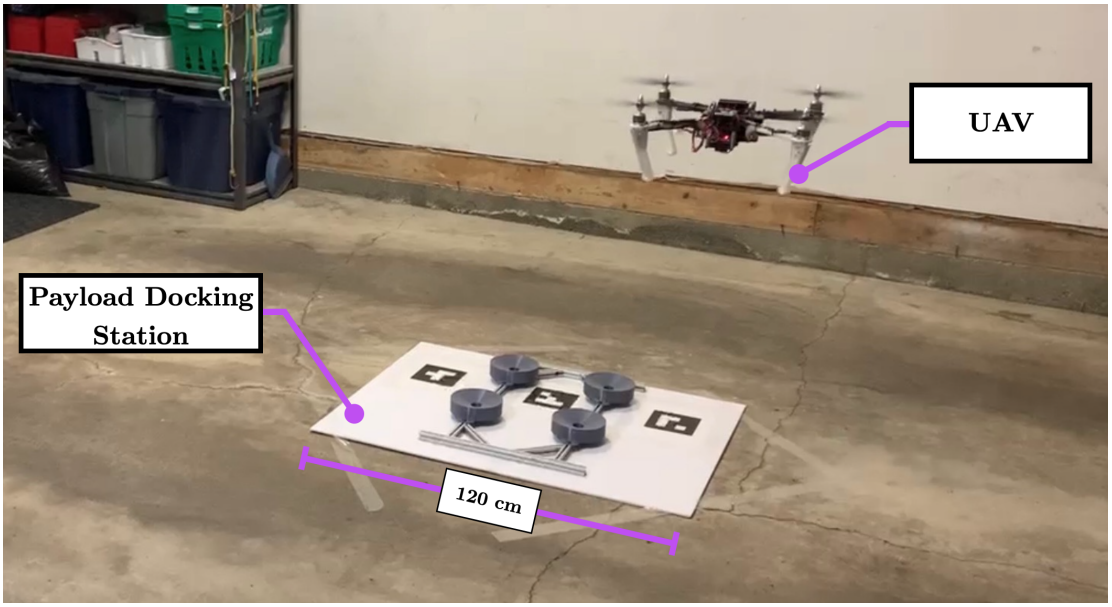


Figure 3.19: UAV Payload Docking Station with Approaching UAV

ground using clear packing tape to limit the effects on the vision system and prevent the system from shifting from rotor wash. A photo of the docking station can be observed in Figure 3.19.

With the UAV system's layout and quick connect development complete, a novel payload is designed in the following chapter to interface and demonstrate the capabilities of the UAV and its universal payload system.

Chapter 4

Payload Development Example

4.1 Conceptual Design

4.1.1 Design Space

With a laboratory focus on machine diagnostics, three primitive designs for sensing elevated temperature trends, vibration spectral features, and retrieving lubricant samples were developed [51, 52]. To demonstrate the quick connect's full capabilities, a single payload was selected for detailed development. A payload for accessing a sump lubricant reservoir was selected, as the scenario is of general interest. Providing a torque with a UAV is applicable to other use cases, such as opening doors, twisting process control knobs, and actuating the handwheel of a valve. Additionally, acquiring a lubricant sample at an access port can be achieved with a contact inspection, which is a reasonably well-researched area [116, 149, 150]. The presented design only focuses on creating an access point to retrieve a sample; the extraction of a sample is left for a future payload.

There is no standard sump cavity; therefore, the payload design must be functional for a variety of configurations. Considering the different configurations of sump cavities, a vertically orientated cap with a right-hand thread appeared to be common. Caps securing the reservoir come in various shapes and sizes, and are dependent on factors such as the amount of lubricant or type of lubricant. Circular caps are the most prevalent; accordingly, undoing nonmagnetic caps with diameters ranging from 25 mm to 100 mm was selected as a primary scenario of interest.

Two main challenges must be addressed in opening and closing a cap. Firstly, the cap must be interfaced with; and secondly, a sufficient controlled torque in the axis of cap rotation must be applied. The following two sections describe the conceptualization for addressing the design problems.

4.1.2 Cap Manipulation

To interface with various diameter caps, the manipulator must be adaptable, while still being able to provide sufficient gripping force to prevent slip when a torque is applied. With the benefits of rapid prototyping, four designs were manufactured (as pictured in Figure 4.1) and empirically tested.

Each of the manipulators was 3D-printed using polylactic acid (PLA) filament on a Prusa MK3 printer. The multiple parts were assembled with standard stainless steel metric hardware ranging from M2 to M4. An elastomeric surface was added to each design's inner contact surfaces to increase the coefficient of friction between the cap and the manipulator. For active devices, a Hitec HS-645MG ultra torque servo converted electrical energy to mechanical energy. The servo was controlled by a PWM signal from a microcontroller and powered by a 5 V regulated supply.

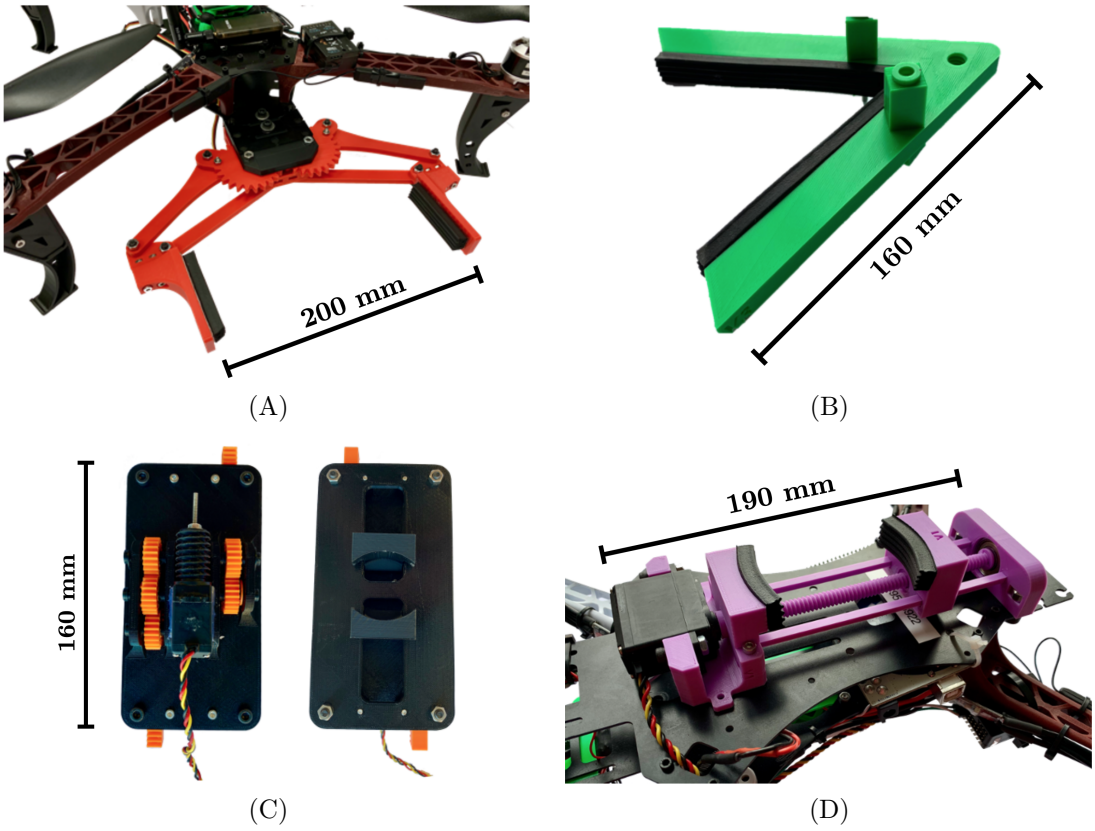


Figure 4.1: Prototyped Cap Manipulators: (A) Adaptive Gripper, (B) Static Gripper, (C) Rack and Pinion Gripper, and (D) Lead Screw Gripper

The first design, shown in the photograph in Figure 4.1-A, is an adaptive gripper based on the Robotiq 2F-140 [151]. Originally designed for a variety of picking and placing applications, the ability to clamp a cap appeared as another adoptable use case for the manipulator. Using the torque generated by the servo, two gears which are fixed to linkages that actuate the jaws are rotated. The gripper actuation range spans from 0 mm to 200 mm, and has sufficient gripping force to grasp a rigid smooth plastic container weighing 1 kg even in dynamic loading scenarios.

The second design, shown in Figure 4.1-B, is a passive device requiring no actuation to achieve a gripping force. The design was selected due to its overall simplicity, and works by wedging the cap in the jaws of the gripper. Due to the system's geometry, the cap auto-aligns itself at the center of the manipulator. The elastomer surface slightly deforms to further increase the cohesion between the cap and the gripper. A range of gripper jaw angles was explored empirically; 60° was found to provide the best results for caps in the 25 mm to 100 mm diameter range.

The third design, shown in Figure 4.1-C, actuates a bottom travelling set of jaws fixed to a rack and pinion. The rack and pinion is driven by a gear train made up of a bidirectional continuous rotation servo and worm gear assembly. The worm gear drives a worm wheel fixed to two sets of gears. Each gear set then uses idler gears to alter the rotation direction and make contact with the rack to transmit the rotational energy into translational energy. Due to the servo's high torque and the massive reduction from the worm gear, large clamping forces were achieved. The system can either open, close, or stay stationary by varying the PWM signal to the servo.

The final design, shown in Figure 4.1-D, shares many similarities to the rack and pinion design; however, in place of the rack and pinion, a lead screw is implemented. One jaw is held fixed while the other translates longitudinally as it is fixed to a lead nut. A thrust bearing counteracts the axial forces and keeps the lead screw secured, while two guide rails keep the system aligned. Similar to the rack and pinion design, the system also opens, closes, and remains stationary by varying the input signal to the servo.

Each design was an early conceptual model and could be optimized; therefore, the four designs were compared qualitatively. The static gripper, Figure 4.1-B, was an optimal design on paper; however, once the cap was released from the reservoir's threads, it would only stay in the jaws approximately 25% of the time. Dropping the cap raised the question as to whether a secondary mechanism would be required to hold the cap once removed, mitigating the simplicity of the design. The adaptive gripper, Figure 4.1-A, was highly functional and reliable; however, it lacked gripping strength compared to the rack and pinion or lead screw design. When comparing the rack and pinion, Figure 4.1-C, and lead screw, Figure 4.1-D, the lead screw had fewer moving parts, a greater gripping force, and faster actuation time. Ultimately, the lead screw design was selected for further development, as it was the most reliable and yielded the best performance.

4.1.3 Torque Generation

With the selection of the lead screw cap manipulator, a method for applying a torque to the cap had to be considered. Fixing the UAV to the surrounding environment (i.e., a steel gearbox with electromagnets) simplifies the problem in terms of controllability. The problem is simplified because the reaction forces are mechanically constrained, and the focus can be placed on the mechanical apparatus generating the torque rather than designing a controller. Alternatively, the UAV can stay airborne and fix itself to the cap stem's throat, and a secondary device can be used to introduce torque to the cap. Both methods and any derivatives can produce large torques and are scalable through modifications in mechanical ratios. The constraints with such devices are the many assumptions made of the surrounding environment, leading to limitations in the system's autonomous capabilities. Additionally, the devices may require multiple actuators and systems, leading to a potentially large and heavy apparatus outside the specifications of the current UAV and quick connect system.

A methodology to counter fixing the UAV to the surrounding environment was to consider a design space that uses the four motors already on the quadrotor UAV. The motors' use allows for the actuators, or group of actuators, to achieve multiple functions: flight and torque generation. Although there is a finite amount of torque available, initial estimates projected that the torque would be sufficient for removing or replacing a cap. Additionally, the design space does not limit the autonomous capabilities of the payload. For the concept of removing a cap and introducing a torque with the UAV, it was assumed that the UAV could rigidly fix itself to a cap with the lead screw manipulator.

Fixed-arm multicopters, such as the quadcopter platform, achieve lift by rotating propellers to generate thrust and lift. The direction of rotation of a propeller is opposite to that of its neighbour and

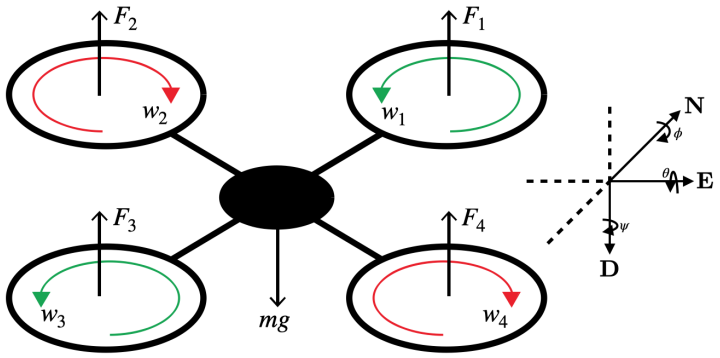


Figure 4.2: Free Body Diagram of a Quadcopter While Hovering

is the same as its adjacent counterpart, as depicted in Figure 4.2. The staggering of the propeller's directions is due to the propeller's rotation, creating an opposing torque. The staggering of the propeller rotations creates a system that is in a net neutral state in terms of angular momentum. Therefore, to unbalance the system and maintain level flight, two non-adjacent propellers must spin faster than the other pair of non-adjacent propeller pairs. This difference in the rotation will create a net aerodynamic torque and cause the UAV to rotate in the yaw direction. Using the free body diagram from Figure 4.2, clockwise yaw is achieved in Equation 4.1, and counterclockwise yaw is achieved in Equation 4.2, where F_i is the thrust produced by motor i, and is approximately proportional to the torque generated by a fixed pitch propeller. The relationship is due to the propellers cutting through the air to create thrust and the corresponding opposing torque; the faster the motor spins, the greater the thrust, and the greater the opposing torque.

$$F_1 + F_3 > F_2 + F_4 \tag{4.1}$$

$$F_1 + F_3 < F_2 + F_4 \tag{4.2}$$

The larger the difference between sets of adjacent propellers, the faster the UAV will yaw, and the greater the torque. However, the system must maintain enough thrust to keep its altitude as in Equation 4.3, where F_i remains the thrust produced by some motor i, and mg is the weight of the UAV. Additional thrust may be required due to disturbances such as wind; however, in an ideal scenario, a quadcopter will ascend if the summation of forces is greater than the quadcopter's weight, and it will descend if the summed forces are less than the weight of the quadcopter.

$$+\uparrow \sum_{4}^{i} F_i = mg \tag{4.3}$$

Should the cap manipulator provide sufficient gripping strength to hold the UAV without aid from a thrust vector, the maximum ideal torque is achieved when two sets of adjacent motors are operating at 100% while the other pair remains off. Although ideal, only half of the available motors are used to generate a torque as the remaining rotors spin the wrong way and would require reconfiguration to aid in torque generation.

To take advantage of all four actuators, a rotation of the thrust vector was conceptualized. By having the UAV execute a rolling or pitching motion while secured to a cap by a moment arm, the UAV can exhibit a torque to the cap. For example, rolling the UAV will transition the vertical thrust vector to include a horizontal component at an angle ϕ as pictured in Figure 4.3.

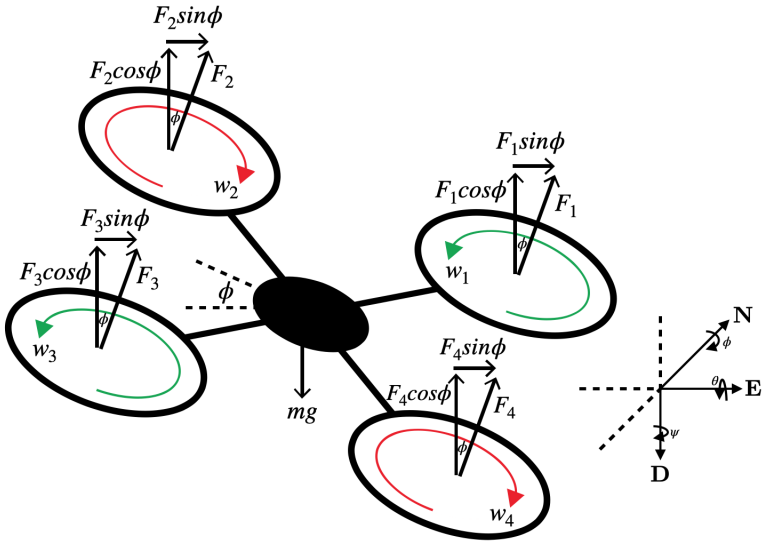


Figure 4.3: Quadcopter Executing Positive Roll ϕ for a Counterclockwise Torque

The maximum counterclockwise torque occurs when ϕ is equal to 90°; however, the UAV should still maintain a vertical thrust vector to sustain flight and minimize the load on the moment arm. The minimal lifting force required is established in Equation 4.4, where F_{max_i} is the maximum thrust produced by motor i, ϕ is the roll angle, and mg is the weight of the UAV when there is no acceleration.

$$+\uparrow \sum_{4}^{i} F_{max_i} cos(\phi) = mg \tag{4.4}$$

Using the angle ϕ, which yields the minimum required lifting force from Equation 4.4, provides the maximum horizontal force P when substituted into Equation 4.5.

$$P \overset{+}{=\!\!\rightarrow} \sum_{4}^{i} F_{max_i} sin(\phi) \tag{4.5}$$

Using the force P with the length of a moment arm r, the torque τ generated to remove the cap can be calculated as in Equation 4.6.

$$\tau = r \times P \tag{4.6}$$

Assuming quasistatic conditions, the generated torque is scalable in this configuration because many parameters can be changed, such as the minimal hovering force. If some of the vertical load is taken by the moment arm, then a larger force P can be generated as the angle ϕ increases. Additionally, as the length of the moment arm is increased, the torque also increases. The limitation with this method is the device for applying the torque. Even if it were infinitely strong, the physical environment in which the UAV could operate would be finite. The space surrounding the cap must be free of any obstacles whose radius is equal to at least the length of the moment arm plus the length of the UAV.

A third method is being able to independently rotate each rotor arm with respect to the UAV body, as observed in Figure 4.4, which turns an under-actuated quadcopter into an over-actuated machine. Each motor arm is capable of rotating with respect to the body frame B of the UAV while the body frame moves relative to a fixed ground frame G. Because the selected UAV platform has fixed arms and a single attachment point for payloads, the method cannot be easily implemented; however, over-actuated systems have been numerically and experimentally validated [152, 153]. The benefit of controlling each rotor independently allows for all four motors to produce torque (as in the second proposed method) while still having a small actuation footprint (as in the first method). The over-actuated system yields promising results, as it could theoretically sit on the cap and convert all of the thrust power into torque, thereby yielding a high-density torquing footprint.

Considering the conceptualized models, achieving torque through yawing the UAV was selected as

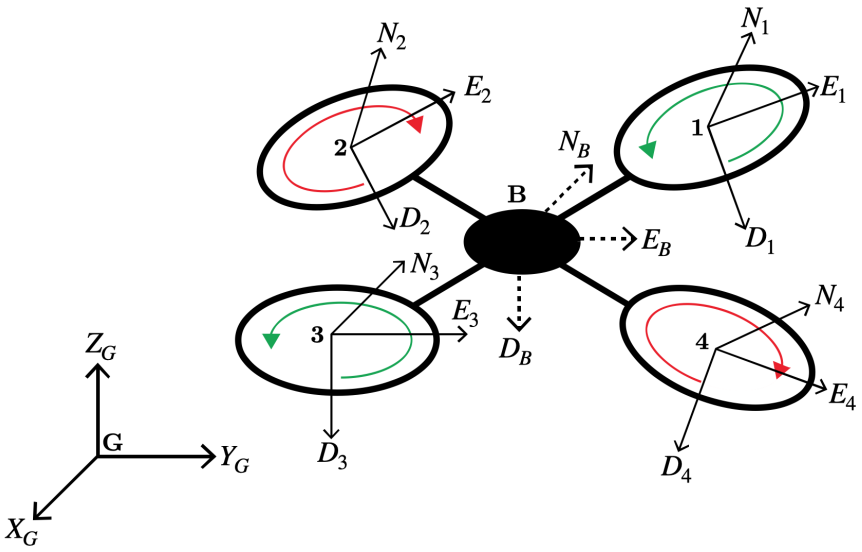


Figure 4.4: Over-Actuated Quadcopter Concept

the focus for future development. Fixing the UAV to its environment and having a secondary device apply a torque was eliminated from consideration; that approach makes too many assumptions about the surrounding environment, reducing the number of applicable use cases. Rotating the thrust vector would allow for large torques to be generated; however, a large actuation area is required, and the system also makes assumptions about its operating environment. The over-actuated UAV design was eliminated from consideration because it requires a full UAV redesign and does not focus on payload development. Ultimately, using the UAV's yaw was selected because it was the most general solution for multiple environments and was validated in other use cases, such as actuating a valve [36, 154] and unscrewing a light bulb [35].

4.2 Final Design

The following four sections provide an overview of the cap manipulator's final design and quantification of the available torque the UAV can produce from executing a yaw maneuver. The first three sections focus on the design of the cap manipulator. The mechanical section outlines a general overview of the system, while the electrical section highlights specific component selection and the respective signals and data streams processed. The software section brings together the mechanical and electrical sections and explains how the system functions. Finally, torque data is gathered from the UAV to establish a payload specification.

4.2.1 Mechanical Design

The final mechanical design of the cap manipulator with labelled components can be observed in Figures 4.5 and 4.6. The design features the custom-designed female quick connect used for interfacing with the UAV's universal payload system. A custom PCB communicates with the UAV using a transceiver, and a microcontroller performs onboard data acquisition, processing, and control. The majority of the mechanical structure is 3D-printed out of PLA filament and assembled with stainless steel metric bolts. Two hollow aluminum tubes act as guide rods for the dynamic jaw, which is driven by a Hitec HSR-2648CR servo attached to a stainless steel 6 mm diameter, 1.33 mm lead pitch, hex start lead screw. Limit switches bound the endpoints of the system, and the dynamic jaw can travel 95 mm along the lead screw's length, allowing for manipulation of caps ranging from 25 mm to 100 mm in diameter.

As with the original design, one jaw (the static jaw) is fixed in place, while the dynamic jaw is driven longitudinally by a servo fixed to a lead screw. A thrust bearing balances axial forces and provides a smooth rotation point on the outermost edge. Upon successfully clamping a cap, the torque produced by yawing the UAV is transferred through the quick connect system and directly to the grasped cap. An elastomer surface is placed on the inside of the jaws to increase the coefficient

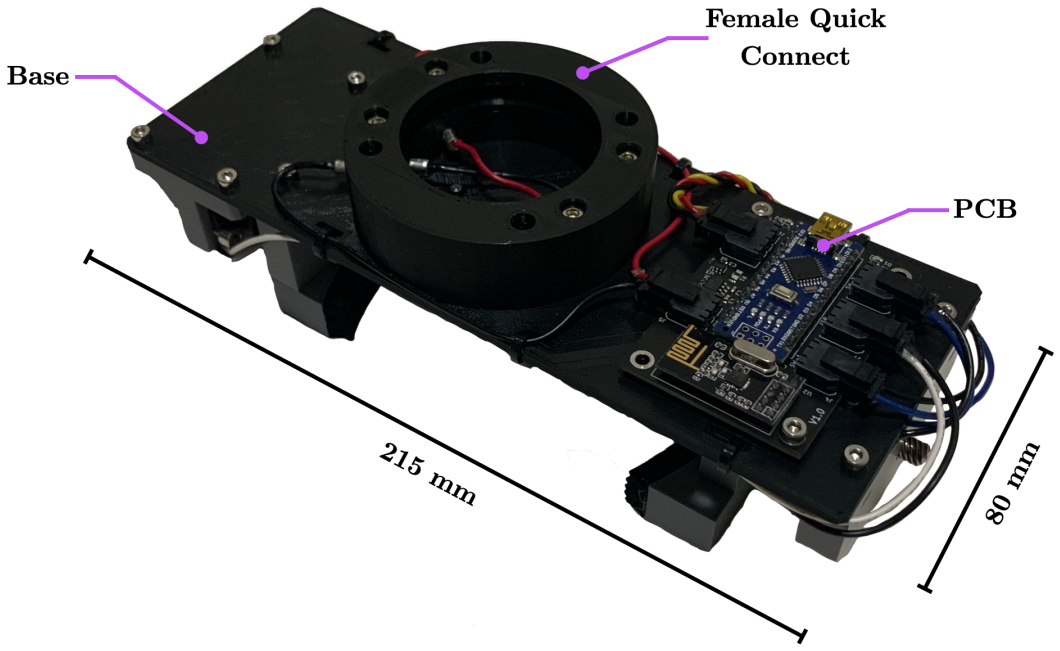


Figure 4.5: Cap Manipulator Top View with Labelled Components

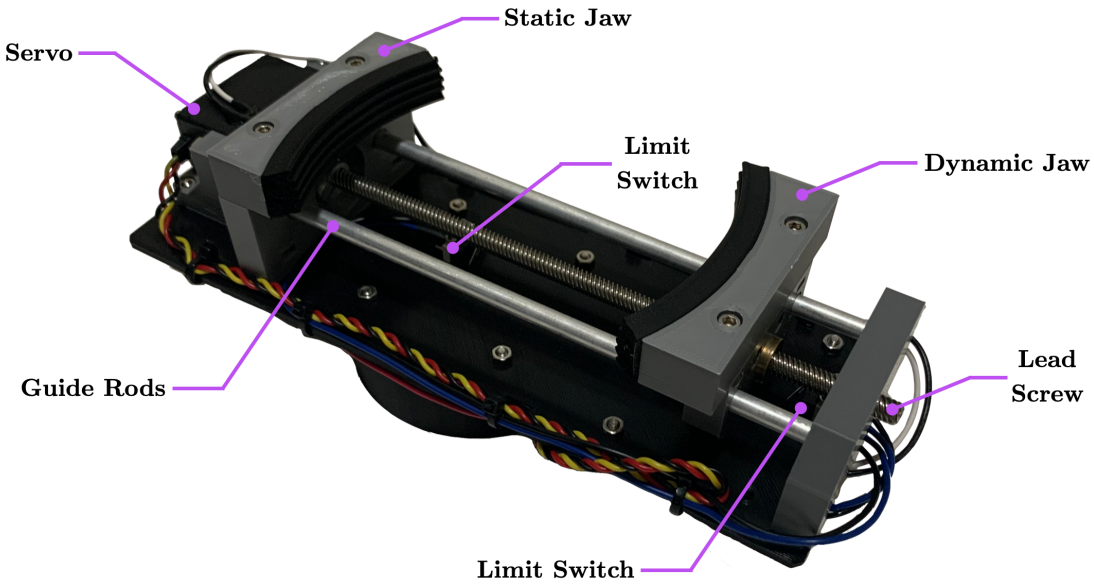


Figure 4.6: Cap Manipulator Bottom View with Labelled Components

of friction between the manipulator and a cap of interest. A selection of sensors provides feedback on how tightly a cap is gripped and keeps the dynamic jaw within its operating conditions.

Treating the lead screw as an ideal screw, the maximum clamping force capable of the system can be determined by using Equation 4.7, where F_{out} is the maximum clamping force in N, T_{in} is the input torque applied to the lead screw in Nm, and l is the thread lead of the lead screw in m. According to manufacturer data, the HSR-2648CR servo can produce 0.784 Nm of torque and the thread lead is 8 mm, which yields a maximum theoretical clamping force of 615 N.

$$F_{out} = \frac{T_{in} 2\pi}{l} \tag{4.7}$$

In reality, screws have large frictional energy losses, and Equation 4.7 should include an efficiency term η for a realistic estimate. To estimate the efficiency of a lead screw system, the helix angle α must be calculated as in Equation 4.8, where D is the thread diameter and l is the thread lead, both requiring the same units. The thread diameter, D, was measured to be 5 mm, and thread lead, l, is 8 mm, yielding a helix angle of 27°.

$$\alpha = \arctan \frac{l}{\pi D} \tag{4.8}$$

To calculate the efficiency η, Equation 4.9 is used, where α is the helix angle previously calculated in radians and μ_s is the static coefficient of friction between the lead nut and lead screw. The lead nut is brass, and the lead screw is stainless steel; therefore, a value of 0.51 was selected [146]. With the given values, the efficiency was found to be 37%.

$$\eta = \frac{\tan \alpha}{\tan(\alpha + \arctan \mu_s)} \tag{4.9}$$

Applying the efficiency to the ideal screw yields an updated estimated clamping force of 228 N. Although the efficiency significantly reduced the maximum theoretical clamping force of the system, a lead screw is self-locking if the efficiency is below 50% [155]. Such a system will not back drive, allowing the cap to be securely gripped without the need for additional input once contact has been established.

4.2.2 Electrical Design

The electrical design of the cap manipulator is summarized in Figure 4.7. The quick connect device provides up to 15 W of power at 5 V and is mainly used by the Arduino Nano microcontroller and Hitec HSR-2648CR servo. A wireless 2.4 GHz connection is established between Kuman nRF24L01

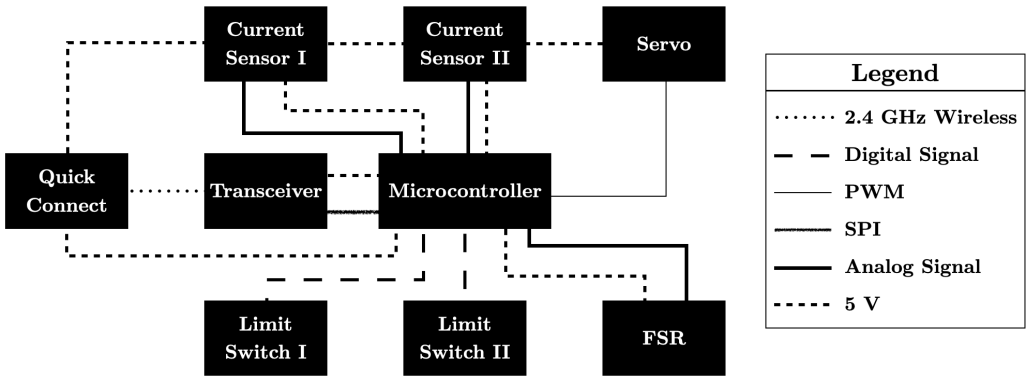


Figure 4.7: Cap Manipulator Electrical and Signal Diagram

transceivers onboard the UAV and payload facilitating communication between the systems. The transceivers interface with their respective microcontroller using SPI communication. The servo's current draw is monitored by two current sensors (Allegro MicroSystems ACS723 and Texas Instruments INA181A3) placed in series, which output analog signals read by the microcontroller. The microcontroller reroutes power from the quick connect to power the two current sensors and other peripherals such as the transceiver and an Interlink Electronics 30-49649 force sensitive resistor (FSR). The microcontroller provides a PWM signal to the servo to control the servo's direction and, in turn, control whether the dynamic jaw should open or close. Two Gikfun EK1713 limit switches are placed on the limits of the dynamic jaw's travel path to provide feedback as to where the dynamic jaw might be located. Lastly, the FSR provides additional feedback to the system when contact has been made with a cap.

The cap manipulators electrical design is centered around a custom two-layer 45 mm x 65 mm PCB. The PCB provides a single location for all the various electrical components to either be placed or terminated. The assembled PCB and a screenshot of the routed board can be viewed in Figure 4.8, and a list of the components, manufacturers, and part numbers can be found in Table 4.1.

As with the quick connect PCB design, 90-degree traces were eliminated, vias were minimized, and traces were sized to keep board temperatures below 40°C during peak operating conditions. A copper ground plane was used to establish a common ground between components and aid in overall thermal regulation, acting as a large heat sink. The board was designed to include mounting holes, and to validate the system, the entire system was prototyped using a protoboard and breakout boards before being manufactured.

All physical connections to the board used Molex Micro-Fit 3.0 connectors to keep commonality between the quick connect and payload. Additionally, the locking tabs, low-resistance contacts and

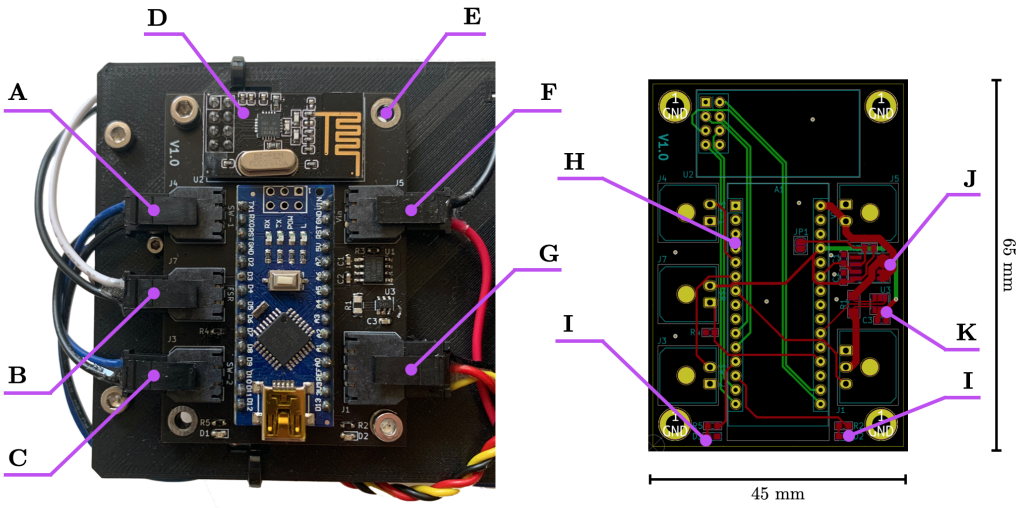


Figure 4.8: Cap Manipulator PCB Board: (Left) Assembled Board with Wiring, (Right) Board Layout with Hidden Fill Zones

Table 4.1: Cap Manipulator Major PCB Components and Peripherals

ID	Part Name	Manufacturer	Manufacturer Part Number
A.1	Connector	Molex	0436500200
A.2	Limit Switch I	Gikfun	EK1713
B.1	Connector	Molex	0436500200
B.2	FSR	Interlink Electronics	30-49649
C.1	Connector	Molex	0436500200
C.2	Limit Switch II	Gikfun	EK1713
D	Transceiver	Kuman	nRF24L01 + 2.4GHz Antenna
E	Custom PCB	JLCPCB	-
F	Power In Connector (5 V)	Molex	0436500200
G.1	Connector	Molex	0436500300
G.2	Servo	Hitec	HSR-2648CR
H	Microcontroller	Arduino	Arduino Nano v3.0 (A000005)
I	Indicator LEDs	-	-
J	Current Sensor I (Hall)	Allegro MicroSystems	ACS723LLCTR-05AB-T
K	Current Sensor II (Amp)	Texas Instruments	INA181A3IDBVR

crimps, and polarized casings minimize potential system failures, such as a connector falling out or a sensor being incorrectly plugged in. Surface mounted components such as capacitors, resistors, and LEDs were selected in place of through-hole components to minimize failure due to vibrations. Parts at risk of failing due to vibrations were reinforced with epoxy, such as the servo crimps in their housing.

The nRF24L01 transceiver was required in the design due to the communication protocol of the quick connect. An Arduino Nano microcontroller was selected as the central control unit because the payloads' system requirements shared many similarities to the quick connect and would allow for streamlined development between devices. Additionally, the Arduino Nano has a small footprint and onboard analog to digital converter (ADC), removing the need for additional components. Micro limit switches from Gikfun were selected to monitor the manipulator's endpoints, since the hardware had a minimal footprint and low mass. The Arduino's internal pull-up resistors avoided the need for external circuitry to prevent floating switch states. The HSR-2648CR servo was selected because it is a digital continuous rotation servo. Being digital, the servo provides faster response times and is less sensitive to disturbances than its analog counterpart. Additionally, the servo accepts an M3 x 0.5 mm socket head screw for additional holding power when adapting the servo to the lead screw.

Three sensors were used to provide feedback as to whether a cap has been gripped or released: an FSR and two current sensors. An FSR is a device that decreases in resistance with an increase in applied force. The 30-49649 FSR selected from Interlink Electronics is extremely small and lightweight, giving it the ability to be mounted behind the static jaw's elastomer surface. The FSR is not designed for precision measurements; however, it is optimized for touch control, such as when a cap is pressed into the static jaw. To measure the force applied to an FSR, power, VCC, is applied to one lead, while the second lead is connected to a grounded resistor R and an ADC. The combination of the FSR and grounded resistor creates a voltage divider. Converting the raw analog value to a voltage, V_{FSR}, the FSR's resistance, R_{FSR}, can be calculated as in Equation 4.10.

$$R_{FSR} = \frac{(VCC - V_{FSR})R}{V_{FSR}} \tag{4.10}$$

Using the FSR's calculated resistance and manufacturer supplied data, the resistance can be equated to a force. In the clamping manipulator, the FSR is protected by the elastomer surface, causing a distribution of the applied force and an underestimate of the actual force. Because the FSR is used as a relative comparison for gripping strength and not for quantitative data collection, there is no need for correction factors.

Two current sensors monitor the dynamic jaw. An FSR is not used on the dynamic jaw because two wires are required to read the FSR's output and would likely be caught in the actuating device. The first current sensor, an ACS723LLCTR-05AB-T from Allegro MicroSystems, is a precise low-offset linear hall sensor. As current flows through a copper path inside the integrated circuit (IC), a magnetic field is generated which is sensed by an integrated hall IC and converted to a proportional voltage to be read by the microcontroller. The IC required additional circuitry on the PCB, such as decoupling capacitors to reject power supply noise, output filtering capacitors, and a pull-

down resistor for selecting bandwidth. The circuitry was added according to the manufacturer's recommendations, and a 20 kHz bandwidth was selected for increased low-pass IC filtering and reduced sensor noise. The full-scale sensor's range is ±5 A and was advertised to have an error of ±3%. The raw analog value was mapped to a current measurement, I_{load}, by converting the analog value to an equivalent voltage, V, and then compensating for the voltage offset, V_{offset}, and applying a sensitivity scaling factor, $sens$, as observed in Equation 4.11.

$$I_{load} = \frac{V - V_{offset}}{sens} \tag{4.11}$$

A second current sensor was implemented in series with the linear hall sensor to provide a secondary current data stream. A current sense amplifier, INA181A3IDBVR from Texas Instruments, was selected and allowed for a more precise selection of the full-scale range. The current sense amplifier is paired with a current shunt resistor. The resistor is placed in the current path and typically requires a low resistance, high power rating, and precisely manufactured resistance (16 mΩ, 1 W, and 1% respectively for the following design). It is essential to have these parameters so that the resistor does not affect the current flow, can handle large loads, and is accurate enough that the hardware does not become a significant source of error. The current sense amplifier measures the voltage across the shunt resistor, which is proportional to the current passing through it, and amplifies the signal so that an ADC can read it. As with the linear hall sensor, the manufacturer provides recommendations for external circuitry (such as a decoupling capacitor) and proper routing techniques (such as Kelvin connections). The current shunt resistor is selected to minimize power loss, maximize the full-scale sensing range, and maximize accuracy (estimated to be 3%). To calculate the measured current I_{load}, the analog value is converted to an equivalent voltage, V, which is divided by the resistance, R_{sense}, of the current shunt resistor, and gain, $GAIN$, of the current sense amplifier as in Equation 4.12.

$$I_{load} = \frac{V}{R_{sense} GAIN} \tag{4.12}$$

Current data was recorded for an entire cycle of the dynamic jaw travelling from its outermost limit switch to the innermost limit switch to observe the two current sensors' response. The results are plotted in Figure 4.9. The first observation is that both the ACS723 linear hall effect sensor and INA181A3 current sense amplifier are extremely noisy, which was concluded to be a function of the digital servo motor. To filter the signal, a low-pass 50 point moving average filter was implemented. The number of points provided a useable signal and introduced minimum relative phase lag, which was empirically compensated for in the conditional arguments of the developed controller. The ACS723 was found to measure current approximately 40 mA higher than the INA181A3. Because

both values followed the same trend, and in order to further filter noise, the two sensor values were averaged to be used as a final value. Lastly, the cyclic current response observed in the plot appeared to be originating from the servo itself and not from the manipulator's mechanical assembly.

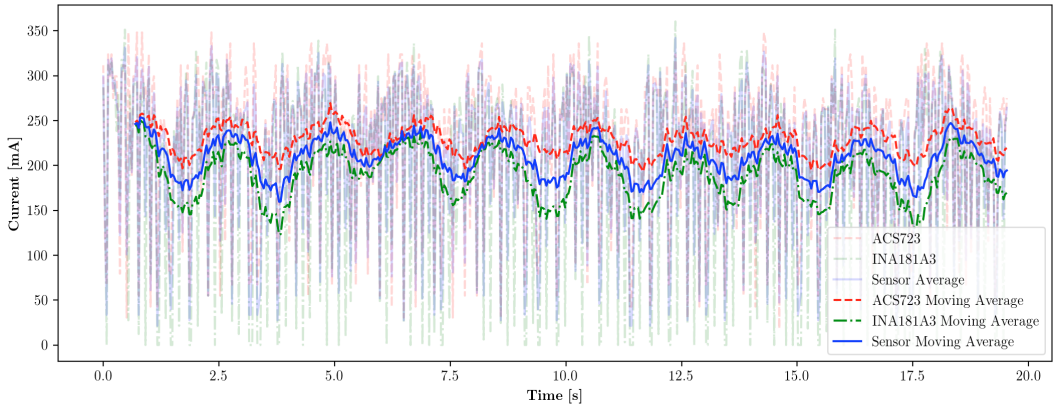


Figure 4.9: Raw and Filtered Current Readings with Dynamic Jaw Moving From Open to Close

4.2.3 Software Design

The software design of the clamping mechanism includes many similarities to the master quick connect system. Using a microcontroller, C/C++ was the programming language of choice, and the same five libraries as the mainboard were used, including the Arduino, Servo, SPI, and two nRF24L01 libraries. The libraries were used to interface with the Arduino pinout, generate the signals required to control the servo, and establish communication with the transceiver. The two nRF24L01 libraries handle the communication between transceivers. The main program running on the clamping payload features a wireless 2.4 GHz link with the UAV. Using the pass-through message protocol from the main program running on the SBC, state messages are sent to control the payload. A flow chart of the main program can be viewed in Figure 4.10 and the open and close switch cases are expanded on in Figures 4.11 and 4.12 respectively.

The main program begins by initializing the transceiver radio and setting the servo to be in an off position. An infinite loop reads incoming messages from the radio connected to the upstream SBC. The received messages are applied to a switch case, with functions to open or close the jaws. The default case implements a boundary controller which monitors the two limit switches and sets the servo to actuate in the opposing direction of the activated limit switch. A small half-second delay ensures that the limit switch is cleared and prevents a spike in the current readings caused by the servo initially starting up. The servo then returns to a stationary position and sends a message informing the UAV that it is in a floating state awaiting either an open or close command.

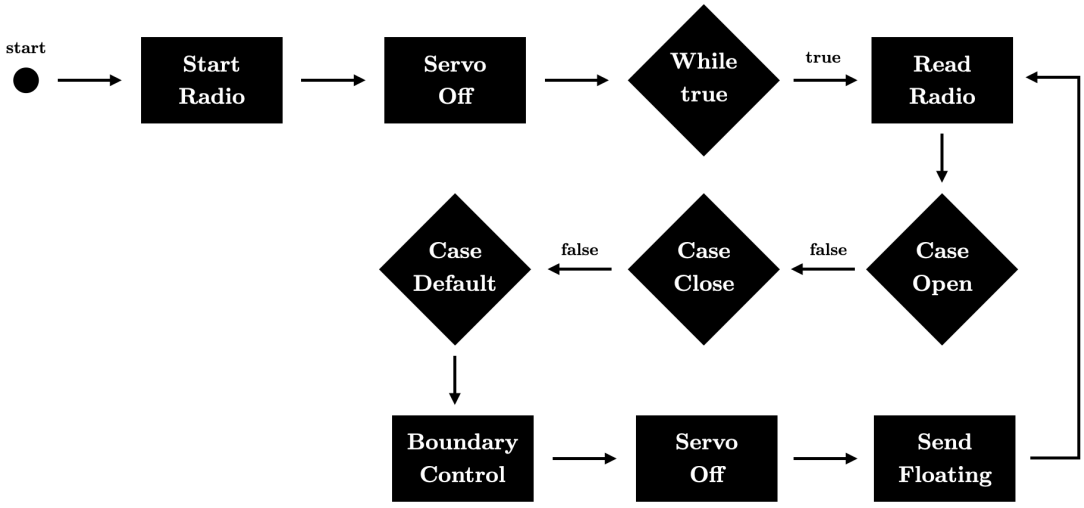


Figure 4.10: Main Cap Manipulator Code Flow Chart

The open case switch, Figure 4.11, begins by sending the servo a pulse to rotate until the outermost limit switch is activated. While the outermost limit switch has not been activated, the radio is read to check if a new command has been sent to close the jaws instead of open them. If true, the servo is set to an off state, and the program returns to the main loop, awaiting a second confirmation that the close state should be initiated. If the system does not need to close, then a value is acquired from the FSR. If the value is approximately zero, a message is sent stating the jaws are released; otherwise, a floating message is transmitted. Once the switch has been actuated, the servo is set to an off position, and the program returns to the main loop awaiting a new command.

The close case switch, Figure 4.12, begins by providing the servo with an initial PWM pulse to close the jaw. As the jaws close, a floating message is sent to the SBC, and the radio is monitored for incoming messages. If an open message is received, the system stops the servo and returns to the main loop to await confirmation. Otherwise, the limit switches are monitored, and the boundary controller is implemented. While the dynamic jaw continues to close, the FSR and current sensors are monitored. If both the measured current and force exceed their given thresholds, the force at that instant is saved. The servo is then set to a stopped position, and an integral controller either increases or decreases the servo's signal to maintain pressure on the cap in the unlikely case the lead screw is back driven or there is drift in the PWM signal. In case the integral controller should fail, the boundary control runs the entire time, preventing the system from over-actuating. A clamped message is sent to the SBC, and the radio is continuously read to monitor when the system should return to the main loop.

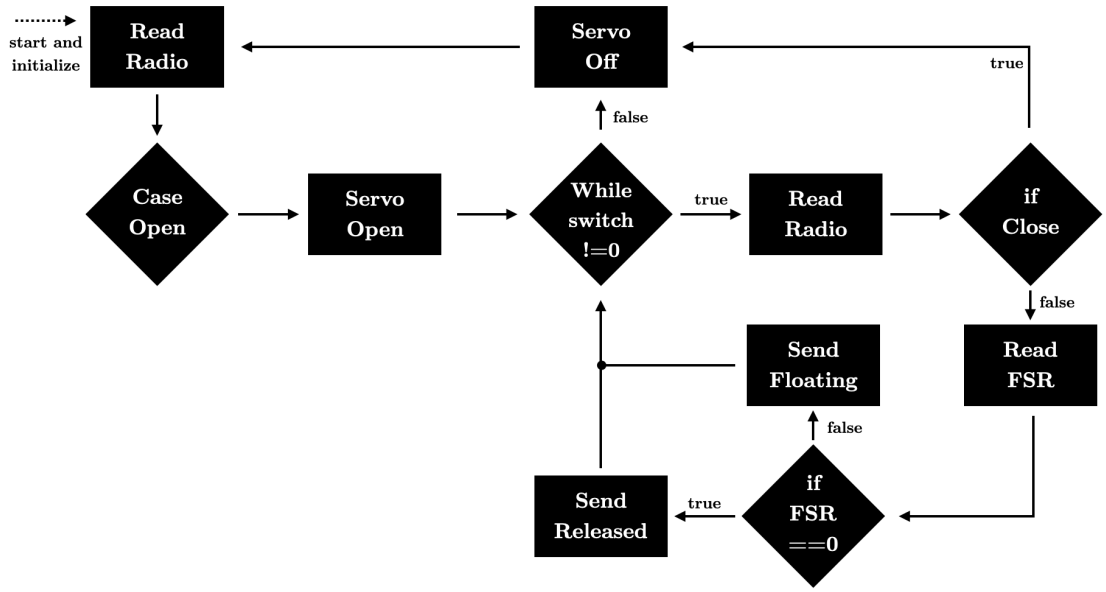


Figure 4.11: Open Case Switch Flow Chart (Continued From Figure 4.10)

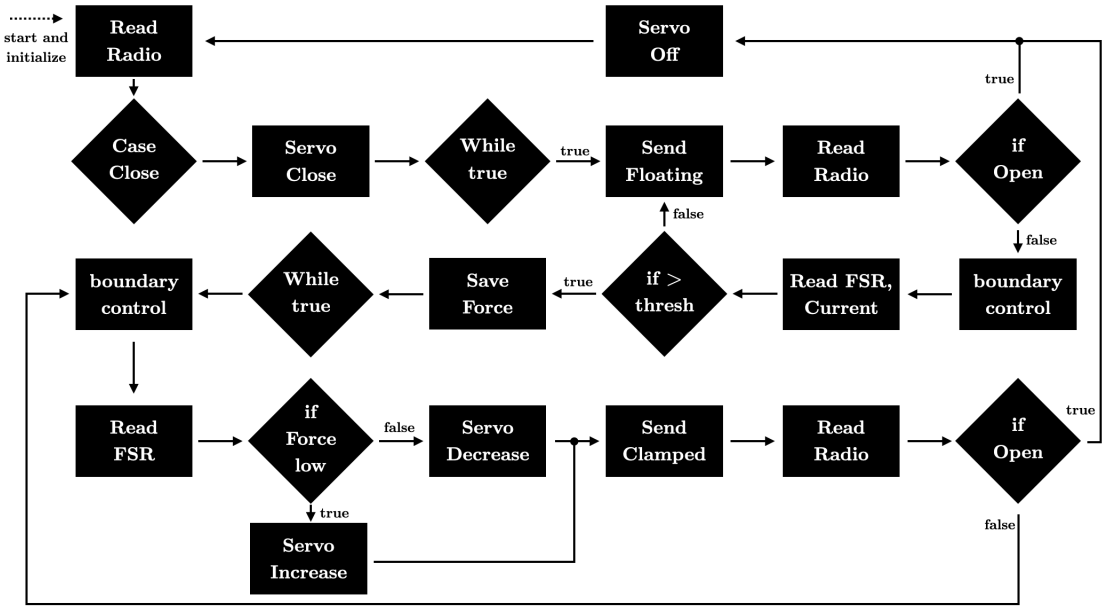


Figure 4.12: Close Case Switch Flow Chart (Continued From Figure 4.10)

To further illustrate the clamping process, Figure 4.13 shows the response of gripping a cap. For the first 7.5 seconds, while the dynamic jaw is closing, the FSR registers zero force, and the measured current is consistent other than some cyclic noise. At approximately 8 seconds, contact is made with a cap, and both the force and current readings increase linearly until their respective thresholds are both met at 10.5 seconds. The servo is then stopped, and the integral controller monitors the error in actuation force and adjusts the PWM signal to the servo motor to keep a consistent gripping force. The current drops below 100 mA as the lead screw's geometry provides the required locking force, and only minor adjustments are required from the servo. The cap is then securely gripped, and the UAV can apply a torque to the cap. The speed at which the jaw actuates or retracts is set empirically and is close to the maximum operating conditions of the servo motor.

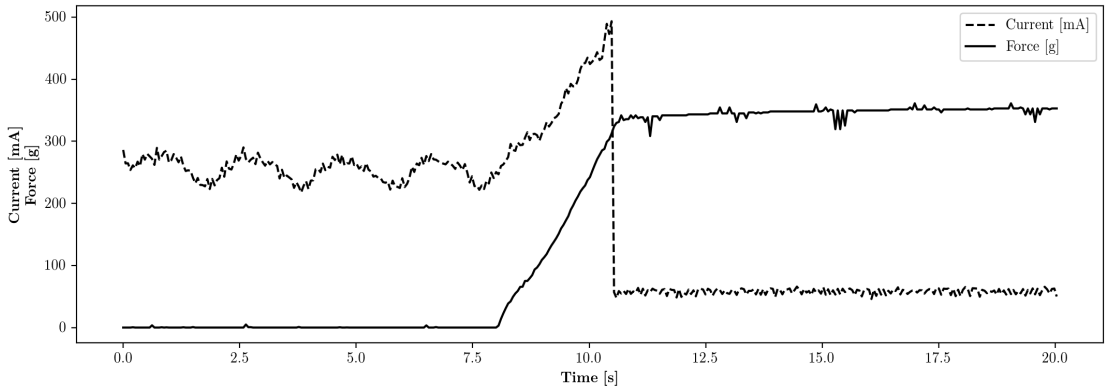


Figure 4.13: Clamping Sequence Current and Force Sensor Readings

4.2.4 Control Design

To quantify the maximum torque the UAV can produce by exhibiting a yawing motion, the torsional platform developed in [156] was used. The torsional platform was initially designed as a novel device for accurately estimating inertia tensors of objects with complex geometries such as robots. Using select outputs of the logged data and creating a custom-designed mount, the torsional platform could measure torque forces generated by the UAV.

To provide an initial benchmark of the maximum yaw torque the UAV can generate, manufacturer data provided with the XA2212 980KV brushless motors from EMAX was substituted in Equation 4.13, where τ is the maximum torque produced by a single motor, P is maximum power consumption, and w is the maximum angular velocity of the motor.

$$\tau = \frac{P}{\omega} = \frac{181\ W}{232\pi\ rad/s} = 0.249\ Nm \tag{4.13}$$

In a quadcopter configuration, two motors work together to provide a maximum torque, which yields a final theoretical value of 0.498 Nm. The result assumes that the system is 100% efficient in converting the electrical energy to thrust and torque. The actual efficiency is highly dependent on various other factors such as propeller efficiency and copper and iron losses in the brushless motor. To estimate the efficiencies is a non-trivial solution dependent on many factors (such as propeller advance ratios), and is often experimentally determined. Therefore, the result is an upper bound to compare against empirical results.

The UAV was fixed to the torsional platform using a shaft collar and custom 3D-printed mount. The mount was fixed about the center of rotation of the four propellers and bolted directly to the UAV frame. Four main tests were conducted, including motor ramp and step inputs in both clockwise and counterclockwise directions. For each test, the experiment was repeated five times to ensure the results were repeatable and increase the final results' confidence.

Each motor is controlled by an ESC which converts direct current (DC) power into alternating current (AC). The ESC converts the DC into AC based on a PWM input signal. For the 20 A Racerstar RS20A V2 ESCs used, input values range from approximately 1000 to 2000 μs depending on their calibration values. To generate a ramp input, a 2 μs pulse is incremented every 0.1 seconds beginning at 975 μs until the ESCs automatically turn off at approximately 2250 μs. The signal is supplied to a pair of opposite motors while the other pair is unplugged. The results of the clockwise and counterclockwise ramp inputs can be observed in Figures 4.14 and 4.15, respectively. The data is trimmed only to show non-zero torque values and to align all the different trials. Additionally, the counterclockwise results are inverted to become positive torques for ease of comparison.

The ramp experiment results show a slight second order but a mostly linear trend of increasing torque as the ESC input also increases. The oscillations are due to the torsional platform using a linear torsion spring, which is required by the apparatus to measure inertia tensors accurately. The oscillations are relatively small compared with the rest of the data and can be filtered should a single data point be required. Using the last five seconds of data, where the system is at a steady-state and holding a maximum torque, the results across all the trials were averaged, which yielded a torque value of 0.25$\pm$0.01 Nm in the clockwise direction and 0.22$\pm$0.01 Nm in the counterclockwise direction. The two directions differ by approximately 13%.

The second input signal is a step impulse. The input to the ESCs is slowly ramped to overcome inertial, friction, and aerodynamic forces. Once all motors were spinning and non-zero torque was being registered, the ESC was given its maximum signal. The step was held for 10 seconds before the motors were stopped and data logging was terminated. Figures 4.16 and 4.17 provide the results for the clockwise and counterclockwise trials respectively. Data is again trimmed to show only non-zero torque values, and the counterclockwise results are inverted to be positive.

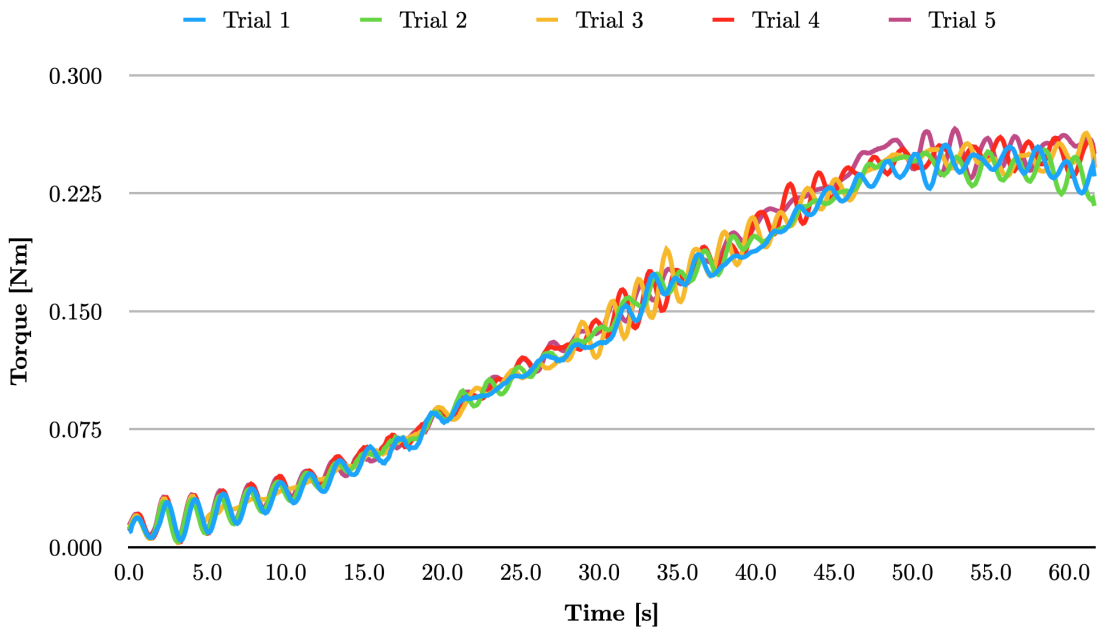


Figure 4.14: Clockwise Ramp Input: Torque as a Function of Time

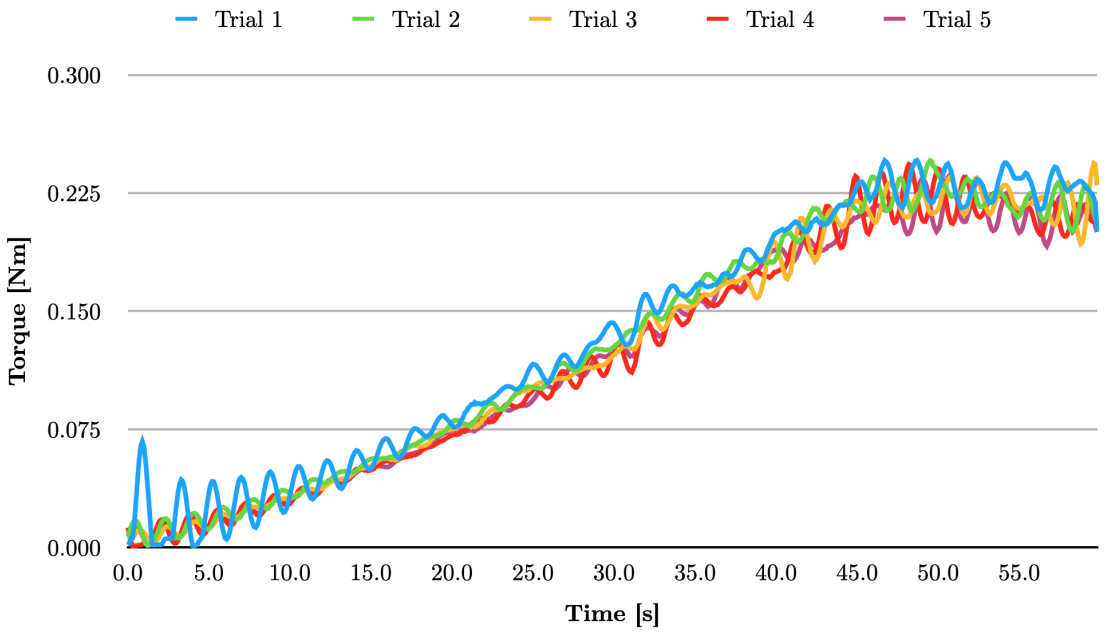


Figure 4.15: Counterclockwise Ramp Input: Torque as a Function of Time (Torque Converted to Positive)

The damped oscillatory nature of the linear torsion spring is more evident in the step response plots than the ramp input plots. The counterclockwise data is consistent between trials, while the clockwise response trials has some variability between trials. The first clockwise peak and steady state value is consistent; however, the periods and rate of dampening differ. Because the experimental setup, input signal, and system did not change between trials, the difference is due to some performance coupling of the clockwise motors, propellers, and ESCs which does not occur in the counterclockwise direction. Because the first peak is always consistent between trials, the largest five values from each trial were averaged to find the maximum impulse torque. The maximum clockwise value was calculated to be 0.40±0.02 Nm, and the counterclockwise value was 0.45±0.03 Nm. The two directions also differ by approximately 13%; however, the counterclockwise value is greater than the clockwise direction opposed to the ramp input.

Due to the quick connect system's current design and the way the torsional platform measures torque, the impulse torque recorded will not be achievable in practice. The impulse value will not be achievable, as the UAV can build up angular momentum in the experiment due to the linear torsional spring and the value recorded is a combination of momentum and rotor torque. In reality, the system will be static; however, in a future design, a spring clutch-based mechanism can be implemented, yielding results similar to the torsional platform. Therefore, the ramp test is a more realistic result of the currently available torque in the system. To further validate the steady-state ramp numbers, the UAV was again subjected to a step response and was left on until the oscillations were damped entirely. On average, the clockwise and counterclockwise trials took approximately 30 seconds to dampen fully. Averaging the steady-state values yielded a torque of 0.24±0.01 Nm and 0.21±0.01 Nm in the clockwise and counterclockwise directions, respectively. The steady-state values between a step response and ramp input differ by no more than 4%, demonstrating consistency in the responding variable in different experiments.

The results of all three experiments are summarized in Table 4.2, where μ is the average torque value and σ is the standard deviation. The measured step response is not feasible with the current hardware; however, the torque the UAV can introduce has the potential to more than double. The maximum steady-state torque was approximately 14% greater in the clockwise direction than in the counterclockwise direction. The difference can be attributed to flex in the airframe, individual motor performance, thrust blockage, and difference in the propellers' balance. Overall, the data had a small standard deviation compared to its nominal value, suggesting confidence in the acquired values. The experimental data is approximately half of the theoretical estimate of 0.498 Nm, suggesting an overall system efficiency range of 43% to 48% (assuming the manufacturer's data is reliable). In practice, the torque required to release a cap is likely not known. Caps can be over-tightened, build up resistance due to rust, or have very low resistance if a lubricant is applied to the threads. The analysis performed is to provide a maximum specification for the payload and

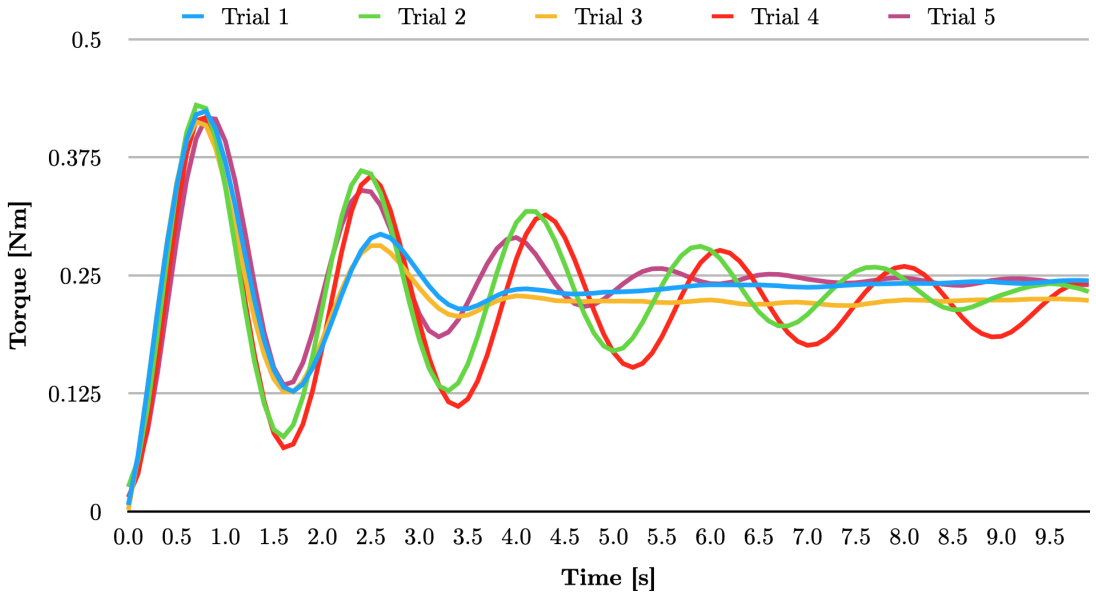


Figure 4.16: Clockwise Step Input: Torque as a Function of Time

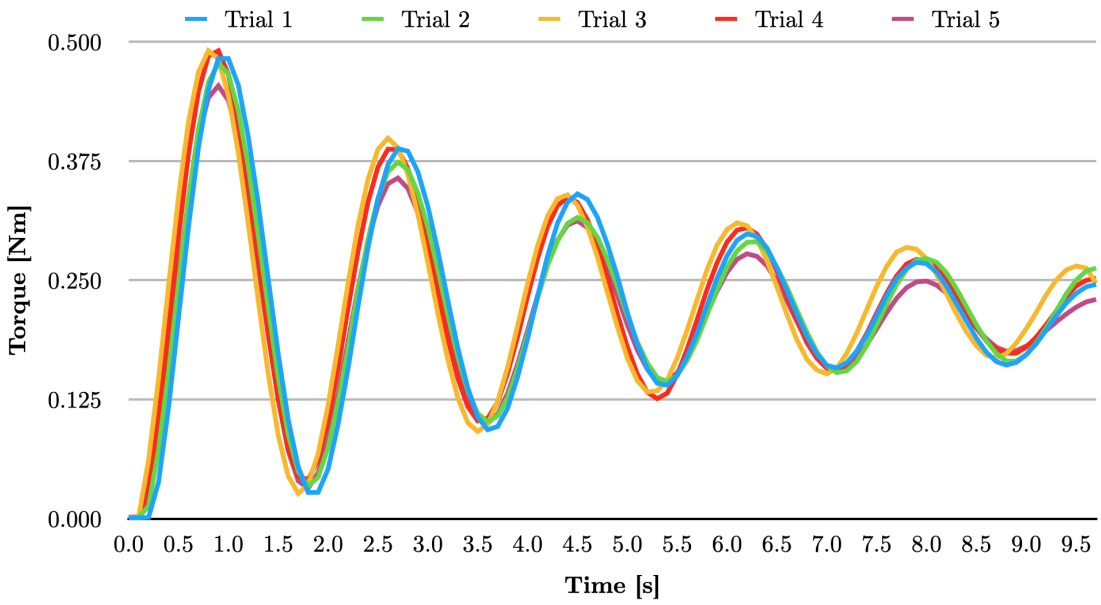


Figure 4.17: Counterclockwise Step Input: Torque as a Function of Time (Torque Converted to Positive)

Table 4.2: Experimental Yaw Torque Summary Table

Direction	**Metric**	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	**Units**
Clockwise	Ramp Steady State	0.25	0.01	Nm
	Step Max	0.40	0.02	Nm
	Step Steady State	0.24	0.01	Nm
Counterclockwise	Ramp Steady State	0.22	0.01	Nm
	Step Max	0.45	0.03	Nm
	Step Steady State	0.21	0.01	Nm

the given UAV so that the system can be applied to tasks within its capabilities.

With the detailed design of the cap manipulator and a functioning UAV and quick connect, the systems must undergo experimental validation as outlined in the following chapter.

Chapter 5

Experimental Trials and System Performance

5.1 Introductory Experimental Testing

Experimental prototyping testing was conducted to observe the complex synergy between the mechanical, electrical, and software designs, which would otherwise be challenging to model (especially the real-world responses). Testing also yielded valuable performance metrics of the developed system and the corresponding limitations.

5.1.1 Software in the Loop (SITL) Simulation

To validate the developed PID controller and MAVLink connection with the autopilot, a software in the loop (SITL) simulation was conducted. The goal of the SITL simulation was not to establish final gain values, but rather to validate the proposed theory and connection with the autopilot.

To run the SITL simulation, the ArduCopter firmware was compiled locally on a desktop computer running Linux Ubuntu 20.04 LTS. The default parameters and model of an Iris quadcopter were loaded into an ArduPilot supplied simulator and linked to a Gazebo virtual environment. Gazebo is a well-respected robotics simulator, and the virtual environment, Figure 5.1, was used to aid in tuning the positional controller by observing the response of the UAV when given different commands. Although ArduPilot and Gazebo are both open source, the model was treated as a black box and assumed to be functional, since it is used for unit testing and logic validation. To streamline the simulation, RTK GPS data was used to simulate ArUco vision data.

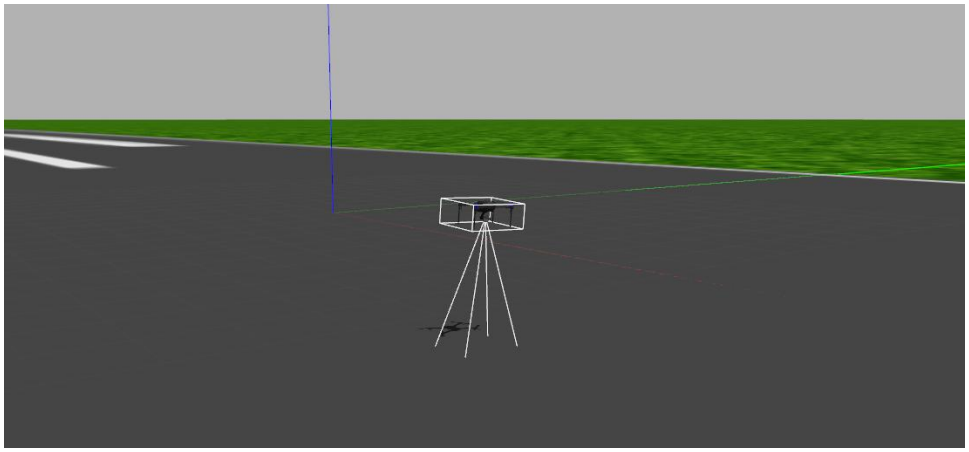


Figure 5.1: Gazebo Simulation Environment

With a configured environment and model, multiple different simulations were conducted. The first simulation had the UAV take off to an altitude of approximately 100 cm. Once this altitude was reached, trajectories lasting five seconds with desired set points of 75 cm and 50 cm for the north and east axes were generated. The trajectory in the down axis was set to last ten seconds and reach a final altitude of 125 cm. The results of the simulation after tuning can be viewed in Figure 5.2. The results show that the PID controller is correctly implemented and that the UAV follows the generated trajectories. There is some overshoot; however, the UAV settles to a steady-state value

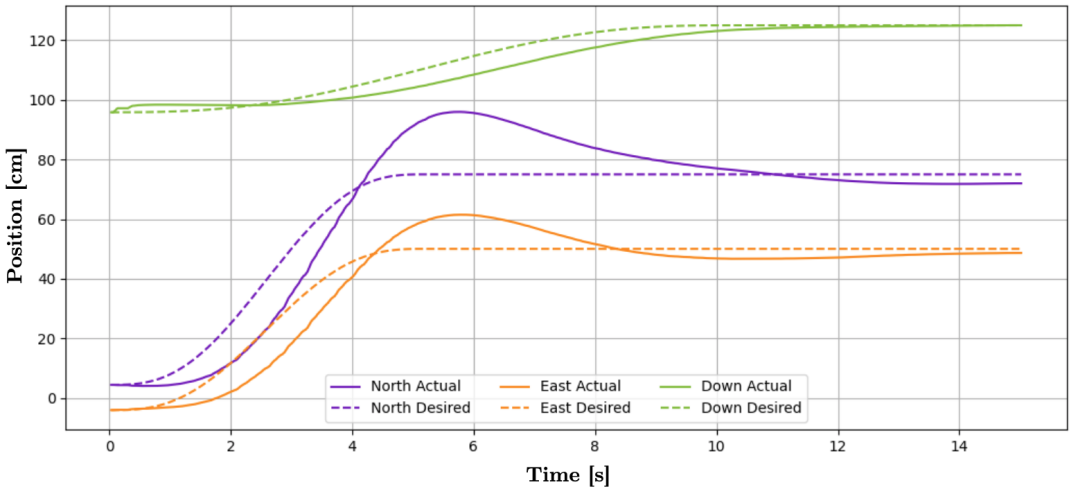


Figure 5.2: North East Down Position Controller Simulation Response

and achieves centimetre accuracy. The controller could be further tuned for a more optimal rise and settling time; however, the initial system was validated.

The second simulation conducted was controlling the yaw of the UAV from a 0° to 15° heading. No trajectory was used for the control, and the results are observed in Figure 5.3. The UAV's yaw responds exceptionally well, and sub-degree accuracy was achieved, further validating the generated code. The final simulation put the UAV at a 90° heading, so roll and pitch were swapped. The positional controller was reinitialized, and the same desired locations were set. The results aligned with Figure 5.2, validating the mixing of the controller's output based on yaw.

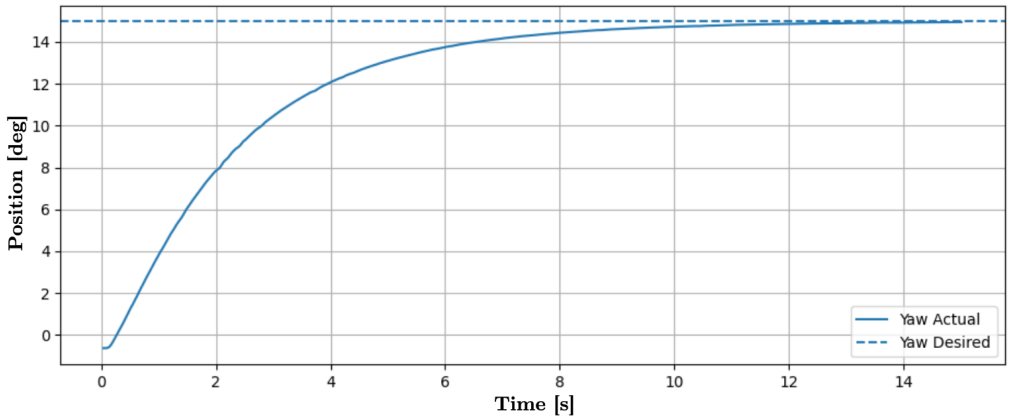
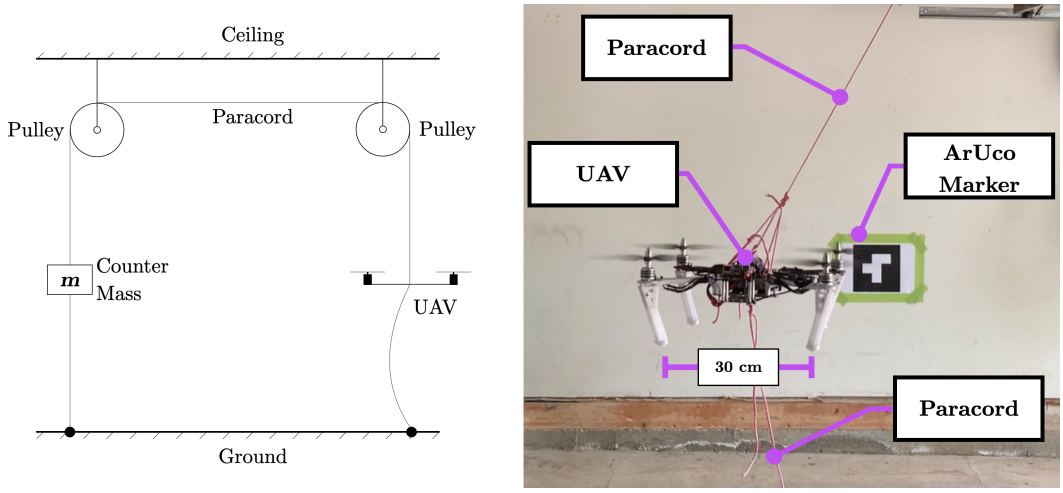


Figure 5.3: Yaw Position Controller Simulation Response

5.1.2 Online Testing with Safety Tether

Due to COVID-19 and the closure of our research group's laboratory, an alternate testing location had to be found and safeguards put into place. A residential garage was used as a testing facility for the remaining experimental trials.

During initial autonomous testing, the UAV was tethered as a safety precaution and to prevent damage in the event of a crash. The system used for tethering the UAV can be observed in Figure 5.4. The system works by constraining the UAV with two lines of paracord. To prevent the UAV from striking the ground, paracord was tied to the UAV's top and fed through two pulleys and then fixed to the ground. A small mass was fixed on the paracord to compensate for the tether's mass and prevent it from being caught in the propellers. To prevent the UAV from flying into the roof, an additional length of paracord was tied to the bottom of the UAV and fixed to the ground. Both directions of tethers worked in tandem to prevent the UAV from flying away in the lateral direction. The paracord's elastic properties and the amount of given slack in the system allowed for an operating volume of approximately 1.5 m x 1.5 m x 1.5 m.



(A) (B)

Figure 5.4: UAV Tethering System: (A) System Diagram, (B) Tethered UAV During Flight

Using the software validation from the SITL simulation, the same algorithms were tested on the hardware to validate the system's response in a controlled environment. Once all systems were validated, the control scheme was tuned for the given hardware. As control schemes were tuned, the tethers constraining the UAV were removed. Even with the addition of the counterbalancing mass and smooth pulleys, the tethers provided dampening in the control scheme, which yielded aggressive gains when the UAV was in free flight; however, the tuning brought the UAV to an initial controllable state.

5.2 Computational Performance

Yappi (Yet another python profiler) was selected to record the computational performance of the developed software. Yappi was chosen because it supports multithreaded programs, as opposed to the standard profilers included with Python. Additionally, the Yappi code is written in C, making it fast and limiting the effect the profiler may have on a program it is analyzing. The profiler was configured to record CPU time due to the threaded workloads and indeterminacy in function execution. Built-in functions were included in the profiled results to monitor whether the selected packages had to be altered or changed entirely. To profile the code, two flights of approximately 30 seconds were conducted (the I/O and logging and vision processes had to be profiled separately because they run in different memory heaps).

To visualize the raw profiling data, gprof2dot was used to generate visual call graphs. A call graph is

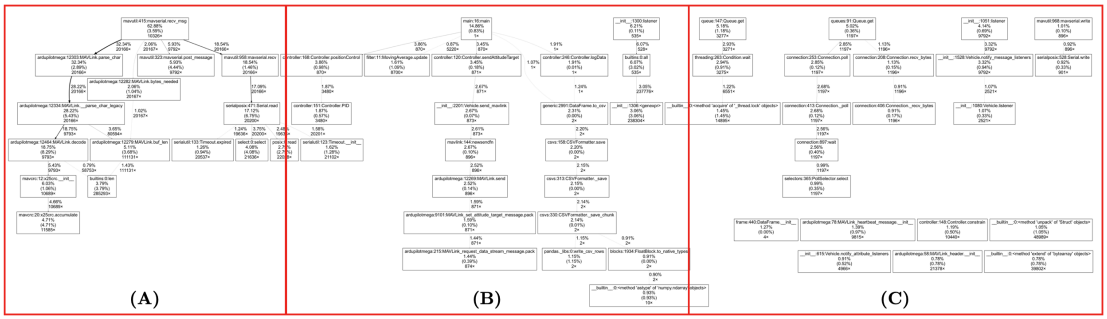


Figure 5.5: I/O and Logging Call Graph with Highlighted Regions of Interest

a control flow graph that displays the relationships between subroutines. Each box has the function's name, the amount of self-time of that function, the total number of calls, and the percentage of the total amount of CPU time spent in that function compared to the rest of the program. Arrows show the relationship between functions, and the number beside the arrow represents the number of times the function interaction was called. The results of profiling the I/O and logging process can be viewed in Figure 5.5. As there are many interactions and functions, areas of interest were highlighted for further analysis.

Using a secondary profiling visualizer, KCachegrind, the profiling data relating to Figure 5.5-A was examined in detail. KCachegrind provides the same graphs as gprof2dot; however, it provides additional analysis tools. The profiled data was filtered in KCachegrind to have a maximum callee depth of two to limit the number of root sub-functions that would appear. The minimum node and call cost were set to 5% to trim the number of functions further and only show those that had a significant computational contribution to the overall program. The results of the filtered profiled data are observed in Figure 5.6.

Figure 5.6 is a thread that used approximately 63% of the process's computation time, and was automatically spawned by the DroneKit API during the initial connection with the autopilot. The thread's main function, $mavserial.recv\_msg$, primarily uses two functions, $MAVLink.parse\_char$ and $mavserial.recv$, which use approximately 32% and 19% of the process's computation time respectively. The sub-functions' responsibilities are to read the autopilot serial port (running at 1500000 baud) and then parse the incoming MAVLink messages for relevant data. Due to the large amounts of data being processed, high computation requirements were expected. Because the I/O and logging process is not CPU bound, as with the vision process, no changes were required in the data streaming rates between the autopilot and main program. Lastly, the $mavserial.post\_message$ function is an additional MAVLink function for organizing messages and consumes approximately 6% of the process's computation time.

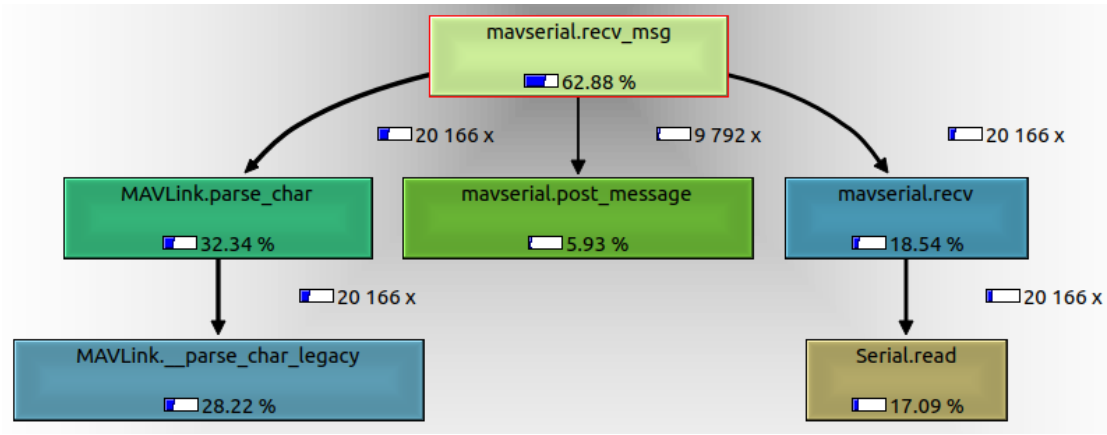


Figure 5.6: I/O and Logging Detail Figure 5.5-A Call Graph

Using the same filters as previously, the subroutines shown in Figure 5.5-B were converted to Figure 5.7. The *main* function is the central function that spawns all processes and threads, logs data, and is relied upon by the overall system. The function uses approximately 15% of the process's computation time. The highest contributing function, at approximately 4%, is the *Controller.positionControl*, which runs the PID controller and provides set points to navigate the UAV. The *Controller.sendAttitudeTarget* is the second most resource-hungry function at approximately 3%; it is responsible for sending the calculated set points to the autopilot. A moving average function provides filtering for a trajectory's initial values should it be requested, and consumes less than a percent of the processes computation time; data logging consumes approximately 2%. The *DataFrame.to_csv* logging archive function is only executed once a mission is complete. Therefore, the function does not affect the real-time system.

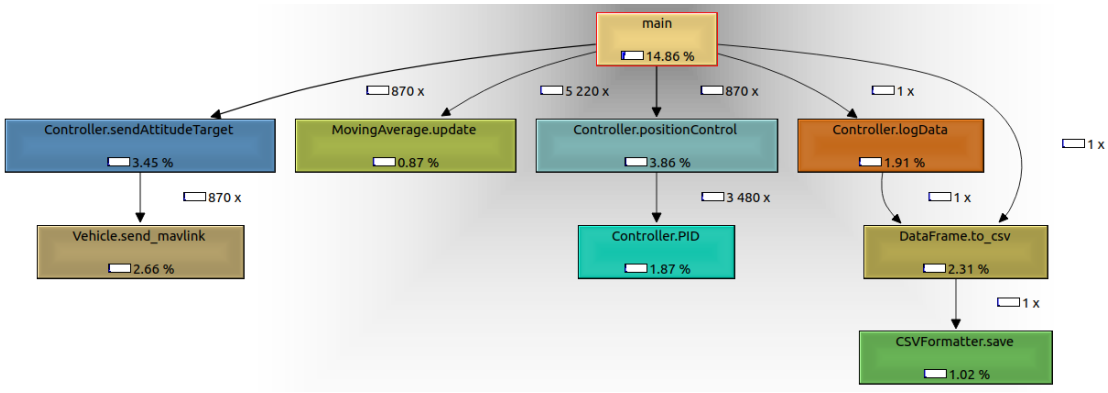


Figure 5.7: I/O and Logging Detail Figure 5.5-B Call Graph

Figure 5.6 and 5.7 total process's computation time sums to approximately 78%. Those in Figure 5.5-C account for the remaining 22% and consist of numerous small functions that primarily occur during the program initialization. The two largest functions, accounting for a majority of the 22%, are built-in methods for exchanging data placed in queues and listening for specific messages in data streams.

The call graphs profiling the vision process are much simpler than the I/O bound process. The filtered call graph of the main thread can be viewed in Figure 5.8. The main thread consumes approximately 5% of the second process's computation time, with most of the CPU clock cycles being used by the Kalman filter to estimate the vehicle's pose by fusing the raw positional data with IMU sensor data. The remaining computation time is used to establish a connection with the Intel RealSense T265 camera during startup.

The CPU-bound tasks are performed on the remaining three threads of the vision process and are graphed in Figure 5.9. Although the system is not entirely CPU-bound, the process was running near maximum capacity. Due to the threads being created manually by the main program, all resources are traced back to *Thread.run*. The call graph's left side shows the resources required for the T265 to acquire IMU data and the corresponding image frames from the two cameras. The most taxing function is *remap*, which converts the fisheye image into a pinhole equivalent image. The mapping of camera models consumes approximately 29% of the process's computation time. Because the developed code uses classes and object-oriented programming, a single group of functions can be used multiple times. Therefore, on the call graph's right side, two instances of *VisionPose.run* are generated. The program initializes the same class for each of the two cameras based on the respective lens calibration values. The largest computational consumption

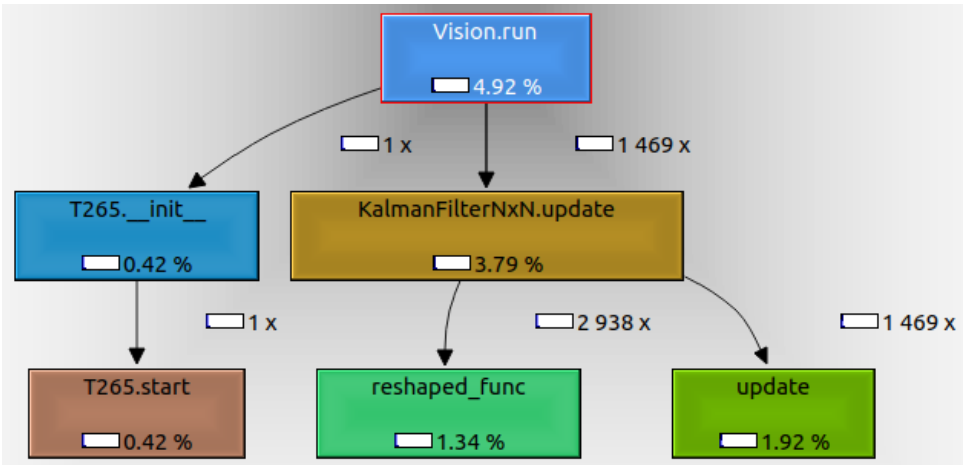


Figure 5.8: Vision Process Main Thread Call Graph

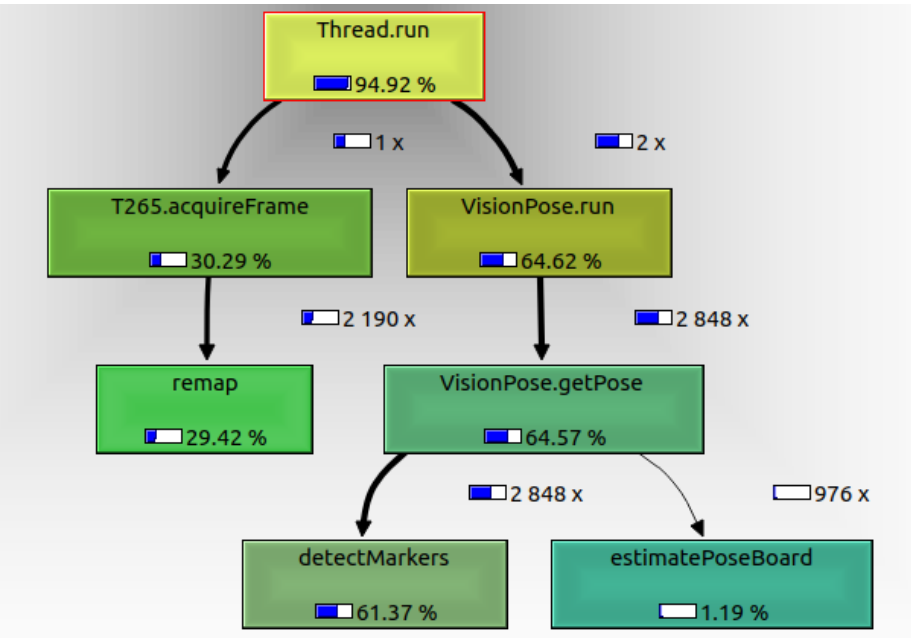


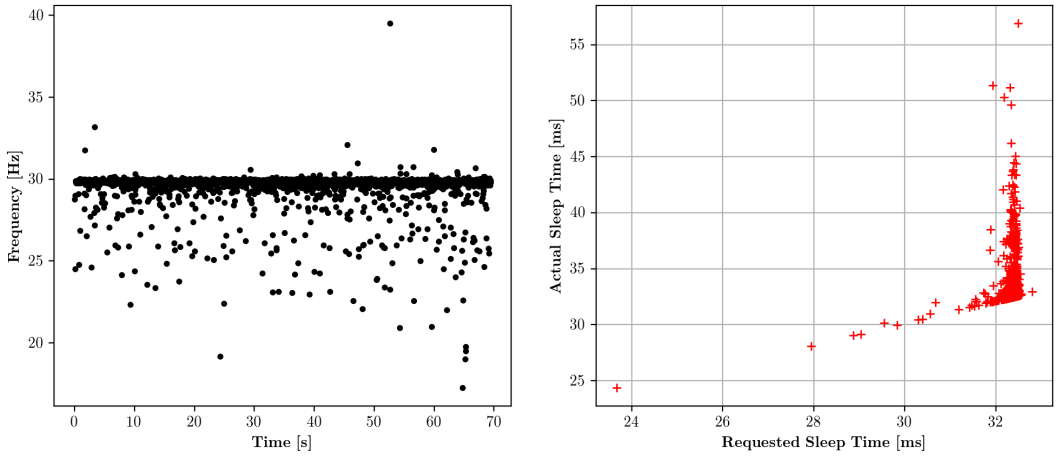
Figure 5.9: Vision Process Spawned Threads Call Graph

in retrieving a pose is detecting a marker, which consumes approximately 61% of the process's computation time. Once a marker is found, the pose estimation is minimal. The call graph for detecting the markers cannot be analyzed further, as Yappi only monitors Python functions, and the *detectMarkers* function is based in C/C++ due to the OpenCV wrapper used. Going through the OpenCV source code and adjusting ArUco parameters, the adaptive thresholding was found to have the most considerable contribution on performance; this is why the number of filtering windows was reduced from the default value of three to two.

Because Python is not truly concurrent from GIL, it is essential to have delays so that other threads can run and loops do not waste clock cycles. Additionally, having a consistent sampling rate is important for tuning of controllers and data logging. To achieve desired sampling rates, a function was written that monitors the time elapsed since it was last executed and then delays accordingly. When using this method, the system needed to be capable of running above a desired sampling rate; otherwise, the loop cannot be bound by the delay.

Testing the function proved to be successful; however, the results were more variable than expected. One example of sampling frequency as a function of flight time can be found in Figure 5.10-A, which had a desired frequency of 30 Hz and an actual frequency of 29.5±1.3 Hz. The results show that sometimes a loop takes longer than it should, and no delay can be executed. However, frequencies

above the desired rate raised questions, and the desired versus actual delay times was plotted as viewed in Figure 5.10-B. The results show that the desired delay is not always achieved. The variability leads to both undershooting and overshooting of delay times and the corresponding loop frequency. Further analysis found that the variability was highly dependent on the OS used due to catching routines and scheduling of activities [144]. When comparing desired and actual delays for other operating systems, including Windows 10 and macOS, Linux Ubuntu had the most consistent performance. Because the deviation of frequency was relatively minimal, and a small number of outliers would not cause significant issues, the limitation of a non-perfect sampling rate was deemed acceptable.



(A) (B)

Figure 5.10: (A) Frequency Rate and (B) Actual Sleep Time Versus Requested Sleep Time

5.3 Localization

5.3.1 ArUco Pose Validation

The primary method for localizing the UAV in an environment was the use of ArUco fiducial markers. Three markers were placed in an array and configured as an ArUco board to allow for occlusions and increase pose accuracy. The markers' size was constrained by the payload docking station's interior and was sized at 13 cm x 13 cm. Markers were spaced 22 cm apart edge to edge, so that a marker was located on the two outermost edges of the docking station, with one directly in the center.

Examples of what the UAV sees during flight can be observed in Figures 5.11 and 5.12. Each Figure has two sets of images due to the Intel RealSense T265 having two slightly offset lenses. The photos

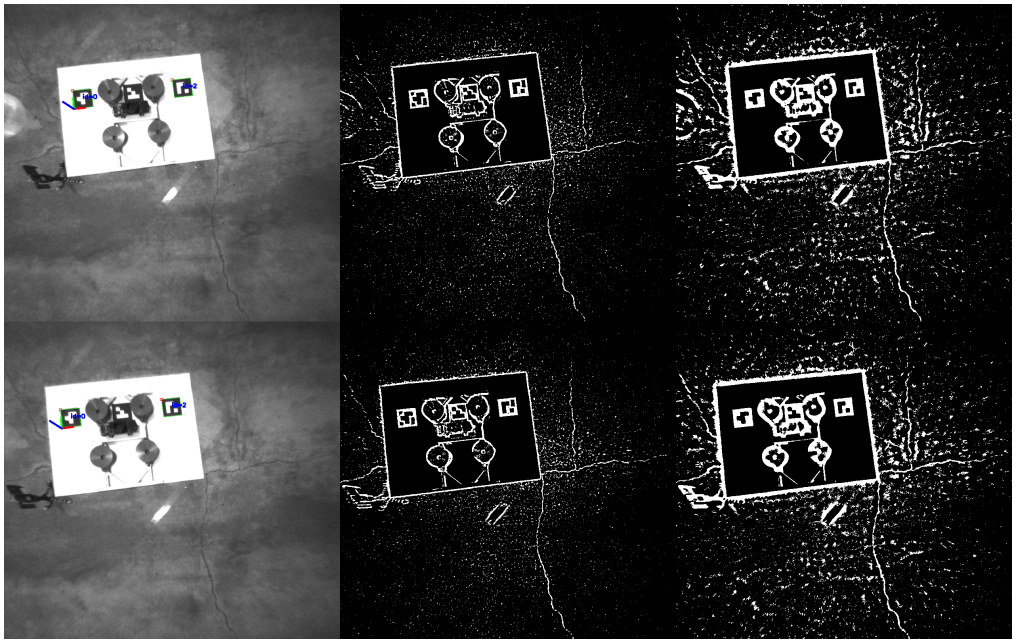


Figure 5.11: UAV Vision with Adaptive Threshold Responses at an Altitude of 75 cm

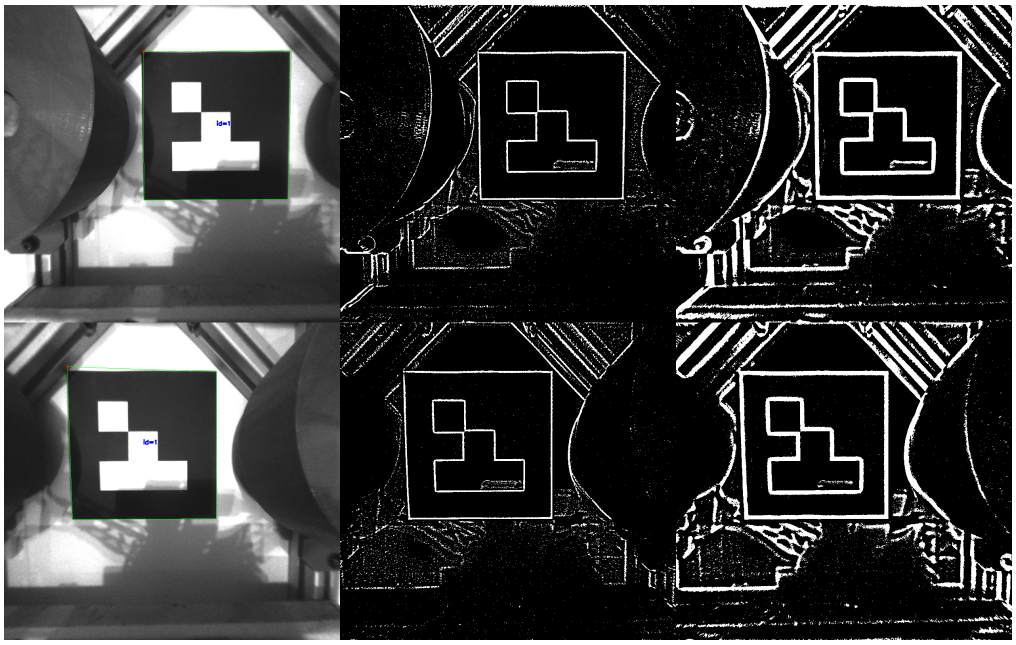


Figure 5.12: UAV Vision with Adaptive Threshold Responses at an Altitude of 11 cm (landing)

at far left are the equivalent pinhole images after correcting for the fisheye distortion of the camera lens. The center and far right images are the adaptive thresholded images, with window sizes of 9 and 23 respectively. The results of the adaptive threshold are raw, and no contours are filtered. The selected filter windows show the adaptive algorithm's robustness, where shadows are filtered, and the markers' contour is detected. Additionally, having multiple different thresholding windows allows for one window to detect the marker when the other window fails, further increasing the system's robustness.

Comparing Figures 5.11 and 5.12, a filter window size of 9 yields thin-walled contours, while a filter window size of 23 yields much thicker walled contours. Flying further from the markers, Figure 5.11, a greater number of contours are registered than when flying close to the markers, Figure 5.12. The difference is due to the amount of information in the image. Flying further from the markers, more of the environment is captured and filtered than when landed. In both Figures 5.11 and 5.12, the left and right lenses of the T265 camera, and the corresponding filtered images, yield similar results. The image set also displays why an ArUco board was selected. When approaching the target, all three markers are detected, as observed in Figure 5.11, and once landed, Figure 5.12, only the center marker is detected. Similarly, as the UAV travels further away from the markers in the north or east direction, the center marker becomes occluded, and the outer markers are used for the pose calculation.

To validate the raw pose accuracy, the UAV was manually placed at increments of 50 cm from the docking station. Only the down axis (the distance from the target) was considered in the analysis, as pose accuracy is highly dependent on the size of the marker in the image frame, which is directly proportional to the target's distance from the camera. The remaining axes will exhibit a similar trend to the down axis, and attempting to precisely measure north, east, or yaw at varying distances from the target would require precision equipment.

Recording the raw position from the ArUco marker and comparing it to the actual measured position revealed that the ArUco calculation had centimetre accuracy in the range of 0 cm to 200 cm from the payload docking station. From 200 cm to 350 cm, the raw values began to exhibit noise and underestimated the position by an average of 10 cm. The marker became pixelated and no longer detectable between 350 cm and 400 cm. Due to the constraints of the flight location, an altitude greater than 200 cm could never be achieved during experiments. Additionally, precise centimetre accuracy is not required at far distances, as the UAV can navigate closer to the marker to receive more accurate data. Therefore, the raw values have sufficient accuracy for estimating the UAV's position in the available environment.

To ensure that the two cameras of the Intel RealSense T265 were in agreement, a flight was conducted and the raw ArUco pose values were recorded. Plotting the response of the two cameras

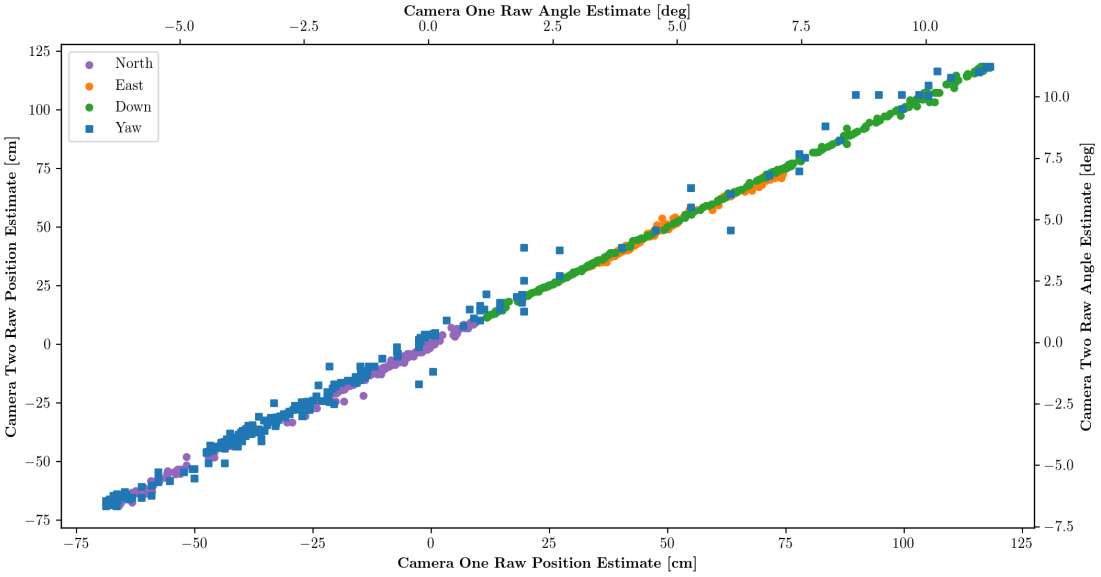


Figure 5.13: Raw ArUco Pose Scatter Plot Comparing the Two Intel RealSense T265 Cameras

against each other created the scatter plot in Figure 5.13. The mean difference, μ, and corresponding standard deviation, σ, between the two cameras are summarized in Table 5.1. Overall, the two cameras agree, with the worst average difference being -0.64±0.83 cm in the down axis. The north and east axis average difference is 0.15±1.06 cm and 0.02±0.80 cm, respectively. The trend is observable in the linear nature of the scatter plot. The yaw's average difference between cameras was 0.00±0.32 degrees; however, the deviation is much worse than the positional deviations as observed in the scatter plot, suggesting that angles are more susceptible to noise than translational measurements. Considering that all the values are raw measurements, there are two data streams, and no filter has been applied, the foundation for localizing the UAV is well-established.

Table 5.1: Raw ArUco Pose Difference Statistics Between the Two Intel RealSense T265 Cameras

Axis	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	**Units**
North	0.15	1.06	cm
East	0.02	0.80	cm
Down	-0.64	0.83	cm
Yaw	0.00	0.32	degrees

5.3.2 Kalman Filter Validation

A Kalman filter was implemented to filter outliers, provide a pose estimation if the target was ever lost, and increase the overall confidence in the localization of the UAV. The system was calibrated

to ensure the filter was correctly fusing the various data streams and had a correct output. Due to constrained resources from world events, no advanced methods for obtaining a ground truth (such as a Vicon motion capture system) could be used. To address this shortcoming, the vision system's output was compared to outputs of the flight controller.

The autopilot responses were considered true, as thousands of users have adopted the ArduPilot firmware across the globe in both industry and research [157, 158]. The autopilot was proven to work for the given applications, as it can level itself and even follow desired headings and altitudes when configured with a GPS in an outdoor environment. Although the autopilot's actual accuracy with the developed UAV cannot be accurately quantified, the system trends in all the expected directions qualitatively. Therefore, the autopilot provided the most reliable comparison metric for the developed localization system.

To acquire data, the UAV was manually controlled to acquire the pose, corresponding filtered data, and outputs from the flight controller. The UAV was manually controlled instead of running a control loop with a preprogrammed path to reduce the two systems' coupling. For example, if the controller's input data is unreliable, then the corresponding output will also suffer. Similarly, if the controller is not tuned to a sufficient level, then it will be challenging to draw conclusions from the acquired data. Manually controlling the UAV allowed a focus be put on the vision system and away from the controller. Additionally, manually flying the UAV reduced the risk of a crash when requesting large inputs from the system.

To validate the developed yaw estimator, the acquired data was compared directly to the autopilot's yaw. The autopilot's internal magnetometer is disabled due to flying indoors within proximity of hard iron distortions. To mitigate gyroscope drift, the autopilot runs advanced state estimation techniques such as dual extended Kalman filters, each reading an independent IMU sensor. The firmware then runs a comparison between the two results and makes compensations where applicable. Additionally, running the UAV for only a short duration limits the buildup of error. Since the UAV has no reference point without a magnetometer, the yaw is calculated relative to its starting position and cannot localize as with the fiducial markers.

Data from a flight is plotted in Figure 5.14, where the UAV was put through various yaw maneuvers such as small- and large-amplitude sinusoidal oscillations, and was maneuvered so that the vision system momentarily lost sight of the markers. The data was post-processed to remove takeoff and landing, which provided no marker detection. The data set was also adjusted so the mean value was centered about zero to compensate for the UAV's heading not aligning with the markers' coordinate system.

At an initial glance, the two systems track each other throughout the flight. The values appear to have a small constant offset at the beginning and end of the experiment. In contrast, the

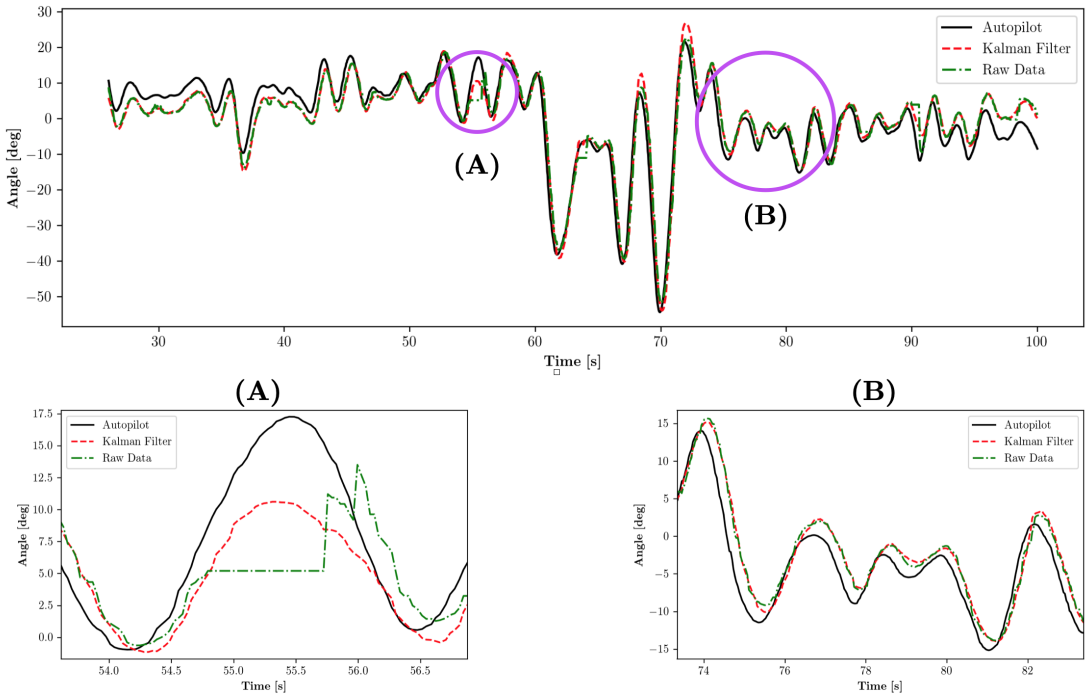


Figure 5.14: Yaw Kalman Filter Validation Using Autopilot as Ground Truth with Highlighted Regions of Interest

experiment's center tracked more closely other than a small overshoot during the large oscillations. The root mean squared error (RMSE) was calculated to quantitatively compare the vision system with the autopilot, where RMSE is a comparison of how similar two groups of numbers are on average. To compute RMSE, Equation 5.1 was used, where n is the total number of data points considered, i is the index for a given array of data, y is the data considered to be true, and $\hat{y}$ is the estimated value which is being compared.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.1}$$

Computing the RMSE of the two data sets, the Kalman filtered data were found to have a value of 3.83° and the raw data a value of 4.02°. As two yaw values are being compared, the RMSE value can also be considered the standard error of the estimator. A lower number suggests a better fit of the data; however, the RMSE values only differ by 5%. The small difference can be attributed to the highlighted "A" region of Figure 5.14. The highlighted region is one area where the vision system lost sight of the markers observed by the horizontal dot-dashed line. The Kalman filter responds by

recognizing that the body's angular velocity is still rotating, and provides an additional estimate that follows the autopilot trend. The Kalman filter's value does not reach the same amplitude of the autopilot due to the raw data's weighting and the programming implementation of the vision system's class-based method saving the last recorded position. Once the vision system regains sight of the target, the Kalman filter begins to track with the new data. Recalculating the RMSE value for the windowed data yielded values of 2.92° and 5.12° for the Kalman filter and raw data, respectively, which demonstrates an even better response of the Kalman filter for the given window then the entire data set. It should also be noted that the Kalman filter removes the high-frequency edge when the pose was again found.

A final observation is the highlighted "B" region of Figure 5.14. There is an approximately 0.2 second phase lag between the autopilot's response and the vision system response, which yields windowed RMSE values of 2.18° and 2.04° for the Kalman filter and raw data, respectively. Although the raw data is slightly better, the difference is negligible compared to the benefits of prediction and low-pass filtering the Kalman filter provides. If the lag were compensated for, the inflection points would track much closer, and RMSE would be further reduced. Because the Kalman filter and raw data both lag by similar amounts, the latency is likely due to processing the pose estimation and passing the data between memory heaps.

The autopilot does not return positional data unless it has a GPS; therefore, a relation between desired input, actual output, and positional data must be established to validate the translation Kalman filters. To establish the relationship, flights were manually conducted so that low-frequency oscillations were introduced in each axis independently. The oscillations provided trackable signals and reduced noise caused by the UAV trying to keep itself level. Since conclusions are to be drawn from the introduced oscillations, only a subset of data can be considered rather than an entire flight's data, as with yaw.

An example data set for validating the north axis is observable in Figure 5.15. The left-hand axis is the position estimate in centimetres, while the right-hand graph has an axis for both the pitch angle in degrees and the controller input to the autopilot in microseconds. The windowed data shows two complete oscillations where the UAV achieved approximately ±10° of pitch, corresponding to an approximately 50 cm amplitude in position. There is a small disturbance in pitch compared to the other trends, located at the bottom of each trough; however, the four signals appear to reasonably track each other.

For the given window, the RMSE was computed between the UAV's pitch and the raw and Kalman filtered positional data. The controller input was not included in the comparison, as the value is a desired setpoint rather than a indicator of how the system responded. Before the RMSE value was calculated, values were normalized to compare the trend instead of mixing units and compar-

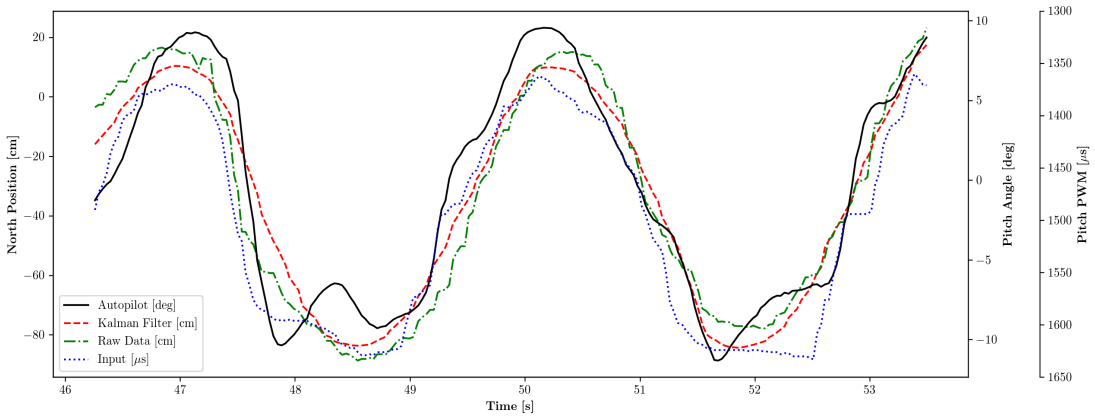


Figure 5.15: North Kalman Filter Validation Using Autopilot Controller Input and Pitch Output for Comparison

ing degrees with centimetres. The values were normalized in the range of $[0, 1]$ using Equation 5.2, where x is the input data vector to be normalized in the range $[a, b]$. Using the normalized values, the RMSE was calculated to be 0.108 and 0.128 for the Kalman filter and raw data, respectively, demonstrating that the Kalman filter's implementation was an improvement over the raw data.

$$x_{normalized} = (b - a)\frac{x - min(x)}{max(x) - min(x)} + a \tag{5.2}$$

The translational Kalman filter exhibits the same trends as the yaw filter. When subjected to missing data, the estimator continues to follow the same trend as the attitude with a slightly damped response. The phase lag also matches the yaw response being approximately 0.2 seconds on average, and the Kalman filter provides a smooth response attenuating high-frequency signals. When comparing the remaining east and down axes, all the responses followed the same trends due to the same equations of motion and corresponding data streams. Overall, the rotational and positional Kalman filtered data, when compared to the autopilot, can be used in the system until more precise validation techniques are available.

5.4 PID Tuning

With the vision system validated, the next step was to tune the PID controller. The fundamental algorithms were verified with the SITL simulation and further validated on hardware using the tether system.

The first axis tuned was the down (thrust) controller, as it was essential to keep the UAV airborne while tuning the other axis. The tuning effort was minimal, as ArduCopter has an internal altitude controller. The inputs provided to the autopilot are commands to either ascend, descend, or hover. The ascend or descend rates are a percentage of a fixed value of 100 cm/s, which was set as a default parameter in the autopilot and found to be satisfactory. Manually controlling the UAV's roll, pitch, and yaw axis, the external down control loop's proportional and integral terms were adjusted until the UAV was capable of maintaining its altitude within ± 5 cm.

Yaw was the second axis to be tuned. Observing the UAV during a manual controlled flight, the system was naturally stable and required minimal input from the hand controller to keep a constant heading; however, the system was still required to rotate to desired set points and be robust to disturbances. To tune yaw, the proportional term was increased until the system began to oscillate, at which time the value was reduced by a factor of four and a small amount of integral was added. Several flights were conducted with slightly higher and lower gains, and the flight logs were analyzed to find the combination which had the least tracking error and fastest response time. The yaw controller was found not to need a derivative term, similar to the native ArduCopter yaw controller. Under normal operating conditions, the controller achieved $\pm 10^{\circ}$ accuracy or better when attempting to maintain a fixed heading.

The tuning of the north and east (pitch and roll) axes was an iterative process. While one axis was unstable, it was extremely challenging to tune the other. Therefore, the axes were tuned independently, and when one axis was brought to a reasonably controlled state, the other axis was retuned, should the results become coupled. The general approach for tuning roll and pitch was to introduce a small step response and observe the output. The proportional term was to be increased until steady oscillations were observed. Upon observing oscillations, the value was slightly reduced, and a derivative term was added to critically dampen the oscillations. The process was repeated until the system response consistently responded to the step response in a desired time. Steady-state error was then corrected with a relatively small amount of integral control.

Using the proposed approach, the north and east axes were iteratively tuned. After more than 100 flights of dedicated north and east gain tuning, the current UAV configuration and PID controller were determined to be insufficient to provide robust, consistent centimetre accuracy. Many factors led to the conclusion; however, the UAV was tuned to be as robust and accurate as possible by comparing errors and the system response across different combinations of gains.

Figure 5.16 is a plot of the response of a fully tuned north axis being given three individual step responses during a single flight. The UAV was flown to be within visual proximity of the ArUco fiducial markers (and remain in proximity after a step response) and was stabilized manually to provide the system with near-zero initial conditions in terms of velocity and acceleration. The UAV

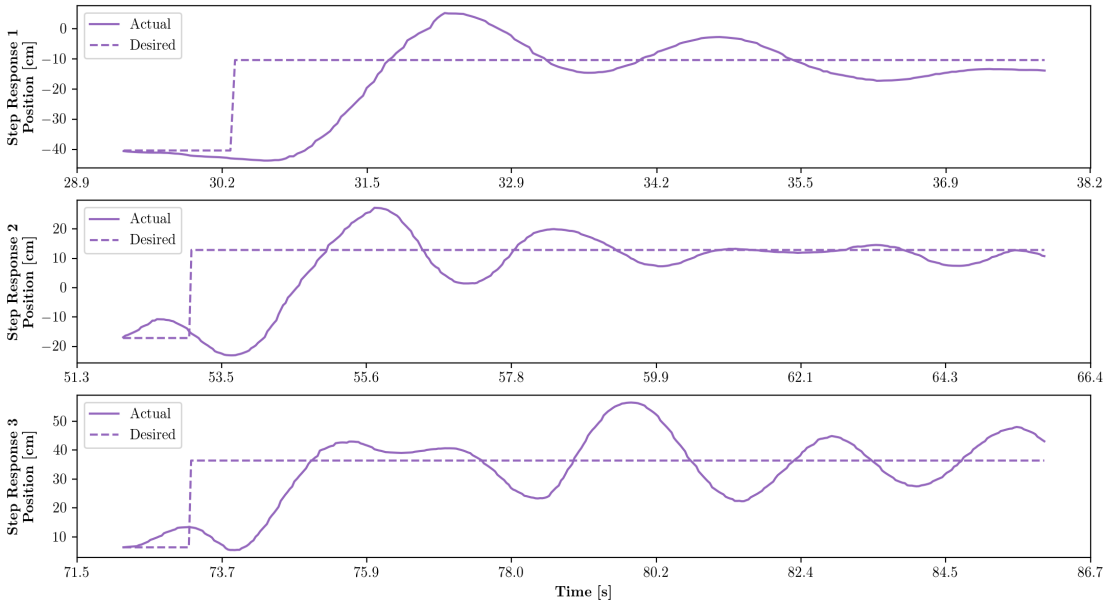


Figure 5.16: Three North 30 cm Step Responses During Single Flight

was switched to autonomous mode, the PID controller assumed command, and the step input was given one second later. In the first step response, the system overshot by 15 cm, and 7.5 seconds from the initial step, there was an error of approximately 5 cm. The second response had the same 15 cm overshoot and, after 8 seconds, achieved sub centimetre accuracy; however, after 2 seconds, the system was disturbed and began to oscillate with a 5 cm error. The third step response only overshot by 7 cm; however, instead of recovering, the system got stuck in continuous oscillations. When observing plots for the east axis, a very similar trend is observed, although the gains slightly differ due to the non-symmetrical airframe with varying moments of inertia.

The plot demonstrates the UAV and control scheme's ability to achieve the desired accuracy at times; however, it also demonstrates that the system does not always function to satisfactory levels. Limitations of the system are related to the controller being sensitive to initial conditions. If the system's initial conditions strayed too far from zero, the controller could not overcome the body's inertia, until there was a large error. Due to the large error and the required compensation angle, the system would overshoot in the opposing direction, causing undamped oscillations.

An additional challenge was the robustness of the controller. When introducing small disturbances, the system would return to its desired position; however, the UAV had to maneuver slowly to prevent the system from becoming unstable. Due to the UAV's thrust in the enclosed testing area, the air became highly turbulent and had a noticeable effect on the UAV. Quantifying the turbulent

air is nontrivial; however, qualitatively, the longer the UAV was in the air, the more challenging it was to control. The self-generated disturbances are possibly one reason the system is non-uniformly oscillating in the third step response. Increasing the integral term did not increase robustness, since the noisy airflow came from random directions, yielding non constant steady-state error.

Overall, the most significant challenge was having the system be responsive enough to handle disturbances and achieve navigation without increasing gains to a level that put the UAV into a perpetual oscillation. To achieve the desired response, flight logs were carefully analyzed to see how the pilot took control of the system once the PID controller had become unstable. The response was nonlinear, suggesting the need for a more advanced controller should the UAV be used in imperfect flight conditions.

Comparing the controller's responses to successful PID implementations in literature [80, 81, 82, 83, 84], the developed system was found to have very similar oscillations and overshoot to the literature when functioning correctly. Even trends similar to the third response from Figure 5.16 were found satisfactory for some applications. When comparing the accuracy of the different implementations in literature, systems that yielded the best accuracy were the smallest UAVs, some of which were less than half the developed system's size and mass. Although the developed PID controller did not function as robustly as predicted, the system achieved centimetre accuracy in ideal scenarios.

In addition to tuning the PID controller gains, various terms were constrained and filtered to aid in the controller's response. Bounds were set as extreme points to protect the UAV from crashing, and were rarely reached. If an extreme point was reached, the system was likely not operating in its optimal conditions or the system was experiencing artifacts of noise. For example, the derivative term range was set to limit the derivative term's contribution when the error suddenly changed, such as a step response. Additionally, the derivative term was filtered with a low-pass moving average filter to prevent the amplification of noise which the Kalman filter may have missed, or in case the sampling frequency became irregular. Integral terms were bounded to reduce integral windup. A summary of the values and the PID controller's final gains for each axis can be found in Table 5.2.

Table 5.2: PID Gain, Bounding, and Filtering Summary Table

Axis	**P Gain**	**I Gain**	**D Gain**	**Output Range**	**I Range**	**D Range**	**D Filter**
North	0.170	0.015	0.080	[-4°, 4°]	[-200, 200]	[-125, 125]	$n = 5$
East	0.160	0.015	0.110	[-4°, 4°]	[-200, 200]	[-125, 125]	$n = 5$
Down	0.002	8E-5	-	[0, 1]	[-500, 500]	-	-
Yaw	0.200	0.020	-	[-10°s^{-1}, 10°s^{-1}]	[-500, 500]	-	-

5.5 Visual Servoing Landing

With the PID controller tuned to be as functional as possible, the UAV was tasked with demonstrating its ability to land in the payload docking station. In place of a step response, trajectory generation was enabled for the north, east, and down axes to smooth the positional controller's desired set points. The trajectory only used positional data and no velocity or acceleration, as these were too noisy. Filters were trialled to smooth the velocity and acceleration data; however, the filters yielded too large of a phase lag to represent the UAV's kinematics accurately. Additionally, the UAV was manually controlled to minimize initial velocity and acceleration of the UAV.

As with PID gain tuning, the UAV was manually flown in visual proximity of the ArUco markers and manually stabilized to approximately zero initial velocity and acceleration conditions. The moment the UAV was set to autonomous mode, a trajectory was generated, and the UAV was entirely in control. To quantify the accuracy and repeatably of landing, two fully charged 5000 mAh Goldbat 10042126 three-cell LiPo batteries were flown one at a time until the measured voltage was 3.75 V/cell, which is when the UAV began to underperform. The batteries yielded 18 flights where one successful landing was made. The results of the successful flight is plotted in Figure 5.17.

The north and east axes had trajectories lasting four seconds in duration, while the down axis was eight. The theory was that the UAV would arrive at the landing position before it touched

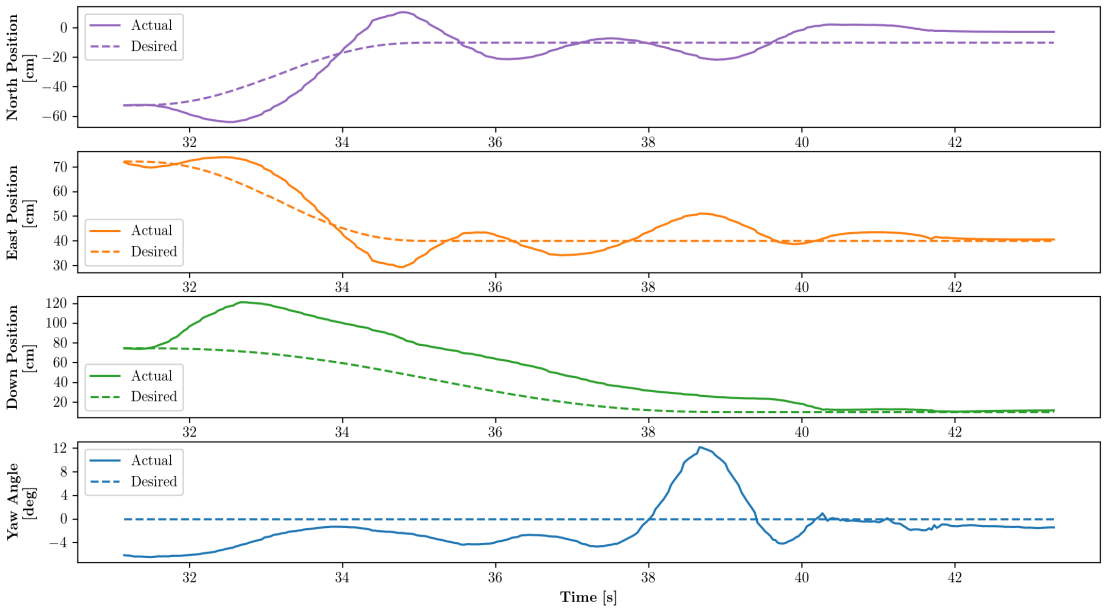


Figure 5.17: UAV Trajectory Tracking and Successful Landing Plot

down and have enough time to compensate for errors. Both the north and east axes exhibit a similar flight profile of overshooting their trajectory and then proceeding to have a low-frequency oscillations about their respective desired point. Once landed in the payload docking station, the east axis shows zero error; however, the north axis is off by 8 cm. The offset is likely due to the fiducial marker being lost during the docking process and the UAV saving its last position, which was incorrect, while the other axes were already correctly aligned. The down axis begins with an overshoot and then tracks the trajectory with a constant steady-state error until landing. The overshoot was due to the MAVLink connection not clearing its buffer before switching flight modes, causing a momentary rise in the throttle. It is not common for the error to occur; however, the system recovers quickly. Lastly, the heading begins with a 6° error, and the control slowly compensates for it. Nearing the landing, a 12° disturbance was introduced, likely due to ground effect, and the controller quickly compensated before the UAV landed about two seconds later. The unsuccessful landings display very similar plots to Figure 5.17, but with slightly more error and larger oscillations. The system was in control for each of the trials; however, it rarely had the ideal conditions required by the controller to execute a precision landing.

A positional scatter plot of all attempted landings is observed in Figure 5.18. The figure also includes a photo of one of the unsuccessful landings to provide a visual scale. It should be noted that the landing locations recorded are where the UAV came to rest and not the original locations where the UAV touched down. At least a third of the UAV trials bumped into the 3D-printed funnels or aluminum extrusions, which caused the system to be disturbed and veer out of control to its final position.

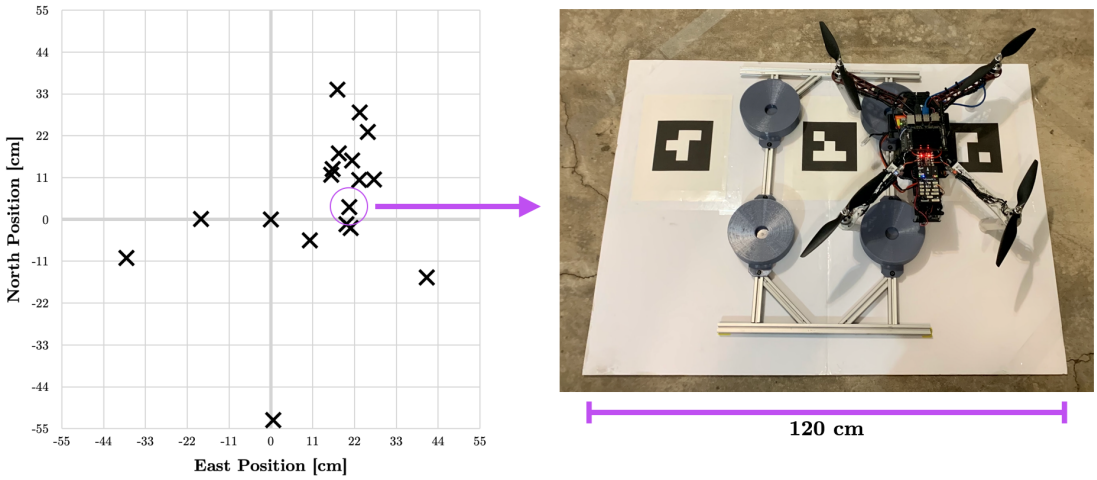


Figure 5.18: Landing Locations Scatter Plot and Failed Landing for Scale

The majority of the failed landings were concentrated to the upper right of the desired landing location. There were additional scattered landings that were reasonably accurate in the north axis but off in the east axis, and there was one outlier that was accurate in the east direction; however, it had more than a 0.5 meter error in the north direction. Considering all 18 trials, the north error was calculated to be 4.5±19.4 cm, the east error 13.6±18.1 cm, and the diagonal distance error 26.9±12.4 cm. Although the landings did not achieve the desired accuracy, the system was shown to be functional.

5.6 Aerial Manipulation

5.6.1 Payload Swapping

The sequence for retrieving or returning a payload begins by landing the UAV in the payload docking station. The UAV internally monitors the error and location of the UAV, and if the UAV's pose error is within a specified tolerance and is below a set altitude, a timer is initiated. A timer is used

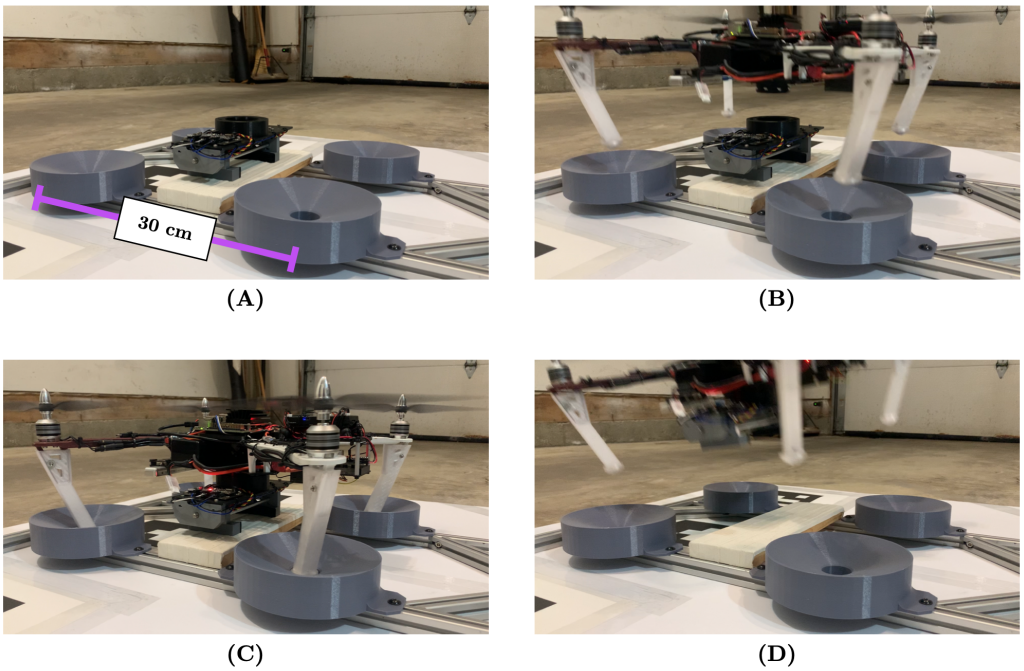


Figure 5.19: Payload Retrieval Flight Sequence: (A) Payload to be retrieved, (B) UAV approaching payload docking station, (C) Engaging Quick Connect and verifying connection, (D) Takeoff and begin mission

to ensure the UAV is in its given location and that system noise did not prompt a false response. Once the timer has exceeded 1.5 seconds, the quick connect engages or releases the payload.

Due to the shortcomings of the implemented PID controller and the corresponding inaccuracy when landing the UAV, a complete autonomous payload retrieval was not achieved. To demonstrate the system, a semi-autonomous verification was conducted, as in Figure 5.19. The UAV was manually flown and landed in the payload docking station. The UAV was then switched to autonomous mode with the desired set points of the manual landing. As the system achieved its desired position, the self-checks were completed, and the payload was grasped by the quick connect. Because the quick connect has indicator LEDs, the system was shown to be ready, and the UAV was again switched into manual mode and flown from the docking station.

5.6.2 Cap Removal

Similar to the payload swapping experiment, the developed navigation system could not reliably perform the positioning requirements demanded to remove a cap; therefore, the experiment was manually performed. Four circular caps measuring 3.9, 5.4, 6.6, and 9.1 cm in diameter were selected for removal to show the cap manipulator's spanning capabilities. As with the lack of standardization of the sump cavity, no standard cap torque specification was found; thus, each cap was tightened to its reservoir so that fluid would not escape if inverted. The UAV was then placed on the cap, the manipulator was closed, and the UAV was manually yawed, yielding four successfully removed caps. Photos of the UAV removing the four caps are observable in Figure 5.20.

During testing, three observations were made about the developed cap manipulator payload system. Firstly, the threshold for the gripping strength was found to be cap-dependent; what worked for one cap, elastically deformed another. If the gripper squeezed too tight, the UAV had to fight an increased release torque, as the normal contact force had also increased. After iterating through the selected caps, a single threshold value was determined for all four; however, a more extensive study of the caps should be considered. Otherwise, the manipulator was functional and was confirmed not to be back driven when the servo was placed in an idle state. While interfaced with the quick connect, the cap manipulator had sufficient gripping strength to hold the UAV upright on the cap. The gripping provided the UAV with the maximum torque scenario where only two adjacent propellers operate, as the remaining propellers do not need to generate lift.

Secondly, the cap manipulator's designed jaws worked for clamping the four caps in the experiment; however, on the smallest-diameter cap, the jaws were rubbing on the stem of the reservoir, thereby increasing friction and making the removal of the cap more challenging. If the cap dimensions are known ahead of time, the jaws should be dimensioned relative to the cap's vertical thickness. The jaws on the manipulator are modular; however, manually changing them before a mission is

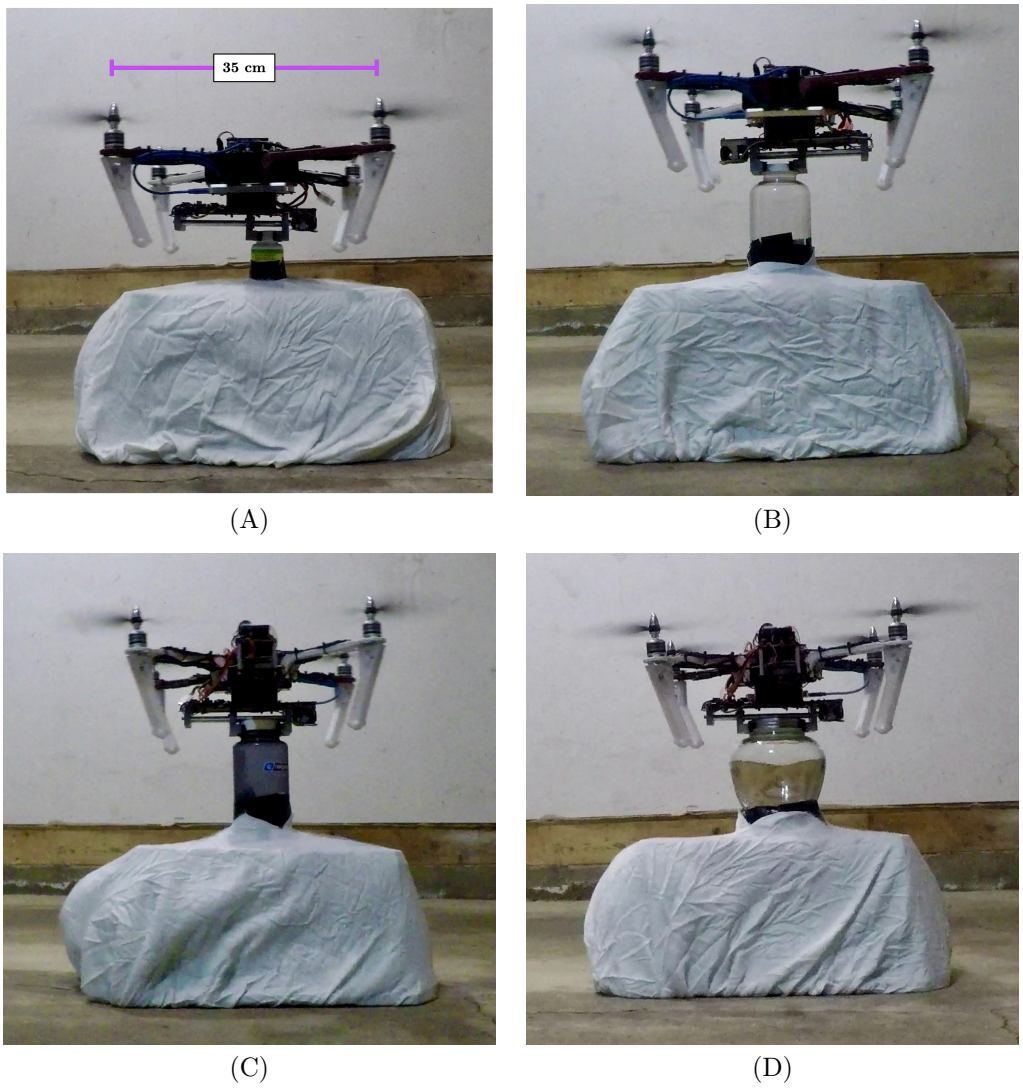


Figure 5.20: Cap Removal: (A) Small Diameter 3.9 cm, (B) Medium Diameter 5.4 cm, (C) Large Diameter 6.6 cm, and (D) Extra Large Diameter 9.1 cm

undesirable. With the quick connect system, multiple versions of the same payload can be developed and selected as desired, or swapped if the cap removal task fails.

Lastly, the UAV's maximum torque specifications were experimentally gathered by directly controlling two ESCs with a microcontroller. When connected to the autopilot, the same signal could not be achieved without modifying the internal control loops. The UAV was always trying to keep

itself stabilized, and all four motors were constantly running, reducing the maximum torque the system was capable of producing. Additionally, the coupling dynamics began to interfere with the standard control loops, causing integral windup and abnormal oscillations while the UAV was fixed but still given a command to actuate. Once the UAV overcame the initial static friction and began to rotate the cap, the controller quickly regained stability.

With the UAV, quick connect, and developed payload fully integrated and tested, the following section will summarize the presented research and make recommendations for future work.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The development of a quadcopter system capable of localizing position and heading based on ArUco fiducial markers, navigating to desired set points with a PID controller, and interfacing with various payloads was presented in detail. The development of the system was conducted using an empirical approach.

The relevant theory and methodology for localizing and controlling a system were first presented, followed by an overview of the UAV and the mechanical, electrical, and software design of the universal payload adapter, termed the quick connect. To demonstrate the quick connect's functionality, the conceptual and detailed development of an example payload for manipulating sump lubricant reservoir caps was provided. Finally, the results of experimental trials validating the developed systems was presented.

The developed software running the localizing and control algorithms was profiled, showing the system's computational performance. A multiprocess, multithreaded program was required to balance the image processing computation requirements with the I/O data streams. The raw ArUco pose calculation yielded values accurate to centimetre tolerances when within 200 cm of the ArUco markers; the maximum dimension of the environment flown in. The Kalman filter was validated using the ArduCopter firmware as ground truth, and the filter sufficiently tracked flight profiles and estimated states when data was lost. Methods for tuning the PID controller were presented. The UAV achieved centimetre accuracy in perfect conditions; however, the controller was not robust to the given environment's disturbances and had challenges overcoming the UAV's non-linear dynamics. The results were further observed when the UAV followed a trajectory to land and yielded an

average error of 26.9±12.4 cm. The UAV was unable to autonomously perform aerial manipulation tasks due to the response of the controller. Therefore, a semi-autonomous approach was taken to validate the quick connect's ability to swap payloads and have the test payload remove caps.

Although the system in its entirety did not meet the demanded performance to swap payloads and manipulate caps consistently and autonomously, the overall system demonstrated elements of functionality. The following section outlines recommendations for future work to improve the overall system; however, the following list provides a summary of the major contributions of this thesis:

- Fabricated a quadrotor UAV capable of handling small payloads ($<$ 650 grams).
- Designed, constructed, and assembled a universal payload system for transferring translational and rotational force. The universal payload system supplies electrical power to payloads and features a custom wireless communication protocol between the main control board and payload.
- Programmed a vision and inertial navigation system capable of localizing the UAV in a given environment using fiducial ArUco markers and an inertial measurement unit. A PID control scheme, including trajectory generation, was capable of manoeuvring the UAV to desired set points in ideal conditions.
- Demonstrated the quick connect, vision system, and controller capabilities by conducting experimental trials, including the deployment of an example payload for removing twist caps off simulated sump lubricant reservoirs.

6.2 Future Work

The present work leads to a number of future topics for technology development and demonstration, which are outlined in the following remaining sections.

6.2.1 Localization

Localizing the UAV using the Intel RealSense T265 camera and OpenCV ArUco library proved satisfactory for the given environment; however, the vision system can be upgraded to be more efficient and adaptable to other environments. Because the dual cameras on the T265 agreed, they should be replaced by a single higher-resolution global shutter camera. The change would yield similar processing times but allow for increased accuracy, especially when further from a fiducial marker. The resolution of the camera could be even further increased if processing times were optimized. Optimization is possible by upgrading the hardware or rewriting the Python code in C/C++, which

would be fundamentally more efficient. Additionally, many optimal C/C++ libraries do not have Python packages. For example, the NVIDIA Jetson Nano has image processing algorithms using Compute Unified Device Architecture (CUDA), which runs on the graphical processing unit (GPU) and thus frees the CPU to complete other calculations. There is also an updated ArUco library, which reported between 17 and 40 times faster marker detection than the ArUco approach currently implemented in OpenCV [130].

The Kalman filter's implementation assumed the system was linear, when in reality it was not. Implementing an extended Kalman or unscented Kalman filter would better fit the design and provide more reliable results. Alternatives to the Kalman filter should also be explored. Additionally, the Kalman filter may benefit from using a different data stream than the T265's IMU and fusion algorithms. The data being received is preprocessed by closed source software, leading to potentially undesirable coupling effects. Lastly, the current and future versions of the vision system should be validated against a reliable ground truth measurement, such as a Vicon motion capture.

6.2.2 Control

The PID controller was the overall limiting factor in achieving the desired functionality of the system. Future work will require evaluating a candidate control scheme, such as an adaptive or sliding mode controller, to address the system's nonlinearities. A potentially quicker solution would be upgrading the current PID controller with a gain schedule or cascade controller. Due to the developed program's objected-oriented nature, future developed controllers can be swapped, and the main code is not required to be rewritten. An additional method to be tested is to not perform any control and let the autopilot handle the positioning. The process would require packaging the local position data to mimic a MAVLink GPS message. Should the correct parameters be set in the autopilot, the UAV could use the fake GPS data to localize itself, allowing for autopilot-based navigation. MAVLink messages and control modes in the autopilot must also be explored so a maximum yaw command can be executed when manipulating a cap.

Additional testing of the UAV should be conducted in alternate environments to monitor the system's response with limited ground effect and external disturbances. Furthermore, the system should be tested in an outdoor setting to monitor the impact of wind and varying light conditions. Testing the handoff between GPS and autonomous modes would provide valuable insight into a given GPS's error, and the corresponding trajectories needed to be generated. Outdoor testing would require a Special Flight Operations Certificate (SFOC) due to the system's autonomous prototype nature; therefore, additional research should be conducted into the proper certifications for flying in a non-laboratory setting.

6.2.3 General System and Universal Payload Design

The fundamental quadcopter platform was an early prototype making use of budget materials and parts. Swapping from plastic injected moulded parts to carbon fibre will decrease overall mass and increase rigidity. Upgrading the motors and ESCs will also increase the overall responsiveness of the system. Integrating the electronics of the quick connect and the UAV would yield a more compact and integrated design. For example, there are three independent 5 V voltage regulators; with a well-designed PCB and careful selection of components, the same systems could run on a single regulator. Alternatively, one PCB could be created to hold all electrical components, leading to an even more compact and optimized design. To further increase the system's capabilities, an over-actuated quadcopter design can be explored in which each arm of the UAV can rotate independently of the others.

The quick connect and actuation system should be manufactured with more robust materials, such as nylon, to limit deflection and increase the maximum loading specifications of the system. The design's physical size is also scalable, and based on future payloads, it can be increased or decreased to save mass or increase the maximum loading specification. The use of Wi-Fi to communicate with payloads should be reconsidered, as the payload could communicate directly with the main program and not have to be passed through the microcontroller. The payload and quick connect system's communication protocol should be updated to use MAVLink messages instead of the custom-developed communication protocol. Having the quick connect and payloads communicate with MAVLink would allow closer integration with the autopilot and reduce the complexity of creating a standard for the quick connect system. Similarly, with the recent release of ROS2 and the increasing trend of using ROS, it is highly recommended to reevaluate the use of ROS with the UAV system and its devices.

Once the UAV can achieve consistent centimetre positional control, the system should autonomously validate payload swapping. Similarly, autonomous cap removal and replacement experiments should be conducted. The cap manipulator worked for the preliminary testing; however, additional work is required to quantify the jaws' optimal gripping strength and geometry. With the fundamental UAV and quick connect system developed, other payloads need to be designed to test the full capabilities of the localization, control, and quick connect system. Upon completion of laboratory testing, the system should be demonstrated in controlled real-world environments. With sufficient testing and implementation of safety procedures and regulations, the system will be ready for full integration into real-world workflows.

References

[1] B. Custers, *The Future of Drone Use: Opportunities and Threats from Ethical and Legal Perspectives*. Springer, Oct. 2016, google-Books-ID: WytEDQAAQBAJ.

[2] J. Buckley, *Air Power in the Age of Total War*. Routledge, May 2006, google-Books-ID: YSSPAgAAQBAJ.

[3] A. G. Renstrom, *Wilbur & Orville Wright: A Bibliography*. U. S. Centennial of Flight Commission and the National Aeronautics and Space Administration, 2002.

[4] K. L. B. Cook, "The Silent Force Multiplier: The History and Role of UAVs in Warfare," in *2007 IEEE Aerospace Conference*, Mar. 2007, pp. 1–7, iSSN: 1095-323X.

[5] J. Keane and S. Carr, "A Brief History of Early Unmanned Aircraft," in *Johns Hopkins Apl Technical Digest*, vol. 32(3):, 2013, pp. 558–571.

[6] A. F. H. a. M. P. (U.S.), *Splendid Vision, Unswerving Purpose : Developing Air Power for the United States Air Force During the First Century of Powered Flight*. Claitors Pub Div, Mar. 2003.

[7] "Captain John Alcock and Lieutenant Arthur Whitten Brown," Jul. 2014. [Online]. Available: http://www.aviation-history.com/airmen/alcock.htm

[8] National Air and Space Museum, "Doolittle and the First "Blind Flight"," 1929. [Online]. Available: https://pioneersofflight.si.edu/node/105

[9] L. S. Howeth, *History of Communications Electronics in the United States Navy*. U.S. Government Printing Office, 1963, google-Books-ID: sfI4i4znFmgC.

[10] D. O'Malley, "Vintage Wings of Canada: The Mother of All Drones," 2019. [Online]. Available: http://www.vintagewings.ca/VintageNews/Stories/tabid/116/articleType/ArticleView/articleId/484/The-Mother-of-All-Drones.aspx

[11] W. William, *Lightning Bugs And Other Reconnaissance Drones: The Can Do Story Of Ryan's Unmanned Spy Planes*, 1st ed. Aero Publishers, 1982. [Online]. Available: https://www.biblio.com/book/lightning-bugs-other-reconnaissance-drones-william/d/1092221821

[12] S. Dunstan, *Israeli Fortifications of the October War 1973*. Bloomsbury Publishing, Sep. 2012, google-Books-ID: DbLvCwAAQBAJ.

[13] R. Bartsch, J. Coyne, and K. Gray, *Drones in Society: Exploring the Strange New World of Unmanned Aircraft*. Taylor & Francis, Dec. 2016, google-Books-ID: 7CglDwAAQBAJ.

[14] R. Whittle, *Predator: The Secret Origins of the Drone Revolution*. Henry Holt and Company, Sep. 2014, google-Books-ID: 7ZZzAwAAQBAJ.

[15] T. P. Ehrhard, *Air Force UAVs: The Secret History*. Mitchell Institute Press, 2010, google-Books-ID: P2cfcgAACAAJ.

[16] "AlphaDogfight Trials Final Event," Aug. 2020. [Online]. Available: https://www.youtube.com/watch?v=NzdhIA2S35w

[17] R. Rashad, J. Goerres, R. Aarts, J. B. C. Engelen, and S. Stramigioli, "Fully Actuated Multirotor UAVs: A Literature Review," *IEEE Robotics Automation Magazine*, vol. 27, no. 3, pp. 97–107, Sep. 2020, conference Name: IEEE Robotics Automation Magazine.

[18] W. R. Young, *The Helicopters*. Time-Life Books, 1982, google-Books-ID: Jr1TAAAAMAAJ.

[19] G. J. Leishman, *Principles of Helicopter Aerodynamics with CD Extra*. Cambridge University Press, Apr. 2006.

[20] "Etienne Oehmichen (FRA) (13093)," Oct. 2017. [Online]. Available: https://www.fai.org/record/13093

[21] I. I. Sikorsky, "Direct lift aircraft," US Patent US1 994 488A, Mar., 1935. [Online]. Available: https://patents.google.com/patent/US1994488A/en

[22] P. Lambermont, *Helicopters and Autogyros of the World: Revised Edition*. Cassel Limited, 1970, google-Books-ID: u4HKuAAACAAJ.

[23] "KEYENCE Gyrosaucer TVCM 1991," Feb. 2012. [Online]. Available: https:www.youtube.comwatch?v=CEjNej4JXUE&feature=emb_logo

[24] Draganfly, "About us - Draganfly - A History of Innovation Since 1998." [Online]. Available: https://draganfly.com/about-us/

[25] B. Johnson, "CES: iPhone-controlled drone unveiled at tech show curtain-raiser," *The Guardian*, Jan. 2010. [Online]. Available: https://www.theguardian.com/technology/2010/jan/06/ces-iphone-controlled-drone

[26] L. Schroth, "Drone Manufacturer Market Shares: DJI Leads the Way - DRONEII.com," Sep. 2019, section: Research. [Online]. Available: https://www.droneii.com/drone-manufacturer-market-shares-dji-leads-the-way-in-the-us

[27] U. D. of Transportation, "UAS by the Numbers," Sep. 2020, last Modified: 2020-09-01T09:26:55-0400. [Online]. Available: https:www.faa.govuasresourcesby_the_numbers

[28] X. Ding, P. Guo, K. Xu, and Y. Yu, "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems," *Chinese Journal of Aeronautics*, vol. 32, no. 1, pp. 200–214, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1000936118301894

[29] K. Máthé, L. Buşoniu, L. Barabás, C. Iuga, L. Miclea, and J. Braband, "Vision-based control of a quadrotor for an object inspection scenario," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2016, pp. 849–857.

[30] H. Yu, W. Yang, H. Zhang, and W. He, "A UAV-based crack inspection system for concrete bridge monitoring," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Jul. 2017, pp. 3305–3308.

[31] L. F. Luque-Vega, B. Castillo-Toledo, A. Loukianov, and L. E. Gonzalez-Jimenez, "Power line inspection via an unmanned aerial system based on the quadrotor helicopter," in *MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference*, Apr. 2014, pp. 393–397.

[32] F. Ruggiero, V. Lippiello, and A. Ollero, "Aerial Manipulation: A Literature Review," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1957–1964, Jul. 2018.

[33] S. Kim, H. Seo, and H. J. Kim, "Operating an unknown drawer using an aerial manipulator," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5503–5508, iSSN: 1050-4729.

[34] H. Tsukagoshi, T. Hamada, M. Watanabe, R. Iizuka, and A. Dameitry, "Aerial manipulator aimed for door opening mission," in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, Oct. 2014, pp. 1–2, iSSN: 2374-3247.

[35] S. Shimahara, S. Leewiwatwong, R. Ladig, and K. Shimonomura, "Aerial torsional manipulation employing multi-rotor flying robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1595–1600, iSSN: 2153-0866.

[36] M. Orsag, C. Korpela, S. Bogdan, and P. Oh, "Dexterous Aerial Robots—Mobile Manipulation Using Unmanned Aerial Systems," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1453–1466, Dec. 2017.

[37] P. Agarwal and M. K. Singh, "A multipurpose drone for water sampling video surveillance," in *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*, Feb. 2019, pp. 1–5.

[38] H. Kim, H. Seo, C. Y. Son, H. Lee, S. Kim, and H. J. Kim, "Cooperation in the Air: A Learning-Based Approach for the Efficient Motion Planning of Aerial Manipulators," *IEEE Robotics Automation Magazine*, vol. 25, no. 4, pp. 76–85, Dec. 2018.

[39] R. Yajima, K. Nagatani, and K. Yoshida, "Development and field testing of UAV-based sampling devices for obtaining volcanic products," in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, Oct. 2014, pp. 1–5, iSSN: 2374-3247.

[40] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, "A UAV system for inspection of industrial facilities," in *2013 IEEE Aerospace Conference*, Mar. 2013, pp. 1–8, iSSN: 1095-323X.

[41] Y. Zhang, X. Yuan, Y. Fang, and S. Chen, "UAV Low Altitude Photogrammetry for Power Line Inspection," *ISPRS International Journal of Geo-Information*, vol. 6, no. 1, p. 14, Jan. 2017, number: 1 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2220-9964/6/1/14

[42] Y. Zefri, A. ElKettani, I. Sebari, and S. Ait Lamallam, "Thermal Infrared and Visual Inspection of Photovoltaic Installations by UAV Photogrammetry—Application Case: Morocco," *Drones*, vol. 2, no. 4, p. 41, Dec. 2018, number: 4 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2504-446X/2/4/41

[43] L. Yu, E. Yang, P. Ren, C. Luo, G. Dobie, D. Gu, and X. Yan, "Inspection Robots in Oil and Gas Industry: a Review of Current Solutions and Future Trends," in *2019 25th International Conference on Automation and Computing (ICAC)*, Sep. 2019, pp. 1–6.

[44] A. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, pp. 1483–1510, Oct. 2006.

[45] M. Dunbabin and L. Marques, "Robots for Environmental Monitoring: Significant Advancements and Applications," *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 24–39, Mar. 2012, conference Name: IEEE Robotics Automation Magazine.

[46] ASTM International, "Practice for Manual Sampling of Petroleum and Petroleum Products," ASTM International, West Conshohocken, PA, Tech. Rep. ASTM D4057-19, 2019. [Online]. Available: http://www.astm.org/cgi-bin/resolver.cgi?D4057

[47] M. Botha, "Electrical machine failures, causes and cures," in *1997 Eighth International Conference on Electrical Machines and Drives (Conf. Publ. No. 444)*, Sep. 1997, pp. 114–117, iSSN: 0537-9989.

[48] J. Yuen, S. Dwyer, A. Kotchon, W. Moussa, and M. Lipsett, *Commissioning and Vibration Isolation of a Low Cost UAS for Industrial and Environmental Remote Sensing Applications*. American Society of Civil Engineers, 2012, pp. 1380–1389. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/9780784412190.147

[49] M. Lipsett, "Using Unmanned Aerial Systems for Condition Monitoring of Rotating Equipment and Structural Health Monitoring," in *Using Unmanned Aerial Systems for Condition Monitoring of Rotating Equipment and Structural Health Monitoring*, Dayton, OH, May 2016.

[50] M. G. Lipsett, N. Olmedo, and B. Tesfay, "A Wireless System for Conducting Remote Vibration Monitoring Using a Remotely Operated Deployment Robot," in *Condition Monitor 365, Inst British Non Dest Testing*, Aug. 2017, pp. 6–9.

[51] M. J. Sherstan, R. J. Augustine, and M. G. Lipsett, "Aerial manipulation for remote robotic machinery diagnostics," *International Journal of Condition Monitoring and Diagnostic Engineering Management*, vol. 23, no. 3, pp. 31–36, 2020.

[52] M. Lipsett, R. Augustine, and M. Sherstan, "AERIAL MANIPULATION FOR REMOTE ROBOTIC MACHINERY DIAGNOSTICS," in *AERIAL MANIPULATION FOR REMOTE ROBOTIC MACHINERY DIAGNOSTICS*, King of Prussia PA, 2019, p. 18pp.

[53] M. Lipsett, "Graduates of Our Group | Michael Lipsett," 2020. [Online]. Available: https://www.mlipsett.com/research/people/recent-graduates/

[54] J. P. Queralta, C. M. Almansa, F. Schiano, D. Floreano, and T. Westerlund, "UWB-based system for UAV Localization in GNSS-Denied Environments: Characterization and Dataset (Accepted)," in *UWB-based system for UAV Localization in GNSS-Denied Environments: Characterization and Dataset*, Mar. 2020, arXiv: 2003.04380 version: 1. [Online]. Available: http://arxiv.org/abs/2003.04380

[55] D. of Defense, "GLOBAL POSITIONING SYSTEM STANDARD POSITIONING SERVICE PERFORMANCE STANDARD (5E)," Department of Defense, Washington, DC, Tech. Rep., Apr. 2020.

[56] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*. Springer Science & Business Media, Dec. 2012, google-Books-ID: F7jrCAAAQBAJ.

[57] S. K. Moore, "Superaccurate GPS Chips Coming to Smartphones in 2018 - IEEE Spectrum," Sep. 2017. [Online]. Available: https://spectrum.ieee.org/tech-talk/semiconductors/design/superaccurate-gps-chips-coming-to-smartphones-in-2018

[58] R. Ross and R. Hoque, "Augmenting GPS with Geolocated Fiducials to Improve Accuracy for Mobile Robot Applications," *Applied Sciences*, vol. 10, no. 1, p. 146, Jan. 2020, number: 1 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2076-3417/10/1/146

[59] M. G. Wing, A. Eklund, and L. D. Kellogg, "Consumer-Grade Global Positioning System (GPS) Accuracy and Reliability," *Journal of Forestry*, vol. 103, no. 4, pp. 169–173, Jun. 2005, publisher: Oxford Academic. [Online]. Available: https://academic.oup.com/jof/article/103/4/169/4598618

[60] S. Gan-Mor, R. L. Clark, and B. L. Upchurch, "Implement lateral position accuracy under RTK-GPS tractor guidance," *Computers and Electronics in Agriculture*, vol. 59, no. 1, pp. 31–38, Nov. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169907001238

[61] R. Odolinski and P. J. G. Teunissen, "An assessment of smartphone and low-cost multi-GNSS single-frequency RTK positioning for low, medium and high ionospheric disturbance periods," *Journal of Geodesy*, vol. 93, no. 5, pp. 701–722, May 2019. [Online]. Available: https://doi.org/10.1007/s00190-018-1192-5

[62] M. Bahrami and M. Ziebart, "A kalman filter-based doppler-smoothing of code pseudoranges in GNSS-challenged environments," *Proceedings of the IUN GNSS 2011 (Portland)*, vol. 3, pp. 2362–2372, Jan. 2011.

[63] D. M. Mayhew, "Multi-rate Sensor Fusion for GPS Navigation Using Kalman Filtering," Thesis, Virginia Tech, Apr. 1999, accepted: 2014-03-14T20:40:46Z. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/33808

[64] Y. Bai and Q. Bai, "Subsea Engineering Handbook," in *Subsea Engineering Handbook (Second Edition)*, 2nd ed. Boston: Gulf Professional Publishing, Jan. 2019, pp. 81–121. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012812622600004X

[65] Y. Wu, D. Zou, P. Liu, and W. Yu, "Dynamic Magnetometer Calibration and Alignment to Inertial Sensors by Kalman Filtering," *IEEE Transactions on Control Systems Technology*,

vol. 26, no. 2, pp. 716–723, Mar. 2018, conference Name: IEEE Transactions on Control Systems Technology.

[66] K. Winer, "Simple and Effective Magnetometer Calibration," Aug. 2017. [Online]. Available: https://github.com/kriswiner/MPU6050

[67] A. Shetty and G. X. Gao, "Covariance Estimation for GPS-LiDar Sensor Fusion for UAVs," in *30th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017)*, Portland, Oregon, Nov. 2017, pp. 2919–2923. [Online]. Available: https://www.ion.org/publications/abstract.cfm?articleID=15165

[68] S. Yousuf and M. B. Kadri, "Sensor fusion of INS, odometer and GPS for robot localization," in *2016 IEEE Conference on Systems, Process and Control (ICSPC)*, Dec. 2016, pp. 118–123.

[69] P. Henkel, A. Sperl, U. Mittmann, R. Bensch, and P. Färber, "Precise Positioning of Robots with Fusion of GNSS, INS, Odometry, LPS and Vision," in *2019 IEEE Aerospace Conference*, Mar. 2019, pp. 1–6, iSSN: 1095-323X.

[70] J. Hu, Z. Zhang, C. Zhao, D. Wang, B. Fan, S. Li, and Q. Pan, "A brief review on the positioning technologies for unmanned aerial vehicles," in *2017 IEEE International Conference on Unmanned Systems (ICUS)*, Oct. 2017, pp. 527–532.

[71] A. Yassin, Y. Nasser, M. Awad, A. Al-Dubai, R. Liu, C. Yuen, R. Raulefs, and E. Aboutanios, "Recent Advances in Indoor Localization: A Survey on Theoretical Approaches and Applications," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 1327–1346, 2017, conference Name: IEEE Communications Surveys Tutorials.

[72] W. Power, M. Pavlovski, D. Saranovic, I. Stojkovic, and Z. Obradovic, "Autonomous Navigation for Drone Swarms in GPS-Denied Environments Using Structured Learning," in *Artificial Intelligence Applications and Innovations*, ser. IFIP Advances in Information and Communication Technology, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, 2020, pp. 219–231.

[73] L. Yang, X. Feng, J. Zhang, and X. Shu, "Multi-Ray Modeling of Ultrasonic Sensors and Application for Micro-UAV Localization in Indoor Environments," *Sensors*, vol. 19, no. 8, p. 1770, Jan. 2019, number: 8 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/19/8/1770

[74] R. Li, J. Liu, L. Zhang, and Y. Hang, "LIDAR/MEMS IMU integrated navigation (SLAM) method for a small UAV in indoor environments," in *2014 DGON Inertial Sensors and Systems (ISS)*, Sep. 2014, pp. 1–15, iSSN: 2377-3480.

[75] A. Benini, A. Mancini, A. Marinelli, and S. Longhi, "A Biased Extended Kalman Filter for Indoor Localization of a Mobile Agent using Low-Cost IMU and UWB Wireless Sensor Network," *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 735–740, Jan. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016336977

[76] M. Brossard, S. Bonnabel, and A. Barrau, "Unscented Kalman Filter on Lie Groups for Visual Inertial Odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 649–655, iSSN: 2153-0866.

[77] A. Gautam, S. P.B, and S. Saripalli, "Autonomous Quadrotor Landing Using Vision and Pursuit Guidance," *IFAC-PapersOnLine*, vol. 50, pp. 10 501–10 506, Jul. 2017.

[78] V. Grabe, H. H. Bülthoff, and P. R. Giordano, "A comparison of scale estimation schemes for a quadrotor UAV based on optical flow and IMU measurements," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 5193–5200, iSSN: 2153-0866.

[79] S. I. Abdelmaksoud, M. Mailah, and A. M. Abdallah, "Control Strategies and Novel Techniques for Autonomous Rotorcraft Unmanned Aerial Vehicles: A Review," *IEEE Access*, vol. 8, pp. 195 142–195 169, 2020, conference Name: IEEE Access.

[80] X. Zhang, Y. Du, F. Chen, L. Qin, and Q. Ling, "Indoor Position Control of a Quadrotor UAV with Monocular Vision Feedback," in *2018 37th Chinese Control Conference (CCC)*, Jul. 2018, pp. 9760–9765, iSSN: 1934-1768.

[81] C. Karlsson, "Vision based control and landing of Micro aerial vehicles," Bachelor Thesis, Karlstad University, Department of Engineering and Physics, 2019. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-73225

[82] M. F. Sani and G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," in *2017 International Conference on Computer and Drone Applications (IConDA)*, Nov. 2017, pp. 102–107.

[83] A. Prayitno, V. Indrawati, and I. I. Trusulaw, "Comparison of PID and Fuzzy Controller for Position Control of AR.Drone," *IOP Conference Series: Materials Science and Engineering*, vol. 190, p. 012006, Apr. 2017. [Online]. Available: http://stacks.iop.org/1757-899X/190/i=1/a=012006?key=crossref.568e6d8c590266cebc4a7f5dc8705d68

[84] C. Hsiu Min, D. He, and A. Namiki, "Autonomous Target Tracking of UAV Using High-Speed Visual Feedback," *Applied Sciences*, vol. 9, p. 4552, Oct. 2019.

[85] S.-E.-I. Hasseni and L. Abdou, "Decentralized PID Control by Using GA Optimization Applied to a Quadrotor," *Journal of Automation Mobile Robotics & Intelligent Systems*, vol. 12, pp. 33–44, Jun. 2018.

[86] A. Adriansyah, S. Amin, A. Minarso, and E. Ihsanto, "Improvement of quadrotor performance with flight control system using particle swarm proportional-integral-derivative (PS-PID)," *Jurnal Teknologi*, vol. 79, Aug. 2017.

[87] S. Bari, S. S. Z. Hamdani, H. U. Khan, M. u. Rehman, and H. Khan, "Artificial Neural Network Based Self-Tuned PID Controller for Flight Control of Quadcopter," in *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, Feb. 2019, pp. 1–5, iSSN: 2409-2983.

[88] J. Dong and B. He, "Novel Fuzzy PID-Type Iterative Learning Control for Quadrotor UAV," *Sensors*, vol. 19, no. 1, p. 24, Jan. 2019, number: 1 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/19/1/24

[89] H. Du, W. Zhu, G. Wen, Z. Duan, and J. Lü, "Distributed Formation Control of Multiple Quadrotor Aircraft Based on Nonsmooth Consensus Algorithms," *IEEE Transactions on Cybernetics*, vol. 49, no. 1, pp. 342–353, Jan. 2019, conference Name: IEEE Transactions on Cybernetics.

[90] M. A. Smirnova and M. N. Smirnov, "Dynamic Modeling and Hybrid Control Design with Image Tracking for a Quadrotor UAV," *International Journal of Applied Engineering Research*, vol. 12, no. 15, pp. 5073–5077, 2017.

[91] C. Li, H. Jing, J. Bao, S. Sun, and R. Wang, "Robust H∞ fault tolerant control for quadrotor attitude regulation," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 232, no. 10, pp. 1302–1313, Nov. 2018, publisher: IMECHE. [Online]. Available: https://doi.org/10.1177/0959651818780763

[92] H. Wang, Z. Li, H. Xiong, and X. Nian, "Robust H∞ attitude tracking control of a quadrotor UAV on SO(3) via variation-based linearization and interval matrix approach," *ISA Transactions*, vol. 87, pp. 10–16, Apr. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019057818304518

[93] J. Sanwale, P. Trivedi, M. Kothari, and A. Malagaudanavar, "Quaternion-based position control of a quadrotor unmanned aerial vehicle using robust nonlinear third-order sliding mode control with disturbance cancellation," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 234, no. 4, pp. 997–1013, Mar. 2020, publisher: IMECHE. [Online]. Available: https://doi.org/10.1177/0954410019893215

[94] H. Maqsood and Y. Qu, "Nonlinear Disturbance Observer Based Sliding Mode Control of Quadrotor Helicopter," *Journal of Electrical Engineering & Technology*, vol. 15, Apr. 2020.

[95] S. Kim, S. Choi, and H. J. Kim, "Aerial manipulation using a quadrotor with a two DOF robotic arm," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 4990–4995.

[96] I. Palunko, P. Cruz, and R. Fierro, "Agile Load Transportation : Safe and Efficient Load Manipulation with Aerial Robots," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 69–79, Sep. 2012.

[97] S. Islam, P. X. Liu, and A. E. Saddik, "Observer-Based Adaptive Output Feedback Control for Miniature Aerial Vehicle," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 1, pp. 470–477, Jan. 2018, conference Name: IEEE Transactions on Industrial Electronics.

[98] Q. Wang, H. Xiong, and B. Qiu, "The Attitude Control of Transmission Line Fault Inspection UAV Based on ADRC," in *2017 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*, Dec. 2017, pp. 186–189.

[99] G. Garimella and M. Kobilarov, "Towards model-predictive control for aerial pick-and-place," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4692–4697, iSSN: 1050-4729.

[100] A. Bounemeur, M. Chemachema, and N. Essounbouli, "Indirect adaptive fuzzy fault-tolerant tracking control for MIMO nonlinear systems with actuator and sensor failures," *ISA Transactions*, vol. 79, pp. 45–61, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019057818301617

[101] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-Level Control of a Quadrotor With Deep Model-Based Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, Oct. 2019, conference Name: IEEE Robotics and Automation Letters.

[102] J. P. Rogelio and R. G. Baldovino, "Development of an automatic tool changer (ATC) system for the 3-axis computer numerically-controlled (CNC) router machine: Support program for the productivity and competitiveness of the metals and engineering industries," in *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, Nov. 2014, pp. 1–5.

[103] M. I. Gökler and M. B. Koç, "Design of an automatic tool changer with disc magazine for a CNC horizontal machining center," *International Journal of Machine*

Tools and Manufacture, vol. 37, no. 3, pp. 277–286, Mar. 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890695596000661

[104] C. Obreja, G. Stan, D. Andrioaia, and M. Funaru, "Design of an Automatic Tool Changer System for Milling Machining Centers," 2013, conference Name: Innovative Manufacturing Engineering ISBN: 9783037857861 ISSN: 1662-7482 Pages: 69-73 Publisher: Trans Tech Publications Ltd Volume: 371. [Online]. Available: https://www.scientific.net/AMM.371.69

[105] A. I. Automation, "ATI Industrial Automation: Automatic / Robotic Tool Changers," 2020. [Online]. Available: https://www.ati-ia.com/Products/toolchanger/robot_tool_changer.aspx

[106] D. Gyimothy and A. Toth, "Experimental evaluation of a novel automatic service robot tool changer," in *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Jul. 2011, pp. 1046–1051, iSSN: 2159-6255.

[107] B. H. Wilcox, "ATHLETE: A limbed vehicle for solar system exploration," in *2012 IEEE Aerospace Conference*, Mar. 2012, pp. 1–9, iSSN: 1095-323X.

[108] J. Hricko and S. Havlik, "Exchange of Effectors for Small Mobile Robots and UAV: Proceedings of the 27th International Conference on Robotics in Alpe-Adria Danube Region (RAAD 2018)," in *Mechanisms and Machine Science*, Jan. 2019, pp. 308–315, journal Abbreviation: Mechanisms and Machine Science.

[109] N. Reker, D. Troxell, and D. Troy, "Universal UAV Payload Interface," Ph.D. dissertation, California Polytechnic State University - Computer Engineering, San Luis Obispo, Jun. 2015. [Online]. Available: https://ardupilot.org/dev/docs/ros.html

[110] T. T. DUNKELBERGER and J. Adkins, "Layered architecture for customer payload systems," US Patent US9 981 740B2, May, 2018. [Online]. Available: https://patents.google.com/patent/US9981740B2/en

[111] H. Ghazzai, H. Menouar, and A. Kadri, "On the Placement of UAV Docking Stations for Future Intelligent Transportation Systems," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, Jun. 2017, pp. 1–6.

[112] Z. Jiang, "An autonomous landing and charging system for drones," Thesis, Massachusetts Institute of Technology, 2019, accepted: 2019-11-22T00:03:11Z. [Online]. Available: https://dspace.mit.edu/handle/1721.1/123029

[113] K. A. O. Suzuki, P. Kemper Filho, and J. R. Morrison, "Automatic Battery Replacement System for UAVs: Analysis and Design," *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1, pp. 563–586, Jan. 2012. [Online]. Available: https://doi.org/10.1007/s10846-011-9616-y

[114] B. Michini, T. Toksoz, J. Redding, M. Michini, J. How, M. Vavrina, and J. Vian, "Automated Battery Swap and Recharge to Enable Persistent UAV Missions," in *Infotech@Aerospace 2011*, ser. Infotech@Aerospace Conferences. American Institute of Aeronautics and Astronautics, Mar. 2011. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2011-1405

[115] T. Ricci, "Unmanned Aerial Vehicles Soar High," Nov. 2012. [Online]. Available: /topics-resources/content/Unmanned-Aerial-Vehicles-Soar-High

[116] A. Ollero, G. Heredia, A. Franchi, G. Antonelli, K. Kondak, A. Sanfeliu, A. Viguria, J. R. M.-d. Dios, F. Pierri, J. Cortes, A. Santamaria-Navarro, M. A. T. Soto, R. Balachandran, J. Andrade-Cetto, and A. Rodriguez, "The AEROARMS Project: Aerial Robots with Advanced Manipulation Capabilities for Inspection and Maintenance," *IEEE Robotics Automation Magazine*, vol. 25, no. 4, pp. 12–23, Dec. 2018.

[117] A. Marut, K. Wojtowicz, and K. Falkowski, "ArUco markers pose estimation in UAV landing aid system," in *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, Jun. 2019, pp. 261–266, iSSN: 2575-7490.

[118] B. H. Y. Alsalam, K. Morton, D. Campbell, and F. Gonzalez, "Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture," in *2017 IEEE Aerospace Conference*, Mar. 2017, pp. 1–12.

[119] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, 2014.

[120] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming," *Pattern Recognition*, vol. 51, Oct. 2015.

[121] G. Bradski, "The OpenCV Library (2020: v4.3.0)," *Dr. Dobb's Journal of Software Tools*, 2000.

[122] J. Kannala and S. Brandt, "A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, pp. 1335–40, Sep. 2006.

[123] J. Courbon, Y. Mezouar, L. Eckt, and P. Martinet, "A generic fisheye camera model for robotic applications," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2007, pp. 1683–1688, iSSN: 2153-0866.

[124] OpenCV, "OpenCV v4.3.0 Documentation," 2020. [Online]. Available: https://docs.opencv.org/4.3.0/d9/d0c/group__calib3d.html

[125] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab," 2004. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

[126] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[127] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, Apr. 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0734189X85900167

[128] D. H. Douglas and T. K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature," *The International Journal for Geographic Information and Geovisualization 2*, vol. 10, pp. 112–122, 1973.

[129] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979, conference Name: IEEE Transactions on Systems, Man, and Cybernetics.

[130] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded Up Detection of Squared Fiducial Markers," *Image and Vision Computing*, vol. 76, Jun. 2018.

[131] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[132] S. Sarkka, *Bayesian Filtering and Smoothing*. Cambridge: Cambridge University Press, 2013. [Online]. Available: http://ebooks.cambridge.org/ref/id/CBO9781139344203

[133] Y. Kim and H. Bang, "Introduction to Kalman Filter and Its Applications," *Introduction and Implementations of the Kalman Filter*, Nov. 2018, publisher: IntechOpen. [Online]. Available: https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications

[134] W. Bath and J. Paxman, "UAV localisation & control through computer vision," *ARC Centre of Excellence in Autonomous Systems CAS*, Jan. 2004.

[135] G. McCaldin, "Omnimac 3DR APM Anti Vibration Mount." [Online]. Available: https://www.thingiverse.com/thing:160655

[136] MAVLink, "MAVLink Developer Guide," 2020. [Online]. Available: https://mavlink.io/en/

[137] ArduPilot, "Ground Effect Compensation — Copter documentation," 2020. [Online]. Available: https://ardupilot.org/copter/docs/ground-effect-compensation.html

[138] NVIDIA, "Jetson Nano Developer Kit," Mar. 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[139] Hardkernel, "ODROID-C4 w/ Performance," 2020. [Online]. Available: https://www.hardkernel.com/shop/odroid-c4/

[140] R. P. Foundation, "Raspberry Pi Community - Projects, Blogs, and more," 2020. [Online]. Available: https://www.raspberrypi.org/community/

[141] Hardkernel, "ODROID - Index page," 2020. [Online]. Available: https://forum.odroid.com/

[142] D. T. ArduPilot, "Companion Computers: Recommended Parts," 2020. [Online]. Available: https://ardupilot.org/dev/docs/odroid-via-mavlink.html#recommended-parts

[143] "IntelRealSense/librealsense: T265_sensor_extrinsics.png," Aug. 2020. [Online]. Available: github.com/IntelRealSense/librealsense/blob/master/doc/img/T265_sensor_extrinsics.png

[144] F. Enner, "A Practical Look at Latency in Robotics : The Importance of Metrics and Operating Systems," Sep. 2016. [Online]. Available: https://ennerf.github.io/2016/09/20/A-Practical-Look-at-Latency-in-Robotics-The-Importance-of-Metrics-and-Operating-Systems.html

[145] A. Baatout, "Multithreading VS Multiprocessing in Python," Dec. 2019. [Online]. Available: https://medium.com/contentsquare-engineering-blog/multithreading-vs-multiprocessing-in-python-ece023ad55a

[146] E. ToolBox, "Friction and Friction Coefficients," 2004. [Online]. Available: https://www.engineeringtoolbox.com/friction-coefficients-d_778.html

[147] TMRh20, "Optimized high speed nRF24L01+ driver class documentation: Optimized High Speed Driver for nRF24L01(+) 2.4GHz Wireless Transceiver," 2020. [Online]. Available: http://tmrh20.github.io/RF24/

[148] ——, "Newly Optimized RF24Network Layer: Network Layer for RF24 Radios," 2020. [Online]. Available: http://tmrh20.github.io/RF24Network/

[149] H. W. Wopereis, W. L. W. v. d. Ridder, T. J. W. Lankhorst, L. Klooster, E. M. Bukai, D. Wuthier, G. Nikolakopoulos, S. Stramigioli, J. B. C. Engelen, and M. Fumagalli, "Multimodal Aerial Locomotion: An Approach to Active Tool Handling," *IEEE Robotics Automation Magazine*, vol. 25, no. 4, pp. 57–65, Dec. 2018.

[150] T. Ikeda, S. Yasui, M. Fujihara, K. Ohara, S. Ashizawa, A. Ichikawa, A. Okino, T. Oomichi, and T. Fukuda, "Wall contact by octo-rotor UAV with one DoF manipulator for bridge in-

spection," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5122–5127, iSSN: 2153-0866.

[151] Robotiq, "2F-85 and 2F-140 Grippers," 2019. [Online]. Available: https://robotiq.com/products/2f85-140-adaptive-robot-gripper

[152] C. Yih, "Flight control of a tilt-rotor quadcopter via sliding mode," in *2016 International Automatic Control Conference (CACS)*, Nov. 2016, pp. 65–70.

[153] M. Kamel, S. Verling, O. Elkhatib, C. Sprecher, P. Wulkop, Z. Taylor, R. Siegwart, and I. Gilitschenski, "The Voliro Omniorientational Hexacopter: An Agile and Maneuverable Tiltable-Rotor Aerial Vehicle," *IEEE Robotics Automation Magazine*, vol. 25, no. 4, pp. 34–44, Dec. 2018.

[154] M. Orsag, T. Haus, D. Tolić, A. Ivanovic, M. Car, I. Palunko, and S. Bogdan, "Human-in-the-loop control of multi-agent aerial systems," in *2016 European Control Conference (ECC)*, Jun. 2016, pp. 2139–2145.

[155] S. Rao and R. Durgaiah, *Engineering Mechanics*. Universities Press, Nov. 2005, google-Books-ID: vRR4FKAkJl4C.

[156] N. A. Olmedo, M. Barczyk, and M. Lipsett, "Experimental determination of the inertial properties of small robotic systems using a torsion platform," *Mechanical Systems and Signal Processing*, vol. 131, pp. 71–96, Sep. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888327019303292

[157] ArduPilot, "ArduPilot." [Online]. Available: https://ardupilot.org/

[158] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey," *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019, conference Name: IEEE Access.