RVC-1701

Coordinated Quadrotor Unmanned Aerial Vehicles

A Major Qualifying Project Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
In Aerospace Engineering

By

[illegible]
Nicholas E. Green

[illegible]
Jonathan C. Griffin

[illegible]
Christopher R. Mastrangelo

[illegible]
Keith P. Rockwood

[illegible]
Aaron M. Vien

Approved by: [illegible]
Prof. Raghvendra V. Cowlagi, Advisor
Aerospace Engineering Program, Mechanical Engineering Department, WPI

Abstract

The objective of this project is to design and implement coordinated estimation and control algorithms for a pair of quadrotor unmanned aerial vehicles (UAVs). One UAV is designated as a ”carrier” (UAV-C), and primarily carries payload. The other UAV is designated as a ”sensor” (UAV-S) and carriers several sensors to perceive the environment and to track UAV-C. The estimation and control algorithms enable UAV-S to reduce navigational uncertainty for UAV-C, which has a relatively small number of sensors. The sensors installed on UAV-S include a LiDAR depth sensor and a camera. An open-source vision-based tracking software package called Apriltag is used to enable tracking of UAV-C by UAV-S. The commercially available 3DR Solo UAV is used as a platform. For each UAV, this platform is modified by attaching a Raspberry Pi embedded computer, which provides high-level flight commands to the native autopilots. A socket server is developed to enable direct wireless communication between the two embedded computers. The linear quadratic regulator (LQR) and Extended Kalman Filter (EKF) tools from control theory are used to design the desired estimation and control algorithms. These algorithms are implemented using the Python programming language to execute on the Raspberry Pi computers. The system is tested with a flight test involving the two UAVs following a straight line trajectory. Other laboratory bench tests, software unit tests, and flight tests are performed to validate the proposed system design.

Contents

List of Tables

Authorship Table

Section	Author	Editor	Project Work
1.1 Introduction	JG	JG, KR, CM	JG
1.2 A Brief History of the UAV	KR	JG, KR, CM	KR
1.3 Guidance, Navigation, and Control	KR	JG, KR, CM	KR
1.4 LiDAR Sensor in Navigation	NG	JG, KR, CM	NG
2.1 Sensor Configuration	NG, KR, CM	JG, KR, CM	NG,KR,CM,AV
2.2 Quadrotor UAVs	KR	All	KR
2.3 Computational Hardware Selection	AV	All	AV
2.4 Inter-UAV Communications	AV	All	AV
2.5 Electrical Powering	JG	JG, KR, CM	JG
2.6 Hardware and Sensor Mounting	CM	JG, KR, CM, NG	CM
3.1 UAV-S Installation	CM	JG, NG	CM, JG, AV
3.2 UAV-C Installation	CM	JG, NG	CM,JG, AV
3.3 Coordination Installation	AV,NG	JG,AV	NG,AV
4.1 Dynamic Model	KR	JG, NG	KR, JG
4.2 Controllers	KR	JG, NG	KR, JG
4.3 Two State Kalman Filter	RVC	JG, NG	RVC
5.1 TIER I	CM,JG	JG,KR	All
5.2 TIER II	CM,JG	JG,KR	All
6.1 Apriltag Field of View	CM	JG,KR	All
6.2 Flight testing with Apriltags	JG	KR,JG	All
6.3 Areas for Further Development	JG	KR,JG	All

1 Background

1.1 Introduction

Flight, and the pursuit thereof, has captivated mankind for centuries. More recently however, military and civilian applications call for aircraft that fly themselves (Everett and Marino pg 271, 2015), to minimize the risk inherent in flying for human pilots. Remotely operated vehicles (ROVs) and unmanned aerial vehicles (UAVs), are the products of this search.
Beyond minimizing risk for pilots, UAVs open many paths to applications previously impractical for piloted aircraft. This is evidenced by the rapid growth in the UAV market, with worldwide UAV production to more than triple in the next decade (Teal Group Corporation, 2015). Applications of UAVs include security, reconnaissance, search and rescue assistance, wildlife monitoring, crop dusting, telecommunications relay, and commonly photography. Each task has a certain set of requirements that make one type of UAV more effective than another. Tasks that require traveling long distances or require extended periods of flight, need larger UAVs with more payload capacity. These tasks are best suited for aircraft with fixed wings. On the other hand, tasks like photography or search and rescue, where maneuverability is the primary concern, are best suited for quadrotor UAVs.
Even within the quadrotor variety of UAVs, there are many different configurations and variations in design based on the desired task. Some UAVs are large with extended flight time and payload capacity increase, but come with increased cost. For a specific task there is a set of parameters that must not only be met, but optimized so that the job can be completed in the most efficient manner. Often times, large UAVs are not the most efficient tool for certain tasks where cost and size are important variables.
This project addresses the specific applications where sending a set of two smaller UAVs is monetarily savy and more efficient in terms of size and noise generation than sending one larger UAV. One possible way to split up the tasks necessary for flight is to allocate one UAV for navigation and guidance (UAV-S), and the other for payload carrying (UAV-C). The goal of this project is to create a proof of concept model showing that it is possible to have coordinated flight between two autonomous UAVs, providing support for their use as a feasible alternative to large single UAV systems. This is done by allowing the UAV-S to provide the UAV-C with a better estimate of its own position than the UAV-C can provide for itself. The project shows proof of concept by having a UAV-S guide a UAV-C from point A to B, which, with future work, means the system can be expanded to complete more difficult tasks. The project also integrates a LiDAR system onto the UAV-S to allow for that better estimate of the UAV-C position, as well as for obstacle avoidance.

1.2 A Brief History of the Unmanned Aerial Vehicle

One of the earliest predecessors to the modern day UAV dates back to 1910. Thomas R. Philips, a consulting engineer from Liverpool England, created a twenty-foot model dirigible. This dirigible was capable of hovering, steering, and conducting other aerial maneuvers like “figure eights” (New York Times, 1910). The aircraft was controlled by radio commands transmitted from a control apparatus on the ground. These control commands directly manipulated certain propellers to spin clockwise or counterclockwise. Philips designed the dirigible to contain bombs in an effort to radically change how wars were fought (New York Times, 1910). Early UAVs shared a similar purpose, often developed through research in aerial torpedoes (Everett and Marino pg. 271, 2015). Both the United States Army and Navy funded research into developing these unmanned torpedoes, however World War I ended before they could be implemented on the battlefield (Yanushevsky pg. 2, 2011). As interest was fading, these unmanned vehicles were still unable to reliably reach or intercept their respective targets.

During the interwar period, the United States military considered utilizing UAVs as training tools. The US Navy was one such military branch, but their project went dormant in 1925 after a failed test (Keane and Carr, 2013). The British Royal Air Force was not as easily dissuaded and created an aerial target during the 1920s. This target helped to train the anti-aircraft gunners of the Royal Home Fleet (Keane and Carr, 2013). The success of the British system caused the US Army and Navy to revamp their underdeveloped aerial torpedo systems and incorporate them in training. The US Navy conducted a series of UAV tests involving piloting bomb-carrying UAVs into moving naval targets (Keane and Carr, 2013). Impressed by their success, the Navy deployed UAVs in the Pacific Theater, where they conducted a number of successful missions against Japanese military targets (Keane and Carr, 2013). The US Army also attempted to design their own remote controlled bomb laden B-17 torpedoes, although their system was inoperable.

The postwar years resulted in further evolution in the design of the UAV. With the onset of the Cold War and the downing of U-2 spy planes, the US Military desired unmanned reconnaissance aircraft. Target drones were retrofitted to include spying capabilities. This time period also birthed the utilization of UAVs as weapon platforms, hunting for submarines for the Navy and launching missiles and bombs in Vietnam (Keane and Carr, 2013). Multirotors are an invention that dates back to the interwar period. Etienne Oehmichen created a rotorcraft in 1924 driven by four rotors and eight propellers. Also known as Helicopter No. 2, this rotorcraft conducted more than one-thousand successful flight tests. It also demonstrated the ability to fly in circular motion. (Spooner, 1924). The drawback to the multirotor design is that the aircraft is inherently unstable. This instability is caused by the weight of the craft being suspended between the rotors. Constant feedback adjustments to the rotor speeds are therefore necessary for level hovering and flight. However, the recent developments in modern control theory and electronic autopilots has allowed for the creation of multirotor UAVs. These autopilots can easily perform the rotor adjustments needed for hovering, leaving human pilots able to focus on guidance and navigation.

1.3 Guidance and Navigation

In the realm of fully autonomous UAVs however, there are no human pilots involved. Instead, on-board flight controllers are responsible for running guidance, navigation, and control algorithms for the aircraft. Guidance algorithms generate a trajectory for the aircraft, steering it from one location to another by determining the needed accelerations (Yanushevsky pg 4, 2011). These guidance algorithms may rely on data from navigation sensors, which provide an estimation of the aircraft's state (i.e. position and heading). The control algorithms take the trajectory generated by the guidance algorithms and utilize the system's actuators to modify the system's state to follow the desired trajectory. For fixed wing UAVs these actuators are elevators, ailerons and rudders, while for multirotors the actuators are the motors of the aircraft (Yanushevsky pg 5. 2011). These control algorithms may also incorporate feedback from navigation sensors to determine when the desired state has been achieved. Working in tandem, the guidance and control algorithms along with appropriate sensor measurements enable autonomous vehicles to conduct a multitude of tasks without the aid of human operators.

1.3.1 Navigational Uncertainty

The goal of any type of navigation system is to determine the position of a location within a certain coordinate frame (Rohel pg. 1, 1937). Navigation systems use a variety of methods to make this determination, including celestial navigation, radio navigation, and inertial navigation. Quadcopter UAVs typically use a combination of radio navigation from GNSS satellites and inertial navigation from on-board inertial measurement units (IMUs). While either one of these sensors would in theory enable accurate navigation, in practice all sensors exhibit measurement noise. For GNSS sensors, this uncertainty primarily stems from atmospheric delays and multipath errors. Uncertainty in IMU based navigation stems from integration of their starting conditions, which means noise in the initial condition measurements causes errors to propagate through time. This uncertainty from navigational sensors limits the determination of concrete position for a location. Instead, the navigational sensors provide a best estimate and the algorithm generates a range of possible positions, which we refer to as a "ball of uncertainty" for the true state of the location. This ball of uncertainty can be reduced by using more precise sensors, or by applying estimation techniques such as a Kalman Filter.

1.3.2 Kalman Filtering

The Kalman filter is an algorithm that incorporates a predictive model and a corrective model to attain a more accurate estimation of a system's state. (Bishop & Welsh pg.26, 2001). The predictive model takes the state and covariance estimates from the last time step and predicts what they should be at the current time step. Next a Kalman gain is computed based on the estimate error covariance and measurement error covariance. This gain constant is then used in a recursive step that compares both the predicted state and estimate error covariance with the measured state and error covariance. Ultimately this comparison yields updated state values and estimate error covariance. These updated values are then fed into the predictive model, and the process repeats. In this way the most accurate value, or best guess, of the actual position is found, given the sensors available to the UAV.

1.3.3 LiDAR Sensor in Navigation

Light Detection and Ranging, or LiDAR, sensors are one of the many ways to determine distances to objects. LiDAR functions by emitting brief pulses of light and recording the time it takes for the particles to be reflected. LiDAR was first used in the early 1960s by the National Center for Atmospheric Research (NCAR). These sensors were used to measure the distance to clouds in the sky (Goyer, 1963). LiDAR works in a similar fashion as radar, which uses radio waves to determine distances from a point source, whereas LiDAR uses lasers of varying frequencies. One of the advantages of LiDAR is the degree of accuracy that can be obtained using this sensor. LiDAR can be used to map the terrain below an aircraft with a resolution of 30cm or less (Carter, 2012). Due to the accuracy obtained by these sensors, LiDAR is the premier method of creating digital elevation models. LiDAR is also used for obstacle avoidance on autonomous aircraft, including the Boeing AH-6 helicopter, which uses LiDAR to navigate without a pre-programmed flight path (Koski, 2010).

1.4 Multiple Unmanned Air Vehicle System

The most influential parameter to consider when designing a small-scale quadrotor UAV system, is the relationship between the vehicle's maximum thrust and weight. An optimally stable configuration would allow the quadrotor to roll and pitch to move transitionally without immediately losing altitude. This optimally stable configuration is typically met when the maximum thrust is equal to twice the weight of the quadcopter. This relationship is important to remember in light of modern UAV trends, namely that many companies are envisioning using quadcopters as delivery vehicles (Anthony, 2013). On account of the desired thrust to weight ratio, any added payload requires twice as much additional thrust. This increased thrust in turn requires a stronger, and most likely a heavier, frame to deal with the increased bending moments from the motors.
Another consideration for small scale UAV systems is the need for on-board computing and sensors. Frequently, additional sensors must be attached to UAVs to conduct specific tasks, for instance camera gimbals for photography or LiDAR for surveying. In turn, computational power is required to control these sensors and process their output. The flight controllers incorporated in commercial UAVs are typically too specialized to handle these computing requirements, and so an additional computer must be attached to the UAV. The UAV must also carry the mounts and cases for these often fragile devices. These various additions to the UAV quickly alter the moments of inertia of the aircraft, which can render the control algorithms to be inaccurate. Not only does the quadcopter become more complex because of this chain of modifications, its price tag grows substantially as well. An alternative to this painful process is to utilize multiple smaller quadcopters instead of a single larger one.

1.4.1 Potential of Multiple UAVS

The coordination of multiple UAVs in the same system to perform a single task allows for a unique separation of duties. This coordination may take on the form of multiple vehicles collaborating on the same specific task, such as two UAVs lifting a large net to clean pollution in the ocean. Alternatively the task could be divided up and each UAV assigned a portion to complete. This concept can be applied to package delivery, with one UAV carrying the package, while the other UAV guides and navigates the UAV pair. UAV-C is the term used for the vehicle that will transport cargo. In this way, a dual UAV system has the potential to be the most efficient tool for the described tasks, and a proof of concept is therefore necessary to determine whether a system of this nature is feasible and economical.

2 System Design and Configuration

To fully develop a multiple quadrotor UAV system there were several design features to consider. These features included sensor type, processor type, UAV communication type, powering type and method, as well as the type of mount attaching the components to the base quadrotor. Due to time constraints, designing a quadcopter from scratch was considered impractical within the time constraints of this project. Instead, the base quadcopters were purchased, selected after considering several off-the-shelf models, so that the required payload capacity was exceeded for the chosen sensors. Other design choices followed, each facilitating the subsequent decision making process, until the final system was determined.

2.1 Sensor Configuration

The sensor selection was at the crux of the overall system design, and was naturally the starting point. The system's sensors were tailored to best complete the desired task, which in the case of this project involved the UAVs navigating from one point to another while avoiding obstacles. Furthermore, the sensors needed to allow the UAV-S to measure the physical state of the UAV-C, in order to provide the UAV-C with greater positional certainty through Kalman filtering. Most quadcopters already contain GPS receivers and Inertial Measurement Units (IMUs). When combined, these sensors can compute the entire state of a quadcopter. Once the state of each UAV is effectively measured, this information can be used in conjunction with environmental information to avoid obstacles while navigating to the desired destination.

2.1.1 Obstacle Avoidance Sensors

In order to expand the possible operational environments for the UAVs, the system requires sensors for obstacle avoidance. With additional sensors the UAVs can safely operate in unknown or changing environments. LiDAR, sonar and cameras were considered for obstacle avoidance. The properties of each sensor type are summarized in Table 2.1.

Attributes	LiDAR	Sonar	Cameras
Measuring UAV-C State	Moderate	Difficult	Moderate
Positional Uncertainty	2D Ellipse	Elliptical Cone	Stereoscopic Images
Sensors Needed for 2D Operations	1	8	4 Stereoscopic
Ease of Use	Moderate	Difficult	Difficult
Sensor Weight	200g	35g each	24g each
Price (USD)	$350.00	$240.00	$210.00
Supporting Documentation	Moderate	Scarce	Plentiful

Table 2.1: Comparison of Obstacle Avoidance Sensor Attributes

As a result of the analysis presented by Table 2.1, the LiDAR is the best choice for local navigation and obstacle detection. While the LiDAR sensor is the most expensive and heaviest option, only one is needed for planar operation if mounted on a gimbal, or a fixed rotating platform mount. The LiDAR mounted on a gimbal operates by emitting a laser, measuring the time until the light returns which is fast enough for the component to take a full rotations worth of measurements in less than a second and lastly outputs indexed data of the distance the laser was able to travel before hitting something.
Most light weight cameras do not possess the refresh rate to be mounted via gimbal and produce as high of fidelity measurements as a simple laser can. As such the practice of multiple fixed sensors was evaluated. Sonar sensors' limited field of view requires eight or more copies of this component in order to cover the same 2-D plane that the LiDAR does. The cameras face a similar challenge. While lighter and cheaper than the LiDAR, there would need to be a camera on the UAV's left, right, fore and aft to achieve the same desired outcome. Additionally, these cameras would need to be stereoscopic to provide depth data and would require significant computational processing resources to process four live camera feeds simultaneously.
The RPLiDAR A2 sensor by SLAMTEC is a 2D LiDAR that is mounted onto a gimbal platform. This sensor utilizes a laser triangulation measurement system and performs well in indoor and outdoor environments without direct sunlight exposure. The sensor can operate at a maximum rate of 4000 samples per second with a range of up to 6 meters. Additionally, this sensor is advertised for many similar applications as a high fidelity, lightweight and relativity low-cost option.

After purchasing and testing the RPLiDAR A2 sensor, it was apparent that it was not feasible for the LiDAR to detect UAV-C alone. As Figure 2 reveals, it is difficult to use LiDAR to discern the location and orientation of specific objects. This meant that using LiDAR alone, UAV-C was indistinguishable from the environment. As a result, this greatly limited the ability to measure the state of UAV-C and increased the uncertainty in the measurements. Due to the described limitations of the LiDAR device additional UAV-C detection methods were considered.

2.1.2 Additional Sensors

Various sensors were considered to measure the UAV-C positional and dynamical state. Pure depth sensors were quickly ruled out, as they would have the same trouble measuring UAV-C's states as the LiDAR. To simplify the decision making process a further assumption was made for the quadrotor UAV system, that UAV-C would always be behind UAV-S. This assumption allows a single rear pointed camera on UAV-S to always have line of sight contact with UAV-C. For this purpose, two types of cameras were considered and are compared in Table 2.2.
The Raspberry Pi camera is superior as it has better documentation and is presently available from previous contributors to this project. It also previously works with the well documented Raspberry Pi computer, while the Realsense requires a special Intel processor to function.

Figure 1: SLAMTEC's RPLiDAR A2

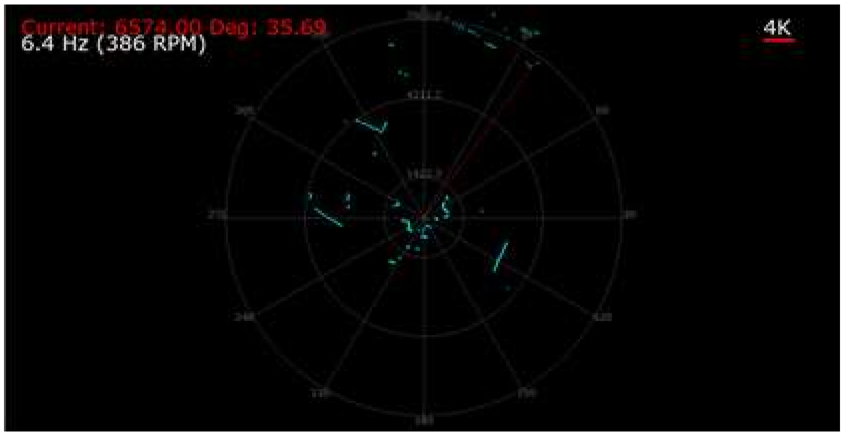


Figure 2: Example of the LiDAR outputs in SDK

2.1.3 Image Processing Software Selection

In order to gain useful information about UAV-C from the camera, image processing software is required. Two image processing libraries were considered. The first library considered was OpenCV, which allows for a high level of customization and

	Raspberry Pi Camera	Intel Realsense
Ease of measuring UAV-C state	=	=
Ease of use	Moderate	Difficult
Total sensor weight	5g	8g
Price (USD)	Already Owned	$150
Supporting Documentation	Moderate	Scarce
Processor Required	Raspberry Pi	Intel

Table 2.2: Comparison of Camera Attributes

creation. The second library considered was Apriltags, which is an application of the OpenCV library applied to specific targets. The Apriltags library was selected because it already contained image processing algorithms while OpenCV would require the creation of such algorithms from scratch. Apriltags recognizes QR code like images that can be created from a normal printer. This software provides distance and orientation data of the QR code based tags it sees. In order to utilize this feature, a tag will be placed on the front of UAV-C to enable UAV-S to calculate the heading and position of the UAV-C in UAV-S body relative coordinates.

2.2 Quadrotors as UAVs

2.2.1 Multiple Quadrotor Unmanned Air Vehicles

In order to fully justify the implementation of a multiple UAV system, a comparison of a single- and multi-UAV system must occur. In reality, larger quadcopters exist that could support the entire payload of a multiple UAV system. These larger quadcopters however, tend to be much more expensive than multiple smaller UAVs. Larger quadcopters must inherently produce more lift, which requires larger or faster spinning propellers and stronger frames to withstand the higher stresses and bending moments. All of these components drastically increase the cost of a single UAV system. Larger and faster rotating blades will also produce more audible noise, which may not be desirable depending on the system's application. Clearly it is possible, for a fraction of the cost of one larger UAV, to create a multiple UAV system of equal efficacy. This comparison alone lends value to the research conducted by this project, and defines an unfilled application for which this project seeks to produce a solution.
[1]

2.2.2 UAV-C and UAV-S Selection

Another important component of the UAV system is the selection of an appropriate pair of quadrotors. This problem was solved by finding a quadcopter that could support a projected payload of around 600g. The 3DR Solo quadcopter was selected

[1] values obtained from drones.specout.com circa October 2016

UAV model	Price ($)	Payload (g)	Manufacturer
Iris+	420	400	3dR Robotics
Solo	400	800	3dR Robotics
Ghost Basic[1]	600	1,000	Ehang
Hexacopter Hawk[1]	1,600	1,200	Skyhawk
M600 Pro[1]	5,000	6,000	DJI
Alta Hexacopter[1]	8,495	6,800	Free Fly Systems

Table 2.3: Comparison of Quadcopter Models.

as both the UAV-S and the UAV-C as each quadcopter had a total payload of approximately 700 grams, which is sufficient for the purposes of this project.
Initially, an IRIS+ was the selection for the UAV-C. This quadcopter was presently available from previous contributors to this project. After some initial flight testing, the IRIS model bequeathed proved to be be unstable as it produced unwanted drift which led to several immediate crashes.

2.3 Computational Hardware Selection

In order to achieve autonomy, it is vital that an on board computer be connected to the quadcopter. Sensors measurements must be processed by this computer to create estimates of the UAV's state. Subsequently, this computer must run scripts that will utilize these state estimations to allow for navigation and obstacle avoidance.
Two different computers were considered for use in this UAV pair: the Raspberry Pi 2 Model B and the Intel Joule. The Intel Joule was considered because it worked with the Intel Realsense Camera, while the Raspberry Pi did not. However, after selecting the Raspberry Pi Camera, the only usable computer is the Raspberry Pi Model B. Additionally, two Raspberry Pi computers were inherited from previous projects, resulting in cost savings.
The next decision was whether to use a Raspberry Pi computer on both the UAV-C and the UAV-S, or only on the UAV-S. Connecting a Raspberry Pi only on the UAV-S would maximize the payload possible on the UAV-C. This setup would require that the UAV-S constantly send commands to the UAV-C. In the event of a break in communications UAV-C would need to wait until communications were reestablished to continue navigating. Utilizing a computer on both the UAV-S and UAV-C however, enables the UAV pair to be significantly more resilient. With a computer mounted to the UAV-C, along with an IMU and a GPS receiver, UAV-C could estimate its own state and therefore continue to navigate alone. While the fidelity of these estimations would be reduced compared with working with UAV-S, UAV-C self navigation allows for contingencies that increase the likelihood of task completion.

2.4 Inter-UAV Communications

Based upon a need for flexibility and seamless dynamic motion, it is pertinent that the UAVs are connected to one another wirelessly. The two most popular wireless connection methods, Wi-Fi and Bluetooth, were considered. The selected communication method would need to be able to consistently send state and input information between the UAVs. A comparison of the characteristics of both methods is listed in Table 2.4 From Table 2.4, it is clear that both methods are

Communication type	Bluetooth	Wi-Fi
Range	10-20 meters	up to 30 meters
Data Transfer rate	up to 3 Mbps	54-150 Mbps
Reliability	Lesser	Greater
Ease of Use	Difficult	Moderate
Dongle Weight	2.5-30 grams	20-30 grams
Price (USD)	\$5-\$15	\$10-\$15
Documentation	Moderate	Well Documented
Establishing Connection	Automatic	Requires Several Scripts
Types of Connection	Direct	Direct and Infrastructure

Table 2.4: Comparison of Communication Attributes

quite similar and meet the needs of the project. Both methods had sufficient transfer speed for the exchange of simple state messages. Both methods were also lightweight and inexpensive to implement on the UAV pair. Additionally, both methods easily had enough range for the UAV pair to communicate. Ultimately, Wi-Fi was chosen as it had greater documentation. Wi-Fi supports a larger variety of connection types for testing and improving reliability. Additionally, the Solo 3DR development guide listed a plethora of supporting documentation that utilized Wi-Fi to connect, communicate and command the quadcopter.

2.4.1 Wi-Fi Connection Between Raspberry Pis

The CanaKit Raspberry Pi Wi-Fi Adapter was used to communicate between both Raspberry Pi computers. This network was created locally, such that when given a common SSID and manually specifying an IP address, they can communicate as if there were a router present. Additionally, this adapter was chosen because it supported both ad-hoc and infrastructure mode communication. Infrastructure mode communication is traditionally how Wi-Fi is used, where the adapter connects to a Wi-Fi network created by a router. An ad-hoc network removes the need for a router or mid-point ground station. Instead this communication mode involves creating a network between two or more devices. Furthermore, the compatibility of this adapter with the Raspberry Pi allowed for simple setup. The necessary drivers were already present on the previously installed Raspbian Operating System distribution.

As a result of the aforementioned network information, the Raspberry Pi computers were then connected over an ad-hoc network. Wireless encryption was not used for

the test network, though it could be adopted for additional security measures. Additionally, each Raspberry Pi computer was scripted to initialize this connection on reboot, otherwise each computer would have had to have been supplied with the connection commands manually.
Alternatively, it would also be feasible to use the infrastructure communication mode. The implementation of this mode would require a router. This router could then act as a relay and ensure that the computers were connected, increasing reliability without affecting the front-end operation. This would be useful in situations where there is a lot of wireless interference.

2.4.2 Communication Protocol with Python

To better integrate communications with MAVLink, the transport layer of communication was done in Python 2.7. MAVLink is a communication protocol that is used by the Pixhawk flight controller which came installed on both UAVs. This flight controller provides access to IMU data, and can control each UAV. Using MAVLink allows the Pi to communicate directly with the UAV, extracting location and flight information to be used in the Kalman filters and control algorithms. Updated inputs can then be generated on the Pi, and sent back to the Pixhawk over MAVLink. A unified encoding protocol was written in addition to the main communication scripts. This protocol allowed for the sending of packets approximately 4 kilobytes in size, which supported sending state and IMU data between UAVs. To support the various simultaneous operations required by the Pis in this project, the main scripts are multi-threaded. This multi-threading allows secondary commands to be sent to both UAVs to modify operation.
The Transmission Control Protocol (TCP) was chosen to send the system's data. This protocol allows the system to time-stamp when a message was received and have it wait to receive a reply. Specifically, this allowed UAV-S to listen for a reply from UAV-C about its current state. Another useful feature of TCP is that it guarantees that packets are received in the required order, preventing data from being scrambled and losing communication. While the risk of data holdups is minimal, since there are no relay points where they packets could get held up, this does still increase the time it takes for information to be sent. Unfortunately, during a communication holdup the communication will stop while waiting for the packet to timeout and announce that it has not been received.
The main script files were setup to allow a variety of communication types. The scripts can be initialized on reboot to begin communicating, and commanded to end communication if a "ground station" is not detected within a specified time. A ground station would be any user-controlled system, allowing for commands to be sent and data to be received so as to monitor the status of the system. This could be potentially expanded to process a larger volume of data, such as location information or environment simulations, before sending it to the necessary UAVs. Additionally, the scripts also possess the ability to communicate on a single response, or continuous response basis. A single response would allow simple sending of information whenever it became necessary, while a continuous response would send the information that was currently processed, even if this included new and old information. For example, a continuous response might include current

velocity but GPS from 2 seconds ago.

2.5 Electrical Powering

After selecting all of the components that were to be used in the two UAV system, the next question was how to most efficiently power the system. A power analysis was conducted, and then powering components were experimentally tested to determine the optimal configuration for powering the on-board vision sensors, computer, and other communication hardware.

2.5.1 Solo 3DR Power Analysis

Component	Current Draw (mA)	Weight (g)
Raspberry Pi	500-1200	45
Raspberry Pi Cam	250	8
LiDAR	400-1500	200
Wifi Dongle	100-500	10

Table 2.5: Component List, Current Draw, and Weight

The first step of the power analysis was to determine the minimum and maximum current draw of each component of the proposed system. This data can be seen in Table 2.5, and sums to a total of 1250-3450 mA. The power analysis compared two main methods: tapping the on-board Solo battery to power every component of the system, and adding separate external batteries. The equations from the power analysis done by the previous contributors to this project were utilized to conduct each calculation. First considering method one, Solo battery power only, the total number of milliamps were found using the following equation:

$$Total\ Current = \frac{Battery\ mAh \cdot 60\frac{min}{h}}{TOF_{unloaded}} \tag{1}$$

$$Total\ Current = \frac{5200_{mAh} \cdot 60\frac{min}{h}}{25_{min}} = 12,480mA \tag{2}$$

Next this unloaded current requirement was combined with the external component current draws. This yielded three different currents, minimum, maximum, and realistic, which equaled 13,730mA, 17,180mA, and 16,180mA respectively. Using equation (3) the time of flight was calculated for each of the three currents, and effects of weight were temporarily neglected.

$$TOF_{unweighted} = \frac{Battery\ mAh \cdot 60\frac{min}{h}}{Current} \tag{3}$$

Once the unweighted time of flight was found, the weight from added components was considered. The equation used to find the weighted time of flight is equation

Current (mA)	Unweighted Time of Flight (min)
13,730	22.72 (max)
15,930	19.59 (min)
14,930	20.90 (realistic)

Table 2.6: Unweighted Time of Flight

(4). The final time of flight calculations for the first method, entirely on-board power, are seen in Table 6. $TOF_{initial}$ refers to the 25 minute as advertised time of flight of the Solo before any modifications were made. "Solo weight" is the weight of the Solo quadcopter and mount, and "Solo actual" is the total weight of the system with everything on board. Finally the $TOF_{unweighted}$ refers to the values found in Table 2.6.

$$TOF_{weighted} = (TOF_{unweighted}) - ((\frac{TOF_{initial}}{Solo\ weight} \cdot Solo\ actual - TOF_{initial})) \tag{4}$$

Actual Time of Flight (min)
18.42 (max)
15.29 (min)
16.60 (realistic)

Table 2.7: Method One: Actual Time of Flight

Moving on to method two, powering the attached components with an external source, the calculations were much simpler and only the weight had to be varied. The equation was of the same form as equation (4), except $TOF_{unweighted}$ simply becomes $TOF_{initial}$, and "Solo actual" varies depending on the source used. This equation is equation (5) and the final results of method two can be seen in Table 2.8.

$$TOF_{weighted} = (TOF_{initial}) - ((\frac{TOF_{initial}}{Solo\ weight} \cdot Solo\ actual - TOF_{initial})) \tag{5}$$

External Source	Actual Time of Flight (min)
USB Power Pack	16.93
8 AA Batteries	17.35
Lipo Battery Option 1	18.57
Lipo Battery Option 2	19.63

Table 2.8: Method Two: Actual Time of Flight

The power analysis made it clear that an external power source would provide longer operational test flight time. Due to a bequeathed 2500 mAH dual USB port power pack and many AA batteries these powering methods were experimentally tested. This test is documented in the Appendix section of this report.
The results of the testing determined that the USB power pack can power the whole system for much longer than the time of flight. However, it is a poor choice because of its excessive weight of 240 grams. Several different configurations of AA batteries, including 4 in parallel, 6 in parallel, and 8 in parallel were tested in the same way. Sets of 4 and 6 were immediately ruled out as they could not provide enough current to the system during LiDAR startup. The 8 AA batteries functioned properly and could potentially work long enough for a whole flight test, but not within a reasonable margin of error. There was a possibility that the on-board systems would run out of power before the Solo itself, which was an unacceptable option. Furthermore, the AA battery pack was not considered to be a "scalable" solution, since even the relatively small Solo already needed at least 8 AA batteries. For these reasons the AA battery source was rejected.
Finally, the Lithium ion polymer or LiPo battery was examined. This battery type can easily provide enough electrical current, and weighs less than all other options considered. Using the LiPo battery as the source allows for the longest operational time of flight while also definitely powering the on-board system for longer than the Solo could stay in the air. The LiPo supply would power everything for approximately 20 minutes per charge assuming that all electrical components were drawing the maximum possible current at all times during a flight. This is an unrealistic assumption which means the battery has a significant safety factor, and will definitely last for the duration of any flight test. The LiPo battery is clearly superior, and was therefore selected as the final power source for the UAV-S system.

2.6 Hardware and Sensor Mounting

The final aspect of the systems physical design is a secure and balanced method of installing hardware to each quadrotor. Since both quadrotors would carry a slightly different configuration of sensors, it became necessary to perform individual flight tests.
The SOLO 3DR houses an easily accessible accessory bay on the underbelly of its chassis. This bay provides four equally spaced M2 sized tapped holes with depths of seven millimeters. These holes are spaced evenly about the Solo's center of mass and are the primary method to connect all components to this UAV.

2.6.1 Payload

The overall goal of the mounting design is to allow the Raspberry Pi computer, the Raspberry Pi camera, the RPLiDAR A2, and any additional power components to attach to the Solo 3DR without impeding any aerobatic capabilities. The maximum payload of the Solo 3DR is conservatively reported as 700 grams, which establishes weight as a major deciding factor in component selection. The weights of each of the selected sensors are reported in Table 2.9 along with the remaining conservative payload as a result of only adding the hardware.

Hardware Component	Weight
Raspberry Pi Model B	45 grams
Raspberry Pi Camera	5 grams
RPLiDAR A2	200 grams
LiPo Battery	16 grams
Remaining Payload	334 grams

Table 2.9: Sensor Weights and Remaining Payload

2.6.2 Prototyping

With a remaining payload of 334 grams, the weight of the mount is a vital parameter driving the design process. Modeling the mount in SolidWorks and selecting materials allows the weight of the mount assembly to be determined easily. GrabCAD.com was utilized in order to acquire preexisting models of each component to aid in the overall mount design. Figure 2 displays the chosen mounting system rendered in SolidWorks:

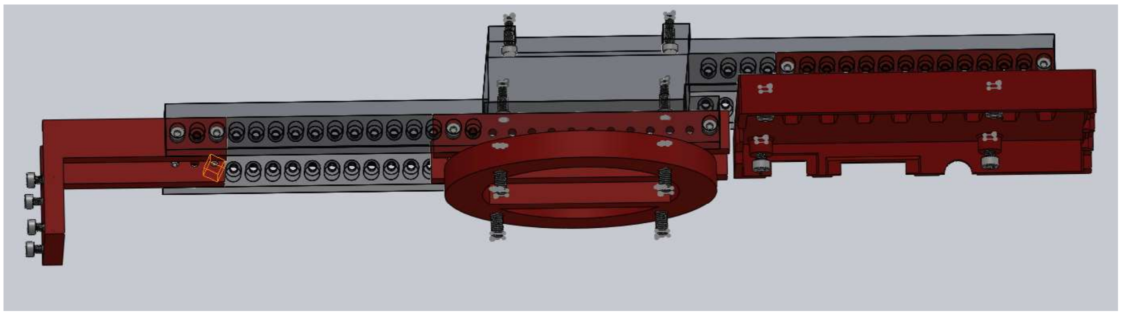


Figure 3: Partially Assembled Component Mount

In order to ensure that the mounted system will not impede flight capabilities, the UAV's moment of inertia about the center of mass must be examined. The desired moment of inertia matrix should resemble a diagonal matrix otherwise populated with zeros. This indicates that the principal axes of the system align with the axes of the body coordinate system.

$$I_{CG,Ideal} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

To evaluate the simulated values of I_{xx}, I_{yy} and I_{zz}, the *Evaluate: Mass Properties* function of SolidWorks was utilized. It is evident that electrical components, such as the Raspberry Pi computer, have nonuniform mass and as such will create non-zero values in the non-diagonal portion of the matrix.

A simulation of the system assembly was constructed after creating a crude model of the Solo 3DR quadcopter using a vernier, using a dimensioned drawing of the accessory bay area and also utilizing preexisting models of the hardware and respective mounting. Instructions for assembling the mount can be located in the Appendix section of this report. After evaluating the mass properties of the Solo 3DR assembled model via SolidWorks it was determined that the effects of the non uniform mass are practically negligible. This claim will need to be tested through hovering flight testing to evaluate the physical effect of the non-diagonal value.

$$I_{CG,Actual} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} 114.220782 & 0.045346 & 2.298259 \\ 0.045346 & 212.391757 & -0.002915 \\ 2.298259 & -0.002915 & 274.203654 \end{bmatrix}$$
$$[kg/cm^2]$$

2.6.3 Manufacturing

Worcester Polytechnic Institute houses facilities that enable students to 3D print various items in an array of various materials across various machines. There are many material options available, including high or low density ABS plastic, as well as a multitude of resin choices. Several possible materials are compared in Table 2.10. The comparisons reveal that ABS Plastic is the best choice. The largest

Material	Density	Price (USD)	Min Thickness	Error	Total Weight
ABS Plastic	$1.05 g/cm^3$	$0.49/$cm^3$	$0.1524 cm$	$0.01524 cm$	135 g
Nylon	$1.15 g/cm^3$	$1.53 – $3.06/$cm^3$	$0.1524 cm$	N/A	147.85 g
Resin	$0.961 g/cm^3$	$1.23 – $1.84/$cm^3$	$0.0254 cm$	$0.002032 cm$	123.56 g

Table 2.10: Comparison of 3D Printable Material Attributes

contributing factor is the fact that it is the least expensive option while still retaining a relatively moderate density. For this application, the minimum thickness and error are negligible. Furthermore the mount weighs considerably less than the maximum leftover payload weight of 334 grams.
The overall dimensions of each mounting component can be located as standard drawings in the appendix. The weights for each mounting component in ABS Plastic are 135 grams in total. Additionally, sixteen M2 socket head cap screws and four M3 socket head cap screws are used to fasten each component to one another weighing a total of 8 grams. This leaves a total of 191 grams of conservative payload remaining.

3 System Installation

With the physical system designed and the individual parts assembled, the next step was to begin system installation. Installation encompasses the assembly of all required subsystems such as hardware and auxiliary power components. Installation also refers to the generation of python scripts that control the system during autonomous flight.

3.1 UAV-S Installation

The UAV-S system needed to carry several sensors, the power system designed ealier, and the raspberry pi, making the design of the mount of special importance. Here, the system installation process is defined, and described.

3.1.1 Mount Attachment

The UAV-S installation began with attaching the sensor mounts to each component and fixing it to the body mount of the quadcopter. First the body mount was attached to the bottom of the Solo by means of four set screws. This mount serves as the base to which the LiDAR, raspberry pi, and pi camera attach. One of the major considerations during the design phase was how the UAV would perform with the additional sensors attached. The added mass changes affecting the inertia matrices could have potentially made the UAV harder to stabilize, or worse, dynamically unstable. In order to have greater control of the center of mass of the UAV, the mounting system was designed to allow the sensors to be mounted in multiple locations. This was accomplished by creating the base mounting structure have beams with holes along their length. Each sensor has a mounting component with similarly sized holes, allowing the sensor to be screwed onto the primary mount at any desired distance longitudinally along the base of the Solo. With the hardware attached the wires utilized for power and data transfer were organized. Due to the limited space on the UAV, compact and efficient organization of the system's cables was essential. Loose wires could interfere with the rotors and disrupt flight. As such, the mounting system was designed such that the middle of primary mount contains a channel for connections. Figure 4 is a block diagram explaining the necessary connections. Lastly, the system required several python based MAVLink packages to be installed. These packages were used to compose scripts that allow the UAV-S to fly and operate autonomously. Download instructions are detailed in the appendix.

3.1.2 Sensor Battery Mounting

As described in the power analysis, the auxiliary power system was powered by a Lithium Polymer(LiPo) battery. A 2400 mAH LiPo battery in the lab was used to power the UAV-S sensors. Originally, the LiPo battery cable had a deans type connector; this connector was removed and a TX60 header was attached to that it could interface with the rest of the on-board power system. The battery itself sits directly on top of the Solo, connected via Velcro, and therefore does not effect the center of mass in the x or y directions.

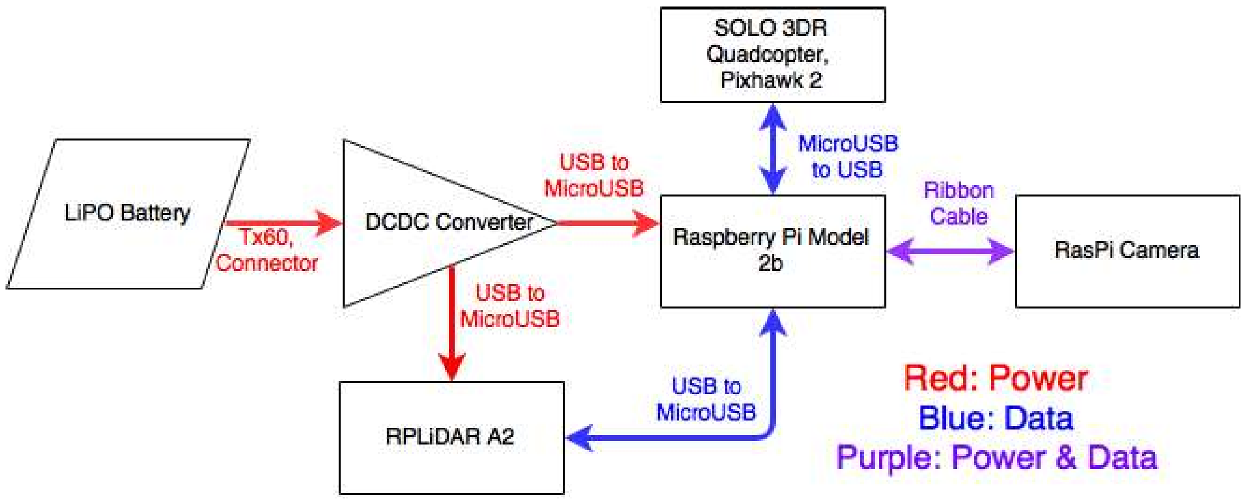


Figure 4: UAV-S system block diagram

3.1.3 MAVLink Flight Scripts

To begin the software development aspect of this project it was necessary to decide upon several benchmarks for autonomous flight. Autonomous arming was the first goal. Subsequently automated takeoff, hover and landing was the next goal. The ultimate goal for the autonomous flight scripts was to incorporate horizontal motion after hovering, and to have the UAVs follow the same path.
The MAVLink software package offers several methods of flying a quadcopter. These methods include setting global level position commands in terms of longitude and latitude, local level position commands from the inertial north end down reference system as well as setting a velocity vector from the inertial plane for a set time. It is worth noting that in order for any quadcopter to reduce drift during hovering there must be an established satellite link and as such it is difficult to operate either UAV indoors.
Final flight scripts have been included in Appendix B.

3.2 UAV-C Installation

The installation of the UAV-C was identical to the UAV-S except the LiDAR sensor and camera were not incorporated. However, a five inch by five inch Apriltag was attached to the front of the UAV.

3.2.1 MAVLink Flight Scripts

To begin the simple flight scripting aspect of the UAV-C, the scripts composed for the Solo were recycled. Once the UAV-C is able to perform simple scripted flight via python and MAVLink the resultant step will be to establish coordination between the UAV pair.

3.3 Communication Installation

The next step in developing the software aspect of this system is to write scripts that aid in the coordination of this autonomous system. This will be performed in two segments. The first segment will be to have the UAV-S follow a flight script while a Kalman filter developed in python improves the positional certainty during the mission.The second segment will be to have the UAV-S fly some designated path while the UAV-C follows it, or visa-versa also requiring an additional Kalman filter.

3.3.1 Sensor Communication and Socket Server

With the UAV-S carrying the Pi camera and LiDAR sensor, a method of sending data collected by the sensors to the Kalman filter running on the Raspberry Pi was needed. There are many algorithms of communication between processes on a computer; this report highlights three different algorithms, and the reasoning behind the final decision.

The first communication algorithm analyzed was Inter-Process Communication (IPC). IPC is an algorithm to allow a computer to coordinate multiple requests at the same time. IPC can achieve this through a the utilization of certain tools and methods to receive and process data from different sources concurrently.

The IPC algorithm contains many different methods, with the first studied being sockets, in order to process communication between different sources. Sockets are used in virtually every computer for various uses, the most common of which is the creation of a server-client system, as seen in Fig 5 below.

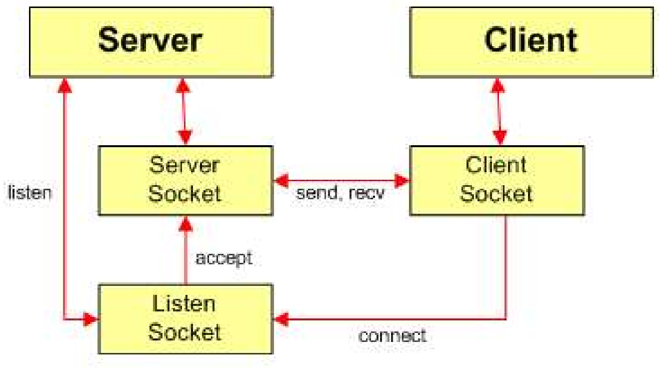


Figure 5: Socket Server Visualization

In this server-client implementation, the client - the on-board sensors - connects and continuously writes data to a socket, while the server - running on the Raspberry Pi - finds the socket and listens to receive the client data. This architecture is used for both IPC as well as network applications. Sockets can be used to send data between two different systems.

The final algorithm studied for communication between the on-board sensors and the Raspberry Pi was the Lightweight Communication and Marshalling (LCM), developed my MIT in 2006. LCM has been used in many robotic and autonomous systems, and seen use in research and products around the globe.

3.3.2 Apriltag

At the crux of this report is the Kalman filter that provides an improved estimation of each quadrotor's state. In order to determine the state of the UAV-C, the Pi camera mounted on the UAV-S is able to track and determine the distance and pose of the UAV-C. The camera is able to track the UAV-C by mounting an Apriltag, shown in Fig 6, on the front of the quadrotor, facing the camera on the UAV-S.

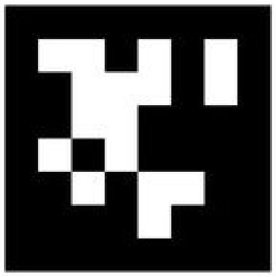
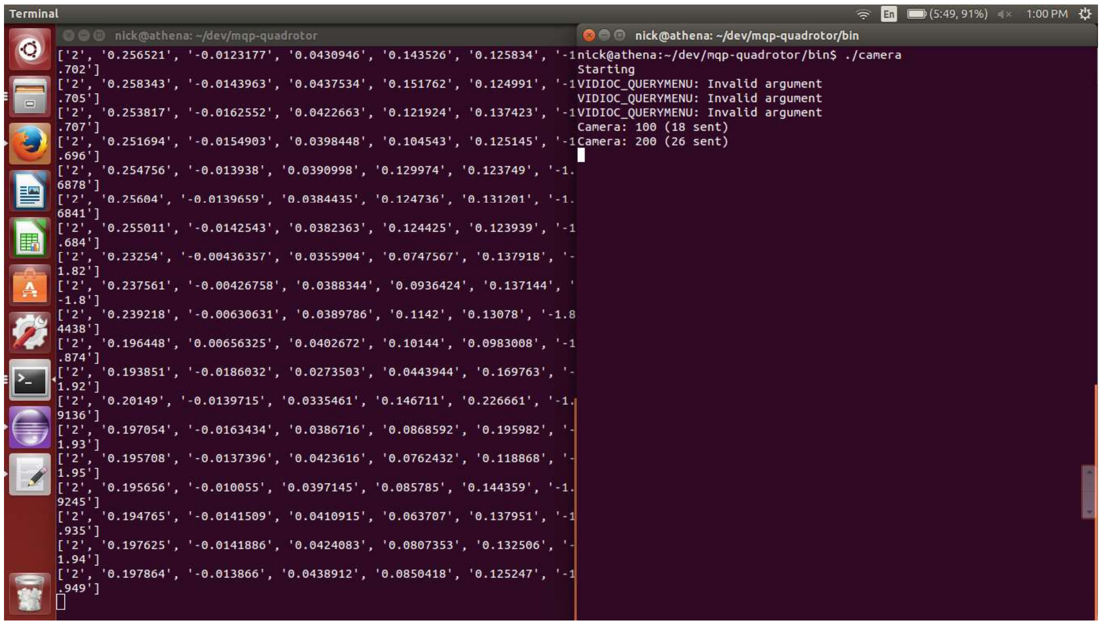


Figure 6: Example of Apriltag Mounted on UAV-C

Using the Apriltag algorithm developed by the University of Michigan April Robotics Laboratory, the UAV-S is able to recognize the Apriltag mounted on the UAV-C to calculate the state of the UAV-C. This state is then fed into the Kalman filter running on the UAV-S to update the state measurements. The code that starts the camera to begin looking for Apriltags first looks for a socket to connect to, then calculates the distance and pose of the Apriltag in view and stores the data in a string. This string is then sent through the socket, where the awaiting server receives the string and sends it into the Kalman filter running on the UAV-S. For debugging purposes, the server also prints the data received from the camera through the socket. Figure 7 shows the format of the string sent through the socket by the camera code. Each value in the array is used to identify the distance and pose of the Apriltag in view.

1. Tag ID: Each Apriltag has an ID to allow for multiple tags in camera view.
2. X Distance: Distance from camera on UAV-S to tag on UAV-C
3. Y Distance: Tag distance from horizontal center of camera
4. Z Distance: Tag distance from vertical center of camera
5. Roll: Tag roll in radians with respect to camera normal
6. Pitch: Tag pitch in radians with respect to camera normal
7. Yaw: Tag yaw in radians with respect to camera normal



```
nick@athena: ~/dev/mqp-quadrotor
['2', '0.256521', '-0.0123177', '0.0430946', '0.143526', '0.125834', '-1
.702']
['2', '0.258343', '-0.0143963', '0.0437534', '0.151762', '0.124991', '-1
.705']
['2', '0.253817', '-0.0162552', '0.0422663', '0.121924', '0.137423', '-1
.707']
['2', '0.251694', '-0.0154903', '0.0398448', '0.104543', '0.125145', '-1
.696']
['2', '0.254756', '-0.013938', '0.0390998', '0.129974', '0.123749', '-1.
6878']
['2', '0.25604', '-0.0139659', '0.0384435', '0.124736', '0.131201', '-1.
6841']
['2', '0.255011', '-0.0142543', '0.0382363', '0.124425', '0.123939', '-1
.684']
['2', '0.23254', '-0.00436357', '0.0355904', '0.0747567', '0.137918', '-
1.82']
['2', '0.237561', '-0.00426758', '0.0388344', '0.0936424', '0.137144', '
-1.8']
['2', '0.239218', '-0.00630631', '0.0389786', '0.1142', '0.13078', '-1.8
4438']
['2', '0.196448', '0.00656325', '0.0402672', '0.10144', '0.0983008', '-1
.874']
['2', '0.193851', '-0.0186032', '0.0273503', '0.0443944', '0.169763', '-
1.92']
['2', '0.20149', '-0.0139715', '0.0335461', '0.146711', '0.226661', '-1.
9136']
['2', '0.197054', '-0.0163434', '0.0386716', '0.0868592', '0.195982', '-
1.93']
['2', '0.195708', '-0.0137396', '0.0423616', '0.0762432', '0.118868', '-
1.95']
['2', '0.195656', '-0.010055', '0.0397145', '0.085785', '0.144359', '-1.
9245']
['2', '0.194765', '-0.0141509', '0.0410915', '0.063707', '0.137951', '-1
.935']
['2', '0.197625', '-0.0141886', '0.0424083', '0.0807353', '0.132506', '-
1.94']
['2', '0.197864', '-0.013866', '0.0438912', '0.0850418', '0.125247', '-1
.949']
```

```
nick@athena: ~/dev/mqp-quadrotor/bin
nick@athena:~/dev/mqp-quadrotor/bin$ ./camera
Starting
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
Camera: 100 (18 sent)
Camera: 200 (26 sent)
```

Figure 7: Apriltag Output Running in Terminal

3.3.3 Inter-UAV Communication

In addition to the socket server developed for passing sensor data to the Kalman Filter, a server was created to talk between the UAV-S and UAV-C, primarily to pass state information. It was built upon an internal ad-hoc network structure, allowing for testing to be down in a variety of environments with minimal setup. Additionally, if needed, a dedicated access point (AP) could replace this with minimal difficulty.

As the wifi adapters chosen supported both ad-hoc and AP mode, all required drivers were pre-installed on our Raspberry Pi's. Due to this, we began by finding MAC address tied to each UAVs adapter. Using this, scripts were written in a bash script, simulating command line entry, which configured a network called "CQMQP" with the ip of 156.0.0.X. The final number being specified for each device manually. The UAV-S was given 156.0.0.1, and UAV-C 156.0.0.2. The launch conditions of the Pi's were then configured to run these scripts on start, allowing them to be accessed while on a UAV.

Using this network as a backbone, a pair of communication scripts was designed, which could be implemented into code as needed. The primary functions of these scripts was to allow TCP packets of information to be transfered between both UAVs, to be used in local code or as command inputs. In order to accomplish this, an small encoding protocol was also written, allowing for specified packets of information to be sent and committed in a timely fashion, and to minimize possible data loss. This script was configured to allow multi-threading, so the connection could be kept open for the entire flight. A timestamp functionality was also included.

4 System Simulation

Simulations can act as a vital test bed for systems, providing important insight into system performance. The higher the fidelity of the simulation, the more valuable the insights gleaned will be. The primary goals of the simulation for the UAV system were to test and verify the UAV controllers and Kalman filters. First a dynamic model was created in Matlab based on the paper "Trajectory generation and control for precise aggressive maneuvers with quadrotors"(Mellinger et al., 2012). The simple controllers from this paper were then combined with the dynamic model to test the functionality of the controllers. Next, a simple two state continuous-discrete Kalman filter was created. The full simulation was created by combining a 12 state continuous discrete Kalman filter, with the attitude and position controllers, and the dynamic model from the paper by Mellinger et al.

4.1 Dynamic Model

The dynamic model presented by Mellinger et al. provides twelve differential equations that model the changes in the twelve states used to represent the UAV. These twelve states are x, y, and z position, x, y, and z velocity, angular rates p, q, and r, and euler angles ϕ, θ, and ψ. Assuming that the UAV has only these twelve states, these twelve equations can be integrated to provide the state values over time. The equations are as follows:

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{6}$$

$$m \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R_B^W \begin{bmatrix} 0 \\ 0 \\ \Sigma_{i=1}^4 F \end{bmatrix} \tag{7}$$

Where F is the force produced by each rotor and R_B^W is the matrix to convert from body coordinates to inertial coordinates:

$$R_B^W = \begin{bmatrix} c(\psi)c(\theta) - s(\phi)s(\psi)s(\theta) & -c(\phi)s(\psi) & c(\psi)s(\theta) + c(\theta)s(\phi)s(\psi) \\ c(\theta)s(\psi) + c(\psi)s(\phi)s(\theta) & c(\phi)c(\psi) & s(\psi)s(\theta) - c(\psi)c(\theta)s(\phi) \\ -c(\phi)s(\theta) & s(\phi) & c(\phi)c(\theta) \end{bmatrix}$$

Note s(θ)=sin(θ) and c(θ)=cos(θ).

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & -cos(\phi)sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\phi)cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{8}$$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} x \; I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{9}$$

Once these equations were implemented in Matlab, they were tested by supplying rotor speeds that should have caused hover, ascension and descension.

4.2 Controllers

4.2.1 Attitude Controller

The first controller implemented was the attitude controller, again taken from Mellinger et al, (2012). This controller took the following form:

$$\begin{bmatrix} w_1^{des} \\ w_2^{des} \\ w_3^{des} \\ w_4^{des} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & 0 & 1 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} w_h + \Delta w_f \\ \Delta w_\phi \\ \Delta_\theta \\ \Delta_\psi \end{bmatrix} \tag{10}$$

where w_h is the rotor speed for hover, and the Δ terms are:

$$\begin{aligned} \Delta w_\phi &= k_{p,\phi}(\phi^{des} - \phi) + k_{d,\phi}(p^{des} - p) \\ \Delta w_\theta &= k_{p,\theta}(\theta^{des} - \theta) + k_{d,\theta}(q^{des} - q) \\ \Delta w_\psi &= k_{p,\psi}(\psi^{des} - \psi) + k_{d,\psi}(r^{des} - r) \end{aligned}$$

This controller was tested by supplying desired angles and analyzing the system response. The various k gain values were tuned for a quick system response with minimal overshoot.

4.2.2 Position Controller

The second controller implemented was the position controller, again taken from Mellinger et al, (2012). This controller took the following form:

$$\begin{aligned} \phi^{des} &= \frac{1}{g}(\ddot{r}_1^{des} sin\psi_T - \ddot{r}_2^{des} cos(\psi_T) \\ \theta^{des} &= \frac{1}{g}(\ddot{r}_1^{des} cos\psi_T + \ddot{r}_2^{des} sin(\psi_T) \\ \Delta w_f &= \frac{m}{8k_f w_h}\ddot{r}_3^{des} \end{aligned}$$

where k_f is a constant relating rotor speed to thrust, and the desired $\ddot{r}$ values are found through PD feedback. For example, the desired $\ddot{r}_1$ is:

$$\ddot{r}^{des} = k_{p,1}(r_1^{des} - r_1) + k_{d,1}(-\dot{r}_1) \tag{11}$$

Ultimately this controller compares the current position with the desired position, and calculates the necessary accelerations. These accelerations get put into the position controller, which in turn generates the desired ϕs and θs and additional thrust to supply those accelerations. Lastly these desired angles get placed into attitude the controller. This generates the desired rotor speeds to achieve the desired accelerations.

While this controller worked adequately before measurement noise was added to the simulated sensor measurements, with the introduction of measurement noise the simulation could only run for eight seconds before oscillations caused the simulation to crash. Tuning the controllers' gains fixed this crashing problem, but the system performance was far from ideal. For example, during simple hover tests the UAV would have semi-random oscillations of 20 cm about the desired z position, and would randomly move in the x and y axes.
In light of these control problems a linear quadratic regulator(LQR) was selected to replace the PD control segment of the simulation. Since each segment of flight had different desired rotor speeds, a separate LQR was created for each flight segment. These LQRs created gain matrices that translated the current state into changes in rotor speeds to achieve the desired state. There are five of these LQRs in total, one for rise, one for hover, one for forward motion, one for sideways motion, and one for descent. The gain matrices were generated in Matlab and hard-coded into the on-board python scripts due to the lack of a capable discrete time Riccatti equation solver library.

4.3 Two State Kalman Filter

In order to provide a proof of concept for the eventual 12 state Kalman filter, a simple 2-D version was first created. This filter was based on a 2-D model that only incorporated position and velocity in two directions. The notation used in the model is located in Table 4.11.

4.3.1 Equations of Motion and Measurement Model

$$\dot{\xi}_s = A\xi_s + B\mathbf{u}_s, \tag{12}$$
$$\dot{\xi}_c = A\xi_c + B\mathbf{u}_c, \tag{13}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Clearly these are very simple equations of motion, but are still enough to simulate system dynamics. These system dynamics are then measured using the following measurement model:

$$\mathbf{z}_{ss}[m] = C_{ss}\xi_s[m] + \eta_{ss}[m], \quad \mathbf{z}_{cc}[n] = C_{cc}\xi_c[n] + \eta_{cc}[n], \quad \mathbf{z}_{cs}[k] = C_{cs}\xi_c[k] + \eta_{cs}[k], \tag{14}$$

Symbol	Meaning
ξ_s	State of UAV-S, $\xi_s = (\mathbf{p}_s, \mathbf{v}_s)$
ξ_c	State of UAV-C, $\xi_c = (\mathbf{p}_c, \mathbf{v}_c)$
$\mathbf{u}_s$	True input to UAV-S
$\hat{\mathbf{u}}_s$	Measured input to UAV-S
$\mathbf{u}_c$	True input to UAV-C
$\hat{\mathbf{u}}_c$	Measured input to UAV-C
$\mathbf{z}_{ss}$	Measurement by UAV-S about itself from its own sensors
R_{ss}	Measurement error covariance for $\mathbf{z}_{ss}$
$t_{s,ss}$	Sample time for measurement $\mathbf{z}_{ss}$
$\mathbf{z}_{cc}$	Measurement by UAV-C about itself from its own sensors
R_{cc}	Measurement error covariance for $\mathbf{z}_{cc}$
$t_{s,cc}$	Sample time for measurement $\mathbf{z}_{cc}$
$\mathbf{z}_{cs}$	Measurement by UAV-S about UAV-C from sensors on UAV-S
R_{cs}	Measurement error covariance for $\mathbf{z}_{cs}$
$t_{s,cs}$	Sample time for measurement $\mathbf{z}_{cs}$
$\hat{\xi}_{ss}$	State of UAV-S, self-estimated by UAV-S
P_{ss}	Estimation error covariance for $\hat{\xi}_{ss}$
$\hat{\xi}_{cc}$	State of UAV-C, self-estimated by UAV-C
P_{cc}	Estimation error covariance for $\hat{\xi}_{cc}$
$\hat{\xi}_{cs}$	State of UAV-C, estimated by UAV-S
P_{cs}	Estimation error covariance for $\hat{\xi}_{cs}$

Table 4.11: Notation used in 2-D Kalman filter

where

$$C_{ss} = C_{cs} = I(4), \qquad C_{cc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{15}$$

The notation $[m], [n], [k]$ indicates values at time instants $t = mt_{s,ss}$, $t = nt_{s,cc}$, and $t = kt_{s,cs}$, respectively. C_{cc} is only a 2x4 matrix to highlight the state estimation improvements achieved with the help of UAV-S. While UAV-C can only measure its own position, UAV-S can measure both UAV-C's position and its velocity.

4.3.2 Estimators

UAV-S Self-navigation

Initialize with $\hat{\xi}_{ss}^{+} = \hat{\xi}_{ss}(0), P_{ss}^{+} = P_{ss}(0)$, then iterate: For $t \in [mt_{s,ss}, (m+1)t_{s,ss}]$

$$\dot{\hat{\xi_{ss}}} = A\hat{\xi_{ss}} + B\hat{u}_s, \qquad \hat{\xi}[m] = \hat{\xi_{ss}}^{+} \tag{16}$$
$$\dot{P}_{ss} = AP_{ss} + P_{ss}A^T + Q_{ss}, \qquad P_{ss}[m] = P_s^{+} \tag{17}$$
$$K_{ss} = P_{ss}C_{ss}^T(R_{ss} + C_{ss}P_{ss}C_{ss}^T)^{-1} \tag{18}$$
$$\dot{\hat{\xi}}_{ss}^{+} = \hat{\xi_{ss}}[m+1] + K_{ss}(z_{ss} - C_{ss}\hat{\xi_{ss}}[m+1]) \tag{19}$$
$$P_{ss}^{+} = (I(4) - K_{ss}C_{ss})P_{ss}[m+1] \tag{20}$$

UAV-S observation of UAV-C

initialize with $\hat{\xi}_{cs}^{+} = \hat{\xi}_{cs}(0), P_{cs}^{+} = P_{cs}(0)$, then iterate: For $t \in [mt_{s,cs}, (m+1)t_{s,cs}]$

$$\dot{\hat{\xi_{cs}}} = A\hat{\xi_{cs}} + B\hat{u}_c,\ \hat{\xi_{cs}}[k] = \hat{\xi_{cs}}^{+} \tag{21}$$
$$\dot{P}_{cs} = AP_{cs} + P_{cs}A^T + Q_{cs}, P_{cs}[k] = P_{cs}^{+} \tag{22}$$
$$K_{cs} = P_{cs}C_{cs}^T(R_{cs} + C_{cs}P_{cs}C_{cs}^T)^{-1} \tag{23}$$
$$\dot{\hat{\xi}}_{cs}^{+} = \hat{\xi_{cs}}[k+1] + K_{cs}(z_{cs} - C_{cs}\hat{\xi_{cs}}[m+1]) \tag{24}$$
$$P_{cs}^{+} = (I(4) - K_{cs}C_{cs})P_{cs}[k+1] \tag{25}$$

UAV-C Self-navigation

Assuming $t_{s,comm} > t_{s,cc}$, initialize with $\hat{\xi}_{cc}^{++} = \hat{\xi}_{cc}(0), P_{cc}^{++} = P_{cc}(0)$, then iterate: For $t \in [mt_{s,cc}, n+1)t_{s,cc}]$

$$\dot{\hat{\xi_{cc}}} = A\hat{\xi_{cc}} + B\hat{u}_c, \qquad \hat{\xi}[n] = \hat{\xi_{cc}}^{++} \tag{26}$$
$$\dot{P}_{cc} = AP_{cc} + P_{cc}A^T + Q_{cc}, \qquad P_{cc}[n] = P_{cc}^{++} \tag{27}$$
$$K_{cc,1} = P_{cc}C_{cc}^T(R_{cc} + C_{cc}P_{cc}C_{cc}^T)^{-1} \tag{28}$$
$$\dot{\hat{\xi}}_{cc}^{+} = \hat{\xi_{cc}}[n+1] + K_{cc,1}(z_{cc} - C_{cc}\hat{\xi_{cc}}[n+1]) \tag{29}$$
$$P_{cc}^{+} = (I(4) - K_{cc,1}C_{cc})P_{cc}[n+1] \tag{30}$$

If $(n+2)t_{s,cc} < (\ell+1)t_{s,comm}$, $\hat{\xi}_{cc}^{++} = \hat{\xi}_{cc}^{+}$, and $P_{cc}^{++} = P_{cc}^{+}$. Otherwise:

$$\dot{\hat{\xi_{cc}}} = A\hat{\xi_{cc}} + B\hat{u}_c, \qquad \hat{\xi}[k] = \hat{\xi_{cc}}^{+} \tag{31}$$

$$\dot{P_{cc}} = AP_{cc} + P_{cc}A^T + Q_{cc}, \qquad P_{cc}[k] = P_{cc}^{+} \tag{32}$$

$$K_{cc,2} = P_{cc}(P_{cs}[l+1] + P_{cc})^{-1} \tag{33}$$

$$\dot{\hat{\xi}}_{cc}^{++} = \hat{\xi_{cc}}[n+1] + K_{cc,2}(\hat{\xi}_{cs} - \hat{\xi}_{cs}[n+1]) \tag{34}$$

$$P_{cc}^{++} = (I(4) - K_{cc,2})P_{cc}[l+1] \tag{35}$$

4.3.3 Twelve State Kalman filter

The twelve state Kalman filter followed the same principles laid out in the 2-D model, but was more complex in a number of ways. Due to the nonlinearities in the UAV dynamic model, the filter had to be an Extended Kalman Filter. This made the A matrix significantly more complicated, as each entry in A was a partial differential equation of one of the twelve equations of motion. Additionally the control inputs were included in the A matrix, such that the motion equation was of the form $\dot{\xi} = A\xi$. The full A matrix can be found in the appendix.

5 System Testing and Experimentation

Iterative unit flight tests were necessary in order to evaluate how well the system improved navigational certainty. To better organize the flight tests, a tier based system was established. The TIER I unit flight tests are the initial steps that made sure each component of the system was working properly before moving to more complex tasks. TIER II encompassed all of the unit tests involving the Apriltag and tandem flight maneuvers.

5.1 TIER I

The first unit test in TIER I was a basic script written to test required functionality of the UAV using the dronekit python library. The script established a connection with the SOLO, allowed all sensors to initialize, armed the drone, printed several parameters, and disarmed the drone.
In order to test the accuracy of the state estimation with the Kalman filter, the UAV was manually piloted while the Kalman Filter estimated the state of the UAV. These estimates were then recorded and compared to the actual flight characteristics to determine if the Kalman filter was working.
The next unit test was to develop an altitude hold script. The script commanded the UAV to fly to a target altitude and hover for 30 seconds. The evaluation was conducted by observing the UAV as it hovered and by measuring the drift in the xyz coordinate directions relative to the desired hover location. This evaluation was conducted for both the UAV-S and the UAV-C.
Steady level flight in a straight line was the next test, and the UAV was given commands to take off, hover briefly, and then travel three meters forward towards the north. The UAV-C was given this task on it's own for TIER I because it established a baseline performance that was compared with later test results. These TIER I unit tests served to establish user familiarity with the necessary technical components, and determined that each individual piece of the entire system was functioning properly before moving on.

5.2 TIER II

TIER II incorporates tandem flight tasks with both UAV-S and UAV-C flying autonomously while communicating. The first test was to have both UAVs take off and hover, measuring the amount of drift in all xyz directions. This was minimal and therefore inconclusive as to the improvement of the navigational certainty. This test was iterated three times to see if it the behavior was repeatable.
Next, the UAV pair was commanded to take off, hover, roll to the positive y direction, and recover to the hover position. This test was iterated several times to see if the behavior was repeatable at different velocities and roll angles. The UAV-C was programmed to follow the UAV-S by attempting to keep its position equal to zero with respect to the UAV-S's coordinate frame. Additionally, the z and y values will be monitored but remained at zero.
Next the UAV pair was commanded to travel in the z direction only, that is directly

upward. In this series of flight tests the UAV-S climbed to an altitude of 5 meters at several different rates and once 5 meters was achieved, it held that altitude for 15 seconds. Next, the UAV-S descended to 5 meters and paused for 15 seconds and then landed. The UAV-C was evaluated by its ability to keep the same consistent altitude as the UAV-S. The x and y position values were monitored but remained equal to zero.

After this, the x and y planar unit test was conducted. This involved the UAV pair flying at a specified altitude of 2 meters in a square shape, while attempting to maintain the displacement between UAVs. In this way two similar squares were "drawn" in the air by the UAVs several meters apart from one another. This flight test ensured that the UAV-C could maintain a constant altitude while maneuvering in a pattern significantly more complex than a straight line.

6 Results and Conclusions

6.1 Apriltag Field of View

In order to ensure UAV-S detected UAV-C consistently, the Raspberry Pi camera's ability to track the Apriltag was evaluated. This evaluation was completed by determining the maximum distances of detection using linear x displacements, y displacements at 3 different x displacements and z displacements at 3 different x displacements. The following tables were constructed by viewing a live terminal feed of Apriltag measurements while incrementally moving the 5"x5" inch tag away from the Pi camera. The limits were determined by the distances at which Apriltag measurements were no longer received. Note that these maximum distances are taken from very slow and meticulous movements of UAV-C. If these limits are approached quickly, the software loses its reading of the Apriltag. Ultimately, this quality reduces the maximum speed the UAVs can operate in order for the UAV-C to receive consistent corrective telemetry commands.

Distance (m)	Trial 1 (m)	Trial 2 (m)	Average (m)	Actual - Avg (m)	% Error
0.305	0.271	0.274	0.272	0.033	11.96%
0.610	0.563	0.563	0.563	0.046	8.25%
0.914	0.846	0.833	0.840	0.075	8.92%
1.219	1.129	1.133	1.131	0.088	7.80%
1.524	1.141	1.410	1.276	0.248	19.47%
1.829	1.708	1.697	1.703	0.126	7.42%
2.134	1.991	1.984	1.988	0.146	7.35%
2.438	2.264	2.270	2.267	0.171	7.56%
2.743	2.537	2.542	2.540	0.204	8.02%
3.048	2.851	2.852	2.852	0.197	6.89%
3.353	3.170	3.170	3.170	0.183	5.77%
3.658	3.485	3.442	3.464	0.194	5.60%
3.962	3.744	3.723	3.734	0.229	6.13%
4.267	4.057	4.045	4.051	0.216	5.34%
4.572	4.388	4.408	4.398	0.174	3.96%
4.877	4.588	4.634	4.611	0.266	5.76%

Table 6.12: Indoor X Displacement Apriltag Evaluation

6.1.1 X Displacement

The first field of view experiment conducted was the indoor and outdoor Apriltag x displacement evaluation, longitudinally away from the UAV-S. The results of this experiment are documented in Table 6.12. Since both UAVs are equipped with propeller guards, the minimum separation distance for the two UAVs is 1 foot or 0.305 meters. The maximum x displacement of the Apriltag is 16 feet or 4.877 meters. Note there still existed a significant percentage of error even after calibrating the Pi camera.

6.1.2 Y Displacement

The next evaluation completed was to determine the maximum y displacement line of sight, or the system's field of view lateral to the UAV-S. This evaluation was conducted by measuring the maximum y displacement at 3 different longitudinal displacements. By utilizing both of these values and basic trigonometry, an average angle was determined in order to predict a projected y displacement at any x displacement. The results of this experiment can be found in Table 6.13. The orientation of the determined angle is displayed in table 8. Since $\beta_{avg} = 23.58\,\text{deg}$, at the maximum displacement of 4.877 meters the maximum y displacement is 2.129 meters.

X Actual (m)	Y Actual (m)	Y Reading (m)	Actual - Reading (m)	% Error	β(deg)
0.9144	0.3683	0.399	-0.0307	-7.69%	21.94
1.8288	0.8128	0.854	-0.0412	-4.82%	23.96
2.7432	1.27	1.325	-0.055	-4.15%	24.84

Table 6.13: Indoor Y Displacement Apriltag Evaluation

6.1.3 Z Displacement

X Actual (m)	Z Actual (m)	Z Reading (m)	Actual - Reading (m)	% Error	Φ(deg)
0.9144	0.4318	0.235	0.1206	34%	5.60
1.8288	0.6477	0.435	0.1365	24%	5.89
2.7432	1.016	0.632	0.3078	33%	5.86

Table 6.14: Indoor Z Displacement Apriltag Evaluation

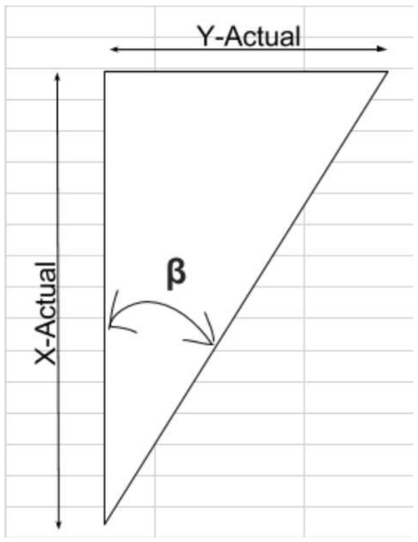


Figure 8: Y Displacement Angle Orientation

The final field of view evaluation performed was based in the z or vertical direction. The maximum z displacement was determined at three different x displacements to determine an average angle of view. The results of this evaluation are found in table 6.14. In summary, the average angle of view is 5.78 deg.

6.2 Flight testing with Apriltags

Due to weather related difficulties, extensive flight testing proved impossible. Instead, a proof of concept demonstration for UAV collaboration became the primary goal for flight testing. Preliminary results of the flight testing suggested that a two UAV system could collaborate to complete a task, making it a feasible solution for future development and potential commercialization. The two UAVs successfully flew in tandem while live Apriltag data was being captured by UAV-S's Pi cam and streamed to a file saved on UAV-S's Pi. This test highlighted UAV-S's capability of taking measurements of UAV-C in real time, which is a vital requirement for collaboration amongst the UAV pair. In separate tests, individual UAVs were able to successfully navigate along pre-chosen trajectories. These navigation tests emphasized the feasibility of fully automated UAV navigation. While this success is important, it would be foolish not to consider the limitations of the current system. One important limit is that the system has no higher level navigation capabilities. The system requires trajectories to be specified before a flight has begun. The system is further limited by a requirement for a GPS fix, effectively forcing all testing to be conducted outside. Another limitation is the range of the Apriltag, and the speed at which data can be sent and received, as well as processed. These hardware based limitations further reduce the mission readiness of the system, but in no way effect the proof of concept as they are primarily financially limited. The Raspberry Pi and Pi camera could be switched for much more powerful processors and cameras in a system that was designed to function in

a real world scenario. Due to the budget constraints of the project, the hardware selected was optimized to complete the task without exceeding the budget.
It must also be addressed that several of the initial goals created at the beginning of this project were not completed. One of these goals was to conduct fully automated flight tests in tandem. With the system built, and flight testing underway, several issues arose. First of all it was winter, and the weather became a large issue. Flight tests needed to be conducted outdoors and therefore a large open area free of snow was required. Additionally, the performance of the UAV controllers degraded in the cold, making it difficult to prepare the UAVs for automated flight. Time itself was also a constraining factor, resulting in only a few days that were clear enough and warm enough to allow for reliable testing. These constraints led to a reworking of the original goals which had been based on assumptions. Instead these goals were replaced with the proof of concept goal, namely having tandem flight with live state estimation via the Apriltag code. This revised goal was successfully achieved.
In summary, while there are definitive and undeniable holes in the capability of the system, according to realistic goals set at the time of completion, this system does indeed complete the task of being a proof of concept for a low-cost alternative to larger UAV systems. There are however, areas that could be further researched to create a system that is significantly more mission capable.

6.3 Areas for Further Development

Several areas that could be considered for future development include: Kalman filter integration, predetermined path following in tandem flight, and testing to quantify the increase in navigational certainty while in tandem flight. First off, scripts for the integration of a Kalman filter into the system were already created. These scripts were not able to be debugged in time for the flight testing, but they worked in simulation. Utilizing Kalman filters would improve the accuracy of UAV navigation and increase the value provided from Apriltag measurements. Path following in tandem flight would be another realistic area for further development. The building blocks for this goal were established, but were unable to be used in conjunction to achieve the automated tandem flying. Tests that can quantify how much the navigational certainty improves after implementing the Kalman filters would also provide valuable numerical data to confirm that this system functions in a superior manner to a single UAV system in terms of navigational certainty.
These three areas are the logical next steps that should be taken if further work is to be done on this project. With the data and experiment results from such research, a more practical system could be created, and a more thorough explanation of which tasks such a system would be suited for could be defined. This project therefore suggests the three steps to any who consider furthering research into coordinated quadrotor UAV systems.

Acknowledgments

Our team would like to thank Professor Raghvendra Cowlagi, Ruixiang Du, and Bryan Healy for all your help and guidance. This project would not look the same without your help.

Bibliography

Anthony, S. (2013). Amazon unveils 30-minute Prime Air quadcopter delivery service, but it's completely impractical. Retrieved from http://www.extremetech.com/extreme/171879-amazon-unveils-30-minute-prime-air -quadcopter-delivery-service-but-its-completely-impractical

Blythe, J. D., & Borowicz, K. A.,& Hollander, A. N. (2016). Autonomous Quadrotor Navigation and Guidance. Retrieved from https://web.wpi.edu/Pubs/E-project/Available/E-project-032316-184225/

Bishop, G., Welch, G. (2001). *An Introduction to the Kalman Filter.* Retrieved from http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf

Carter, Jamie; Keil Schmid; Kirk Waters; Lindy Betzhold; Brian Hadley; Rebecca Mataosky; Jennifer Halleran (2012). "Lidar 101: An Introduction to Lidar Technology, Data, and Applications." (NOAA) Coastal Services Center" (PDF). Coast.noaaa.gov. p. 14.
Compare Drones. (2015). Available from http://drones.specout.com/

Friedland, B.(1986). *Control System Design - An Introduction to State-Space Methods.* Retrieved from https://app.knovel.com/web/toc.v/cid:kpCSDAISS1/viewerType:toc/root_slug:contr ol-system-design/url_slug:kt00B0CJD2

Goyer, G. G.; R. Watson (September 1963). "The Laser and its Application to Meteorology". Bulletin of the American Meteorological Society. 44 (9): 564–575 [568].

Keane, J. F., Carr, S. S. (2013). A Brief History of Early Unmanned Aircraft. John Hopkins APL Technical Digest, 32(3).Retrieved from http://www.jhuapl.edu/techdigest/TD/td3203/32_03-Keane.pdf

Koski, O. In a First, Full-Sized Robo-Copter Flies With No Human Help. Wired, 14 July 2010.

New York Times. (1910, May 22). Torpedo Airship Controlled by Wireless is the Latest Invention. New York Times. Retrieved from http://query.nytimes.com/mem/archive-free/pdf?res=9D02E2D71139E333A25751C 2A9639C946196D6CF

Mellinger, D.,Powers, C., Kumar, V. (2015). Quadrotor Kinematics and Dynamics. *Handbook of Unmanned Aerial Vehicles*, 1: 307-328.

Rohel, E.(1937). *Navigation.*Retrieved from https://ia800500.us.archive.org/16/items /navigation028934mbp/navigation028934mbp.pdf

Siouris, G. M. (2004). Missile guidance and control systems. New York: Springer.

Spooner, S. (1924). A Successful French Helicopter. Flight, 16(4). Retrieved from https://www.flightglobal.com/pdfarchive/view/1924/1924%20%200047.html

Teal Group Corporation. (2015). *Teal Group Predicts Worldwide UAV Production Will Total $93 Billion in its 2015 UAV Market Profile and Forecast.* Retrieved from http://www.prnewswire.com/news-releases/teal-group-predicts-worldwide-uav-production-will-total-93-billion-in-its-2015-uav-market-profile-and-forecast-300128745.html

Yanushevsky, R.(2011).*Guidance of unmanned aerial vehicles.* Retrieved from http://www.crcnetbase.com.ezproxy.wpi.edu/isbn/978-1-4398-5096-1

Appendix A: CAD Detailed Drawings

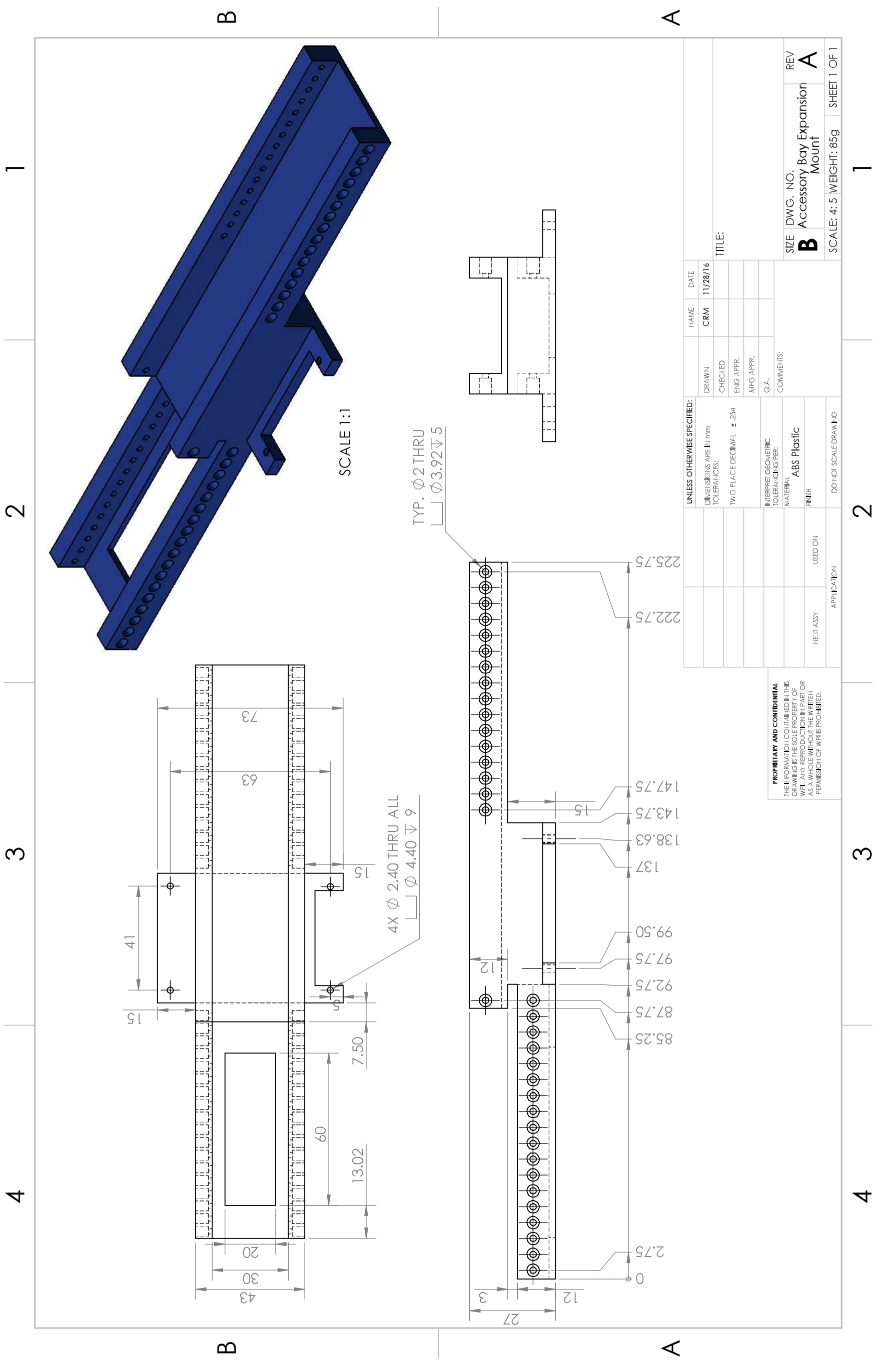


Figure 9: Accessory Bay Expansion Mount Detailed Drawing

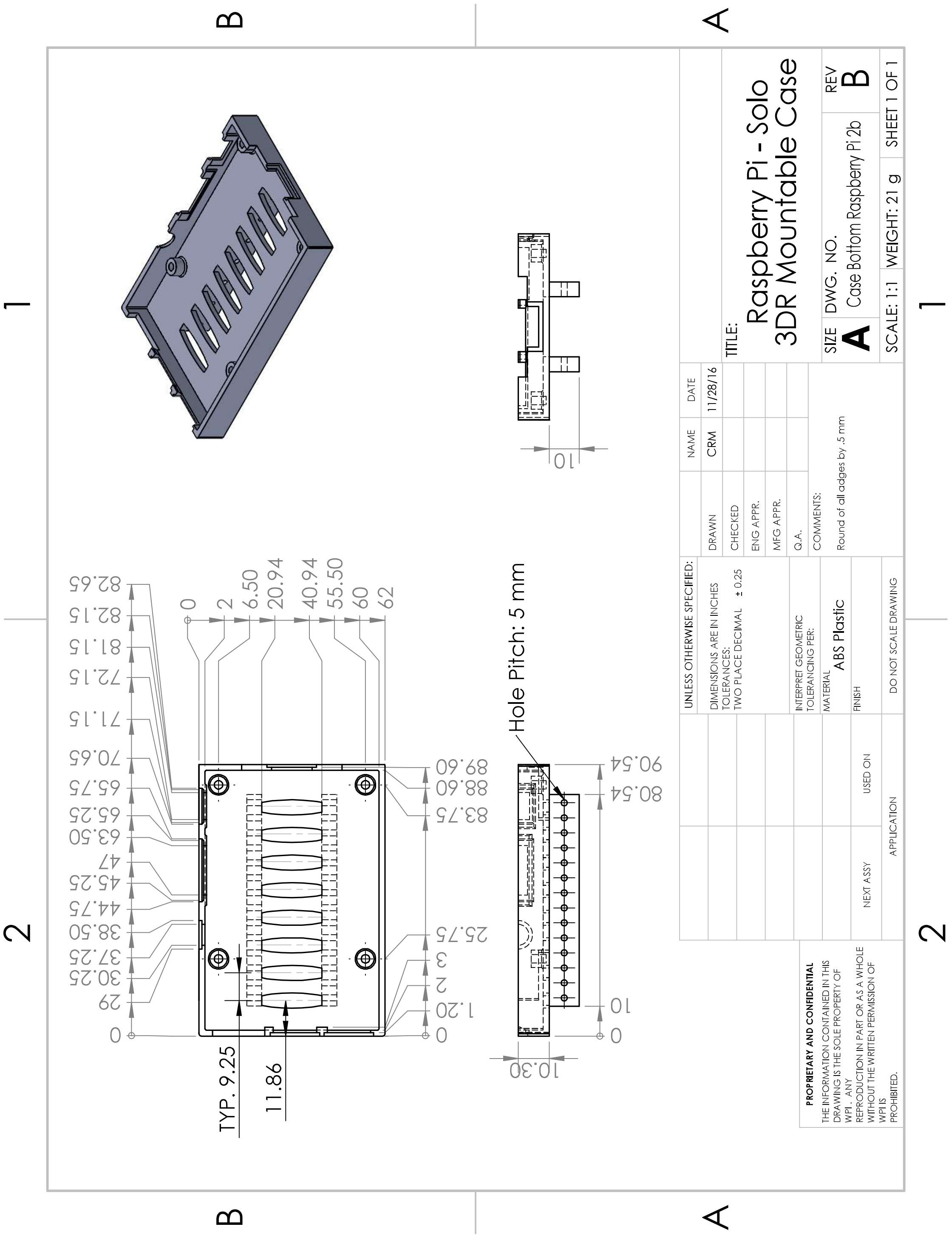

Raspberry Pi - Solo 3DR Mountable Case
DWG. NO. Case Bottom Raspberry Pi 2b
SIZE A
REV B
SCALE: 1:1
WEIGHT: 21 g
SHEET 1 OF 1
DRAWN CRM 11/28/16
MATERIAL ABS Plastic
Round of all edges by .5 mm
Hole Pitch: 5 mm
TYP. 9.25
11.86
PROPRIETARY AND CONFIDENTIAL

Figure 10: Case Bottom for Raspberry Pi 2b Detailed Drawing

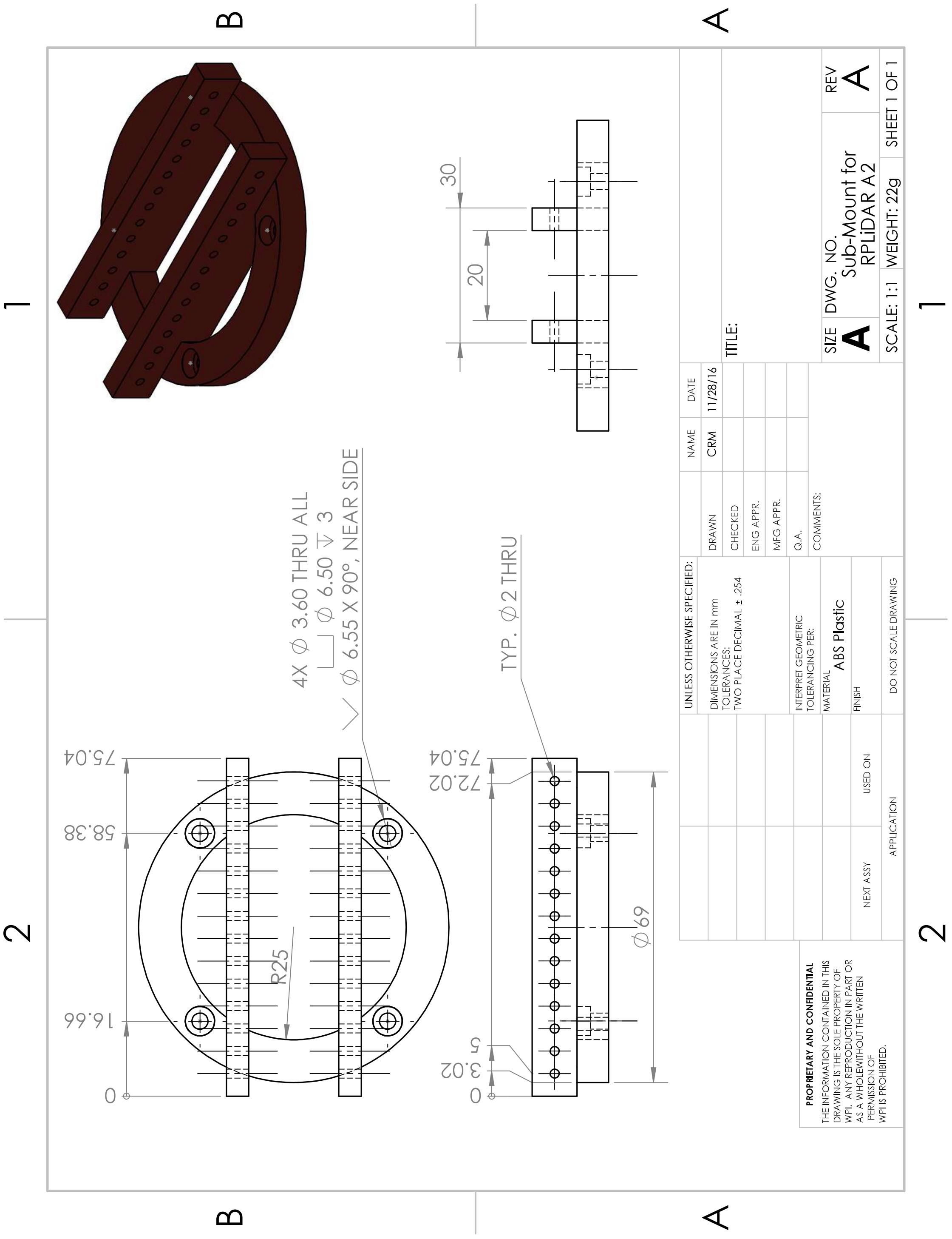


Figure 11: Mount for RPLiDAR A2 Detailed Drawing

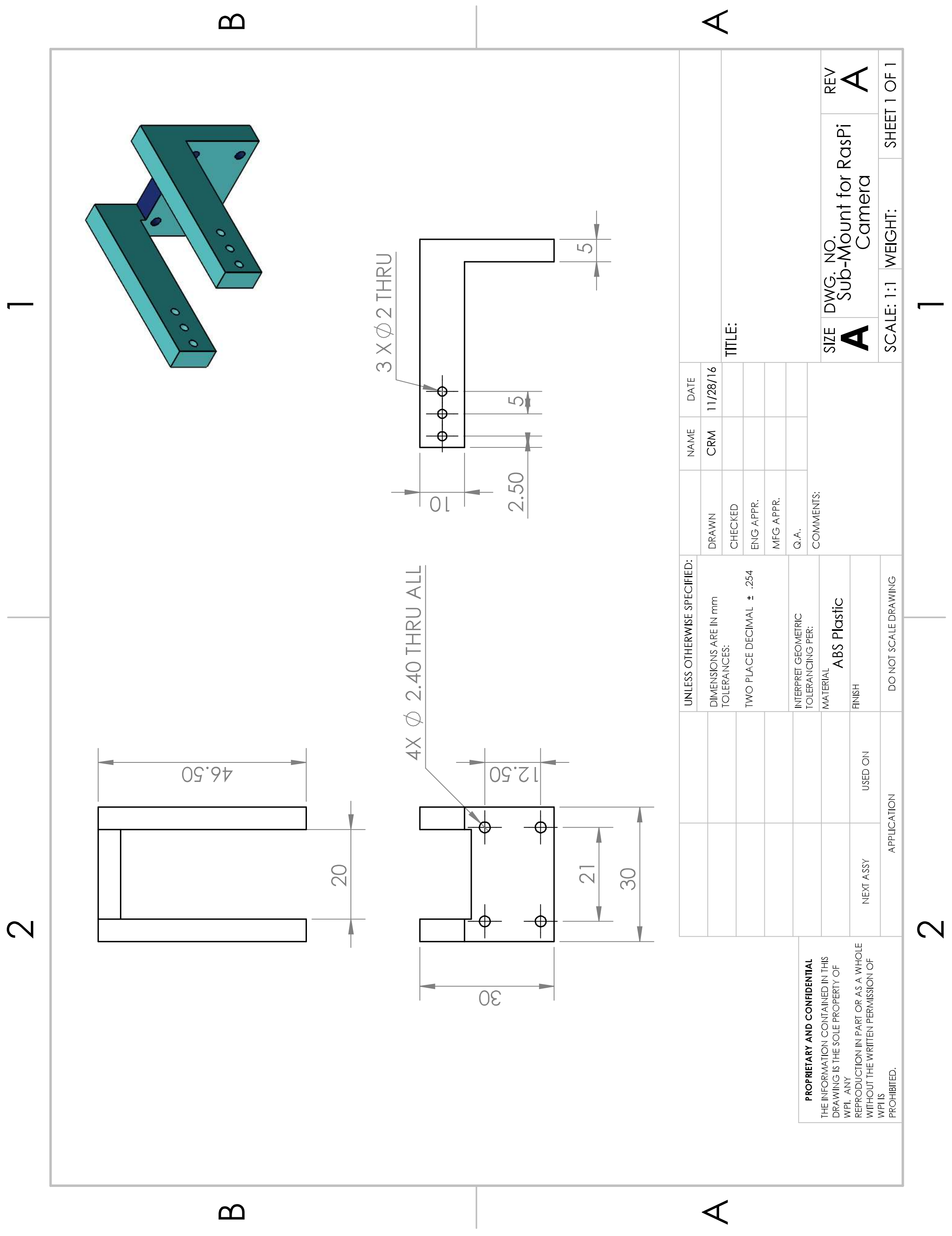


Figure 12: Mount for RasPi Camera Detailed Drawing

Appendix B: Programming

All code has been uploaded to shared repository. See link below:
https://bitbucket.org/rvcowlagi/srcl_mqp/src