Multi-MAV Deployment

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/Mechanical & Robotics Engineering

Submitted by:

Adam Campisi, Aerospace Eng.

Alexander Hindley, Mechanical Eng.

Samuel Daley, Aerospace Eng.

Joseph Danner, Robotics Eng.

Samantha Hilerio, Aerospace Eng.

James Kirk, Robotics Eng.

Christopher Sanchez, Aerospace Eng.

John Pearsall, Robotics Eng.

Advisors:
Michael A. Demetriou

mdemetri@wpi.edu

Stephen Nestinger

ssnestinger@wpi.edu

Abstract

The goal of this project was to develop a system of coordinated micro aerial vehicles along with an unmanned ground vehicle in order to advance the development of collaborative systems. The design objectives were to maximize flight time and mobility of a quad-rotor, and to minimize the size of the system. Analysis, design, construction, and testing of an autonomous quad-rotor and ground vehicle for collaborative operations were completed. The resulting system was capable of deployment and hover.

Acknowledgements

We would like to acknowledge first our advisors, without whom we would have no project. Professor Nestinger and Professor Demetriou were helpful in any way possible, and for that we thank them.

We would also like to acknowledge Tracey Coetzee for all of her assistance with every order we needed to place. Thank you Tracey.

Finally, we would like to thank Dr. Adriana Hera for all of her help with our Matlab simulation. When all hope seemed lost in adding noise to our simulation, she saved the day. Thank you, Dr. Hera.

Table of Contents

List of Figures

List of Tables

1 Introduction

Micro aerial vehicles (MAVs) have risen to the forefront of the technological world in recent years. Whether gathering vital intelligence or launching a strategic missile attack thousands of miles away from its operator, the current generation of MAVs has near limitless capabilities, but the majority of MAVs have gained notoriety through military action. Very few have found their way into the civilian world, where there exists significant potential for the MAV industry to grow. They would be able to conduct a wide variety of missions from assisting rescue workers, providing police officers another pair of eyes, or even surveying surrounding landscape. The ability to provide information from an aerial vantage point, while avoiding potential risks to individuals, would prove to be a valuable asset to many areas of society.

The multiple MAV deployment project focused on the design, analysis, and construction of autonomous micro aerial vehicles that would collaborate with an autonomous ground vehicle (UGV) in order to relay valuable information regarding the MAV's surroundings. The final objective was to construct three MAVs, and to have them receive instruction from a single ground robot that would not only be the source of their instruction but also their landing and charging pad.

The controls system for each MAV, as well as the UGV, has been developed in order to create a properly functioning and coordinated system. The UGV would have the ability to command the MAVs to take-off, fly in formation to another room and then land back on the UGV to charge, all without any human input. At the same time, the UGV would be able to track the locations of the MAVs such that the UGV could follow the MAVs so that they could return to the landing pad without having to come back to the UGV in the initial room.

Another feature the team planned to implement was the ability of the MAVs to pick up small amounts of cargo. The ability to remain stable in flight while experiencing turbulent conditions was another requirement of the MAVs. The design constraints required the MAV to be able to withstand five mph wind gusts outside.

This report will outline the details of the design, assembly, and testing that went into accomplishing the project's objectives. Extensive background research divulges the inner workings of MAVs, UGVs, and the systems that will be required for them to work cohesively. Further investigation into the benefits and disadvantages of the various MAV and UGV designs revealed which is most advantageous to accomplishing the goals of the project. A wide range of tests involving aeronautics, mechanics and robotics have been conducted to insure the accuracy of MAV and UGV components, and determined the functionality of the systems. These topics are covered in detail throughout the report in order to provide the most complete sense of the scope and outcome of the project.

2 Background

Unmanned aerial vehicles (UAVs) are often considered a recent technological development. However, UAVs have been in production and use since the early 1920s. While newer generations of UAVs have far surpassed earlier models, there is a lot that can be learned by understand the development process of the first UAVs.

2.1 History of UAV Tech

Early UAVs were very primitive in both design and function. In 1918, Charles Kettering, an American engineer, developed an unmanned flying vehicle that was controlled by a gyroscope. It was designed to carry an explosive that would detonate after the propeller had completed a predetermined number of rotations. Known as the Kettering Bug flying bomb, it was used by the Army Signal Corps as a form of long-range artillery in the mid-1920s [4]. The Kettering Bug was the first in a long line of aircraft designed to fly without an on-board pilot. A large number of unmanned drones were employed by American military forces sporadically throughout the first and second World. While the United States military made use of many different unmanned aircraft, the vast majority of these early pilotless aircraft were actually missiles, designed to deliver an explosive payload and not intended for reuse. These would actually help to form the foundation for the development of precision guided missile technology [4]. It would not be for many years after the first pilotless aircraft that a true UAV was developed.

The development of an unmanned surveillance vehicle began during the Cold War to help facilitate the intelligence gathering efforts of US agencies. A project dubbed “Red Wagon” was intended to repurpose targeting drones for intelligence gathering missions due to the discrepancy in size between the UAV and traditional surveillance aircraft. Unfortunately, this project was quickly cancelled so that funding could be reallocated to the Oxcart Project, which would lead to the production of the SR-71

Blackbird [4]. “Red Wagon” was among the first of countless US intelligence funded projects that led to the development of several different variations of UAV surveillance vehicles. Through each of these projects, the UAVs became more technologically advanced, employing state of the art sensor packages, navigation equipment, and communications devices [2]. The Predator drone, perhaps the most commonly known UAV, is an example of how far UAV technology has come since the Kettering Bug. With the ability to gather intelligence and launch an attack thousands of miles away from its operator, the Predator is a prime example of the advanced technology of today’s UAVs.

2.2 Micro UAVs

While the military continues to invest in the development of large, traditionally sized unmanned aircraft, the development of miniature sized UAVs is becoming more prevalent. Known as micro UAVs (MAVs), these aircraft are designed to accomplish many of the same goals as their larger counterparts, although MAVs have one significant advantage over other UAVs: portability. Many UAVs, such as the Predator drone, require near full sized runways for takeoff and landing, and must require much more storage space. MAVs provide the user which much more flexibility, as many models are small enough to be carried around. The ability to execute vertical take-off and landings as well as to be hand-launched, allow MAVs to operate in more enclosed environments.

Many different MAV designs exist, and more are constantly being developed. The small size of the aircraft allows for many unconventional designs to be tested and implemented. Each MAV design has its advantages, and many MAVs are designed for specific mission parameters. The most common MAV mission is surveillance, as a small and highly maneuverable craft is able to supply important intelligence without risking the safety of its operators. Due to their small size, MAVs have so far been limited to gathering intelligence; there is not enough available space to include a weapons payload.

While most traditionally sized unmanned aircraft are used exclusively by the military, MAVs have the potential for civilian use as well. MAVs could serve as aids to search and rescue missions, communications aids (possibly too create temporary internet/telecommunication access), or hobby use. MAVs may be able to serve many different functions in the civilian world.

2.3 UGV Overview

Micro UAVs allow for increased precision maneuvering in confined spaces, but their major limiting factor is flight time. At best, an MAV may experience ten to twenty minutes of flight time, depending on the power drain of its motors and the size of its battery. Such a small amount of flight time severely limits the operating radius of the MAV, which in turns limits its potential for gathering intelligence. While a simple solution would be to add more battery power, this would add significant weight to the UAV and may in fact actually lower flight time. One solution to this problem may be to mount the MAV(s) on a UGV platform that would be able to transport the MAVs closer to the desired location. UGVs are becoming more prevalent, and would make an ideal carrier for a swarm of MAVs.

UGV development began in the late 1960s, and quickly expanded. The majority of the UGV projects were militarily funded, and thus focused on reconnaissance, surveillance, and target acquisition missions [3]. UGVs were found to be successful ventures, and several of the projects included combat oriented UGVs, which acted as mobile ordnance bases. These UVGs utilized a complex array of sensors, including TV cameras, ultra sonic range finders, and various forms of radar, all to assist in the autonomous or semi-autonomous navigation of terrain.

In regards to MAVs, a UVG would be able to function as a base of operations, carrying the MAVs onboard until they were ready to deploy, in which case the MAVs would be able to autonomously take-off, and then land upon their return. The UGV would serve as the command center of the MAV/UGV network, receiving data from the MAV and then using that data to plan the MAVs next movements. Not only does this symbiotic relationship between the MAV and UGV allow for the MAV to increase its range of operation, but it will also allow the MAV to charge and redeploy from the UGV, extending the total mission time.

3 Methodology

The Multi-MAV Deployment project focuses around designing, analyzing, and constructing autonomous micro aerial vehicles that will collaborate with a ground vehicle in order to relay valuable information regarding the MAV's surroundings. The final objective is to construct three MAVs, and to have them receive instruction from a single ground robot that will not only be the source of their instruction but also their landing and charging pad as well. The following section outlines the methods employed to accomplish the project objectives.

3.1 Design of MAV

Size and stability are the main priorities in this project. Therefore, this project included looked through several types of MAVs, including the Vertical Take-Off and Landing Fixed Wing, the Tri-Rotor, and the Single Rotor aircrafts. The entire design process is described in further detail below.

3.1.1 Preliminary Design

In order to move forward with the project, a basic design for the MAV needed to be selected. Several different configurations were put forward for consideration by the group. Many unique designs were considered, but the four main contenders were a basic quad-rotor, a quad-rotor with collapsible wings, and two vertical take-off and landing (VTOL) flying wing designs.

3.1.1.1 Quad-rotor

First, the quad-rotor is one of the more basic designs for an MAV. As shown in Figure 1, four individual motors provide thrust, and variable motor speeds allow the quad-rotor to perform precise maneuvers. Its small size also enables the quad-rotor to easily operate indoors as well as outdoors. The major drawback presented by a quad-rotor is the power required to operate four high-speed motors and therefore prevents lengthy flights. Quad-rotors typically have a flight time of only 10-15 minutes.

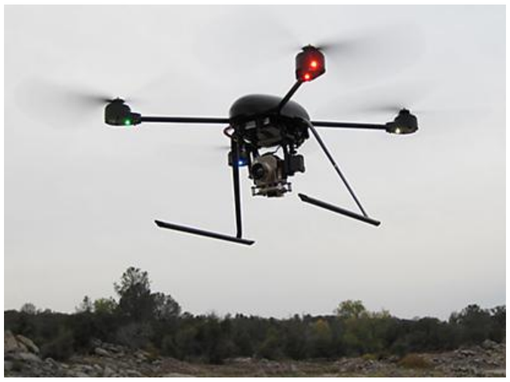


Figure 1: Quad-rotor in flight

3.1.1.2 Collapsible Quad-rotor

The second design differed from a traditional quad-rotor by having two motors located on collapsible wings as seen in Figure 2. This configuration would allow for the MAV to be stored in a confined space, while still being able to operate with the same agility and performance as a conventional quad-rotor. The wings would remain in a downward position until the MAV lifted off under power of the two main motors. Once it reached the desired hovering height, the MAV would activate the two wing-mounted motors and the wings would be lifted into position, with flight continuing under the power of all four motors. The complexity of designing the wings to function properly would make this configuration require much more work than a traditional quad-rotor.

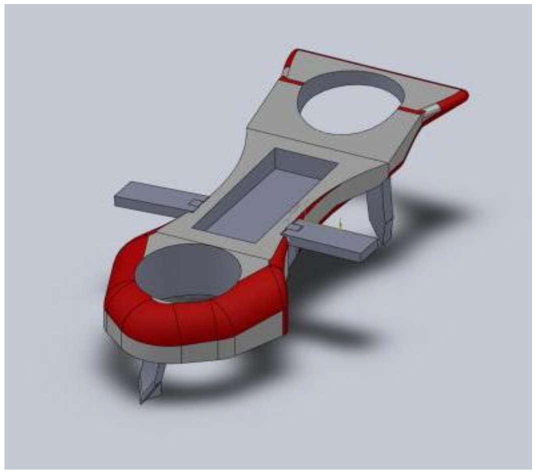


Figure 2: MAV Design #1

3.1.1.3 Flying Wing

Next, two different VTOL flying wing designs were considered. The first design in Figure 3 required the MAV to takeoff in a vertical orientation, with the airfoil perpendicular to the landing surface. This design would have rear mounted motors to provide the thrust for takeoff. Once the MAV had reached a desired height, ailerons located in the wing would be used to guide the MAV onto a horizontal trajectory. Once the MAV is able to achieve stable flight, the motors would be able to operate at low power, as the airfoil design would provide an enough lift to keep the MAV airborne. Landing maneuvers would require the MAV to perform a vertical maneuver to induce a controlled stall in order to properly return to its landing position.

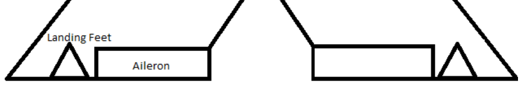


Figure 3: MAV Design #2

The second VTOL flying wing considered is shown in Figure 4. This design placed two motors within the surface of the wing, with rear facing motor to provide forward thrust. The motors mounted within the wing would provide thrust for takeoff, and once a desired hovering height was achieved the rear motor would activate and provide forward thrust. The airfoil body would also provide lift, reducing the need for the motors to operate at high speeds for extending periods of time.

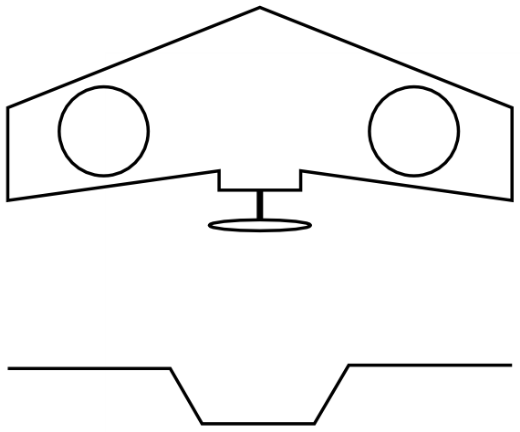


Figure 4: MAV Design #3

3.1.1.4 Design Comparison

Each of the four designs was rated against each other and a rubric, as seen in Table 1, was used to objectively compare each of the designs against the other. On a scale of one to five, with five being the highest, each design was rated on nine different criteria. The design that received the highest score was chosen as the final design of the MAV. The ratings were determined as a group, based on the expected performance of each design. No actual calculations were involved in the determination of the ratings.

Table 1: Evaluation Rubric

Criteria	Quad-rotor	Collapsible Quad-rotor	VTOL Design 1	VTOL Design 2
Number of Motors Required	4	4	2	3
Number of Servos Required	0	0	2	2
Stability Capabilities	5	3	2	2
Footprint (On Landing Pad)	2	3.5	5	3
Control System Complexity	5	2	1	2
Maneuverability	5	4	2	2
Power-to-Weight Ratio (Takeoff)	4	3	2	3
Power Required (Flying)	4	3	5	3
Manufacturability	4	1	2	1
Totals:	33	23.5	23	21

This rubric examined the number of motors each MAV would require and the number of servos required for moving parts. The more motors and servos involved, the more complicated the manufacturing and controls would become. Stability was also factored in, as native stability would increase the simplicity of the control scheme. Overall size and power requirements are also important characteristics, as larger MAVs require more power and larger landing areas. Due to the limited size of the UGV landing pad, large and cumbersome designs were less favorable.

3.1.1.5 Design Iteration #1

Based on this rubric, the team chose the generic quad-rotor design as the ideal candidate for the MAV. A basic quad-rotor was constructed using SolidWorks, and work began to further optimize the design. The original prototype went through several iterations before a main design was reached. First, the original prototype seen in Figure 5: MAV Design Iteration #1 included a rectangular design, with the motors located at different distances from each other. This rectangular configuration was chosen to minimize the overall size of the MAV.

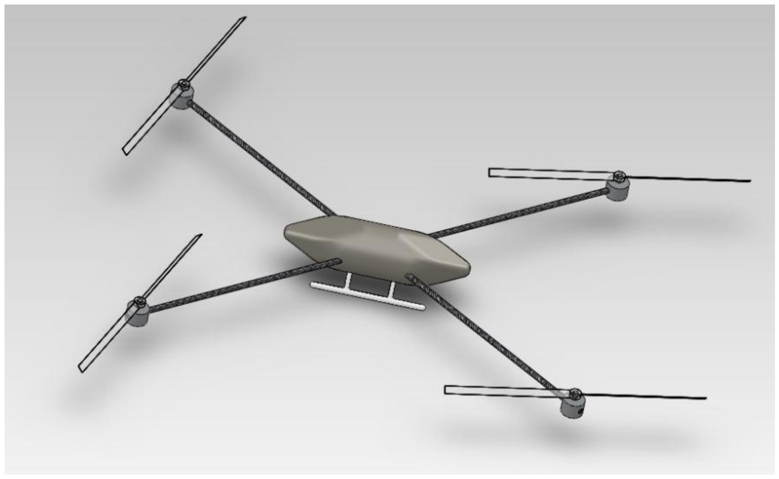


Figure 5: MAV Design Iteration #1

A rectangular motor orientation was found to not be ideal, as the force produced by each motor would not be equally distributed. For the sake of reducing the control system complexity, the design was altered to place the motors equidistant to the adjacent motors. The design of the MAV body was also reconsidered; the original design offered an enclosure for the internal components, and this was determined to be unnecessary weight as the MAV would not be flying in conditions that would be hazardous to exposed electronics.

3.1.1.6 Design Iteration #2

These considerations led to the design of the second prototype. As shown in Figure 6: Quad-rotor Design Iteration #2, this iteration incorporated a much more simple design; the arms were mounted in shafts to the top of the main MAV body, forming a twelve inch by twelve inch square. This footprint was chosen as it would fit snugly onto the surface provided by the UGV. Additionally, the decision was made to use square tubing rather than round and to include a support that held the motors at the ends of the arms. Square tubing was chosen to provide the motors with a level surface to be mounted on, insuring that they are producing thrust in the proper direction. The landing gear was also changed to incorporate an inverted, pyramid shaped guided-landing-pad design. An inverted pyramid landing pad would funnel the MAV into the landing pad, insuring that it is held securely for transport and charging purposes. Lastly, and to subsequently increase the overall stability of the MAV, the heavier components were moved below the main body. This resulted in a lower center of gravity, and thus lowering the moment of inertia, which would improve the MAVs tendency to resist significant and sudden attitude changes. As such, a space was created below the body for the electronic control components and battery. A cargo bay was incorporated to provide ample room for the aforementioned components. This revision also had the added benefit of reducing the overall weight of the MAV.

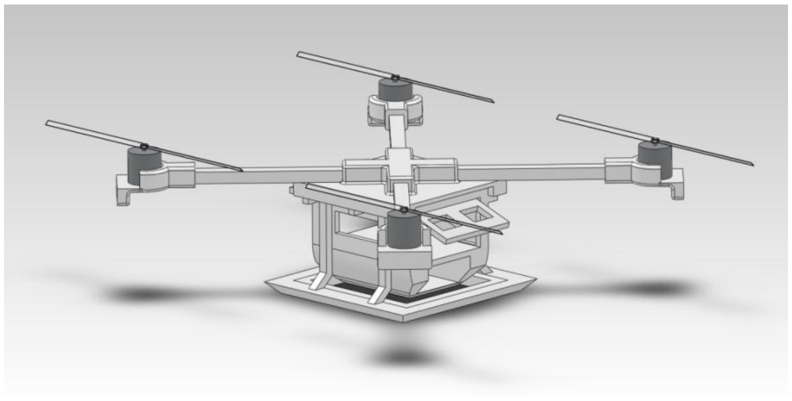


Figure 6: Quad-rotor Design Iteration #2

3.1.1.7 Design Iteration #3

Because the previously mentioned cargo bay proved to be oversized, it was replaced with strips of aluminum that will secure the battery and several electronic components. Additional electric components would be mounted to the bottom of the MAV using adhesive strips. An added benefit of replacing the cargo bay with aluminum strips was further weight reduction. The final MAV design can be seen in Figure 7.

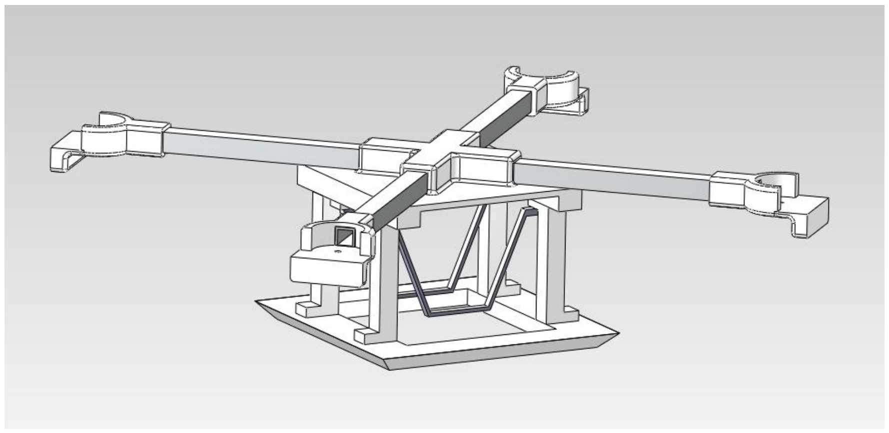


Figure 7: Final MAV Design

3.2 MAV Construction and Iterations

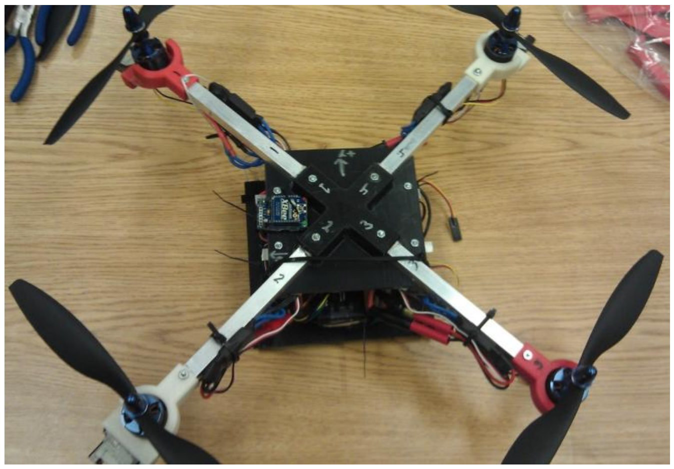


Figure 8: Completed MAV

This portion of the report discusses the assembly, design considerations, and part selection to realize the final MAV. There were four prototypes.

To begin assembly, all of the parts were rapid prototyped or purchased. The parts that were rapid prototyped were the upper body portion, the motor mounts, and the lower bay area (including the landing portion).

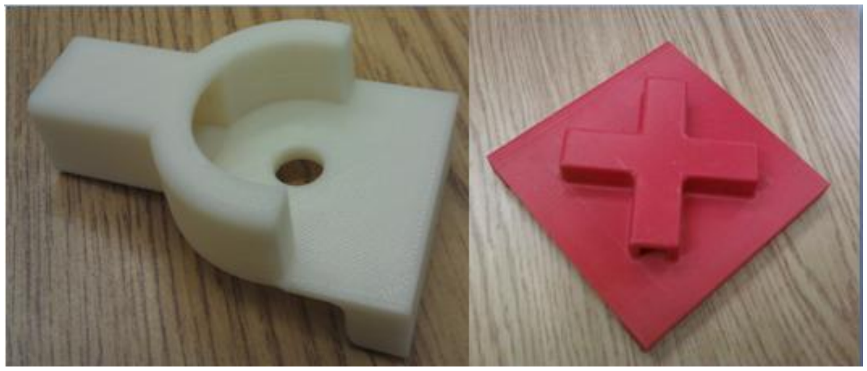


Figure 9: Motor Mount (left), Upper Body Portion (right)

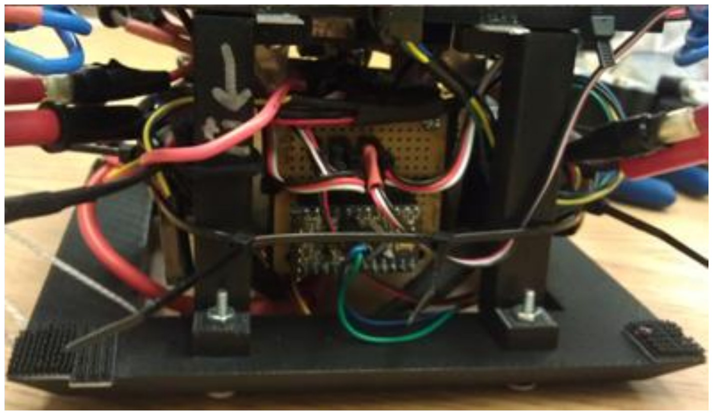


Figure 10: Lower Bay area and Landing Pad

These rapid prototyped parts were brought to the lab to utilize the drill press for through holes. The through holes were drilled through the upper body portion where it was to be connected to the arms, and where it was to be connected to the legs (which held the landing gear on). Through holes were drilled through the aluminum arms in addition, for easier attachment to the upper body.

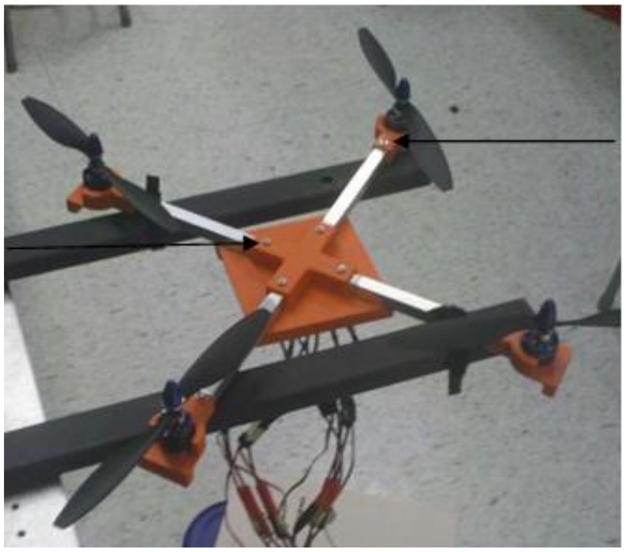


Figure 11: Through Holes on Upper Body and Motor Mounts

Through holes were then drilled through the motor mounts; where the screws for the motor would attach to the mount, and where the IR sensors would be attached to the outward face of the mount.

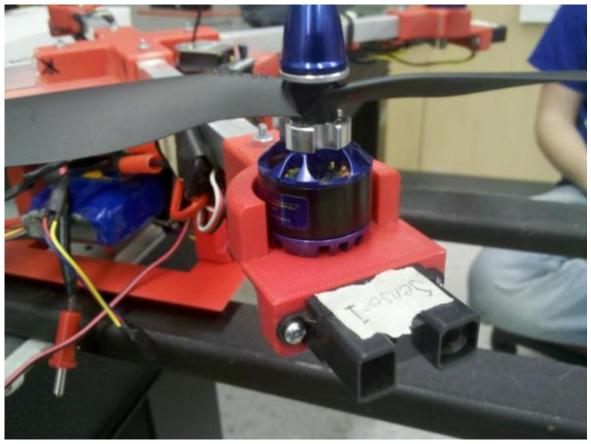


Figure 12: Motor Mount with Motor and IR Sensor Attached

The motor mounts were the first to be assembled after all of the through holes were drilled. As stated before the IR sensors were mounted facing outwards from the motor mount, and the motors were mounted to the top as in Figure 12: Motor Mount with Motor and IR Sensor Attached. Magnets were used as RPM readers for the Hall Effect sensors, and mounted on the bottom end of the shaft of the motors. The Hall Effect sensors were hot glued to the underside of the motor mounts, as close as possible to the magnets such that reading the polar changes would not prove to be an issue.

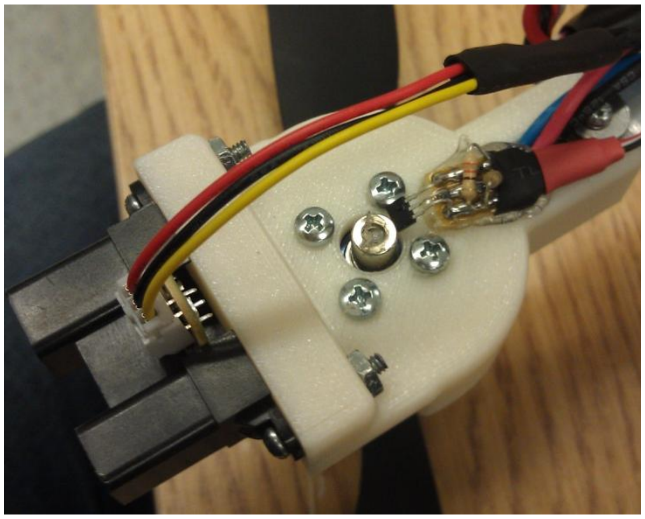


Figure 13: Underside of Motor Mount with Hall Effect Sensor Attached

Once the arms were attached to the upper portion of the MAV structure (using nuts and bolts, as well as securing them down with hot glue to decrease the intensity of the vibrations), the speed controllers were plugged into the motors and mounted on the side of the arms.

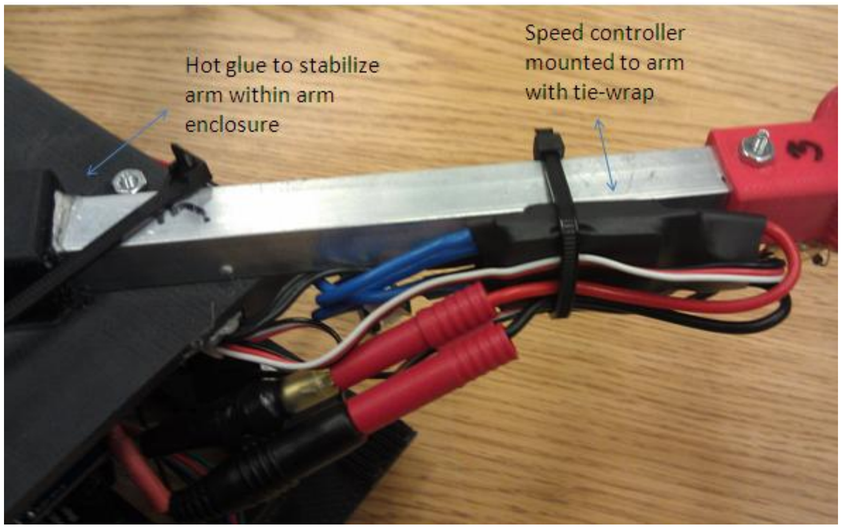

Hot glue to stabilize arm within arm enclosure
Speed controller mounted to arm with tie-wrap

Figure 14: Attached Arms and Mounted Speed Controllers

An aluminum bay area was created to hold the battery. Two aluminum strips were folded and attached to the underside of the upper body portion for the bay. Velcro straps were hot-glued to the bay, to keep the battery from sliding out of the MAV.

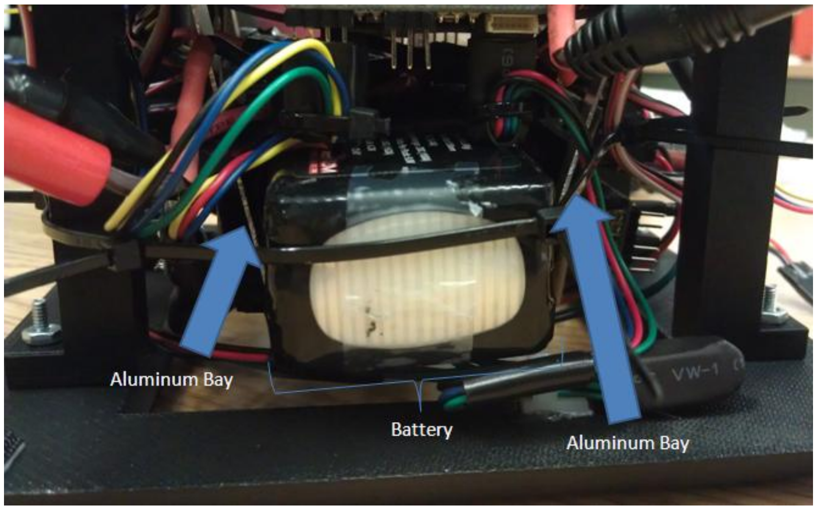


Figure 15: Aluminum Bay with Battery

Velcro was used on the underside of the upper portion of the structure to mount the Ardupilot securely such that it would not move during flight and was capable of being taken out if needed.

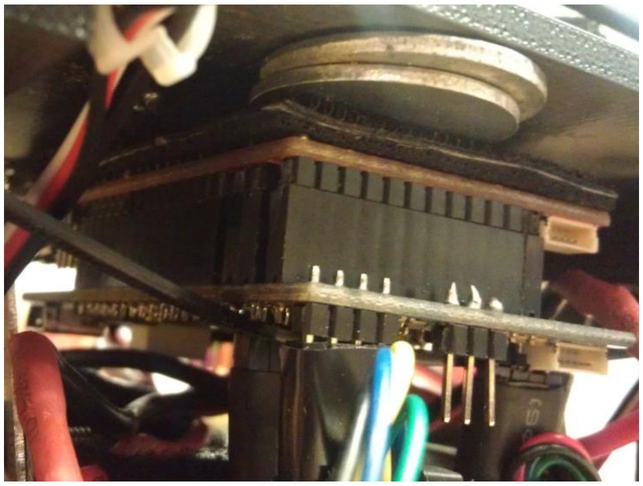


Figure 16: Ardupilot Mounted on Underside of Upper Body

The Ardupilot Pro-minis were attached using the aluminum bay area. They were each mounted on opposite sides with a crossbeam.

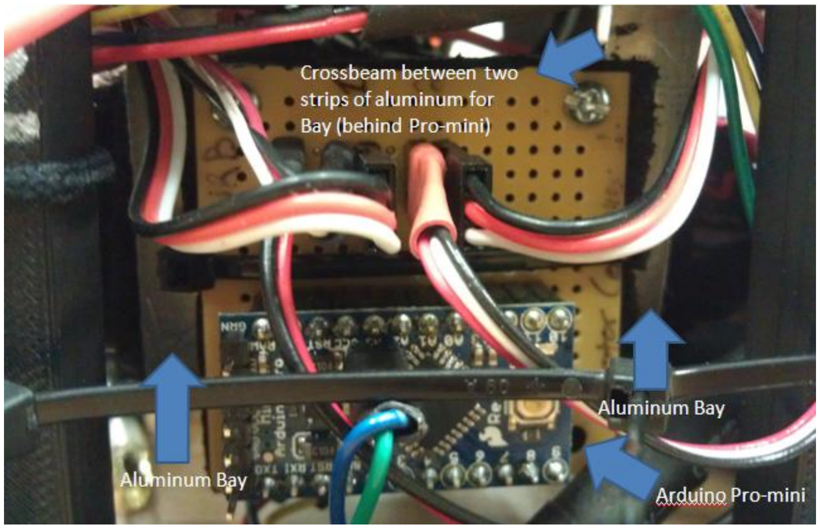

Crossbeam between two strips of aluminum for Bay (behind Pro-mini)
Aluminum Bay
Aluminum Bay
Arduino Pro-mini

Figure 17: Arduino Pro-mini Mounted on Bay Area

After everything had been wired and mounted, the base of the MAV was attached and the ultrasonic was mounted to the underside of the aluminum bay. The Xbee was attached to the top of the MAV structure pointing forward (where yaw was set to zero) using Velcro.

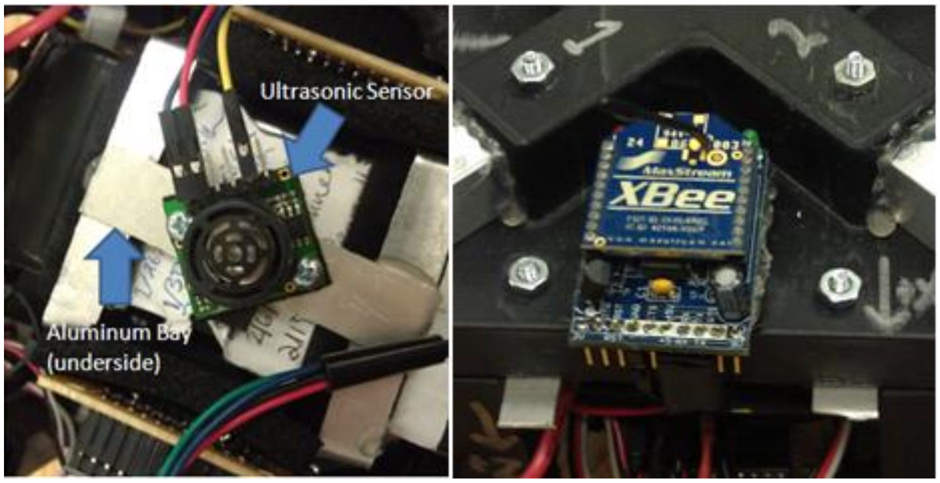


Figure 18: Ultrasonic Sensor Mounted on Bay and XBee Mounted on Top of MAV

3.2.1 Prototype 1

The first prototype consisted only of the main upper body portion (where the arms are attached) and a single motor mount. The purpose of this iteration was to be sure that the ABS plastic rapid-prototype manufactured parts would be sturdy enough for the design. This iteration was then tested to find how much force was necessary before the part failed. The motor mount was used primarily to test the sizing compared to the motors available. It was found that the main body portion needed to be redesigned so it was not easy to pull apart. It was originally rapid prototyped such that it was manufactured in line with the horizontal axis, which allowed a stress fracture along the base of the arm attachment to the rest of the main body. The motor mount required some resizing, because the mount was not wide enough for the motor to have some clearance and to avoid friction with the mount.

3.2.2 Prototype 2

The second MAV was the first to have the aluminum arms attached, as well as the motor mounts and motors (propellers included). It was this prototype that the initial testing was performed on, including motor speed controller testing and separate motor testing. This prototype never lifted off- it was used mainly as a proof of concept for the choice of electronics and to see if the initial code would work. The parts for the base of the MAV were then rapid prototyped and were mostly for proof of concept for the bay area.

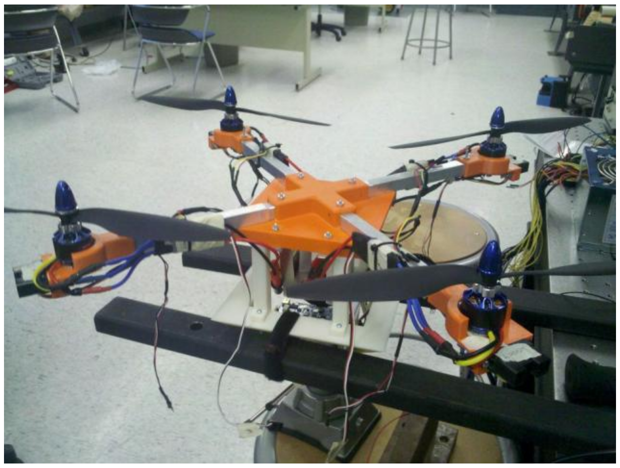


Figure 19: MAV Prototype 2

3.2.3 Prototype 3

Prototype 3 required design resizing. The first alteration was of the arm enclosures; this was to decrease the opening of the rapid prototyped part so that there were fewer vibrations extending from the motors to the main body. This made the aluminum arms fit tighter within the upper body portion.

The next change was in the diameter of the hole in the motor mount as well as the actual diameter of the motor mount. The diameter of the hole was increased so that the magnet used to read the RPMs from each motor could fit through. This was to avoid the issue of trying to attach the magnets after the motor had already been attached to the mount.

The diameter of the motor mount was changed in order to accommodate for the new motors, which were a bit larger than the original motors.

Prototype 3 was to be used for flight tests, however it ended up being more of a proof of concept (please see 3.2.5 Lessons Learned sections for details).

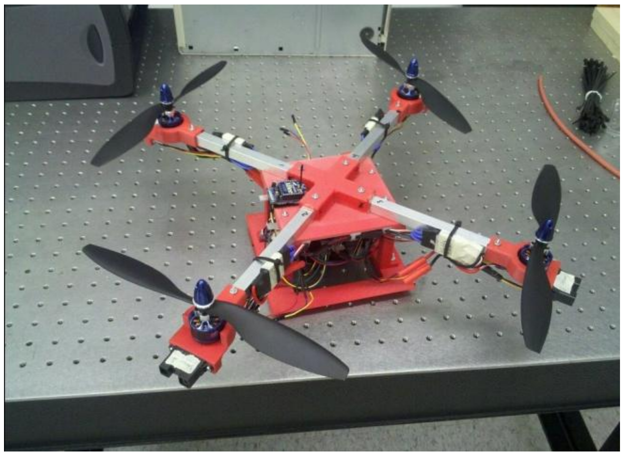


Figure 20: MAV Prototype 3

3.2.4 Prototype 4

The fourth prototype was assembled and used for flight testing. The only changes this prototype went through were the motor mounts. There were not resized but motor mounts broke frequently during testing and were replaced. This was the final prototype.

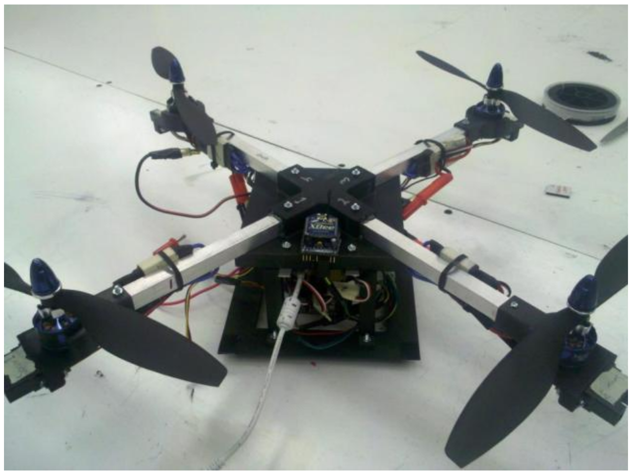


Figure 21: MAV Prototype 4

3.2.5 Lessons Learned

In assembling the third prototype, the last step included adding Loctite 242 to each individual nut and bolt to be sure that the nuts would not fly off the MAV (due to extreme vibrations). However, due to insufficient testing prior to adding Loctite 242 to the ABS plastic structure, this chemical reaction caused the entire structure to melt beneath the harsh adhesive. In building the black MAV, no Loctite was used.

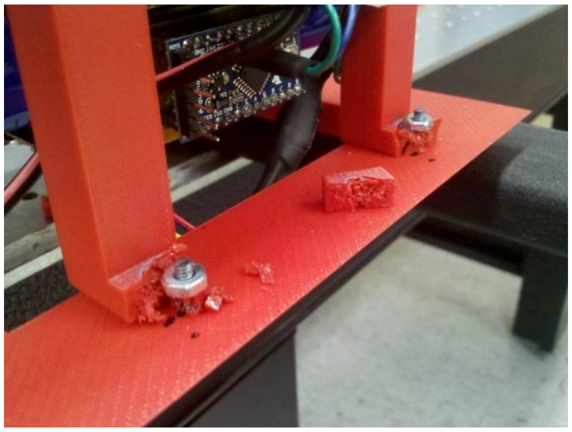


Figure 22: Loctite 242 Melting ABS Plastic

If there had been more time available, the motor mounts would have been rapid prototyped at a different angle. Motor mounts were broken frequently due to the ease with which they could break because they were printed in the horizontal axis (which was where the thrust force was acting).

3.2.6 Propeller Selection

From previous MQPs, an array of various propellers was available. The distinguishing factors between each propeller are the length and pitch of the blade. Since one of the main design requirements was to minimize size, we chose a maximum propeller length of 10 inches, which would provide a large amount of surface area, while still maintaining a minimalist design. A maximum prop length of 10 inches would increase the overall size of the MAV to 20 x 20 inches, from propeller tip to propeller tip. With these dimensions, the MAV would still be capable of easily landing on the UGV; however, it would require a larger amount of clearance for take-off. Several different blade pitch values were chosen to better determine the differences between low and high blade pitch values. Six different variations of propellers of a length 10 inches or less were available from previous MQPs. Each of the six propellers were tested to determine with blade would provide the largest amount of thrust.

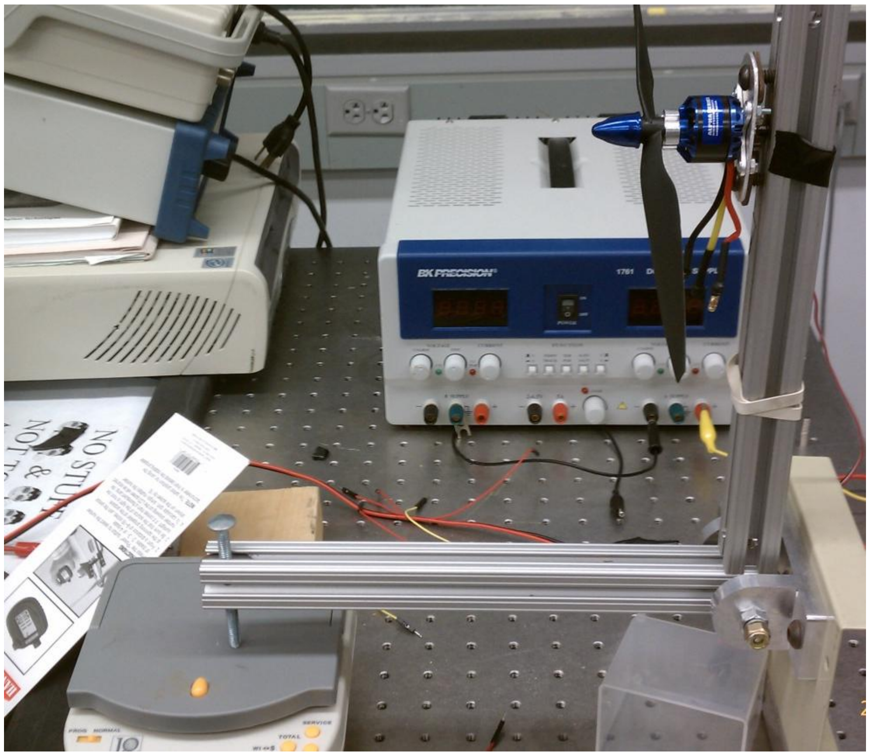


Figure 23: Thrust Testing Stand

To gather data to accurately compare each type of propeller, the team utilized a thrust test stand built by a previous MQP. The thrust stand, shown in Figure 23, was balanced on a digital scale which provided a reading of how much force the propeller was generating. The readings on the scale were multiplied by a factor of $\frac{10}{9}$ to account for the different distances to the pivot point due to different arm lengths. Each propeller was tested at 40%, 70%, and 100% motor power to generate a curve. The data was recorded and graphed using Microsoft Excel. The resulting data is listed in Table 2, and graphed in Figure 24.

Table 2: Propeller Test Chart

Propeller	Percent Power	Scale Reading (g)	RPM (Hz)	Real force (g)
9x6 gray 3 prop	40%	38	2700	42.22
	70%	275	7200	305.56
	100%	335	7740	372.22
9x6 gray 2 prop	40%	62	2760	68.89
	70%	560	7800	622.22
	100%	565	7800	627.78
8x4 black 2 prop	40%	52	3720	57.78
	70%	482	9360	535.56
	100%	505	9600	561.11
10x5 grey 2 prop	40%	75	2640	83.33
	70%	620	7320	688.89
	100%	645	7380	716.67
Large 10” Prop	40%	57	2028	63.33
	70%	600	5760	666.67
	100%	590	5760	655.56
Chosen 10x4.5	40%	75	2300	83.33
	70%	625	6500	694.44
	100%	620	6200	688.89

Based on this experimental data, the 10x4.5 propeller was selected due to a high amount of thrust. The 10x5 propeller was able to generate an overall larger amount of thrust; however it also required higher RPM speeds to reach a similar level. At 70% power, each motor equipped with a 10x4.5 propeller would be able to produce approximately 695g of thrust, giving the MAV a total overall thrust potential of approximately 2.780 kg, or just over 6 pounds.

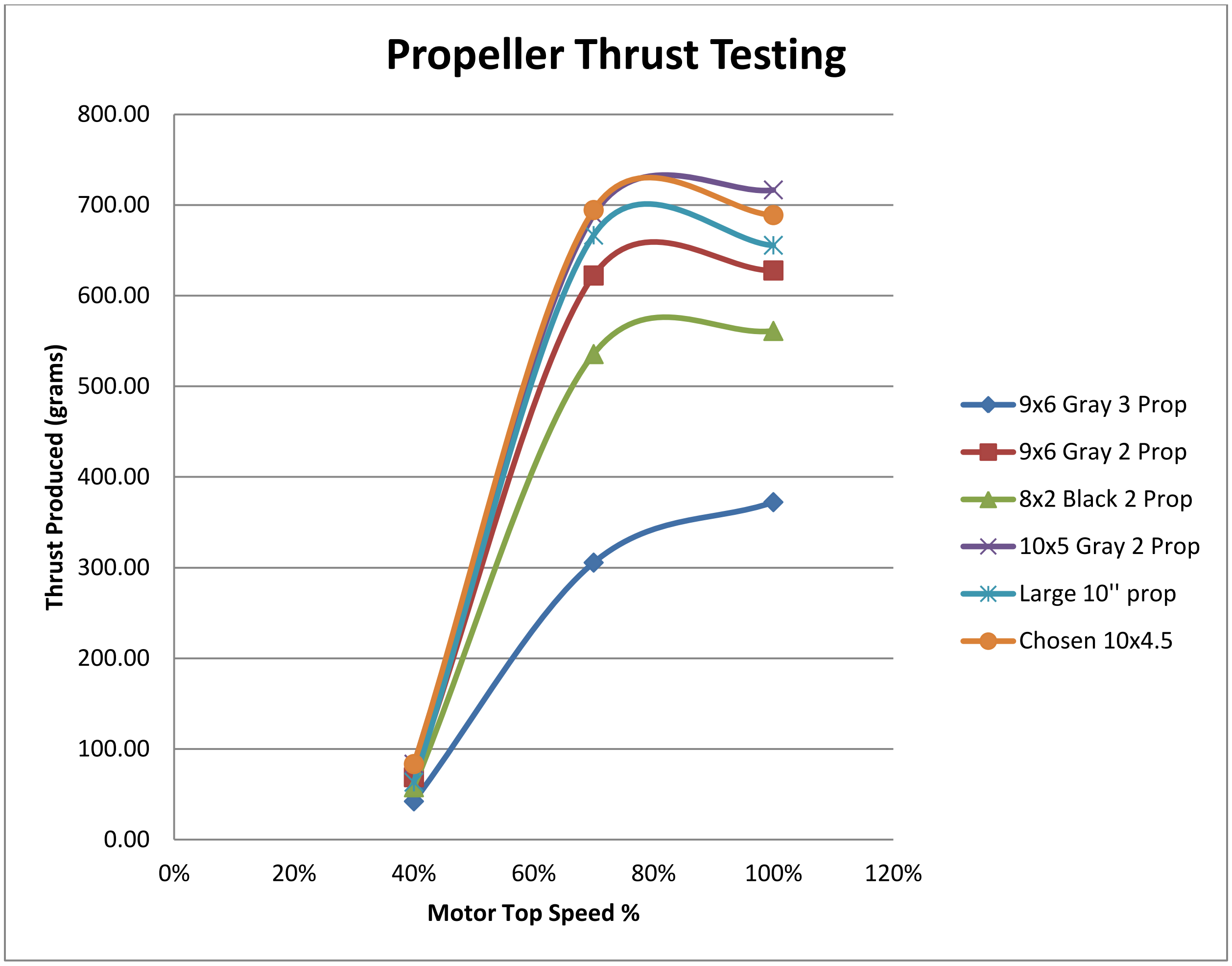


Figure 24: Propeller Test Graph

With a suitable propeller chosen, additional data was collected to determine the thrust and torque curves for the Alpha 370 1200kv motors. Thrust and torque curves are needed to provide the control software to determine how much power to give the motors to perform maneuvers. The graphs are shown in

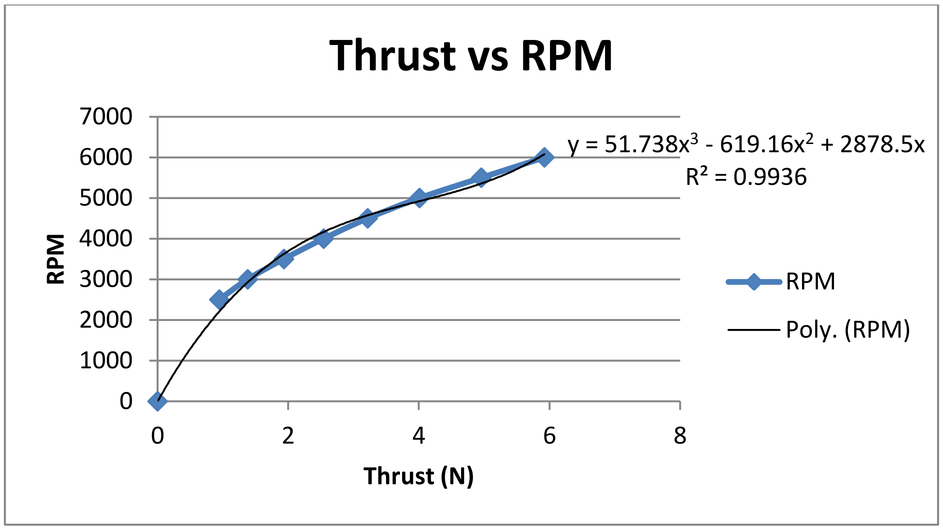


Figure 25: Thrust vs RPM

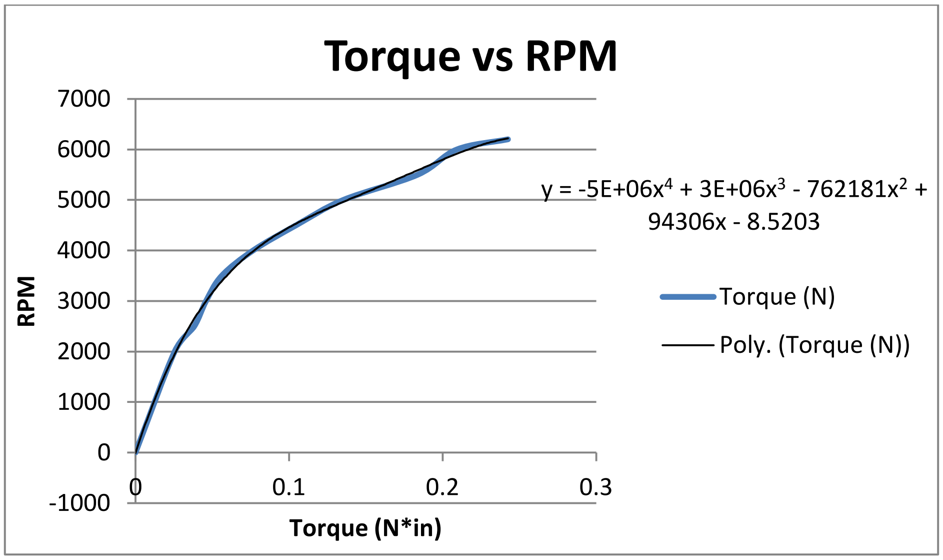


Figure 26: Torque vs RPM

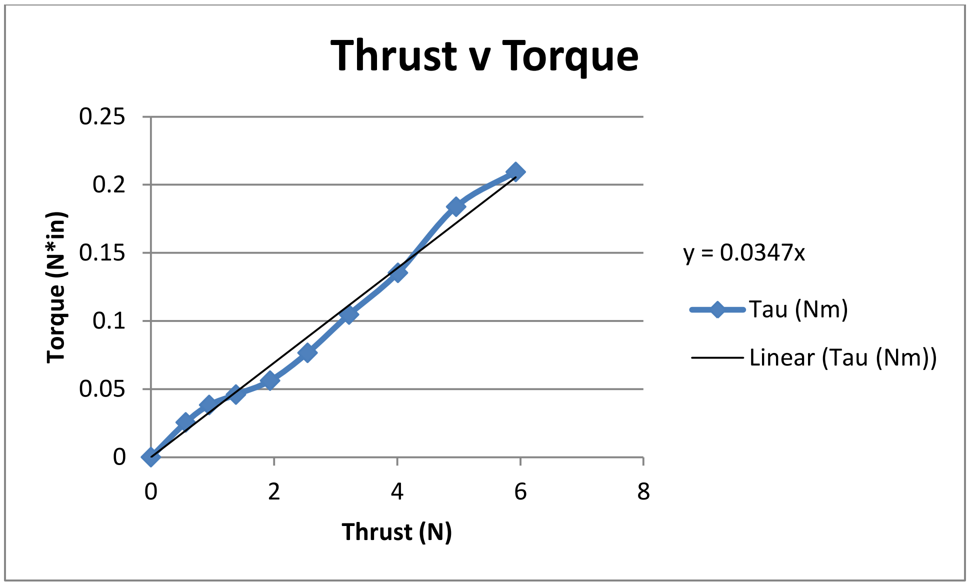


Figure 27: Thrust vs Torque

3.2.7 Propeller Balancing

Due to the high rotational speeds imposed on the blades, improper blade balance could significantly impact the performance of the MAV. Therefore, the blades were balanced prior to installation.

Propeller balancing is accomplished by determining that neither blade is heavier than the other. Two tests were developed to balance the blades. Test 1 had the propellers placed upon a stationary shaft and spun ten times each to be sure that upon stopping the props never stopped in the exact same location. If they had, this would mean that one side of the prop was heavier than the other, and the lighter side of the two would require a weight to equal the heavier side. The weight chosen was simply a strip of scotch tape. However, the weight was not necessary as the props were all properly balanced in this regard.

Test 2 placed the propellers at different angles and observed if the props moved at all (for example, placed at a forty-five degree incline, thirty degree decline, etc). Ideally, the props would be balanced if they did not move from their assigned positions. If they had, that too would have indicated that the props were not properly balanced, and would require weights.

3.2.7 Materials

One of the most critical components of MAV design is material selection. In determining the proper materials for the MAV, it is important to keep in mind certain attributes of the materials. Generally, when looking for a material, some of the most common attributes are weight, strength, stiffness, and cost. Through discussion and design of the MAV, the arms that hold the motors extend from the central location of the UAV, extending outwards in equal lengths. In analysis, it can said to treat the arms as extending beams. To decide on a material type, the group must go further in finding out how these properties define the materials chosen.

3.2.7.1 Stress

Stress is defined as the force, or load, that is acting on the structure over a given area. This is expressed in the following equation:

$$\sigma_{avg} = \frac{F}{A}$$

However, for the stress on a beam, like the arms of the MAV supporting the motors, the type of load must be made clearer. In beams, the transverse load is the load that acts perpendicular to the longitudinal axis of the arm. When these loads act on a beam, they cause the beam to bend or deflect.

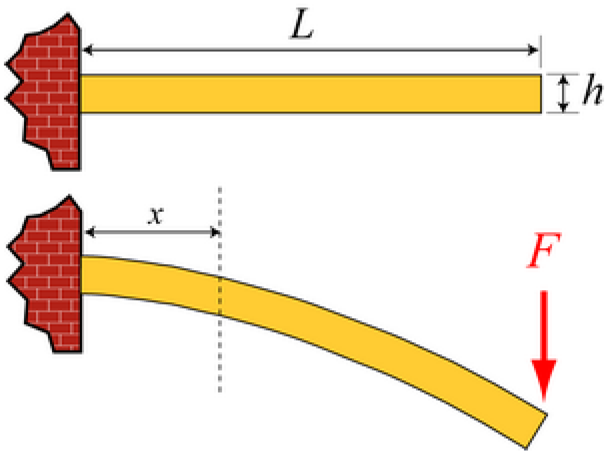


Figure 28: Model of Beam in Bending

The distance the arm will bend, also known as the deflection, depends on the load acting on the beam. If the load is small, then the deflection will also be small. However, at the same time, the radius of curvature, represented by ρ, will be large. The opposite will result with a high load, meaning the deflection will be large and the radius of curvature will be small. We can measure the sharpness of the bend by defining the curvature as κ, and relating it to ρ.

$$\kappa = \frac{1}{\rho}$$

Upon using the following longitudinal strain of the beam:

$$\varepsilon_x = -\frac{1}{\rho} * y$$

Where y is the distance from the normal plane to the desired point in the y-direction, and combining it with Hooke's Law, which is also defined as:

$$\sigma_x = E * \varepsilon$$

Where E is the Elastic Modulus. In applying the longitudinal equation into Hooke's Law, we arrive to the equation below which gives us the variation of the normal stress with the distance y from the neutral plane.

$$\sigma_x = -\frac{E}{\rho} * y$$

From this, we can gather that a material with a high E will have a greater normal stress than a material with a lower E, meaning it can handle higher loads.

3.2.7.2 Young's Modulus and other Criteria

In determining the appropriate materials needed for the MAV, the load that would be applied from the weight of the motors were predetermined using the given weight of the motors as well as the other components that was being located on the arm. From the materials that were found, they can be grouped into 3 categories: wood, polymers, and metals. The defining criteria were high Young's Modulus, relatively low cost, easy machinability, and high stiffness.

3.2.7.3 Wood

The most common material used for model and Remote Control (R/C) aircraft was a selection of woods, varying from cherry, balsa, basswood, mahogany, and other various wood types. Though the densities of the materials are comparatively low to metals, the drawback is the low Young's Modulus and shear strength. As wood offers ease of machinability, low cost, and low density, it serves as a perfect material for models and shells for R/C aircraft.

3.2.7.4 Metals

Metals have been the most commonly used material for structures in aircrafts, from R/C models, to passenger and military planes flying today. In searching for a metal that would have a high strength but low weight, the team found that aluminum was the best option. As a current material used in R/C models, aluminum offers high Young's Modulus with low weight.

Table 3: Mechanical and General Properties of Aluminum Alloys 6061 & 6063

Material	Tensile Strength (ksi)	Young's Modulus (*10^6 psi)	Density (lb/in^3)	Cost ($/lb)
Al 6061	24.9-35.	9.86-10.4	0.0975-0.0985	1.08-1.19
Al 6063	33.8-37.3	10.1-10.6	0.0961-0.0983	1.07-1.17

As displayed in Table 3, the two most common and accessible aluminum alloys, 6061 and 6063, both display excellent properties of strength, density, and cost. In comparison to other metals such as steel, iron, and other advanced alloys, though they may offer stronger properties, weight and cost are what made aluminum a stand-out choice when looking at metals. With their easy machinability, 6061 and 6063 both were top choices.

However, the concern about these metals was vibration. Though the metals would be able to handle the load from the motors well, the vibration from the motors is a cause of concern. Heavy vibrations from the motors would cause structural problems as well as interference in the motors. Though high strength is a desirable quality, a material that could absorb vibration is also desired for the MAV.

3.2.7.5 Polymers/Plastics

The last group of materials researched was polymers and plastics. The advantage that plastics have over metals is their low density, which results in lighter materials. As well, plastics can absorb vibrations better than its metal counterparts, and have higher Yield strengths compared to woods, which means that if bent, it can return to its previous shape better than woods or metals. In completing a search through certain suppliers, it was observed where the range of products was limited to. Though a common material found in today's R/C models, carbon fiber, although with its unique strong properties and low density, was found to be too expensive for the application the MAV needed. Ranging from prices higher than \$20-\$30 for tubing no longer than twelve inches, it was deemed that carbon fiber might not be the suitable choice. In research and comparison into other plastics, the issue arose of machinability, since plastics become fragile when damaged and are not easily repaired. In this focus, one of the purposes for the arms on the MAV was to house the wiring leading from the main central controls of the MAV to the motors that provided the lift. Thus why hollow squared tubing was determined to be the best choice, because it would allow a space where the wires could flow through without being affected by the outside environment, as well as an acting platform for the motor mounts to sit on.

Table 4: Mechanical and General Properties of Select Plastics

Material	Tensile Strength (psi)	Young's Modulus (10^6 psi)	Density (lb/in^3)	Cost ($/lb)
PTFE	1500-3000	0.2-0.239	0.0795-0.0831	6.3-6.93
Polycarbonate	8000-16000	0.304-0.312	0.043-0.048	2.3-2.53
Structural Fiberglass	7000-40000	2.47-2.61	0.058-0.070	1.3-1.95
Polyethylene	1200-1700	0.0249-0.041	0.033-0.034	0.839-0.923

As shown in Table 4, when comparing the different materials, structural fiberglass displays certain traits that are favored for the MAV. With a high range of tensile strength, high Young's Modulus, and low density and cost, this material showed considerable promise. Next, polycarbonate materials also displayed high tensile strength, high Young's Modulus, and low density and cost. As from the graphs in Appendix A, the green area represents the plastic foams, which are certainly not suitable for the UAV. However, the plastics such as the polycarbonate and the fiberglass occupy the dark blue region, revealing high Young's Modulus in comparison to density, which is lower compared to the metals. Carbon fiber plastics, occupying the top right section of the plot, though exemplifying high Modulus and density, come at an expensive cost.

3.3 MAV Dynamics and Control

MAVs require proper control and electrical systems to maintain stability. They commonly use a bevy of onboard sensors to gather and process data, but in order to convert all of that data into commands for the MAV, a series of control equations must be developed. [1]

The generalized coordinates of the orientation and location of the MAV (relative to the Inertial frame) are:

$$q = (x, y, z, \psi, \theta, \varphi)$$

These coordinates may be split up into translational and rotational. The translational coordinates, ξ= (x, y, z) denote the position of the center of gravity of the quad-rotor in respect to the

Inertial frame. The rotational coordinates, η= (ψ, θ, ϕ) denote the orientation of the quad-rotor as the Euler angles of yaw, pitch, and roll, respectively.

Using the Lagrangian equation to summarize the system requires the total kinetic energy with the loss of the potential energy. Translational energy is written as:

$$T_{trans} = \frac{m}{2}\dot{\xi}^T\dot{\xi}$$

where m is the mass of the quad-rotor. Similarly, rotational kinetic energy may be written:

$$T_{rot} = \frac{1}{2}\dot{\eta}^T J\dot{\eta}$$

J is simply the inertia matrix in terms of the generalized rotational coordinates. Finally, the only potential energy that needs to be accounted for in the case of a quad-rotor is the standard gravitational potential energy:

$$U = mgz$$

m is again the mass of the craft, g is the acceleration of gravity, and z is the position of the UAV in the z direction which is point upward. This is also known as the altitude of the craft.

Having acquired the kinetic and potential energies, the summary of the system may be attained using the Lagrangian:

$$L = T_{trans} + T_{rot} - U = \frac{m}{2}\dot{\xi}^T\dot{\xi} + \frac{1}{2}\dot{\eta}^T J\dot{\eta} - mgz$$

To complete the dynamic summary of the system, the Euler-Lagrangian is required:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = F$$

F is a function of F_ξ and τ. F_ξ represents the net translational force on the MAV, and τ represents the generalized moments realized on the MAV. Writing the force vector:

$$\hat{F} = \begin{pmatrix} 0 \\ 0 \\ u \end{pmatrix}$$

Clearly the force vector is zero in the x and y planes because the only force required to be accounted for is the upward force from each of the rotors, or the combined thrust.

$$u = f_1 + f_2 + f_3 + f_4$$

where:

$$f_i = k_i {\omega_i}^2$$

This represents the force of each motor, $i = (1, 2, 3, 4)$. K is a positive constant and ω is the angular momentum of each respective motor (again, represented by i). Using these expressions, it is possible to define:

$$F_\xi = R\hat{F}$$

F_ξ is the translational force on the quad rotor due to the control inputs, and $\hat{F}$ is the upward thrust (the cumulative force from the rotors in the z-direction). R has not been defined as of yet- "…R is the transformation matrix representing the orientation of the rotorcraft."

$$R = \begin{pmatrix} \cos\theta\cos\psi & \sin\psi\sin\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\varphi - \sin\psi\cos\varphi & \sin\psi\sin\theta\sin\varphi + \cos\psi\cos\varphi & \cos\theta\sin\varphi \\ \cos\psi\sin\theta\cos\varphi + \sin\psi\sin\varphi & \sin\psi\sin\theta\cos\varphi - \cos\psi\sin\varphi & \cos\theta\cos\varphi \end{pmatrix}$$

Defining the generalized moments on the yaw, pitch, and roll angles follows:

$$\tau = \begin{pmatrix} \tau_\psi \\ \tau_\theta \\ \tau_\varphi \end{pmatrix}$$

Each of these represent the moment on each respect angle (as shown below):

$$\tau_\psi = \sum_{i=1}^{4} \tau\, M_i$$

$$\tau_\theta = (f_2 - f_4)l$$

$$\tau_\varphi = (f_3 - f_1)l$$

Recall that τ represents the generalized moments on the angles, and so τM_i represents the couple produced by each motor (let M_i be each motor, where as stated above, $i = 1, 2, 3, 4$). l is simply the distance from the center of gravity of the quad rotor to each of the motors, which ideally would be entirely equal, as seen in the above expressions.

Because it is possible to write each the kinetic energy with separate translational and rotational expressions, it is therefore possible to write out the dynamics separately.

Translational Dynamics:

$$m\ddot{\xi} + \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = F_{\xi}$$

This expression is the net force of the translational generalized coordinates. The force is the mass of the UAV times acceleration of each coordinate, summed with downward acceleration of gravity.

Rotational Dynamics:

$$J\ddot{\eta} + \dot{J}\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}\left(\dot{\eta}^T J \dot{\eta}\right) = \tau$$

This expression is the net generalized moment about each of the angles (yaw, pitch, and roll). This net moment is made up of the acceleration of the rotational coordinates (multiplied by the inertial matrix and the velocity of the inertial matrix), and the velocity of these generalized rotational coordinates.

To take into account the Coriolis Effect, we must first define this Coriolis vector:

$$\bar{V}(\eta, \dot{\eta}) = \dot{J}\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T J \dot{\eta})$$

$$J\ddot{\eta} + \bar{V}(\eta, \dot{\eta}) = \tau$$

$$\bar{V}(\eta, \dot{\eta}) = \left(\dot{J} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T J)\right)\dot{\eta} = C(\eta, \dot{\eta})\dot{\eta}$$

Again, J is the inertial matrix in terms of the generalized rotational coordinates, and $\dot{J}$ is the velocity of that inertial matrix. Altogether, $C(\eta, \dot{\eta})\dot{\eta}$ accounts for the Coriolis Effect, in that it accounts for the gyroscopic and centrifugal terms. Since $J\ddot{\eta} + \bar{V}(\eta, \dot{\eta}) = \tau$ it is possible to then come to the conclusion that:

$$m\ddot{\xi} = u\begin{pmatrix} -sin\,\theta \\ cos\,\theta\; sin\,\varphi \\ cos\,\theta\; cos\,\varphi \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix}$$

$$J\ddot{\eta} = -C(\eta, \dot{\eta})\dot{\eta} + \tau$$

In order to simplify this expression, redefine the generalized moments on the yaw, pitch, and roll:

$$\tilde{\tau} = \begin{pmatrix} \tilde{\tau}_\psi \\ \tilde{\tau}_\theta \\ \tilde{\tau}_\varphi \end{pmatrix}$$

Substituting this definition into the prior expression:

$$\tau = C(\eta, \dot{\eta})\dot{\eta} + J\tilde{\tau}$$

Thus:

$$\ddot{\eta} = \tilde{\tau}$$

Because of this, it is possible to rewrite the new equations of motion:

$$m\ddot{x} = -u\sin\theta$$

$$m\ddot{y} = u\cos\theta\sin\varphi$$

$$m\ddot{z} = u\cos\theta\cos\varphi - mg$$

$$\ddot{\psi} = \tilde{\tau}_\psi$$

$$\ddot{\theta} = \tilde{\tau}_\theta$$

$$\ddot{\varphi} = \tilde{\tau}_\varphi$$

where m is the mass of the quad rotor, and u is the total thrust (the sum of the upward forces from each rotor). Each $\tilde{\tau}$ is the generalized moment about the yaw, pitch, and roll angles (the generalized coordinates of $\eta = (\psi, \theta, \varphi)$). g is the acceleration due to gravity, and $\ddot{x}$, $\ddot{y}$, and $\ddot{z}$ represent the translation acceleration (where x and y make up the plane along a landing surface and z is the altitude of the UAV).

3.3.1 Control Strategy: Stabilizing at a Hover

Following Castillo, Lozano, and Dzul's suggestion, the order in which to control the quad-rotor at a hover is to begin with altitude control, then move to control of the yaw, roll angle and y-axis horizontal translational position, and lastly, pitch angle with x-axis horizontal translational position (as seen in Table 5). For detailed derivations of the following equations, please refer to Appendix B.

Table 5: Control Strategy

Phase	*Name*	*Description*
1	Control altitude	u is used to make the altitude reach a desi-red value
2	Yaw control	$\tilde{\tau}_\psi$ is used to set the yaw displacement to zero
3	Roll control	$\tilde{\tau}_\phi$ is used to control the roll ϕ and the ho-rizontal movement in the y-axis
4	Pitch control	$\tilde{\tau}_\theta$ is used to control the pitch θ and the ho-rizontal movement in the x-axis

3.3.2 Control of Altitude (Vertical Position)

Altogether, the expression for the control of the altitude of the quad rotor is:

$$u = \left[\left(a_{z_1}\dot{z} - a_{z_2}(z - z_d)\right) + mg\right]\left(\frac{1}{\cos\theta\cos\varphi}\right) = f_1 + f_2 + f_3 + f_4$$

3.3.3 Control of Yaw Angular Position (ψ)

The expression for the control of the yaw angular position of the UAV (the generalized rotational coordinate ψ about the z axis) is as follows:

$$m\ddot{x} = -(r_1 + mg)\frac{\tan\theta}{\cos\varphi}$$

$$m\ddot{y} = (r_1 + mg)\tan\varphi$$

$$\ddot{z} = \frac{1}{m}(-a_{z_1}\dot{z} - a_{z_2}(z - z_d))$$

$$\ddot{\psi} = -a_{\psi_1}\dot{\psi} - a_{\psi_2}(\psi - \psi_d)$$

3.3.4 Control of Roll Angular Position (φ, y)

The expression for the control of roll in a quad rotor is as follows:

$$\tilde{\tau}_{\varphi} = -\sigma_{\varphi_1}(\dot{\varphi} + \sigma_{\varphi_2}\left(\varphi + \dot{\varphi} + \sigma_{\varphi_3}\left(2\varphi + \dot{\varphi} + \frac{\dot{y}}{g} + \sigma_{\varphi_4}\left(\dot{\varphi} + 3\varphi + 3\frac{\dot{y}}{g} + \frac{y}{g}\right)\right)\right))$$

As stated prior, see all detailed derivations of the following equations in Appendix B.

3.3.5 Control of Pitch Angular Position (θ, x)

The expression for the control of pitch of the quad rotor is very similar to that of the control of the roll angular position:

$$\tilde{\tau}_{\theta} = -\sigma_{\theta_1}(\dot{\theta} + \sigma_{\theta_2}\left(\theta + \dot{\theta} + \sigma_{\theta_3}\left(2\theta + \dot{\theta} + \frac{\dot{x}}{g} + \sigma_{\theta_4}\left(\dot{\theta} + 3\theta + 3\frac{\dot{x}}{g} + \frac{x}{g}\right)\right)\right))$$

3.4 Electrical System

In designing the electrical system for a MAV model there where several diagrams and schematics that were first created to allow for a more precise choosing of the on-board electronics. First, a simple electrical diagram was created to gain a better idea of what needed to be sense and what other components needed to be present on the MAVs. Below is the first draft of the final electrical diagram that was used to understand how everything would work together.

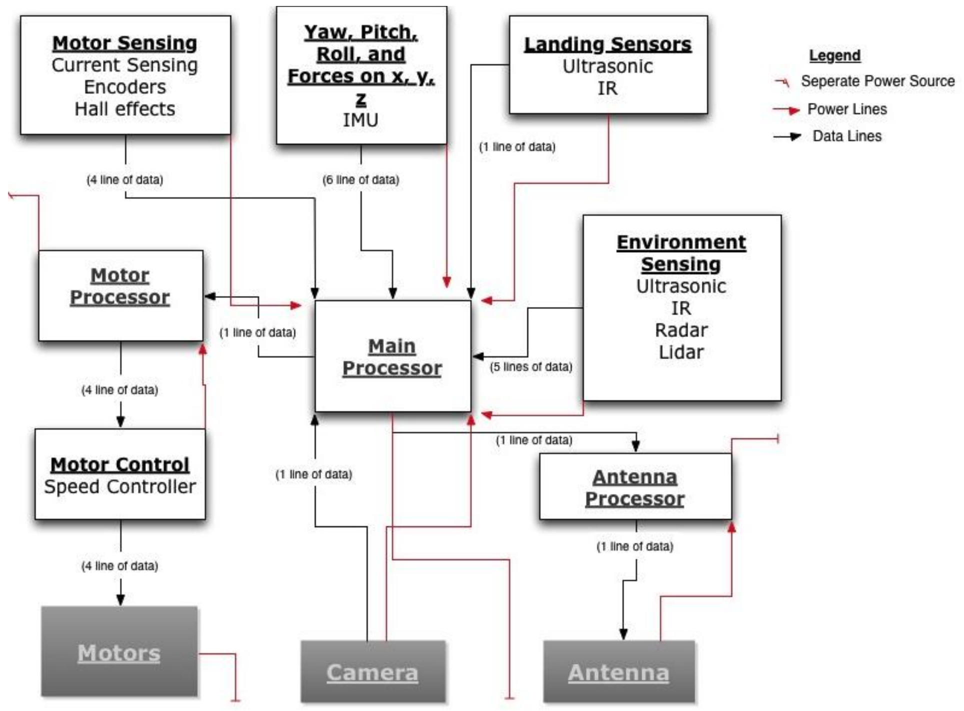


Figure 29: Electrical Diagram Draft 1

From the above model it is clear that there are several different aspects that are needed to be measured in the MAV model. It was discovered that sensors were needed for the environment, landing, motors, and flight dynamics, which included yaw, pitch and roll. In addition, a processor was needed for motor control, antenna feed and a main processor. The final electronics were motors, a camera for the payload and an antenna to transmit the control signal going to and from the UGV.

Next, a more detailed electrical diagram was created to allow for a better understanding of how the numerous sensors were going to be used to measure each factor. The diagram in Figure 30 is more detailed and not only allows for an understanding of the number of sensors and electronics needed, but it of what the main processor will have to be able to compute.

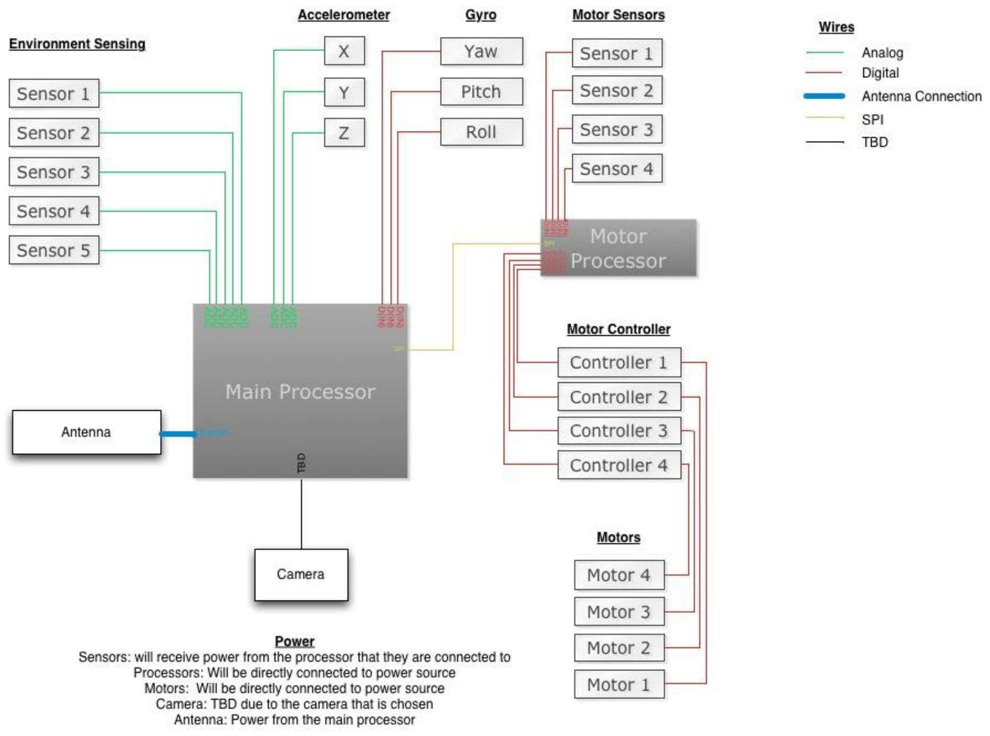


Figure 30: Electrical Diagram Draft 2

From the more complex electrical diagram, it was understood what the main processor and motor processor would require. Research was done on different sensors, processors and controllers that could be used on for the MAV, while making sure to keep weight, power consumption, and operating speeds in mind.

3.4.1 Environment Sensing

To be able to determine the surrounding environment of our MAVs, there were several sensors that could have been used to accomplish this task. In our research we found different range sensors, of which two types stood out to accomplishing our task while still being light, small power use and accurate: Infrared (IR) ranging finding and Ultrasonic sensors.

An IR range finding sensor works by emitting an infrared beam out and receives the beam at an angle. Depending on angle, the IR range finding sensor will be able to calculate the distance an object is away by using triangulation.

An ultrasonic sensor works by emitting a sound pulse, at a specific beam pattern, and then waits for the return of a given pulse. When a pulse is received, the ultrasonic sensor will calculate a change in voltage, due to the strength of the signal that can be converted into a specific range.

By understanding how each sensor works, we decided upon using four IR range finding sensors for detection of possible objects in the way of our MAV. These were chosen over ultrasonic sensors due to how the beam pattern of ultrasonic sensors. As ultrasonic sensors detect further away, their beam pattern increases in size, allowing for detection that may not necessarily in front of the MAV itself. However, an ultrasonic sensor was chosen as the Z-direction sensor to ensure that our MAV will be at the correct height. The ultrasonic was chosen due to continuous changes on the earth's surface and the height that the MAV will be flying. Data sheets for the following sensors can be seen in Appendix C.

Sharp IR Distance Sensor GP2Y0A02YK

The Sharp IR Distance Sensor GP2Y0A02KY was chosen due to its accurate distance measurement, power consumption, and availability. The Sharp IR sensor can measure from 8 to 60 inches and has a power consumption of approximately 115mW. The sensor also has a very small footprint and a weight of 3.5g.   This sensor has all of our requirements that were initially set, as well as being readily accessible and cheap.

LV-MaxSonar-EZ0

The LV-MaxSonar-EZ0 ultrasonic sensor was chosen due to the distance measurement and the wide range that is needed for maintaining an accurate height. The MaxSonar can take measurements from 0 to 254 inches and an approximate power consumption of 15mAW.   Due to its small footprint and lightweight, 4.3g, it falls directly into the requirements initially set.

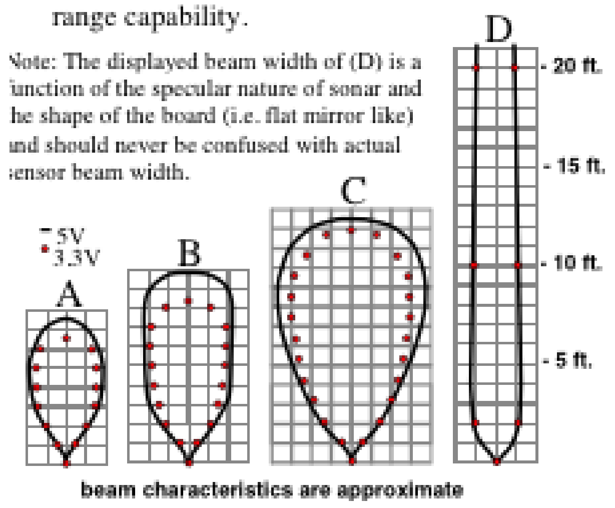

range capability.
Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.
5V
3.3V
A
B
C
D
20 ft.
15 ft.
10 ft.
5 ft.
beam characteristics are approximate

Figure 31: Beam Pattern Characteristics of LV-MaxSonar-EZ0

The beam pattern characteristics for the EZ0 can be seen in Figure 31. The grids shown on the figure are 12 inch squares. The beam pattern for the EZ0 shows that at different ranges the width of the pattern changes. It can be seen that each pattern has a very large width, which could cause much noise in our Z-direction measurement. To help solve this problem, more than one ultrasonic sensor was researched and ruled as a possible solution to our project.

LV-MaxSonar-EZ4

The LV-MaxSonar-EZ4 ultrasonic sensor has all of the same characteristics as the previously mention ultrasonic sensor, however there is one crucial difference: the beam pattern. Figure 32 shows the difference in pattern compared to the EZ0. The same grid formation, twelve inches per block, was used when referencing the pattern to physical distance.

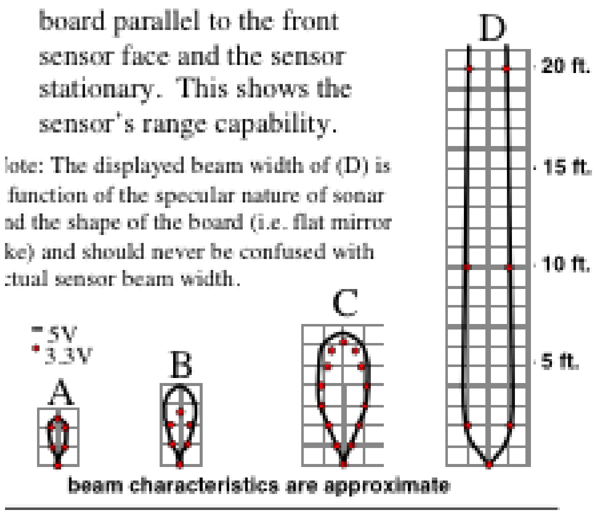

board parallel to the front sensor face and the sensor stationary. This shows the sensor's range capability.
lote: The displayed beam width of (D) is function of the specular nature of sonar nd the shape of the board (i.e. flat mirror ke) and should never be confused with ctual sensor beam width.
5V
3.3V
A
B
C
D
20 ft.
15 ft.
10 ft.
5 ft.
beam characteristics are approximate

Figure 32: Beam Characteristics of LV-MaxSonar-EZ4

As it can be seen, Figure 32 shows how there is a much narrower beam pattern for the EZ4 compared the previous EZ0. With a narrower beam pattern, much surrounding objects that cause noise in the measurements can be resolved. To ensure that either of these sensors will work, test must be run to decide which will be the best sensor for our project.

3.4.2 Motors

One electrical component that held great importance to our project, were motors. Motors were important to this project because they were what allowed the MAV to fly. When thinking about requirements for the motors, there were many aspects that needed to be discussed and looked over. The motor requirements were the power consumption, weight, price, RPM (revolutions per minute), and recommended propeller size. These factors were chosen due to the area and size of the MAV itself.

Originally only one type of motor was to be used on the MAV. However, two motor types were needed due to a lack of available resources.

Alpha 370 (1200kv) Motor

The Alpha 370 motor was chosen due to its high RPM output, low power consumption, and recommended propeller size (9x6). The Alpha 370 uses 133W and can produce 1200 RPM/V, which can be used to produce sufficient lift. The Alpha 370 is not very expensive and has a small footprint; however it weighs 50g making it slightly heavier than expected. Even though the Alpha 370 has a higher weight, it can provide approximately 3.3lbs of lift by only being powered 75% of full capacity, allowing it to have great lift for less power.

Optima 370 (1360kv) Motor

The Optima 370 brushless motor is a slightly more powerful version of the Alpha 370. Considering that they are from the same manufacturer, there are some similarities between the two motors, including footprint. The Optima 370 uses 150W and can produce 1360 RPM/V. This means that it will take more power than expected, but will also produce more than expected. The Optima 370 can produce just as much lift as the Alpha 370 and then some.

3.4.3 Motor Sensing

To ensure that the motor of choice will operate at the specific speeds needed for control, there were two types of sensors used: a speed controller and a Hall Effect sensor. These two electronics were chosen due to the size and power consumption.

US1881 Hall Effects

A Hall Effect sensor is a sensor that detects a change in magnetic field and will return a high or a low voltage. There were several factors to consider when selecting a Hall Effects sensor: size, power needed, and sensitivity. Considering that there are many different brands of Hall Effect sensor, our team chose the US1181 Hall Effect sensor. This sensor can be powered up to 24V, but its minimum value is 3.5V. It has a rise and fall time of 10 microseconds and a switching frequency of 10Hz. For more information on the US1881 please refer to Appendix C.

To make the Hall Effects work properly on the MAV, neodymium magnets were added to the shafts of the motors. These magnets were used so that when the motor would rotate the magnet would too, allowing the Hall Effect to measure the RPM of the motor. To output the signal from the Hall Effect, a voltage divider was created to allow for a significant change in power be seen.

Exceed RC Proton 30A Speed Controller

When selecting a speed controller there are many different brands to choose from, just like the Hall Effects. Based on the criteria of price, maximum amperage, and functioning voltage the Exceed RC Proton 30A Speed Controller was chosen. The Proton 30A could handle the high amperage that was needed to power each motor. The Proton was also fairly priced and referred to by several places that were mainly using motors such as ours. For more information on the Proton 30A please refer to Appendix C.

3.4.4 Microcontroller and IMU sensor

When choosing a processor or microcontroller for any embedded system project there are certain attributes needed to be considered. The first thing to consider is about how many I/O devices the microcontroller will need to handle and how those devices will need to communicate it. From the electrical diagram (Figure 30: Electrical Diagram Draft 2) the number of I/O devices was now known and therefore the microcontroller could be chosen. From research, the team found out that there is a large community of people that develop their own remote controlled and autonomous aerial vehicles. A common microcontroller used by these groups (DIY Drones and AeroQuad) was built using the Arduino architecture. DIY Drones likes to use the ArduPilot Mega, which is essentially an Arduino Mega 2560. AeroQuad, although they have other Arduino architecture boards, uses the Arduino Mega 2560. The reason the team wanted to find if there was a commonly used microcontroller was because there would be a community of people able to help if there was ever a problem the team could not fix on its own. After looking through DIY Drones and AeroQuad, it was decided that the ArduPilot Mega microcontroller would be used, which for all purposes is an Arduino Mega 2560. The difference between the two boards is that the ArduPilot already has some libraries developed for autonomous flight that could be useful. The other difference is that the ArduPilot has an IMU sensor board developed for it and these libraries.

An inertial measurement unit (IMU) is what is used primarily by the MAV for attitude control. An IMU normally consists of a 3-axis accelerometer and gyroscope. The reason this is useful is because the accelerometer gives the MAV data on its acceleration in all directions. The gyroscope gives the MAV data on the rate at which the yaw, pitch, and roll of the MAV are changing. If the team had chosen the generic Arduino Mega 2560 then another selection would have to have been made for the IMU. The IMU board for the ArduPilot has a connection on it for more sensors that can be integrated right into any project with the ArduPilot libraries. Some of the additional supported sensors on the IMU board are an Xbee antenna connection, a GPS connection, and a magnetometer. All of these sensors were considered for use on the MAV and made the IMU board appealing more attractive option. Another useful sensor package the IMU shield has is a barometer for sensing pressure which can then be related to the MAV's height. Lastly, the gyroscopes on the IMU board are vibration resistant inverted gyroscopes. This helps when programming the MAV because some of the noise will already be filtered out.

The team decided that the ArduPilot and corresponding IMU shield would be the microcontroller and IMU sensor for this project. The available sensor packages and connections along with the libraries and community support make the ArduPilot a clear choice for an MAV project.

3.4.5 Arduino Pro Mini

The Arduino Pro Mini was selected to be the motor processor that would take commands from the main processor (ArduPilot), run the PD control loops for motor speeds, receive data from the Hall-effect sensor and compute motor RPM, and communicate to the Exceed-RC speed controllers. The reason the team wanted these motor processors was to decrease the load on the main processor when it had to compute the control equations for flight. The team wanted a small processor to play the role of the motor processor and we wanted to option to use PWM, external interrupts and Analog/Digital inputs/outputs.

A few Arduino boards were looked at to fill this processor spot, the Arduino Mini, the Arduino Pro Mini, and the Arduino Nano. Previous iterations used a Picoduino which is an ATMEL 328 on a breakout board, this was used for motor control. After looking at the specs for the Arduino boards the choice was to select the Arduino Pro Mini. The Arduino Mini, Arduino Pro Mini and the Ardunio Nano all use the ATMEL 168 (3.3V) or the ATMEL 328 (5V) processors and the Mini and Pro Mini are practically the same. The reason the team went with the Pro Mini was because we could program the Pro Mini with a single FTDI cable (the same cable that programs the ArduPilot) and the Nano uses mini USB. The Arduino Mini needed a second programming board in order to program the Mini and we di not want to spend more money for the programmer when we were already buying more FTDI cables to program the ArduPilot. Both the Mini and Pro Mini are relatively the same price when the programmer board would double the price of the Mini. Looking back the Nano would be a viable option because it does not need the extra programming board and mini USB is the same programming cable for the Arduino IMU Shield which is very common. The only downside to the Nano is the board is not made by Sparkfun and was not on the Sparkfun website where the team ordered most of the electronics from.

3.4.6 XBee

During testing and operation of the MAVs the team needs to be able to communicate with the MAVs and having a wire dangling from the MAV makes it hard for the MAV to complete its mission. Instead we need a way of wireless communication. The team's choice was to use XBee RF modules for wireless communication. The original decision was to use XBee-Pro modules because they had an increased range of one mile line of site outside but in finding the parts from previous iterations the team found normal XBee modules that would fulfill our needs. XBee-Pro has a range of 300' in indoor and urban environments where the XBee has a range of 100' indoor/outside (XBee datasheet). Along with the decreased range the XBee has lower power requirements and TX/RX max currents than the XBee-Pro which should increase our battery life. Originally we selected the XBee-Pro because of its inceased range and the modules can be swapped out without changing the requirements in the microprocessor.

In programming the XBees for communication the team use the program XTCU that is free to download. XTCU is a program written to provide up to date drivers and firmware for the XBees as long as signal selection, baud-rate selection, and communication options. This program was very helpful in paring the XBees for communication and debugging communication problems when we thought that the XBees were bricked but it was actually a programming error.

3.4.7 Electronics Packaging and Schematic

When the group was nearing the end of motor testing, the next critical step was packaging. One of the key issues that the group faced was weight, where the determined maximum weight of the MAV was 3 pounds. To make sure the MAV maintained under that tolerance, reductions in weight in the MAV structure were made. In one initial concept design, the MAV was fitted with a cargo bay to carry the electrical components. To help ensure the safety of the ArduPilot and the IMU shield, the cargo bay was to serve as a cage to encase the components and to protect them from any impact. The drawback came in weight, where the cargo bay made the MAV estimated weight higher than the maximum target weight. Through this conclusion, a new solution to the electrical mounting had to be made.

The concept of Line Removable Unit (LRU) became crucial for the electrical components of the MAV, especially during testing. The group knew that in case any of the components had to be replaced, the electrical components had to be easily accessible as well as removable. However, the main problem that came with the mounting of the components was vibration form the motors. Heavy vibration from the four motors would cause any loose mounting of components to become undone, potentially damaging the components critical to the MAV.

Using SolidWorks, the group was able to design different configurations to decide upon which packaging formation would allow for easy removal of the components, as well as the simplest wiring configuration. The first configuration as seen in Figure 33 explored the idea of mounting the components underneath the top portion of the MAV. This revealed that the electrical components had enough space for the wiring as well as proving that the cargo bay was no longer necessary. The problem that was faced with this configuration was the IMU shield. With the blue IMU shield mounted to the MAV, this did not leave any room for the wiring to the XBee antenna. The next issue came from the mounting. Since the IMU mounts on top of the ArduPilot through connections, the only component between the two that had mounting holes was the ArduPilot.

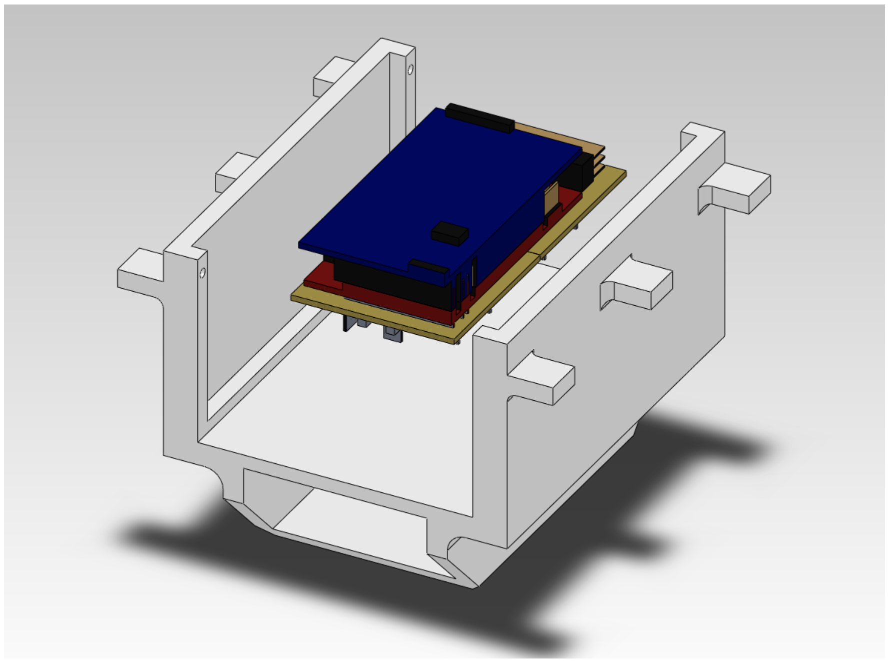


Figure 33: Electronics Packaging Configuration 1

Without the Speed controllers in the assembly, the first configuration was already deemed obsolete due to the problems faced. This led to the second configuration. From what can be seen from Figure 34, the Speed controllers were located underneath the Arduino Pro Minis in a sideways stacked formation. The blue wires on one end of the Speed controllers connected to the motors, while the red and black wires connected to the battery source, providing power to the MAV. Each Arduino Pro Mini controlled a group of two motors, and the speed controllers were oriented to their respective groups. The ArduPilot and IMU shield was placed underneath the Arduino Pro Minis, keeping the orientation the same from the previous configuration. By having space dedicated to the wiring above the IMU, this allowed for easy connection to the XBee. On the other hand, this configuration would have required separate brackets or mounts for the ArduPilot, as well as the two Arduino Pro Minis. As well, the Speed controllers would also need to be mounted in a certain manner that would not have allowed them to move or fall from the MAV.

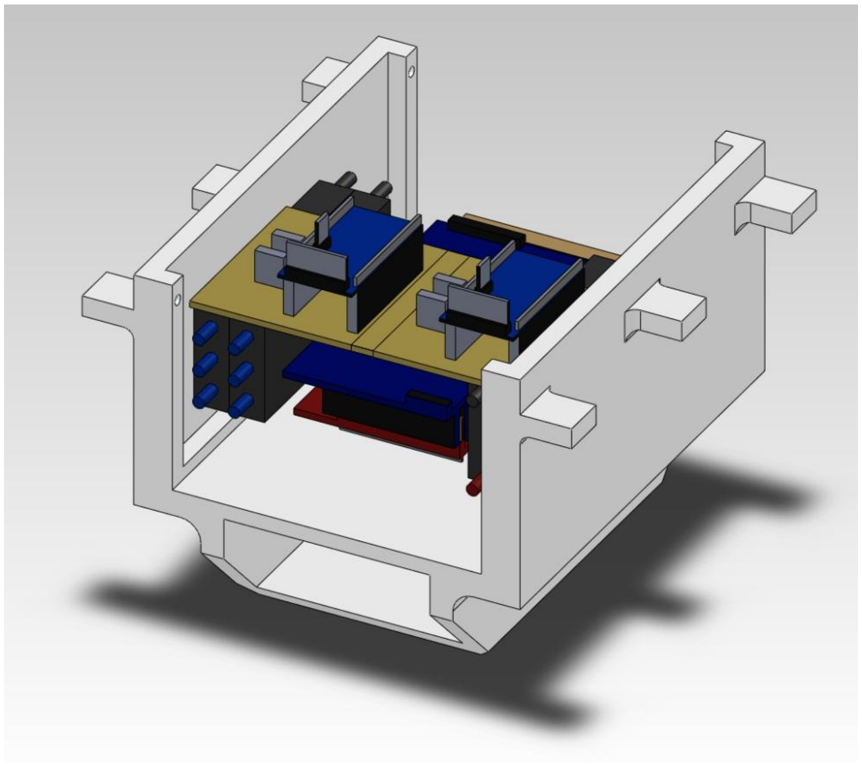


Figure 34: Electronics Packaging Configuration 2

The next configuration that was discussed can be seen in Figure 35. In the third configuration, the components were arranged perpendicular to the sides where the walls of the cargo bay used to be. The wires from the Speed controller had less room to route from the MAV body to the motors. However, this configuration allotted for the components to be mounted horizontally, which meant more room for the wiring as well as for the battery. The ArduPilot was located in the position that allowed for it to be mounted underneath the top structure of the MAV. The intent of the third configuration had the wires follow through the inside of the hollow arms of the MAV to the components mounted on the arm. It was not realized until later that the Speed controllers needed to be moved outside of the main body of the MAV. The reason for this was due to the extreme heat generated from the Speed controllers, it was placing the other components around it in jeopardy of damage. As well, after numerous times of changing wires and components, it was determined that routing the wires through the arms of the MAV would have made the build process much more complicated.

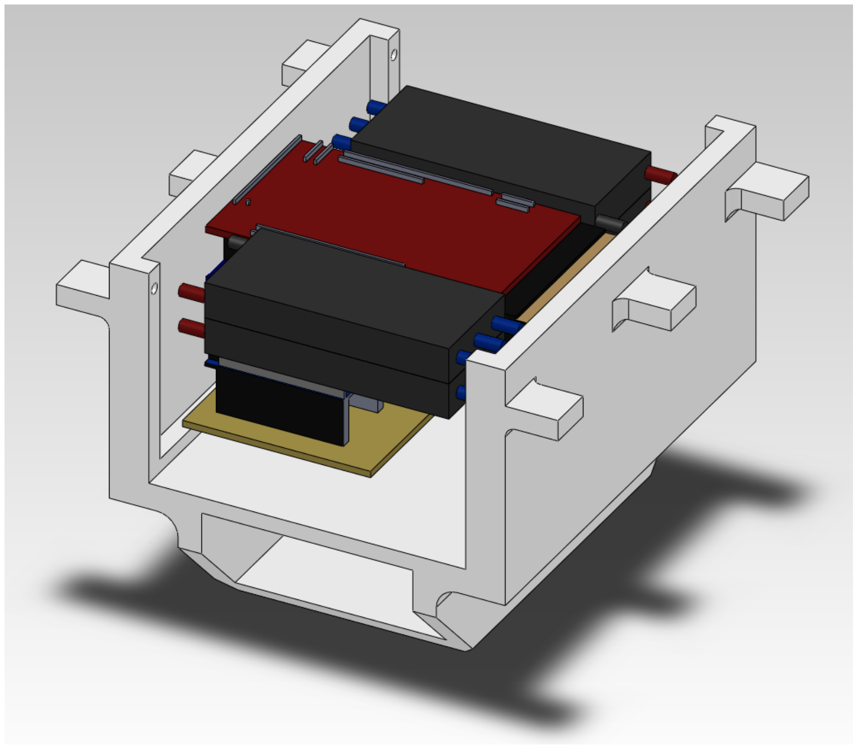


Figure 35: Electronics Packaging Configuration 3

Through these different configurations and lessons, the group arrived to a final configuration that can be seen in Appendix F: MAV Electrical Schematic. As can be seen in the CAD drawing of the electrical schematic, the components of the MAV are all placed, as well as the wiring required. The drawing was divided into sections of the MAV body, arms, top and bottom, that organized the components and wiring bundles. By having the Speed controllers mounted on the arms of the MAV, this allowed for the heat to be dissipated through the arms, preventing any of the other electrical components from overheating. The drawing also illustrated the complexity of the wiring configuration. With the four electrical components mounted on each of the arms of the MAV, the MAV body was dedicated for the IMU, ArduPilot, Arduino Pro Minis, as well as the battery.

The purpose of the schematic was to display the packaging of the MAV, the placement of the components, as well as to display the communication between each of the components. With the schematic, the group could follow the wiring configuration, analyze how the system communicated, and use the schematic for future construction of the MAV.

3.5 MAV Programming

To program the MAV, several requirements had to be researched to better understand how the on-board sensors and flight equations could be used. These requirements may range from low-level to high-level programming. This means that there would be lower-level code that will handle the actual hardware and electronics. There will also be higher-level code that will help with path planning and control of the actual MAV.

To begin, the higher-level programs must be thought of first. The highest-level command will be the control commands that the MAV will receive to react to a situation appropriately. These commands will be relevant to the MAV's overall objective and will receive these commands from the UGV. The preceding level will be path planning for the MAV. This level is where the MAV must use information from calculated from the onboard sensors and overall objectives to learn where it must fly and how to avoid obstacles that are present.

The next level is where the MAV will take in the onboard sensor readings to better calculate where it is, relevant to the UGV, and any obstacles that are present. Using this information the MAV is able to plan a better path in the next higher level. The lowest level of the MAV program will be the code to control all of the sensors. Here is where all of the sensor values are received and processed and used in the level above.

With the understanding of the above levels, it is known that the two higher levels are related with both the UGV and MAV. The two lower-levels are strictly related to the MAV only and all of the hardware that it incorporates. The following UML (Universal Modeling Language) demonstrates how the two lower-level programs are related.

3.5.1 Motor Control

To control the motors there are several smaller functions and hardware that must be written together in such a way that they will operate each motor individually for better control. This functionality mainly includes hardware level programming to control the specific RPM of each motor. To achieve this control, the Hall Effect sensors are incorporate as a digital signal to generate a PWM (high and low) signal. The RPMCalc() then uses these signals to generate the approximate error the motor is off. Then the PIDUpdate() will receive this error and use it to recalculate the specific PD control that is required to obtain the specific RPM required.

3.5.2 IMU

To effectively judge the forces and the angular rotation of each MAV an IMU (Internal Measuring Unit) was attached to return the specific degrees per second of rotation and the amount of force in each of the three axes. To obtain these readings, the IMU has a three-axis accelerometer, a two-axis gyro (for the X-axis and Y-axis), and a one-axis gyro (for the Z-axis). Therefore this lower-level program will return the change of force and rotation for each of the three-axes.

3.5.3 Control Equations

This is the most important aspect of the MAV code, and it is where everything will be put together to better understand where the MAV is locally, the MAV's local environment and how to better correct itself within the local environment. In this second level program, the MAV will pull all of its readings from the three previously stated programs. By doing this the control equations program will be able to maneuver in a local environment and will allow the MAV to control the motors accurately after receiving commands from the higher-level programs.

3.5.3.1 Flight Simulation

Matlab was utilized to simulate the control environment by using the control equations specified above (derived in Appendix B). This simulation was used to find and adjust the gains for pitch, roll, yaw, altitude, and lateral location. The gains were tuned in this order and a stable system was created with noise in each variable to simulate actual readings from the sensors. These noisy values were passed through a filter and then used in the control equations to produce a realistic simulation (please see Matlab code and results in Appendix D).

3.6 Sensor Testing and Calibration

Before any of the sensors are operational, each sensor must run through several tests to ensure that all of the data being received is calibrated, accurate and meets the standards that we require. Each sensor type has its own method for testing that is described in the following section. These sensors were test through the ArduinoMega 1280 to ensure that the readings will be compatible with the main processor on the MAV.

3.6.1 IR Sensors

When first programming IR sensor, they returned a raw Analog-to-Digital Converter (ADC) value. This value seems random, however it returned a constant change when measuring items at different distances. The goal of IR sensor testing and calibration was to find the specific equation that can be used to generate a correlating distance in centimeters.

To effectively test these sensors, code was first written to return the raw values for any given IR sensor. Next several distances were measured out in centimeters to allow for correlations between the raw value and outputted distance. For our testing, we had measured out 150cm in increments of 5cm to allow for accurate data, knowing that the smallest distance the IR sensor can measure is 6 inches or roughly 15.24 centimeters. Then an object with a fairly large flat surface facing the sensor was placed, in front of the sensor, beginning at fifteen centimeters (due to the minimum range labeled in the datasheet, and a value was recorded for each distance leading up to the 150cm mark, creating our sensor to automatically produce centimeters to match the units of our project.

The data was then inputted into Excel and graphed using a scatter plot. Each data point was analyzed to achieve the best-fit line, which would accurately return the desired equation needed to convert the raw ADC values into usable distance values. The following chart and graph demonstrate how this data was interpolated can be seen in Figure 36.

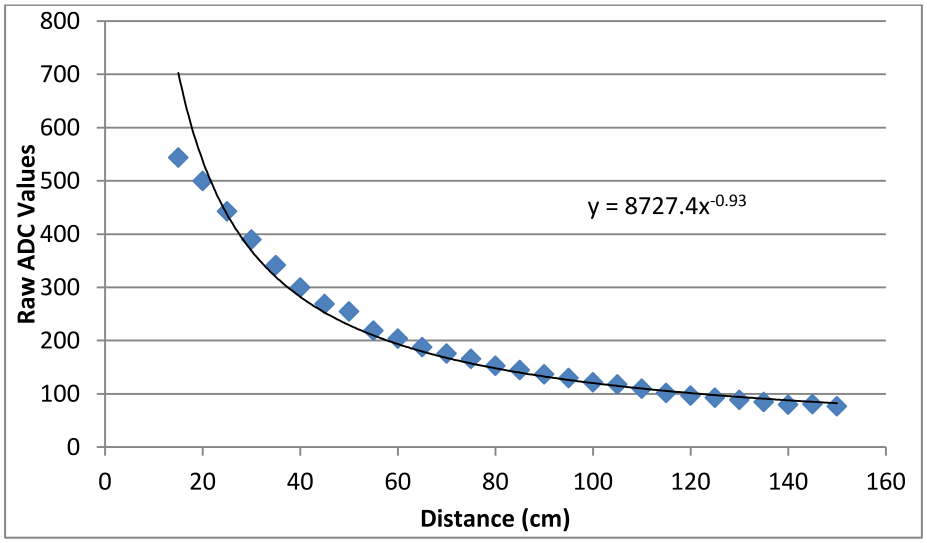


Figure 36: Raw ADC valudes vs Distance

This figure shows that as the distance increase, there was a negative exponential trend. From this, a best-fit exponential line was generated. This equation was then added into the IR sensing code to allow for each sensor to return distance instead of raw ADC values. Each IR sensor has a different equation to be used; therefore the rest of the sensor calibration graphs can be seen in Appendix E.

3.6.2 Ultrasonic

To test and calibrate the ultrasonic, there is an equation involving information from the data sheet must be used. When converting the analog signal first there is an equation to discover the scaling that should be used:

$$V_i = \frac{V_w}{512}$$

where V_i = inch per volts scale and V_{cc} = supply voltage. For both of the ultrasonic sensors on which we were testing turned out to be the same at 0.0098 inches per volt.

Next this value was used in another equation to produce the range. The equation is as follows:

$$\mathrm{R_i} = \frac{\mathrm{V_m}}{\mathrm{V_i}}$$

Where: R_i = the range measured in inches

V_m = the measured voltage from the sensor

V_i = the scaling factor

By using these equations, a specific range can be seen in inches, which for us, was then converted into centimeters to match the rest of our units.

Even though these sensors were run through these series of equations, there was still noise in the system creating our measurements to be very askew. After some research through the manufacturer, it was discovered that for UAV and robotic applications, each sensor must be run through a circuit to eliminate the electronic noise. The website for this specific information is as follows: http://www.maxbotix.com/tutorials.htm.

After applying all of the filtering need for each sensor, they could finally be tested to determine which one would be the proper fit for our project. To test this, each sensor was used to measure specific distances, much like the IR sensors, and the values were then compared. After comparing the values, it was determined that the EZ4 had much more accurate data due to its beam pattern.

3.6.3 IMU

Two separate test and calibrations had to be run on the IMU: one for the accelerometer and the other for the two gyros. The accelerometer test involved the force of gravity, represented in g-force, while the gyros entailed more complex testing.

The accelerometer began with code that would return raw values associated with the specific axis in testing. Starting with the Z-axis, these values were recorded for when gravity was acting in both the positive and negative directions, with negative value being the lowest reading. Then a zero value was recorded for when the accelerometer was positioned in such a way that gravity no longer acted upon that axis, but rather on one of the other two. These values were applied in such a way that the raw values from the accelerometer for that access was subtracted by the zero value and then divided by the difference from the positive and negative values. An example of this equation can be seen as follows:

$$Reading = \frac{\big((raw\ Value) - Zero\ Value\big)}{Positive\ Value - Negative\ Value}$$

The remaining two axes were calibrated in a similar manor

As mentioned before, to calibrate the two gyros on board the IMU was more difficult than the accelerometer. First there are two common equations that were used to better calibrate each gyro using specifications mentioned in the data sheets in Appendix E. The first equation which will convert the raw ADC values to voltages is as followed:

$$ADC\ to\ V = \frac{(Raw\ ADC\ Value) * (Max\ Gyro\ Voltage)}{4096}$$

where 4096 is the integer related to the number of bits for the ADC.

After this conversion there is a second equation that calibrates the gyro to read degrees per second for the change about the rotation of an axis. The equation is as follows:

$$Rotation\ Rate = \frac{(ADC\ to\ V) - \left(Gyro_{zero\ voltage}\right)}{Gyro\ Sensitivity} + offset$$

From here the gyro was calibrated to read zero while not in motion. However these values may not be accurate.

To ensure that the calibration was correct, a test stand was created. This involved a window motor (the motor that was used for our UGV), a 9V battery, an external power source, an Xbee and in our case an empty shoebox. The shoebox was then convert to be placed on the motor in such a way that when the motor rotates so will the box. Next the 9V battery was connected to the Arduino Mega1280 and the IMU shield so that it will power the Arduino, IMU shield and Xbee. Finally the motor was given a low power from the power supply to turn the box. When this happened it rotated the box at a constant rate to allow us to accurately measure rotations per minute, allowing us to calibrate our gyros.

3.7 Quad-rotor Testing

Once the quad-rotor was fully assembled, a controlled environment was built in the lab to ensure the safety of the team as well as the MAV. As shown below in Figure 37, the frame was built out of wood, and plastic netting formed the walls. If the quad-rotor was to accelerate unexpectedly, the strong but flexible netting would protect it from damage. A sheet formed the far wall that allowed for easy access to the inside if changes needed to be made. During testing, the sheet was fastened to the wood frame and therefore fully enclosing the quad-rotor.

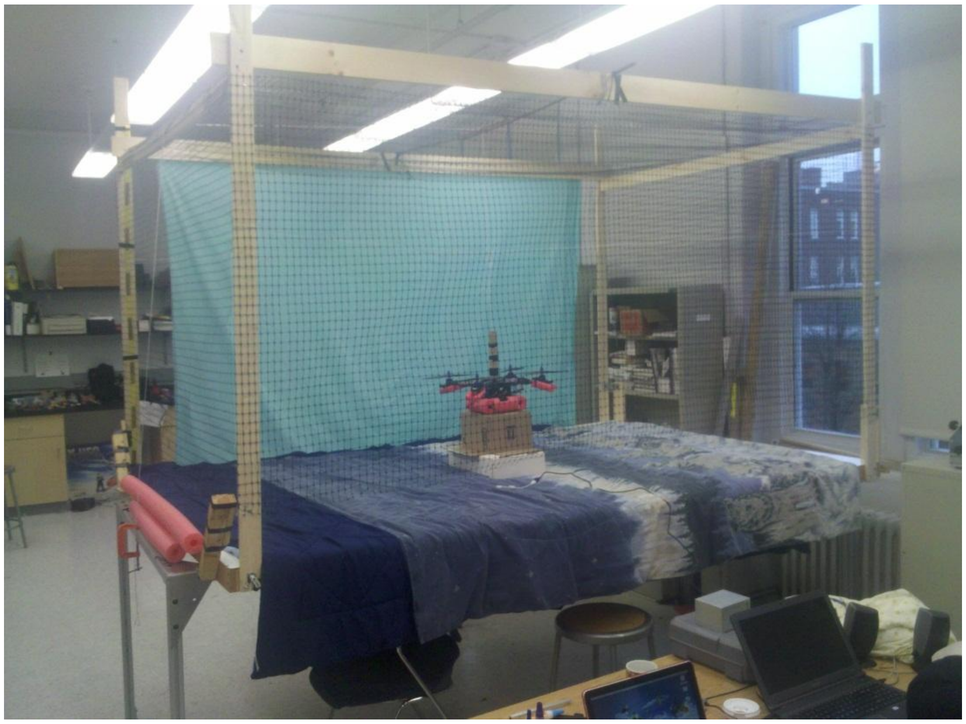


Figure 37: Testing Enclosure

In order to further ensure the main structure and components of the quad-rotor did not sustain damage during testing, the team cut pool noodles (polyethylene foam) in have and placed them around the base and bottom of each motor.

3.8 Design of Unmanned Ground Vehicle

The ground vehicle represents an important part of the overall MAV deployment system developed for this MQP. The UGV consists of several key parts each serving a specific function related to deployment, control, and storage of the MAV systems.

3.8.1 Overview of Pioneer 3DX

The base ground vehicle used in this project is the Pioneer 3DX manufactured by Adept Technology Inc. This is a 3-wheeled robotics platform with a built-in computer system and set of heavy-duty batteries used for powering the UGV as well as charging MAVs when in the field.

The pioneer is about 23cm tall, 38cm wide, and 45cm long. Its maximum forward speed is approximately 1.2 m/s. Unloaded, the pioneer has a weight of 9kg, and can hold up to 17kg (38 lbs) of payload. Each of the three onboard 12V batteries has a capacity of 7.2 Ah, which can power the pioneer (without accessories) for about 8 to 10 hours.

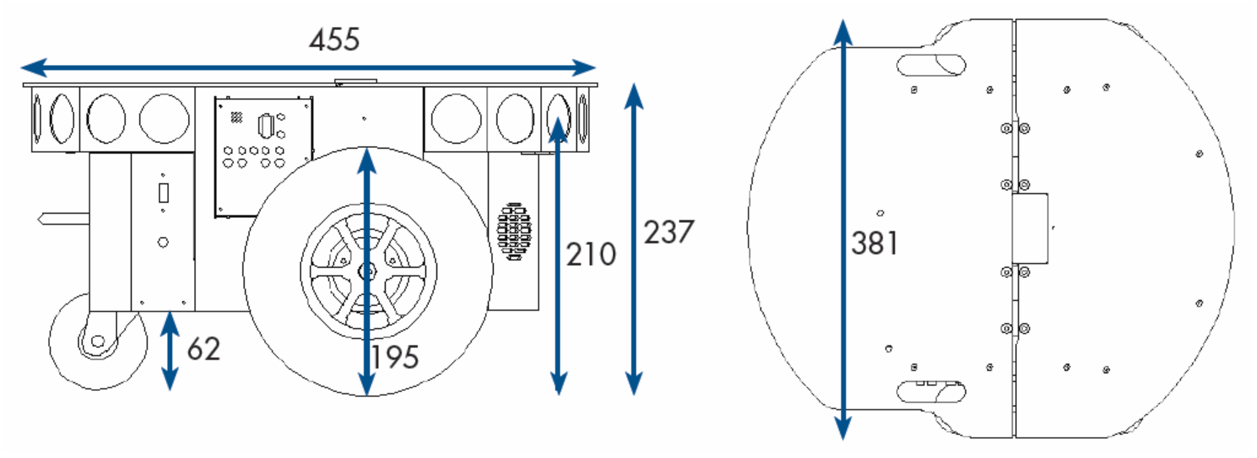


Figure 38: Pioneer Diagram

3.8.2 Deployment Mechanism Design

The Landing pad deployment system consists of a pair of kinematic mechanisms which provide the physical support structure for the MAVs during takeoff, landing, and storage. The purpose of the mechanism is to reduce the overall UGV footprint when the system is not in operation or while the UGV is transporting and/or charging the MAVs.

The first design iteration of the Landing pad deployment system consisted of 3 flat platforms on which the MAVs could land, each supported by 4 arms that could be lifted from a level position up to a stacked position. The resulting mechanism was therefore capable of providing both a level area where the MAVs could land and a stacked configuration for storing the MAVs in a vertical tower to reduce the UGV footprint.

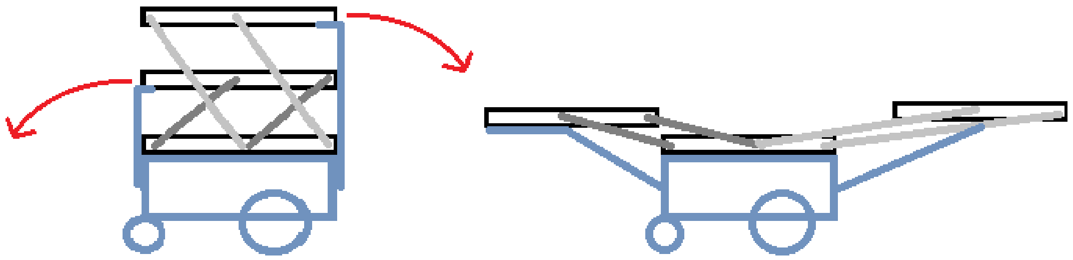


Figure 39: UGV Design 1

Two of the major alternative designs that were considered but rejected early on were a simple sheet platform for all 3 MAVs and a "shelf"-type platform deployment mechanism. The sheet platform design was rejected because of its large size and limited expandability in the case of additional MAVs. The second design consisted of 3 shelves which could be opened laterally. This design was both simple and reduced the UGV footprint, but was ultimately rejected because of the danger varying height platforms posed to the MAVs as they were landing or taking off.

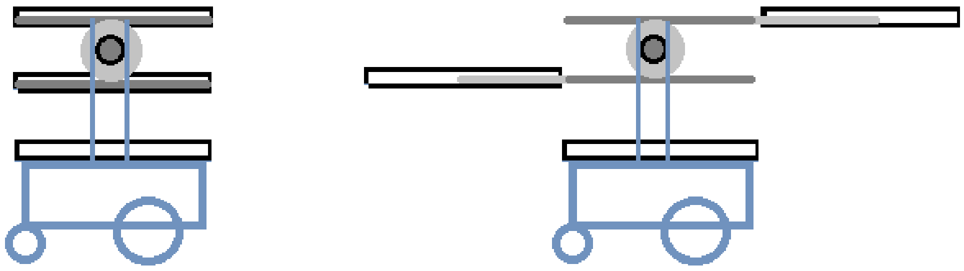


Figure 40: UGV Design 2

Upon analyzing the design of the first iteration of our deployment mechanism two problems became apparent that needed to be resolved before continuing with the design. The first was a simple balance problem relating to the UGVs overall center of gravity during deployment of the MAVs. To stack all of the landing platforms, one platform necessarily had to have longer support arms than the other. When deployed, this creates a difference in weight distribution which shifts the center of gravity towards the longer arm. This could potentially tip the UGV and cause damage to the MAVs or immobilize the UGV. Additionally, and regardless of if the UGV is stable, the extra length on the support arms will increase both the stress on those arms and the needed torque to rotate the mechanism. This is because the torque is related directly to the length of the arms, thus an arm that is twice as long will generate twice as much torque with the same force load.

$$\tau = r * F$$

To resolve both of these problems a new design was created in which both platforms were identical. In this third iteration of the UGV design, the two stacking platforms are each held at the same distance from either side of the pioneer when deployed. The one trade-off that was made to achieve this more stable configuration was growing the overall UGV footprint when the platforms are in the storage position. This is because the two platforms are now unable to stack on top of each other and instead are positioned side-by-side. This was determined to be an appropriate trade of as it made the task of manufacturing the mechanism much easier.

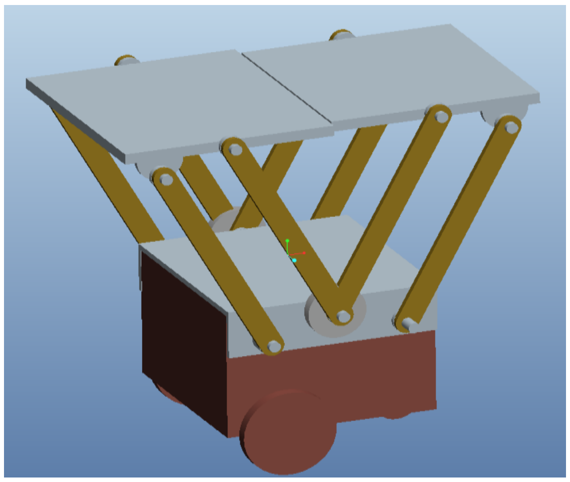


Figure 41: UGV Design 3 - Stored

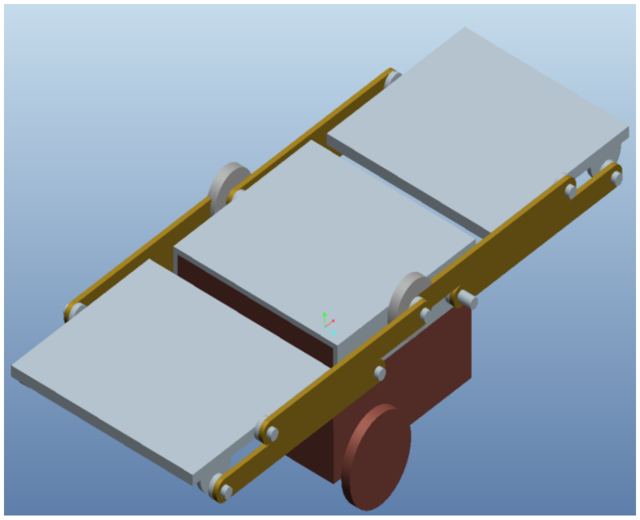


Figure 42: UGV Design 3 - Deployed

Once the general layout of the mechanism had been decided, the next steps were choosing dimensions and physical building materials. At this point in the design of the UGV, some basic dimensions for the MAV had already been chosen and could be used to calculate the needed amounts of each material to support the weight of the MAVs while not physically interfering with the MAV during the deployment motion.

The MAV parameters assumed during the 3nd design iteration of the UGV were as follows. The maximum length and width were each 20 inches, spanning from each end of the outer blade tips of the rotors. The height of the MAV was set at 5 inches and the weight was assumed to be 3lb with a safety factor of 2, meaning that the platforms would be designed to handle up to twice the weight of the assumed MAV. Using these parameters, a platform width of 24 inches was chosen to safely accommodate the MAV. Several materials were then examined as possible candidates for the platform material. Their properties, including modulus of elasticity, rigidity, and average density are listed in Table 6.

Table 6: Material Properties

Materials:	Elasticity (E)		Density (p)		Rigidity (R)	
Aluminum	10	10^6 psi	0.09754	lb/in^3	3.9	10^6 psi
Mild Steel	30	10^6 psi	0.2836	lb/in^3	11.5	10^6 psi
Acrylic	0.3771	10^6 psi	0.03974	lb/in^3		
Polystyrene (EPS)	0.0015	10^6 psi	0.00361	lb/in^3		
Pine Wood	1.3	10^6 psi	0.0177	lb/in^3		

To determine the thickness (h) and overall weight of the platforms for given materials an acceptable deflection was chosen from which the desired measurements could be calculated using beam deflection equations. For this particular case the platform is assumed to be a rigidly supported very wide beam with a point load in the center representing the weight of the MAV. The amount of deflection chosen for making the calculations was 1/10th of an inch. From these equations for moment of inertia (I) and deflection (D), an equation for thickness (h) can be derived. In the equations below, mg represents the weight of the MAVs, W and L represents the width and length of the platform, and E is the elasticity (young's modulus) of the chosen platform material, given in the preceding table.

$$I = \frac{Lh^3}{12} \qquad D = \frac{mgW^3}{48EI} \qquad h = \sqrt[3]{\frac{12mgW^3}{48LDE}}$$

Four materials (Aluminum, Steel, Styrofoam, and Acrylic) were chosen to be analyzed as platform materials. Wood was not chosen, as it has poor characteristics such as variable, and thus unreliable, elasticity and general tensile strength, and may expand or quickly erode in moist conditions or when exposed to water. The masses, volumes, and thicknesses of each material are tabulated below. Length and width are fixed at 8 in and 24 in, respectively.

Table 7: Platform Materials

Platform Materials	Thickness		Mass		Volume	
Aluminum	0.16384	in	3.068344	lb	31.45729	in^3
Steel	0.1136	in	6.185675	lb	21.81127	in^3
Acrylic	0.491053	in	3.746773	lb	94.28216	in^3
EPS	3.134147	in	2.17234	lb	601.7563	in^3

Looking at these results, the best two options for a platform material appeared to be aluminum and acrylic. The steel platform was the best in terms of size but was substantially heavier than the other materials. The Styrofoam platform suffered from the opposite and was too large despite its favorably low weight.

The next structural calculation to be made was determining the width of the support arms for each platform. This is similar to calculating the thickness of the platforms, but requires a slight modification of the deflection equation. This is because the arms are modeled as cantilevered beams while the platform has supports on either side. For these equations, "b" is the thickness of the arm, "Mg" is the weight of the MAV and platform, and all other variables have the same meaning but in relation to the arm instead of the platform.

$$I = \frac{bW^3}{12} \qquad D = \frac{MgL^3}{3EI} \qquad W = \sqrt[3]{\frac{12MgL^3}{3bDE}}$$

In choosing the arm material to support the platforms, Styrofoam was immediately ruled out as it was very likely an unreasonable size would be calculated to achieve the required strength. This left steel, aluminum, and acrylic to be analyzed using the preceding equations. Once again, a deflection of 1/10th was used, the center-center length of the arm was fixed at 22 inches, and an arbitrary arm thickness of 1/4 inch was selected.

Table 8: Arm Materials

AL Platform	Width		Mass		Volume	
Aluminum	1.31018	in	0.702872	lb	7.205991	in^3
Steel	0.908428	in	1.416966	lb	4.996355	in^3
Acrylic	3.926805	in	0.858282	lb	21.59742	in^3
Acrylic Platform	Width		Mass		Volume	
Aluminum	1.331906	in	0.714528	lb	7.325485	in^3
Steel	0.923492	in	1.440463	lb	5.079207	in^3
Acrylic	3.991921	in	0.872514	lb	21.95556	in^3

From the results given, aluminum and steel appeared to be the best materials to use for the support arms based on a combination of size and weight considerations. Assuming the heaviest materials are chosen (in this case, an acrylic platform with steel support arms) the total mass of the mechanism plus the weight of the MAVs comes out to approximately 35 lbs. This total mass is just below the payload capacity of the pioneer and is therefore fairly safe considering twice the expected weight of the MAVs was assumed during the calculations.

After looking over all of the calculations and giving consideration to how each of the components might be obtained or manufactured, the final design iteration was developed based on a few modifications to the previous design (iteration 3). The most substantial change was the reduction of the 8 arms holding each platform into 4 arms. This was accomplished by building the deployment mechanism in the center, below the platforms, rather than having redundant symmetrical systems on either side of each platform.

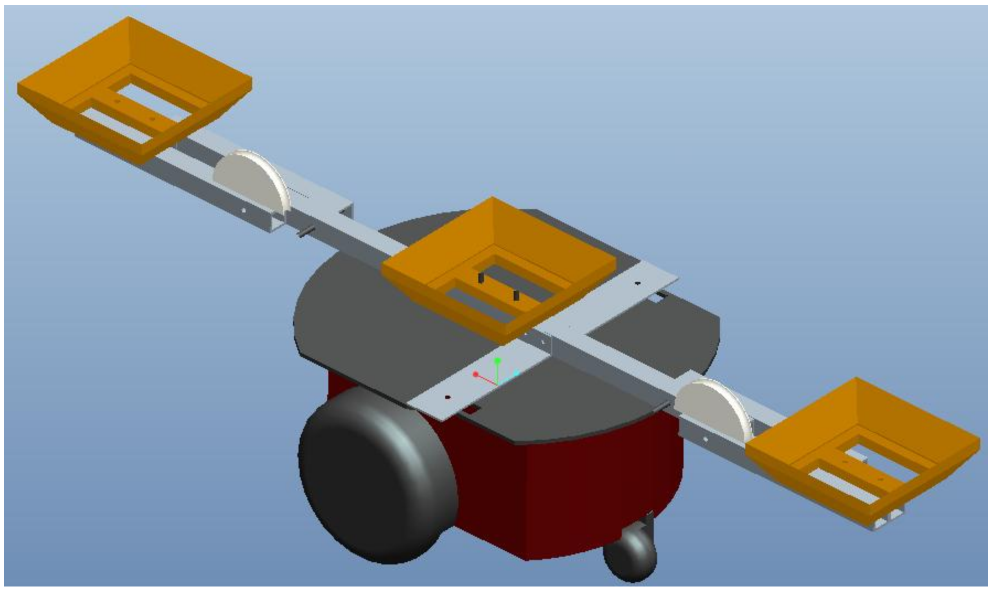


Figure 43: Final UGV Design (deployed)

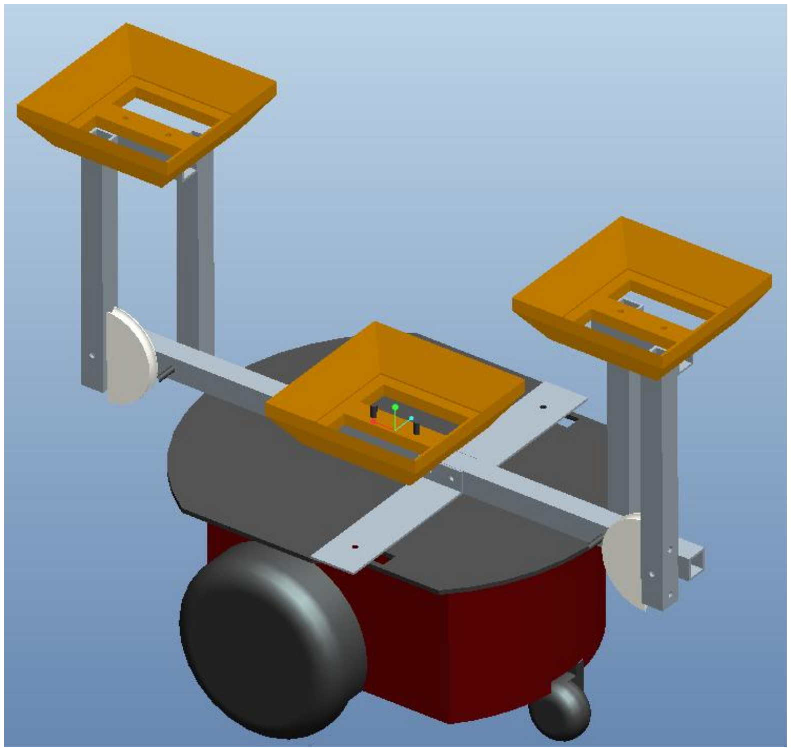


Figure 44: Final UGV Design (Stored)

3.8.3 Landing Pad Design

In addition to physically supporting the MAVs an important function of the landing pads is also the charging of the MAVs. With this in mind, as well as the ability to protect the rotor tips from the arms of the deployment mechanism, it is crucial that the MAVs are able to center themselves on the landing pad. This is necessary for charging because with conductive charging at least, the contacts on the MAV must light up with those of the charging mechanism on the landing pad to make a proper connection.

The two straightforward ways of achieving self-centering, or alignment in general, are active aerial maneuvering and passive geometric alignment. Active maneuvering is generally unfavorable because it requires very accurate real-time sensing on the MAV coupled with a series of controlled maneuvers which must be either pre-programmed or executed manually. The second and much more favorable option is geometric passive alignment. This is most easily accomplished by constructing a tapered inward landing platform and a complimentary tapered landing gear on the MAV.

One issue presented by a conical landing dock, our first design idea, is the fact that there is no rotational control during landing. This was resolved by instead using a pyramid shaped dock (4 sides) which like the cone directs the MAV to the center but also rotates the MAV so that it is aligned with the orientation of the landing pad. Once the MAV has been aligned it will come to a rest at exactly the same point every time, thus making conductive charging possible.

In the latest iterations of the landing pad designs holes were also built into the middle area of the pads to accommodate electrical wiring necessary for MAV charging contacts. Having these access holes as close to the center as possible ensures that the angled sides of the landing dock are unobstructed and therefore smooth enough to slide the MAV landing gear toward the center as they were designed to do rather than having the MAV get caught on a wire or wires going up and over edges.

3.8.4 MAV Charging System

To charge the MAVs, a reliable, efficient, and safe charging system needed to be developed and integrated into the ground vehicle platform. Each MAV was fitted with a single 3-cell lithium battery package which requires a specialized load balancing controller in addition to 12 volt DC lines.

After researching the benefits and drawbacks of inductive and conductive charging systems, it was determined that inductive charging would be both too complex and very inefficient in transferring electricity to the MAV. The chosen method of conductive charging presented a separate difficulty in that there were 6 contacts that needed to be aligned for the charging system to properly interface with the multi-cell battery. Our general approach to this part of the project was to arrange metal contacts along the base of the landing platform which would correspond to other contacts on the underside of the MAV landing gear.

The particular arrangement of the contacts needed to be such that there would be a safe clearance between each contact and also that they would be arranged in a pattern that would fit on the limited area available underneath the MAV. Although the bottom of the MAV is a square there is a hollow section in the center for the Ultrasonic sensor meaning that the contacts could only be placed around the edges of the base.

The next challenge that was foreseen in developing this charging contact layout was the fact that if the MAV were to land in the incorrect orientation it would not make contact and would therefore be unable to charge. Because of the square shape of the landing dock and tapered edges of the MAV landing gear, the numbers of orientations that can possibly occur during landing are substantially reduced. Practically speaking, there are only four possible orientations of the MAV and so there only needs to be accommodation for those four rotations of the charging contacts built into the system.

The simplest solution to this problem of having four possible orientations of the MAV was to layout the charging contacts in groupings of six along all four edges of the landing platform in a rotationally symmetrical fashion. The MAV would still only need the six contacts; however they could be arranged in a way that would maximize space between the contacts while still corresponding to the distance from any given corner equal to the same distance on one of the four sides of the charging platform. The charging platform, having twenty-four contacts, would be able to accommodate the six MAV contacts in any of the four orientations without the need of any software control or complex electronics. To increase contact area, as well as improve likelihood of solid contact, a redundant set of charging leads is positioned at 180 degrees from the first six leads, bringing the total on the MAV to twelve as seen in Figure 45: Charging Schematic.

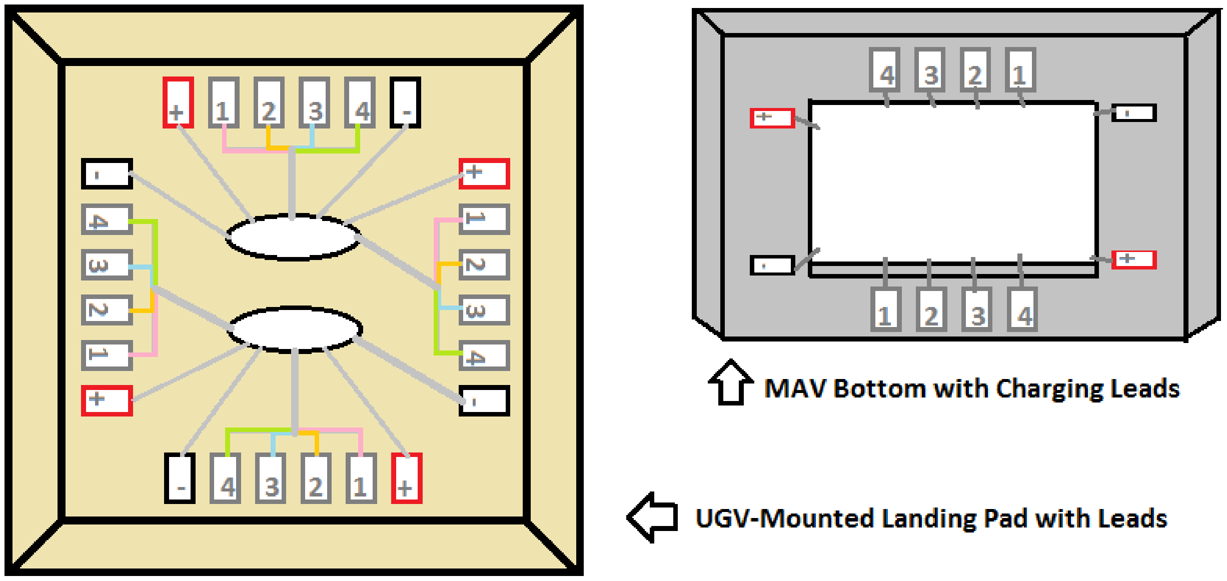

MAV Bottom with Charging Leads
UGV-Mounted Landing Pad with Leads
+
1
2
3
4
-

Figure 45: Charging Schematic

Once the MAV has successfully made contact with the charging platform, the physical charge controller is responsible for delivering a balanced load of power to the three cells of the MAV battery while drawing charge from one or more of the much larger pioneer batteries. The charge controller we have used for this project had two input sockets for drawing power at twelve volts from the twelve volt battery or other external power supply and then six outputs for evenly charging the battery cells of the MAV.

During testing of the MAV, this charge controller was powered directly from a power outlet and power supply system rather than through the batteries of the pioneer. This was done to simplify the testing process and to conserve the energy stored in the pioneer's batteries during lengthy testing procedures. In a real-world situation the charging system would be fully independent of external power and would be handled completely onboard the UGV and only need external power for charging the pioneer batteries at the end of a mission.

3.8.5 UGV Electronics and Software

Even though the UGV is an already existing robotic platform, more electronics were needed for the arms and software was written to control them. In order to control the arms there has to be a way for the system to know where the arms are and there has to be a motor to drive the arms. There were many different ways to solve the problem of how to know where the arms are but the solution used was to put limit switches on the UGV in the maximum up position for the arms and the maximum down position for the arms. Since the arms are connected to the same drive motor and both arms travel the same distance only one set of limit switches was used to detect the arms. As for the motor, it would have to be controlled and again there are a number of solutions. A VEX Victor 884 speed controller was used to control the speed of the drive motor for the arms. All together there needs to be a microcontroller that senses the system and reacts accordingly. Because the Pioneer was designed to work with a specific set of electronic peripherals it was chosen not to use the on board microprocessor to control the arm system on top of the Pioneer but instead another microcontroller would be used and would be sent commands from the Pioneer's on board computer. Since the Arduino family has been primarily used on the MAV it was decided to use an Arduino Mega 2560 as the microcontroller for the arm system. The limit switches were connected to two of the Arduino Mega 2560 digital I/O pins with the open switch being LOW and the closed switch being HIGH. This would tell the controller when the arms were up and down. The next step was to use a PWM port on the Arduino Mega 2560 to send a PWM signal to the Victor 884 speed controller to turn the drive motor. Power for the drive motor would be taken from the Pioneer and the Arduino would be powered and talked to over USB also connected to the Pioneer.

4 Final Results

Following prototyping, testing was performed on both the MAV and UGV and their ability to complete the prescribed tasks were evaluated. Tests were run in a number of different environments presenting various results.

4.1 Flight Simulation Results

The simulation resulted in a stable hover with a reasonable response time. Graphical results are shown in Appendix E: Results of MATLAB Simulation. When the pitch and roll positions were set at ten degrees it took about five seconds for the oscillations to calm to a steady location around zero degrees, and the angular velocities took even less time depending on the noise generated during that simulation. When yaw was set to ten degrees only two to three seconds later the system had turned the yaw to zero, and with minimal oscillation at all. The yaw angular velocity returns to zero after three or fewer seconds.

For altitude testing, the desired altitude was set to five meters and when the system started at zero meters it reached stable hover after four seconds. The velocity shown in the results for the Z position is noisy, but again that is due to the noise generated at that time. With this desired altitude set to the starting altitude, the lateral position was found to move about one meter in the X and Y directions when the pitch and roll were started at ten degrees each. The velocities for lateral movement settle out at five seconds.

4.2 Flight Testing Results

Flight control testing was performed both in the tethered testing enclosure as well as in a free open environment. In the testing enclosure, control gains were adjusted until flight control was perceived to be stable. The tuned quad-rotor was then tested in an open un-tethered environment to verify the controller. The controller was able to maintain stable flight when wind was not a factor. The main control systems for the MAV are dependent on a P-D-DD (Proportional, Derivative, Double Derivate) controller, where behavior often starts with oscillation that slowly decreases to a steady state.

4.3 Testing Challenges

Performing flight testing on the MAV presented many challenges. Among those were complications with motors. The originally chosen Alpha 370 1200kv brushless motors, which had been used by MQP groups in previous year, were seemingly discontinued by manufacturers. Due to a crash during flight testing, several of the Alpha 370 motors became damaged and needed to be replaced. With no replacements, additional motors were needed. As previously mentioned, Optima 370 brushless motors were chosen to replace the Alpha 370 motors.

The fragility of the sensor packages made it necessary to replace certain sensors after periods of use. The hall effects sensors, which are used to determine a motors current RPM, required repeated maintenance. The fragile nature of these sensors meant that any type of collision the MAV experienced during flight testing had the potential to knock a sensor out of alignment. While relatively simple to maintain, several Hall Effect sensors required repairs, causing long delays between tests.

4.4 Manufacturing, Assembly, and Testing of the UGV

The first step in building the UGV systems was cutting the aluminum pieces that would make up the central components of the UGV structures and mechanisms. The materials used in this process were about 10 feet of aluminum tubing, and about 1 square foot of 1/32 inch aluminum sheet. Specifically, the aluminum parts included the central support beam, the lever arms for each of the two pad deployment mechanisms, stabilizers to connect the central beam to the pioneer, and a motor mount.

These pieces were cut in the machine shop using a belt saw and had appropriate fastener holes drilled into them using a drill press.

Once the aluminum components were manufactured, a series of pulleys and spacers were needed to transfer force from the motor to the deployment mechanism arms. Acrylic was chosen as the most cost effective material and could be easily cut into the appropriate shapes on a laser cutter. Most of the pulleys consisted of 3 or 4 glued together layers of 1/8 inch acrylic comprising the two larger guide circles on the outside and smaller track circle(s) on the inside. These pulleys were fastened to the aluminum components using ¼ inch bolts at pre-determined locations that had been drilled when the aluminum components were made. In total there are 7 major pulleys consisting of 3 unique shapes. Four of the pulleys were simple circular pulleys about 1.5 inches in diameter used for guiding the cable. One drive-spool pulley was made with a non-circular axel hole to be mounted on the drive motor and also had secondary holes for securing the wire. The 2 remaining pulleys were semi-circular and direct the cable around the pivot point of the pull arm at a radial distance of 2.5 inches to create a larger moment on the pull arms, thus requiring less motor torque.

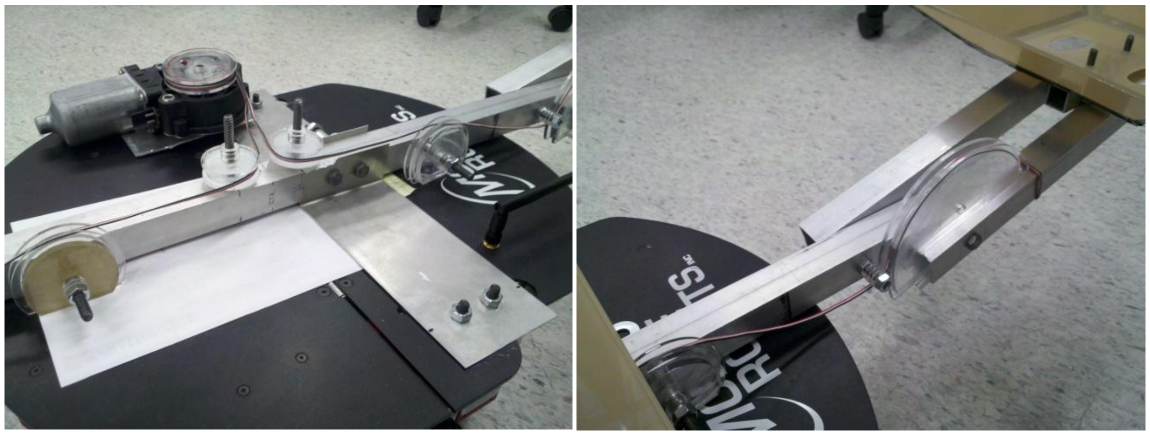


Figure 46: Pulley and Arm System

For the landing platforms, material selection was based on earlier calculations that showed acrylic to be the best material both in weight and ease of manufacturability. To manufacture the platforms acrylic sheets were laser cut into pieces that could then be assembled into the three dimensional platforms. Semi-flexible silicon adhesive was used to glue the pieced together. This particular adhesive was selected as the bonding with acrylic is strong enough to withstand the pressure that a landing MAV would exert, while the flexibility gives the platforms more impact resistance by way of elastically giving instead of shattering or cracking the glue.

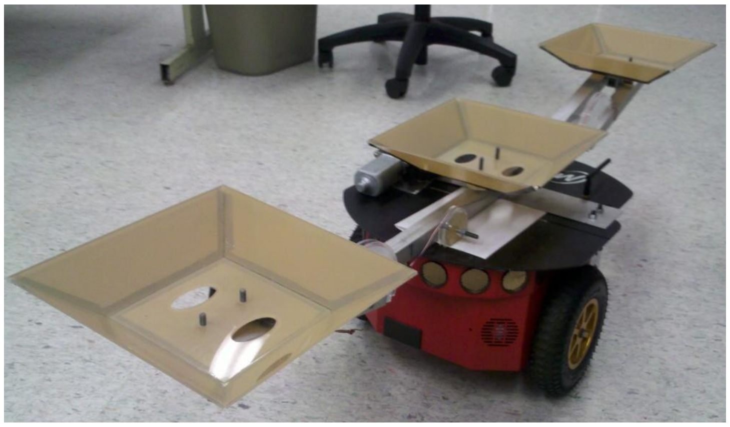


Figure 47: UGV with landing platforms in the deployed position

The electrical components of the UGV were mounted on a series of aluminum sheets or spacers which were bolted to the main cross beam and plate mounting brackets. These electronics include the drive motor for the two lift pulley systems, a Vex speed controller for powering the drive motor, and charge controller for distributing energy to the MAV charging docks at the end of each arm and in the center. Because the landing pads are all raised above the deployment mechanisms, the topside of the pioneer robot was largely open and provided an ideal location for mounting all of the necessary electronic systems that are not already built into the inside of the Pioneer body.

4.4.1 UGV Electronics and Software

The only changed that occurred with the electrical and software system was that the drive motor and Victor 884 speed controller were powered by a spare battery instead of the Pioneer because of connection problems with the Pioneer. While testing the system the speed at which the drive motor was run was increased when bringing the arms up because when tested the arms would struggled under the load but increasing the speed was a solution. The arms performed as expected under a load of 4 lbs.

on each arm, this was roughly estimated to be 16 ft-lbs of torque needed by the drive motor to lift the arms.

Communication protocol for the UGV system was to connect over AD-Hoc Wi-Fi set-up by the Pioneer via an SSH client and control the movement of the Pioneer. Once connect to the Pioneer, a serial connection was made from the Pioneer to the Arduino Mega 2560 and simple commands were transmitted to raise the arms and lower them (“u” for up, “d” for down). The movement of the Pioneer was controlled through the ARIA program loaded onto the platform by the manufacturer. ARIA used the arrow keys to turn and increase speed in the forward and reverse directions to a capped limit as to not have the Pioneer drive away unsecured. Overall the system performed as expected.

4.4.2 UGV Testing and Final Modifications

Once the electronics were working, the first series of tests we performed on the UGV were focused on arm deployment. In the very first test we performed, the speed of the deployment system drive motor was much too slow. In response, we increased what the speed controller was giving the motor and ran more tests, this time with gradually increased payloads placed on one or both of the arms. When the payload approached 3 pounds, we noticed considerable strain in the pulley wires, which eventually led to them breaking.

To alleviate the problem of drive cables breaking, two solutions were explored. The first solution was to add a counterweight to the deployment arms which would serve to dramatically reduce the loading on the cable by counter-acting the moment arm created by the payload. To attach the counterweight, a hook was mounted to the arm on the opposite side of the pivot point from where the payload was. A weight of between 2 or 3 pounds at 3 inches from the center of rotation seemed to work fairly well however it added that much more weight to the pioneer. The second solution was to increase the tensile strength of the cable. To increase the cable’s strength we replaced the copper wire that we had been using with special high-tension string. This solution was ultimately the one we chose to use in the final iteration of the UGV.

Another major issue that became apparent during deployment testing was the fact that some parts on the UGV were bending under the forces exerted by the tension cable on the pulleys while the UGV was trying to lift a payload. Specifically, the center beam on which most of the mechanism was built began to tilt towards the motor and vice-versa as the motor pulled the cable. To resolve this bending, C-clamps were used to hold down the center beam and drive motor plate and keep them level with the pioneer. Eventually we replaced these clamps with smaller, more permanent fixtures that accomplished the same clamping task.

After all of the final components were mounted onto the pioneer, it was taken into the field and tested under real world conditions including on pavement, dirt, grass, and uneven surfaces. While the top speed was not very fast, the handling of the pioneer on inclines up to around 20 degrees and traveling over rough terrain was very good. It never tipped or caused any damage to the payload during this field test and seemed to have no problems with holding the payload during motion. Other things tested during driving of the UGV were the opening and closing of the arms. This was successful as well and demonstrated that the UGV could deploy while still in motion.

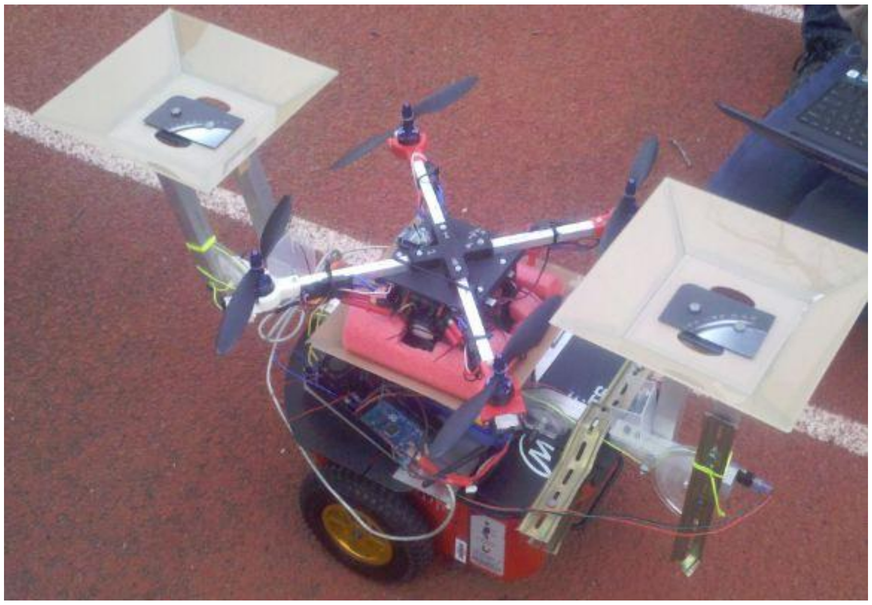


Figure 48: MAV-UGV System

Some of the problems encountered during field testing of the UGV were either related to electrical components not being fully secured or connection problems via wifi. It was discovered that the battery connectors especially were prone to disconnecting under the vibrations experienced by the UGV during movement. This was solved by strengthening the connector linking the control board and the battery cable. The wireless connection problems could have been due to several factors, but were of particular impediment by making it difficult to quickly and reliably connect to the pioneer to engage drive or deployment functions.

5 Reflections of Project

At the start of the project, the team was confident in its ability to meet its goal of having three MAVs and one UGV working in tandem by the end of three terms. With a team of eight members, there are inherent challenges that need to be overcome, such as organization, group dynamics, and productivity. Throughout the project, the team had an easier time with some aspects over others. Arranging weekly meetings proved to be unproblematic as each member's schedule fit well with the rest and everyone was willing to alter their schedule if needed. During the initial stages of the project, the team met almost every day to discuss and plan what needed to get done. As the project progressed, the team continued to meet many times a week in the lab to continue constructing and then later testing. Group dynamics and staying motivated was also not a problem as all the members got along and were fully committed to the project. In retrospect, the team felt that the main problems that were the cause of many setbacks were productivity, monetary and time restrictions, and project goals.

During the first term, the team researched, proposed, and selected the most appropriate MAV design for the project. Once completed, the quad-rotor went through three design iterations and a few material ordering setbacks before the construction actually began a few weeks of the second term. From this, the team learned that the time spent researching various MAV designs was inefficient and the majority could have been better suited to focusing on the quad-rotor and possibly designs used in previous projects, therefore beginning the initial design and construction sooner in A-term. In addition, during the initial research stage, with a team of eight, it could be divided into subgroups with each focusing on a specific task such as optimizing structural design for the MAV and UGV, establishing control equations, or running Matlab simulations. Therefore, different aspects of the project could be worked on initially thereby maintaining productivity as the project continues. A difficulty the team faced involved not setting specific milestones aside from major objectives. In the long run, this caused productivity to slow as members were unclear on what to do next.

Lastly, the team felt the project and project goals needed to be reevaluated. Looking back, the goal of building three autonomous MAVs and one UGV to work in tandem was unrealistic for the time frame and funding given. Therefore, the team suggests that the project become multi-year and possibly span both the Aerospace and Robotics Engineering departments. The benefit of doing so would allow aerospace students to design and analyze certain aspects of the quad-rotor and which would then be passed to the robotics engineering students who would focus on programming and implementing the autonomous system. Then, the team the following year would further refine and optimize the system from the previous. This proposal by the team was in light of the fact that it felt there was not enough work that the aerospace students could help with as they did not have a strong theoretical or practical controls engineering background.

References

[1] Castillo, Pedro, R. Lozano, and Alejandro E. Dzul. *Modelling and Control of Mini-flying Machines*. London: Springer, 2005. Print.

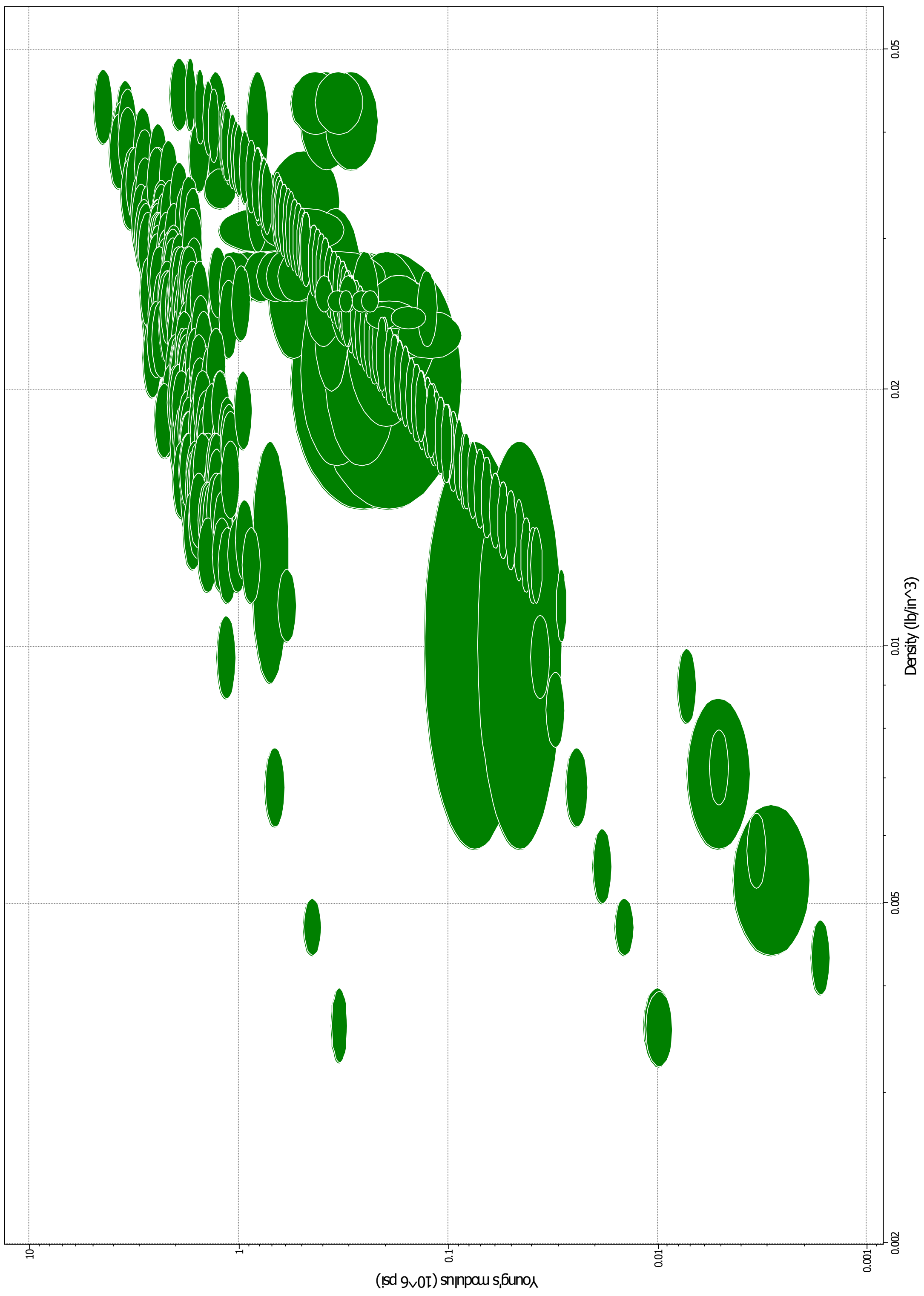
[2] Ehrhard, Thomas P. *Air Force UAVs: The Secret History.* Ft. Belvoir: Defense Technical Information Center, 2010. Print.

[3] Gage, Douglas W. *UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts*. San Diego: Naval Ocean Systems Center, 1995. Print.

[4] Sullivan, J.M. "Evolution or Revolution? the Rise of UAVs." *IEEE Technology and Society Magazine* 25.3 (2006): 43-49. Print.

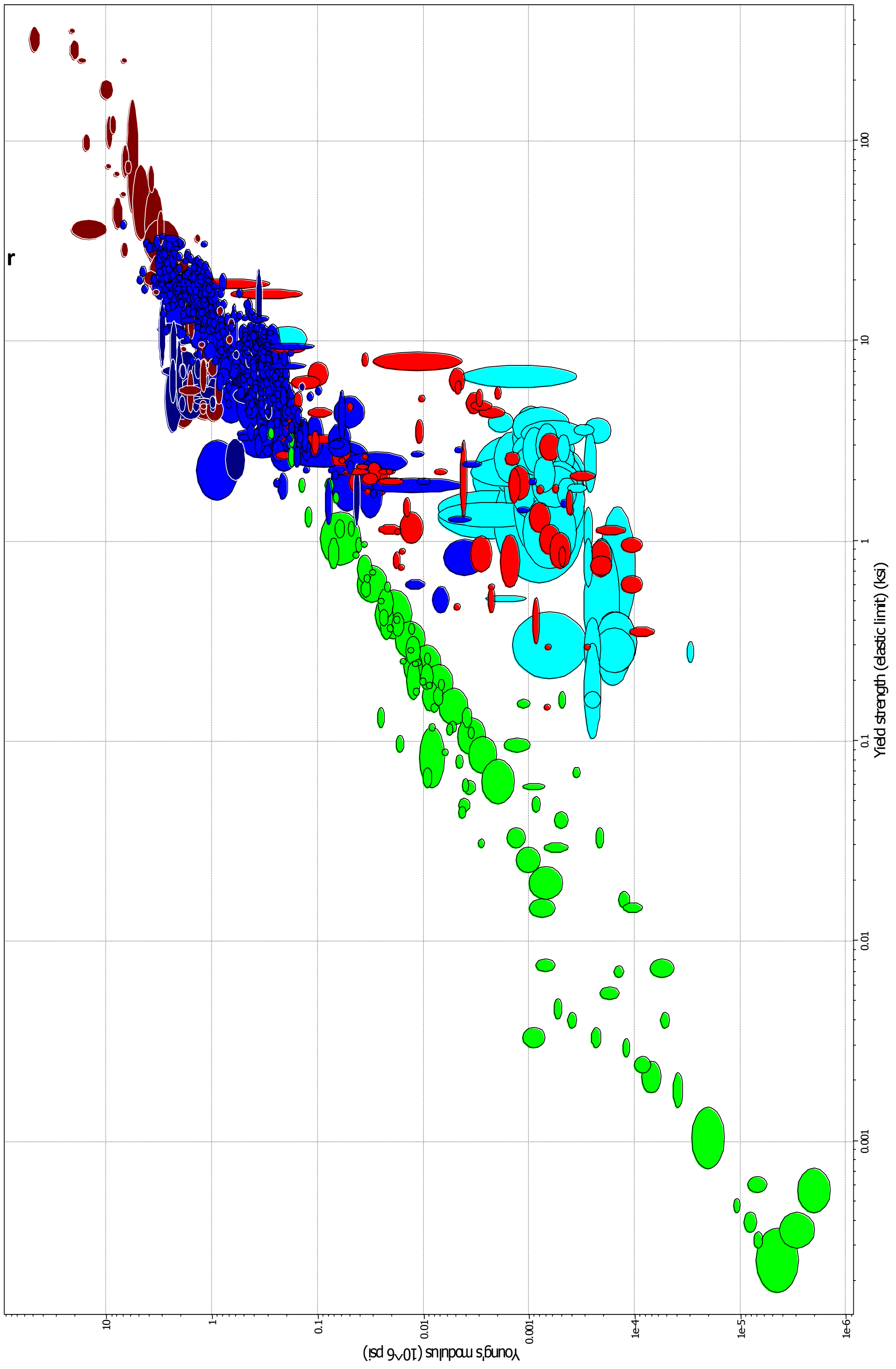
Appendix A: GRANTA Analysis of Current RC Aero Vehicles

Graph of Young's Modulus vs. Density of Woods

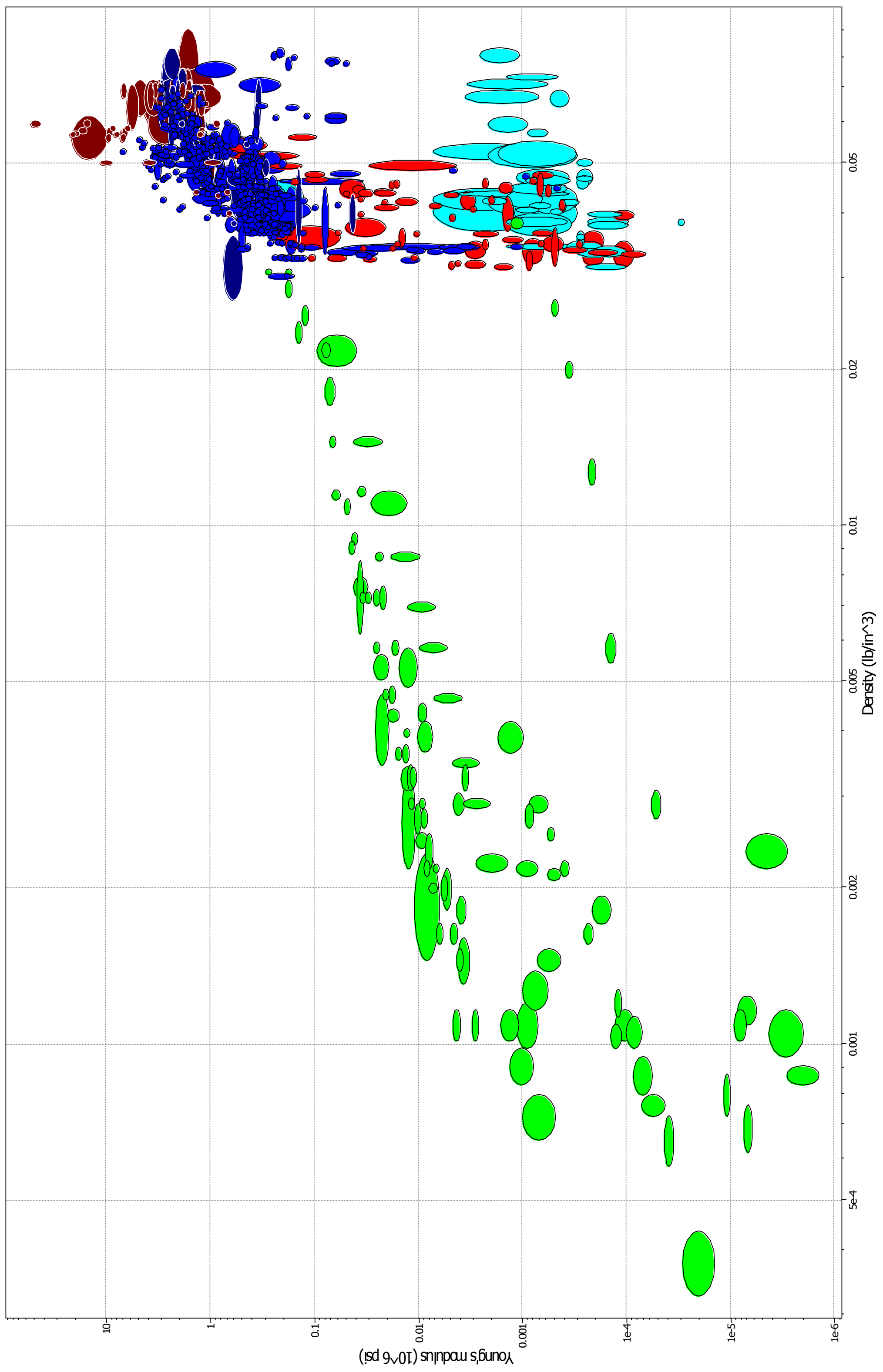


Graph of Young's Modulus vs. Yield Strength of Polymers

r



Graph of Young's Modulus vs. Density of Polymers



Appendix B: Equation of Motion Derivations

Altitude

The expression for the control the altitude of the UAV (the generalized translational coordinate z for the vertical position of the quad rotor) is as follows:

$$u = (r_1 + mg)\frac{1}{\cos\theta\cos\varphi}$$

The only unknown variable within this expression is r_1:

$$r_1 = -a_{z_1}\dot{z} - a_{z_2}(z - z_d)$$

In this expression, $a_{z_1}$and $a_{z_2}$are positive constants and z_d is the desired altitude (the desired position on the z-axis).

Altogether, the expression for the control of the altitude of the quad rotor is:

$$u = \left[\left(a_{z_1}\dot{z} - a_{z_2}(z - z_d)\right) + mg\right]\left(\frac{1}{\cos\theta\cos\varphi}\right) = f_1 + f_2 + f_3 + f_4$$

Yaw Angular Position

The expression for the control of the yaw angular position of the UAV (the generalized rotational coordinate ψ about the z axis) is as follows:

$$\tilde{\tau}_\psi = -a_{\psi_1}\dot{\psi} - a_{\psi_2}(\psi - \psi_d)$$

Within this expression, a_{ψ_1} and a_{ψ_2} are positive constants and ψ_d is the desired yaw angle (about the z axis).

Also, as long as $(\cos\theta\cos\varphi) \neq 0$, then it is possible to write:

$$m\ddot{x} = -(r_1 + mg)\frac{\tan\theta}{\cos\varphi}$$

$$m\ddot{y} = (r_1 + mg)\tan\varphi$$

$$\ddot{z} = \frac{1}{m}(-a_{z_1}\dot{z} - a_{z_2}(z - z_d))$$

$$\ddot{\psi} = -a_{\psi_1}\dot{\psi} - a_{\psi_2}(\psi - \psi_d)$$

a_{z_1} , a_{z_2}, a_{ψ_1}, and a_{ψ_2} are control inputs that must be carefully considered in order to have a perfectly damped and stable system.

Roll Angular Position

The expression for the control of roll in a quad rotor (the control of the generalized rotational coordinate angle ϕ about the horizontal y-axis), is as follows:

$$\tilde{\tau}_\varphi = -\sigma_{\varphi_1}(\dot{\varphi} + \sigma_{\varphi_2}\left(\varphi + \dot{\varphi} + \sigma_{\varphi_3}\left(2\varphi + \dot{\varphi} + \frac{\dot{y}}{g} + \sigma_{\varphi_4}\left(\dot{\varphi} + 3\varphi + 3\frac{\dot{y}}{g} + \frac{y}{g}\right)\right)\right))$$

The only unknown variable within this expression is σ_{φ_i} $(i = 1, 2, 3, 4)$. σ_{φ_i} is defined as the saturation function used to derive this expression. It accounts for each of the motors, M_i, (again, $i = 1, 2, 3, 4$).

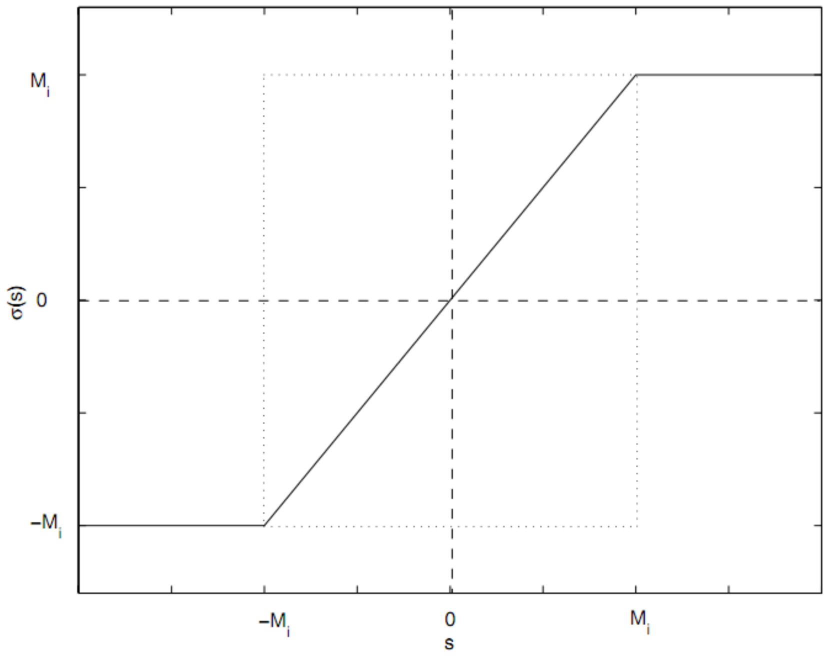


Figure 49: Saturation Function

Pitch Angular Position

The expression for the control of pitch of the quad rotor (the generalized rotational coordinate angle θ about the horizontal x axis) is very similar to that of the control of the roll angular position:

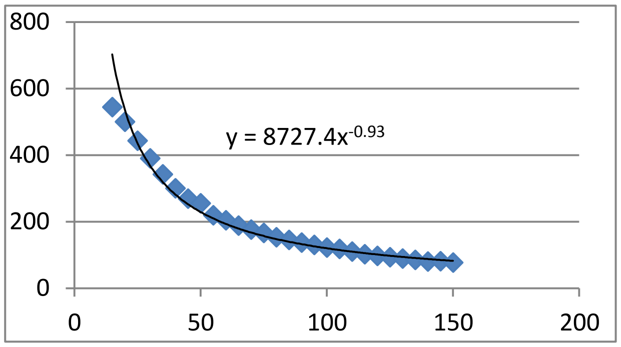
$$\tilde{\tau}_{\theta} = -\sigma_{\theta_1}(\dot{\theta} + \sigma_{\theta_2}\left(\theta + \dot{\theta} + \sigma_{\theta_3}\left(2\theta + \dot{\theta} + \frac{\dot{x}}{g} + \sigma_{\theta_4}\left(\dot{\theta} + 3\theta + 3\frac{\dot{x}}{g} + \frac{x}{g}\right)\right)\right))$$

This is found in the same way as the expression for the control of roll angular position. Both derivations utilize the saturation function, and here it is represented by σ_{θ_i}.

Appendix C: Electronic Information

Infrared Sensor Data

Distance	Old IR Sensor
15	544
20	500
25	443
30	390
35	342
40	300
45	269
50	255
55	219
60	204
65	188
70	176
75	166
80	153
85	145
90	137
95	130
100	122
105	118
110	110
115	102
120	97
125	93
130	89
135	85
140	80
145	81
150	77



Appendix D: MATLAB Simulation Code

quad_flight_sim.m

```
%Multi-MAV MQP Team
%2011-2012
%Quadrotor flight simulation

close all;
clear all;
clc;

disp('Running Quadrotor Simulation....')

global t1 t2 noise1 noise2 filter1 u m sat sat1 g Zd PSId PHId THETAd xDOTd yDOTd aX1 aX2 aY1 aY2 aZ1
aZ2 aPSI1 aPSI2 aPSI3 aPHI1 aPHI2 aPHI3 aTHETA1 aTHETA2 aTHETA3 test test2 incr end_step old_PHI
old_THETA old_PSI old_t;
rng(10);

% Gains and constants for simulation

m = 1.6; %kg
g = 9.81; %m/s^2
sat = 140; %DEG/s^2
sat1 = 1;
Zd = 0; %desired altitude
PSId = 0; %DEG
PHId = 0; %DEG
THETAd = 0; %DEG
aX1 = 2.95;
aX2 = 5;
xDOTd = 0; %m/s (always 0)
aY1 = 2.5;
aY2 = 2.5;
yDOTd = 0; %m/s (always 0)
aZ1 = 3.25;
aZ2 = 2.5;
aPSI1 = 3;
aPSI2 = 5;
aPSI3 = 0;
aPHI1 = 3;
aPHI2 = 5;
aPHI3 = 0;
aTHETA1 = 3;
aTHETA2 = 5;
aTHETA3 = 0;
```

```
incr = [1];
end_step = 6000000; %duration of simulation
filter1 = .8; %must be 0<filter1<1
old_PHI = 0; %always 0
old_THETA = 0; %always 0
old_PSI = 0; %always 0
old_t = 0; %always 0
noise = 0.15; %magnitude of noise in signal

 %time step for simulation
dt = .01; ti = 0; tf = 15;
t = ti:dt:tf;

%noise generation for the simulation. to remove noise, set noise = 0
Dnoise = noise/dt;
sigmaNoise = sqrt(Dnoise);
noiseDamping = sigmaNoise * randn(1,length(t));

 %initial coniditons for simulation
%X = [x y z phi theta psi x_dot y_dot z_dot phi_dot theta_dot psi_dot phi_double_dot theta_double_dot
psi_double_dot]

p0 = [0;0;0; 10;  10;   0;   0;    0;    0;     0;      0;       0;       0;          0;               0];
psol = zeros(length(t),length(p0));
psol(1,:) = p0;
nip = 2;

%runs the simulation
for i = 1:(length(t)-1)
   t1 = t(i); t2 = t(i+1);
   tspan = t1:(t2-t1)/nip:t2;
   noise1 = noiseDamping(i);
   noise2 = noiseDamping(i+1);
   [time,p] = ode45('quadrotor_controls',tspan,p0);

   p0 = p(end,:);
   psol(i+1,:) = p0';

end

disp('Simulation Complelte.')


%store data from simulation for graphing
x = psol(:,1);
Xdot = psol(:,7);
y = psol(:,2);
Ydot = psol(:,8);
```

```
z = psol(:,3);
Zdot = psol(:,9);
PHI = psol(:,4);
PHIdot = psol(:,10);
THETA = psol(:,5);
THETAdot = psol(:,11);
PSI = psol(:,6);
PSIdot = psol(:,12);


 %graph each response from the simulation
figure(1);
subplot(2,1,1),plot(t,THETA);
xlabel ('Time(s)');
ylabel ('THETA (DEG)');

figure(1);
subplot(2,1,2),plot(t,THETAdot);
xlabel ('Time(s)');
ylabel ('THETA Angular Velocity (DEG/s)');

figure(2);
subplot(2,1,1),plot(t,PHI);
xlabel ('Time(s)');
ylabel ('PHI (DEG)');

figure (2);
subplot(2,1,2),plot(t,PHIdot);
xlabel ('Time (s)');
ylabel ('PHI Angular Velocity (DEG/s)');

figure(3);
subplot(2,1,1),plot(t,PSI);
xlabel ('Time (s)');
ylabel ('PSI (DEG)');

figure(3);
subplot(2,1,2),plot(t,PSIdot);
xlabel ('Time (s)');
ylabel ('PSI Angular Velocity (DEG/s)');

figure(4);
subplot(2,1,1),plot(t,z);
xlabel ('Time(s)');
ylabel ('Altitude (m)');

figure(4);
subplot(2,1,2),plot(t,Zdot);
```

```
xlabel ('Time(s)');
ylabel ('Z Velocity (m/s)');

figure(5);
subplot(2,1,1),plot(t,x);
xlabel ('Time(s)');
ylabel ('X-Axis Location (m)');

figure(5);
subplot(2,1,2),plot(t,Xdot);
xlabel ('Time(s)');
ylabel ('X Velocity (m/s)');

figure(6);
subplot(2,1,1),plot(t,y);
xlabel ('Time(s)');
ylabel ('Y-Axis Location (m)');

figure(6);
subplot(2,1,2),plot(t,Ydot);
xlabel('Time(s)');
ylabel('Y Velocity (m/s)');

for i=1:length(test);
   r(i,1) = i;
end

figure(7);
plot(i,test);
xlabel('THETA acceleration (deg/s^2)');
ylabel('increment');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

quadrotor_controls.m

```
function pdot = quadrotor_controls(t,p)

  global noise1 noise2 t1 t2 filter1 u m sat sat1 g Zd PSId PHId THETAd xDOTd yDOTd aX1 aX2 aY1 aY2
aZ1 aZ2 aPSI1 aPSI2 aPSI3 aPHI1 aPHI2 aPHI3 aTHETA1 aTHETA2 aTHETA3 test test2 incr old_PHI
old_THETA old_PSI old_t;

   %initial conditions, received from main sim program
   %X = [x y z phi theta psi x_dot y_dot z_dot phi_dot theta_dot psi_dot]

   u = (-aZ1*p(9)-aZ2*(p(3)-Zd)+m*g)/(cosd(p(4))*cosd(p(5)));

  %update theta and phi desired based on previous sim results
   THETAd = asind(((aX1*(-g*(tand(p(5))/cosd(p(4)))))+(aX2*(p(7)-xDOTd)))/u);
```

```
    PHId = asind(sat_func((-aY1*(g*tand(p(4)))-aY2*(p(8)-
yDOTd))/(u*cosd(sat_func(THETAd,sat1))),sat1));

   %generate noise for sim
    f = noise1+(noise2-noise1)/(t2-t1)*(t-t1);

   %apply noise to signals received from initial conditions
    smooth_x = ((p(1)+f)*(1-filter1))+(p(1)*filter1);
    smooth_Xdot = ((p(7)+f)*(1-filter1))+(p(7)*filter1);
    smooth_y = ((p(2)+f)*(1-filter1))+(p(2)*filter1);
    smooth_Ydot = ((p(8)+f)*(1-filter1))+(p(8)*filter1);
    smooth_z = ((p(3)+f)*(1-filter1))+(p(3)*filter1);
    smooth_Zdot = ((p(9)+(f))*(1-filter1))+(p(9)*filter1);
    smooth_PHI = ((p(4)+f)*(1-filter1))+(p(4)*filter1);
    smooth_PHIdot = ((p(10)+f)*(1-filter1))+(p(10)*filter1);
    smooth_THETA = ((p(5)+f)*(1-filter1))+(p(5)*filter1);
    smooth_THETAdot = ((p(11)+f)*(1-filter1))+(p(11)*filter1);
    smooth_PSI = ((p(6)+f)*(1-filter1))+(p(6)*filter1);
    smooth_PSIdot = ((p(12)+f)*(1-filter1))+(p(12)*filter1);

    %store functions to be integrated
    pdot(1) = smooth_Xdot;
    pdot(2) = smooth_Ydot;
    pdot(3) = smooth_Zdot;
    pdot(4) = smooth_PHIdot;
    pdot(5) = smooth_THETAdot;
    pdot(6) = smooth_PSIdot;
    pdot(7) = -g*(tand(smooth_THETA)/cosd(smooth_PHI));
    pdot(8) = g*tand(smooth_PHI);
    pdot(9) = (1/m)*((-aZ1)*(smooth_Zdot)-(aZ2*((smooth_z)-Zd)));
    pdot(10) = (sat_func((-aPHI1*(smooth_PHIdot))-(sat_func((aPHI2*((smooth_PHI)-PHId))-
(sat_func((aPHI3*p(13)),.25*sat)),.5*sat)),sat));
    pdot(11) = (sat_func((-aTHETA1*(smooth_THETAdot))-(sat_func((aTHETA2*((smooth_THETA)-
THETAd))-(sat_func((aTHETA3*p(14)),.25*sat)),.5*sat)),sat));
    pdot(12) = (sat_func(((-aPSI1)*(smooth_PSIdot))-(sat_func((aPSI2*((smooth_PSI)-PSId))-
(sat_func((aPSI3*p(15)),.25*sat)),.5*sat)),sat));
    pdot(13) = 0;
    pdot(14) = 0;
    pdot(15) = 0;

    pdot = pdot';

   %store data for acceleration plots
    test(incr,1) = pdot(10) + f;
    test2(incr,1) = pdot(11) + f;
    incr = incr +1;

    end
```

Appendix E: Results of MATLAB Simulation

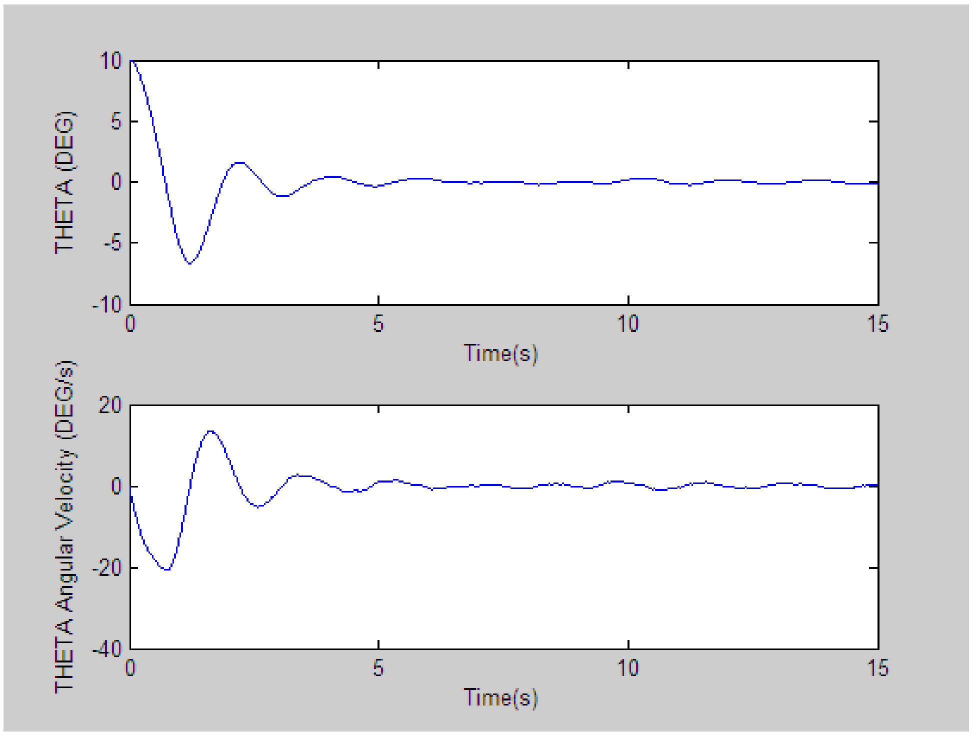


Figure 50: Pitch Control

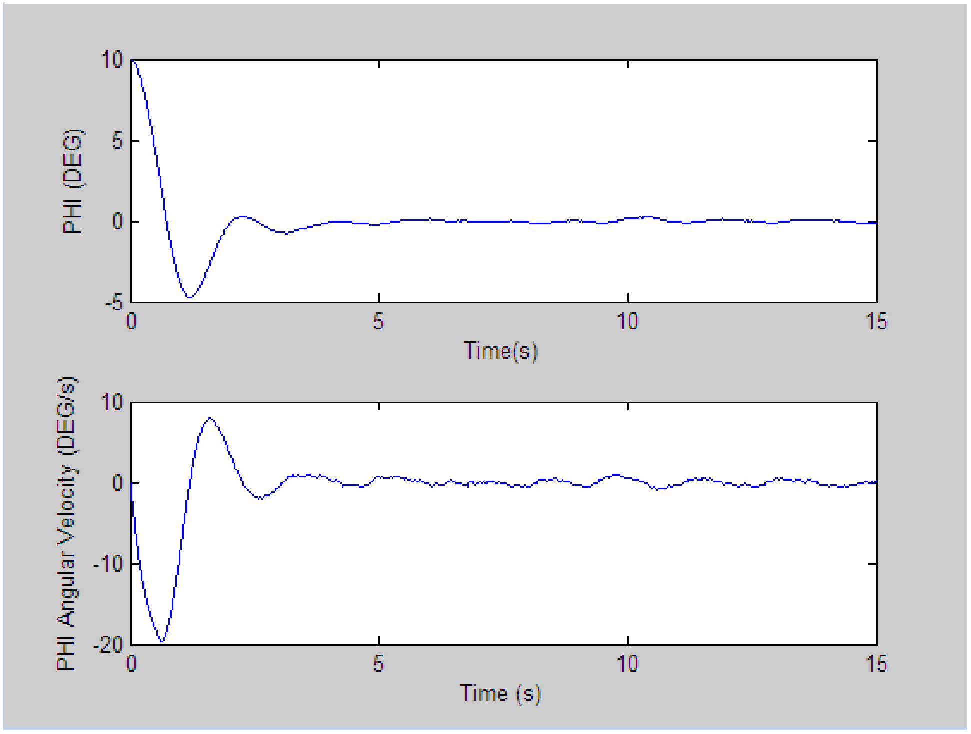


Figure 51: Roll Control

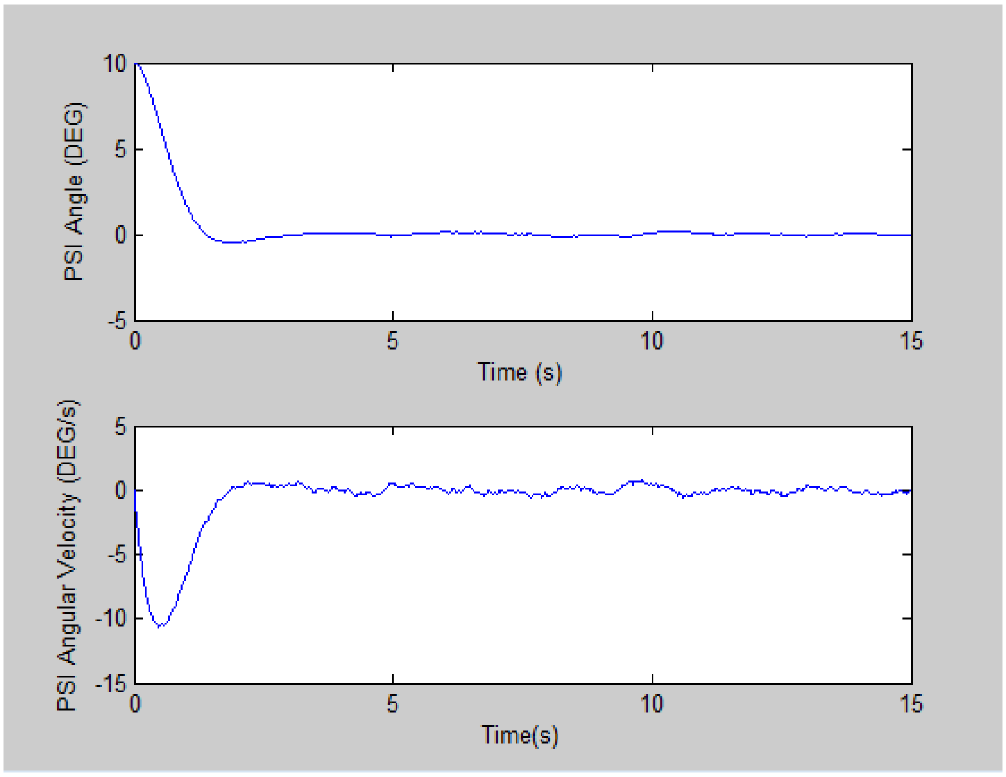


Figure 52: Yaw Control

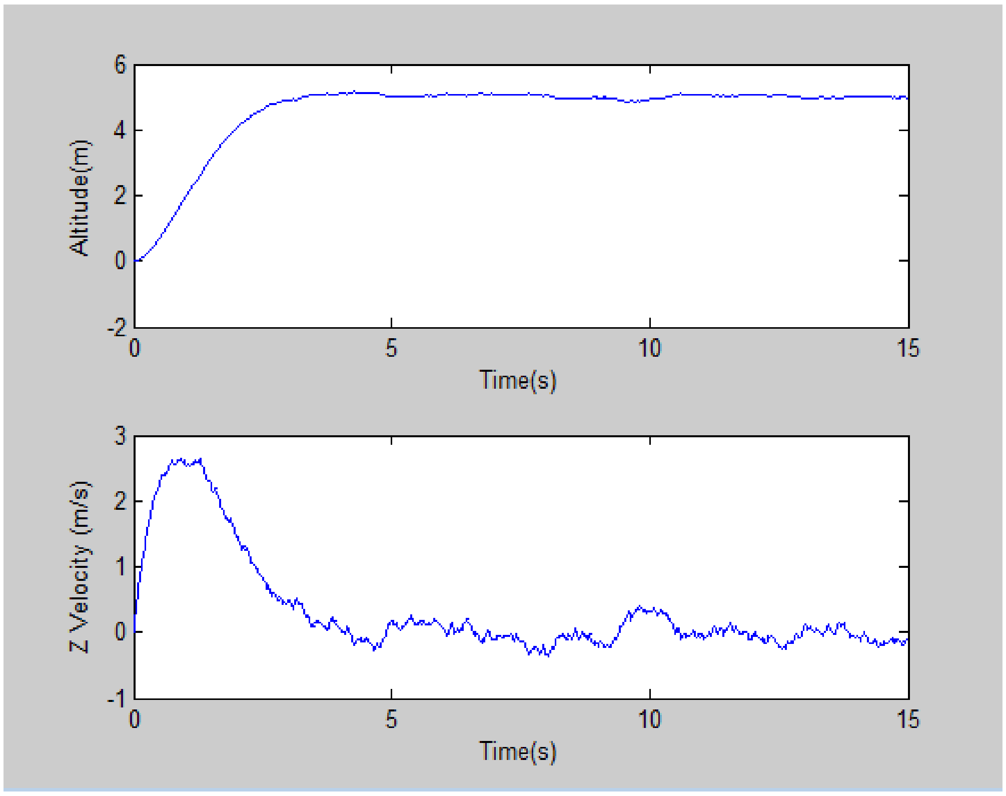


Figure 53: Altitude Control

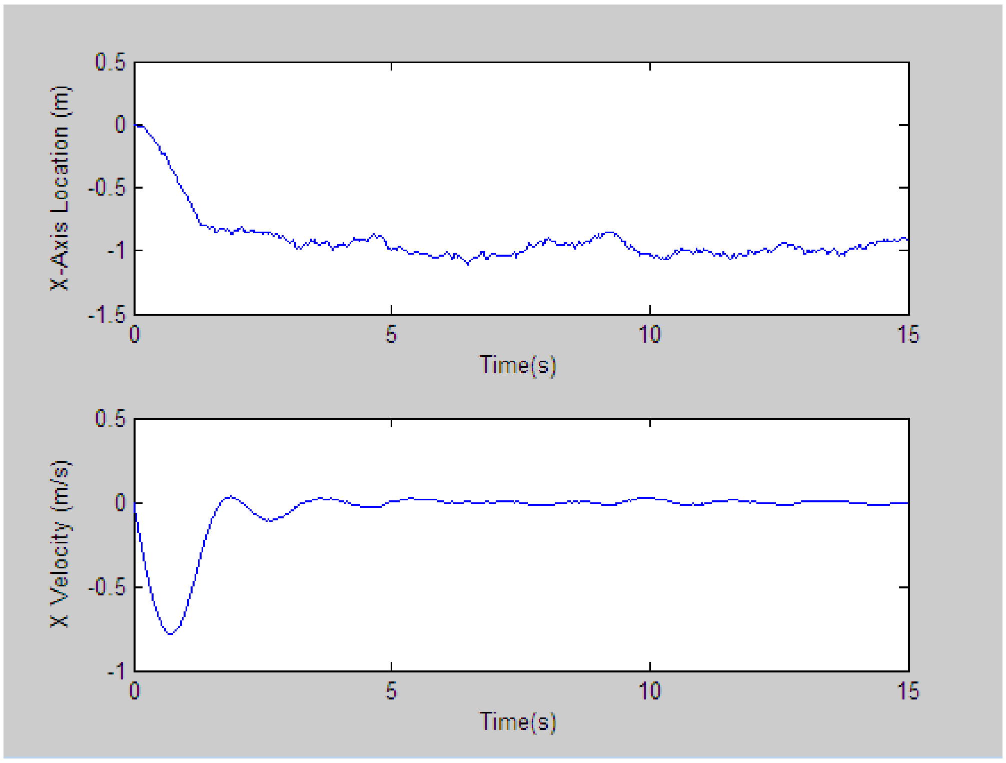


Figure 54: Lateral Control (X Axis)

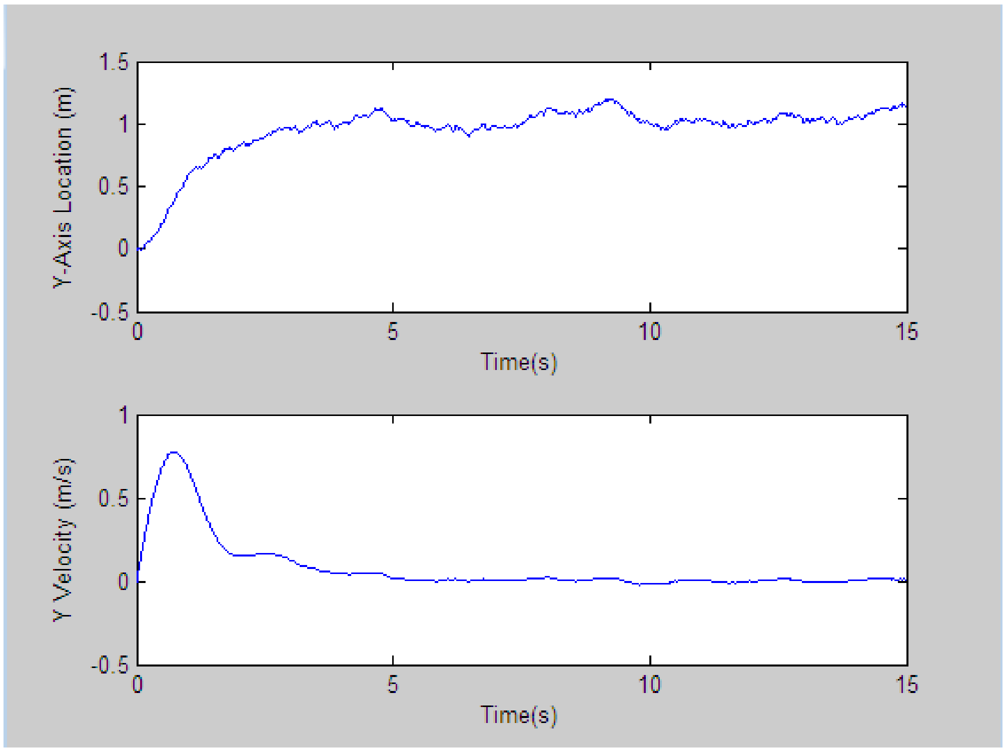
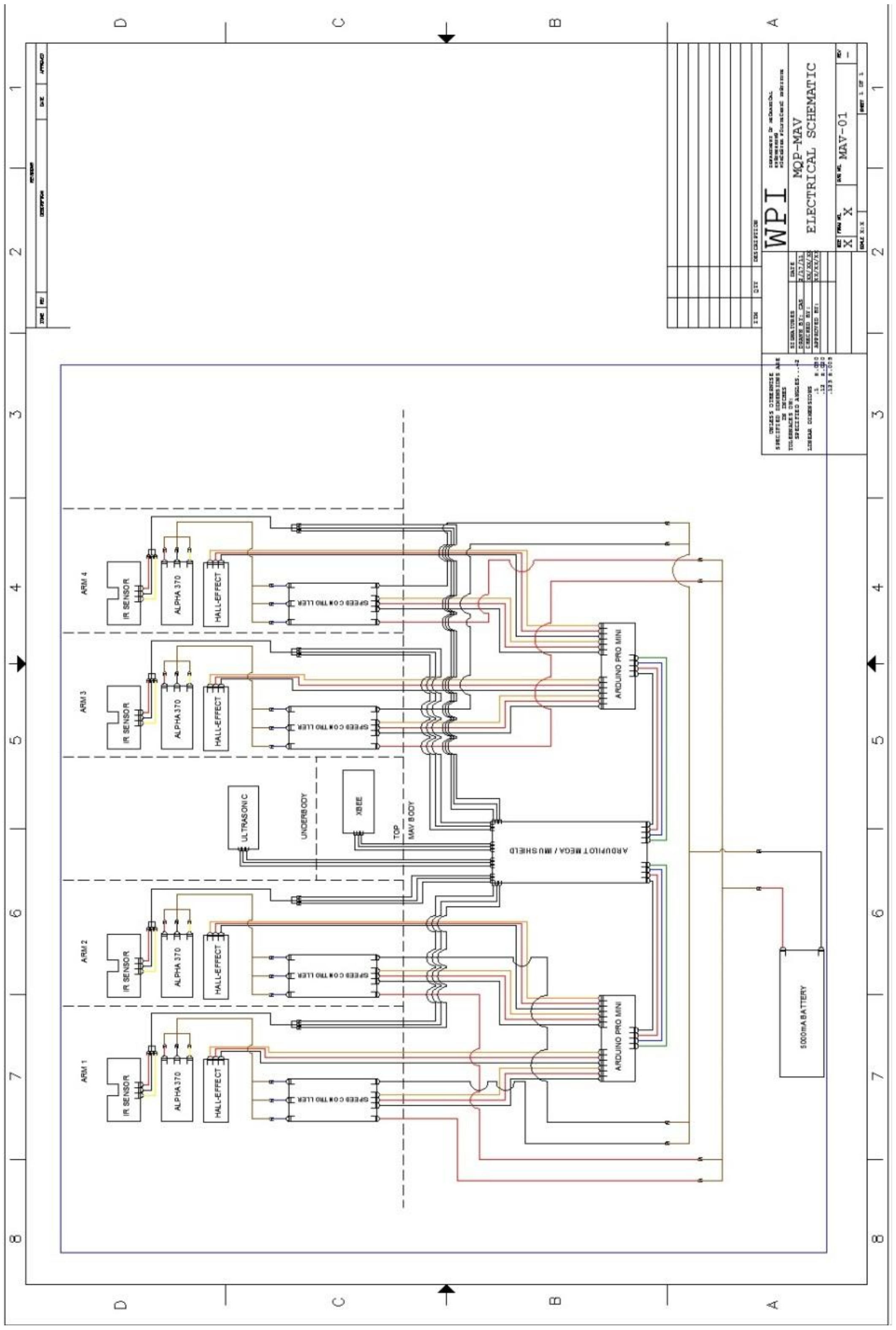


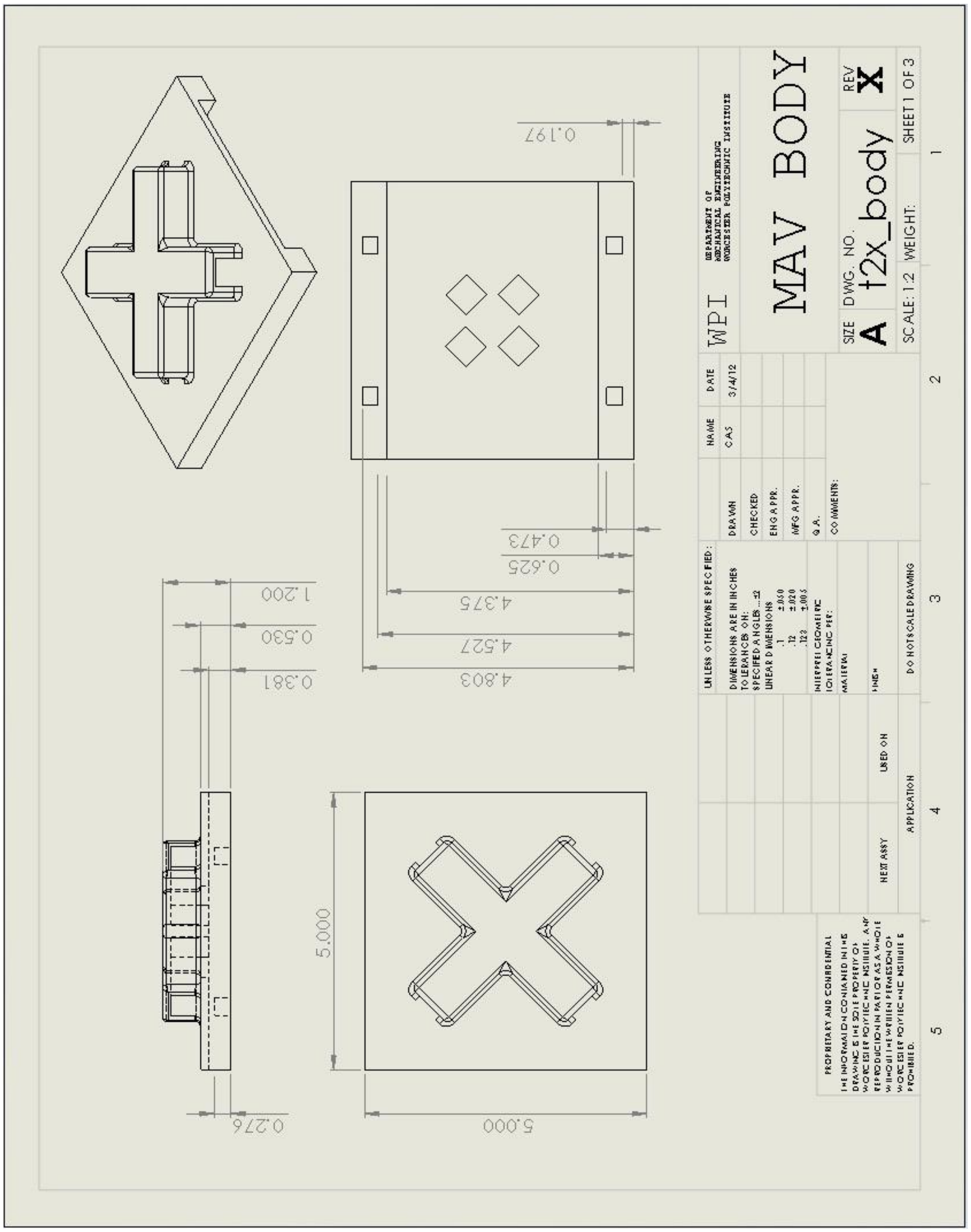
Figure 55: Lateral Control (Y Axis)

Appendix F: MAV Electrical Schematic

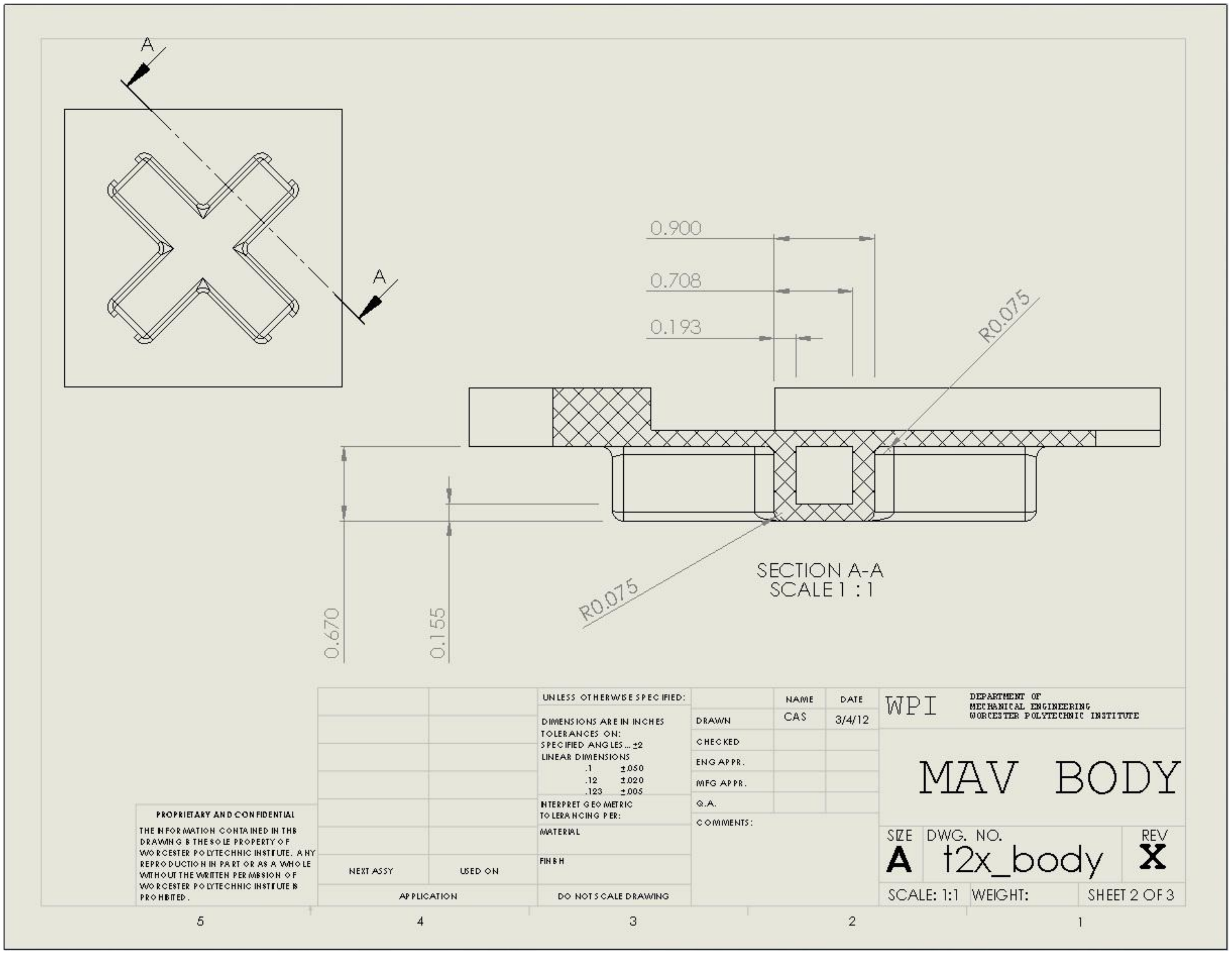


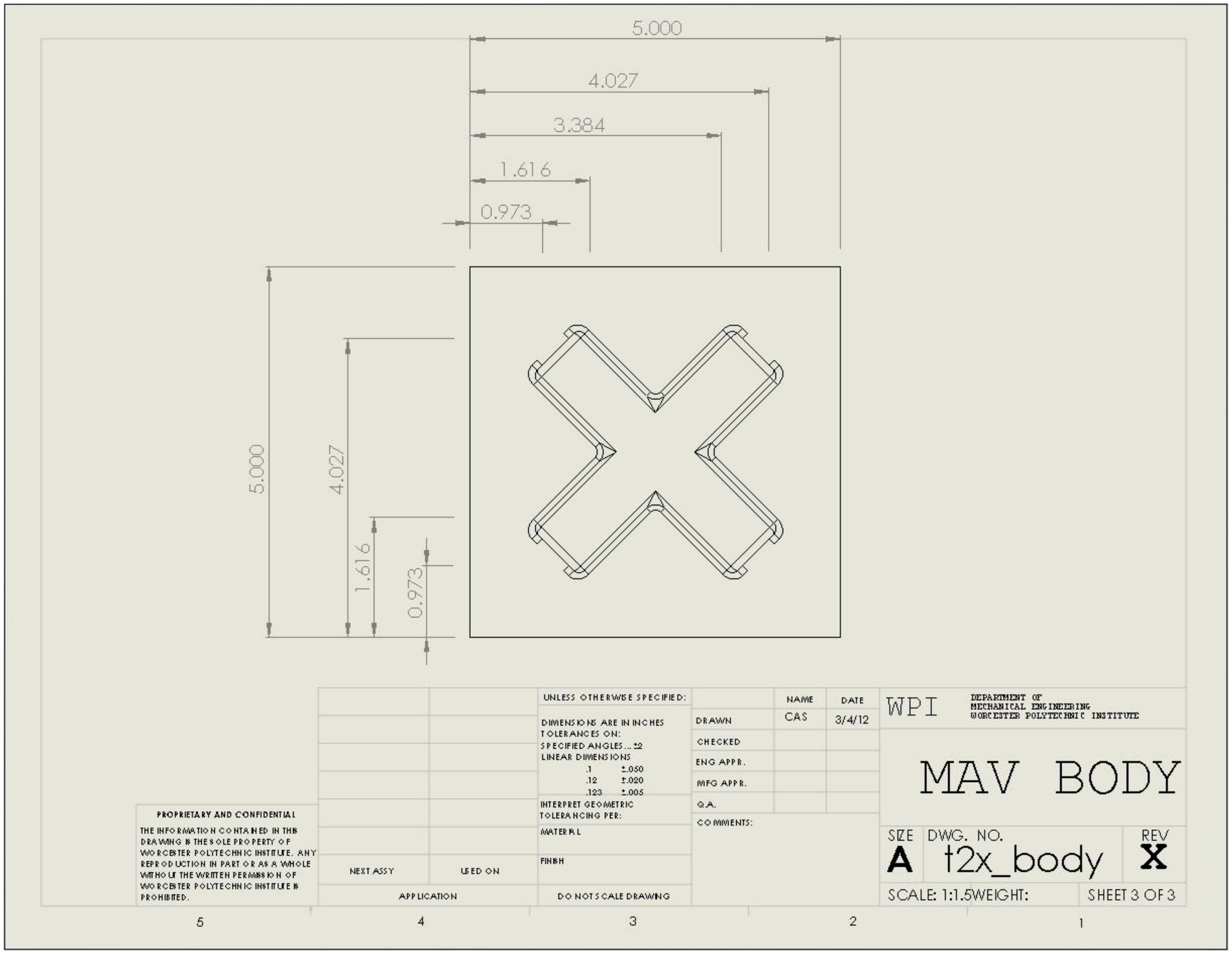
Appendix G: CAD Drawings

MAV Main Body



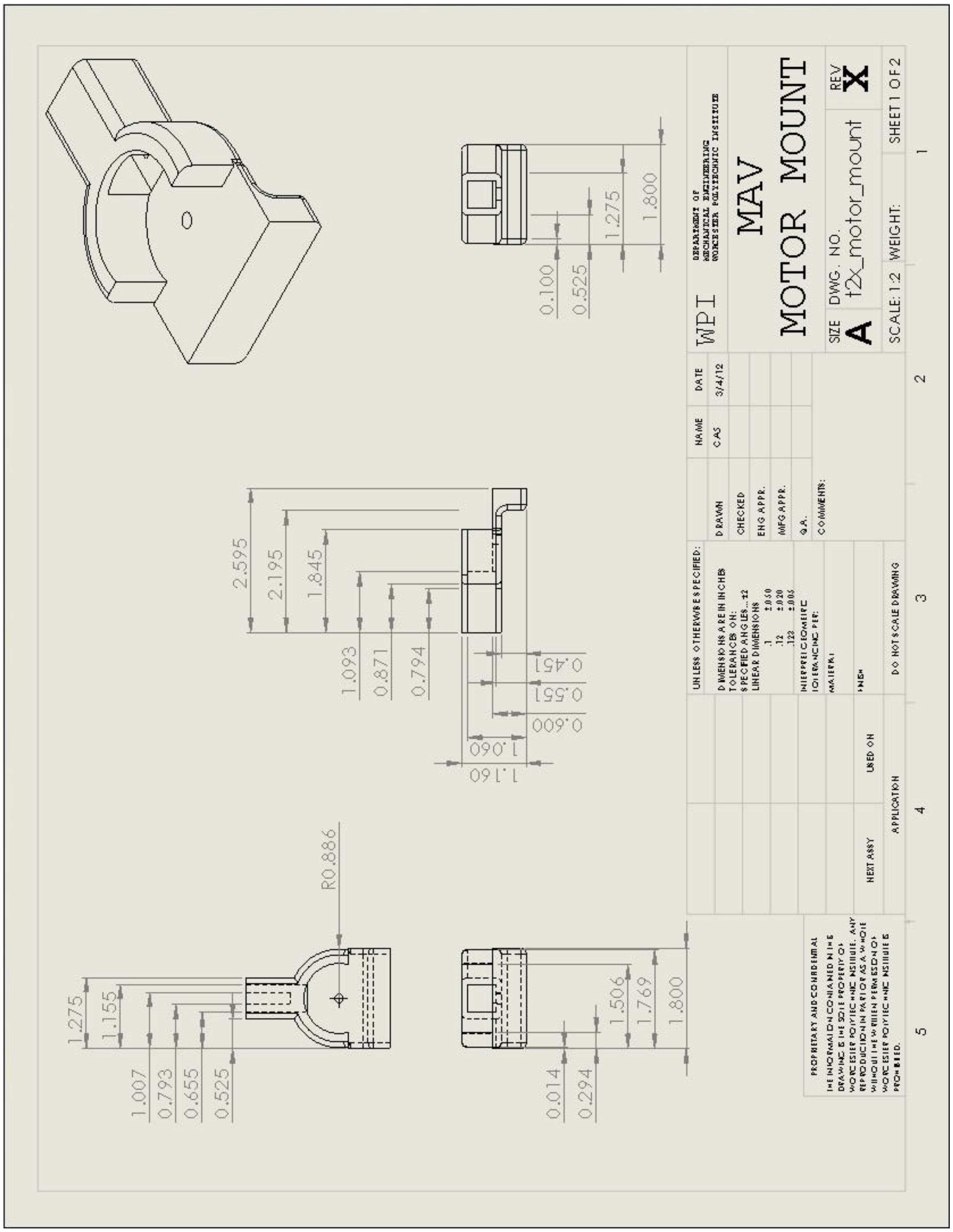
0.197
0.473
0.625
4.375
4.527
4.803
1.200
0.530
0.381
5.000
5.000
0.276
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV BODY
SIZE A
DWG. NO. t2x_body
REV X
SCALE: 1:2
WEIGHT:
SHEET 1 OF 3
NAME
DATE
DRAWN CAS 3/4/12
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFED ANGLES ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
1
2
3
4
5

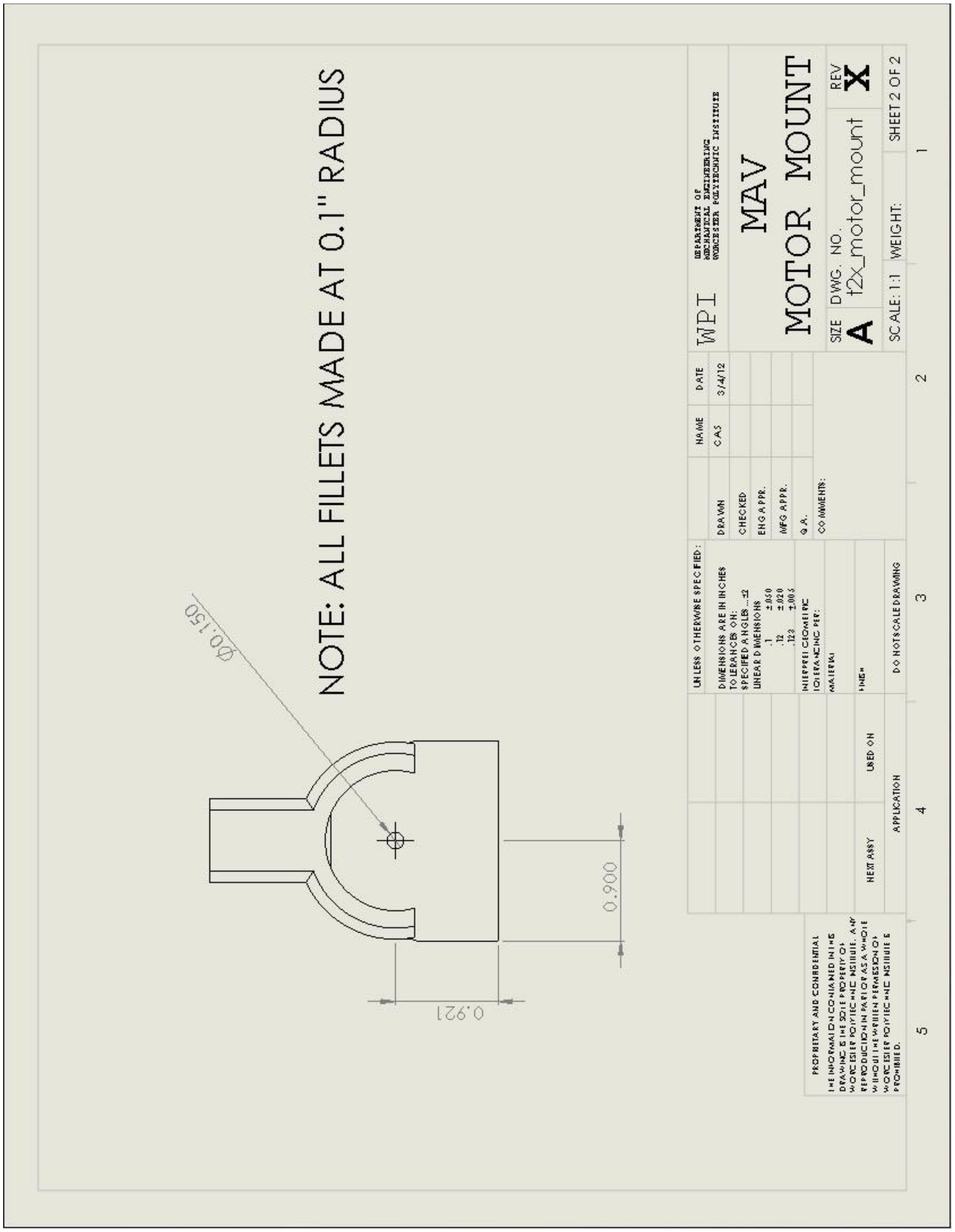

A
A
0.900
0.708
0.193
R0.075
R0.075
0.670
0.155
SECTION A-A
SCALE 1 : 1
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFIED ANGLES... ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CAS
3/4/12
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV BODY
SIZE
A
DWG. NO.
t2x_body
REV
X
SCALE: 1:1
WEIGHT:
SHEET 2 OF 3
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
5
4
3
2
1



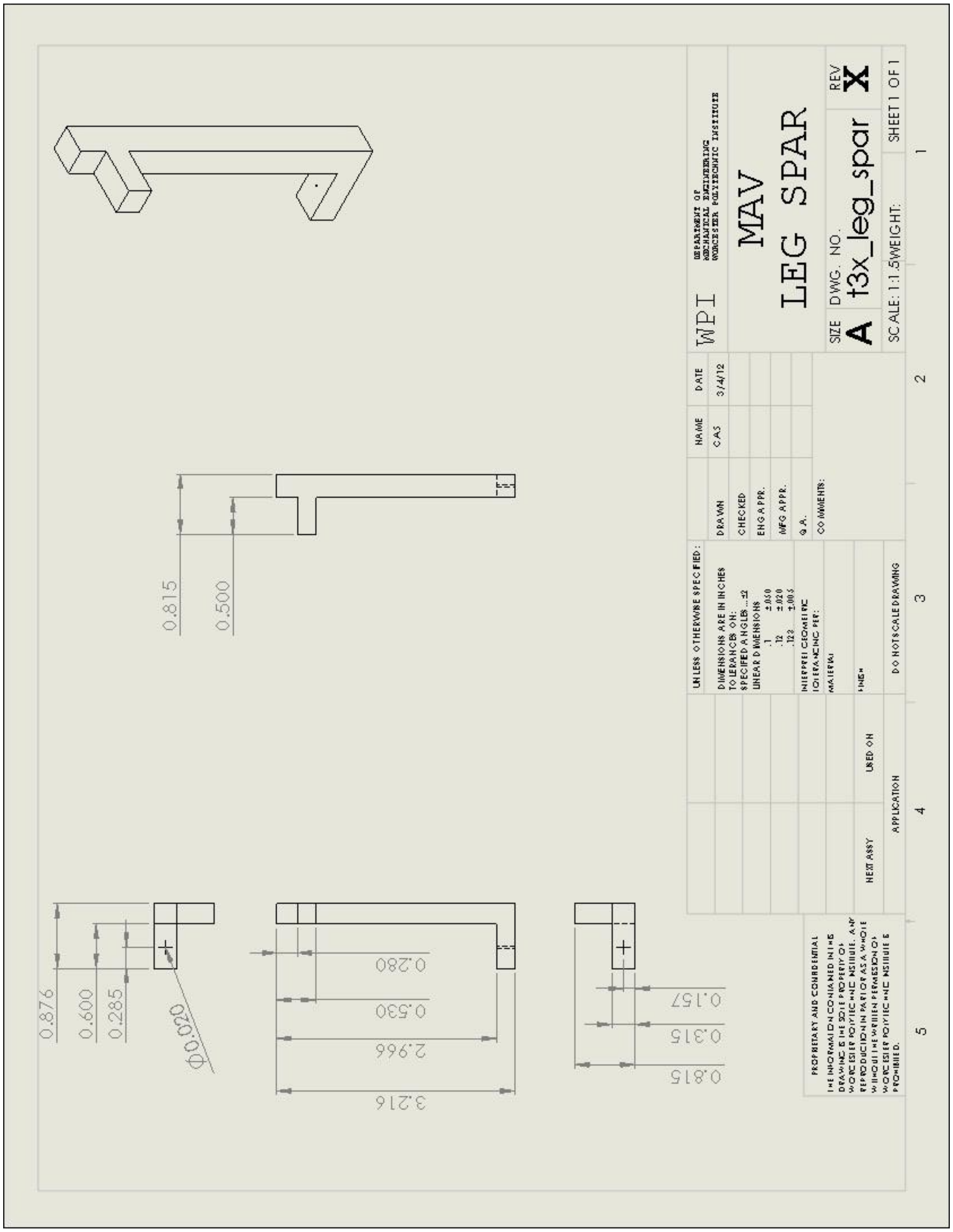
5.000
4.027
3.384
1.616
0.973
5.000
4.027
1.616
0.973
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFIED ANGLES ... ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CAS
3/4/12
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV BODY
SIZE
A
DWG. NO.
t2x_body
REV
X
SCALE: 1:1.5
WEIGHT:
SHEET 3 OF 3
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
5
4
3
2
1

MAV Motor Mount

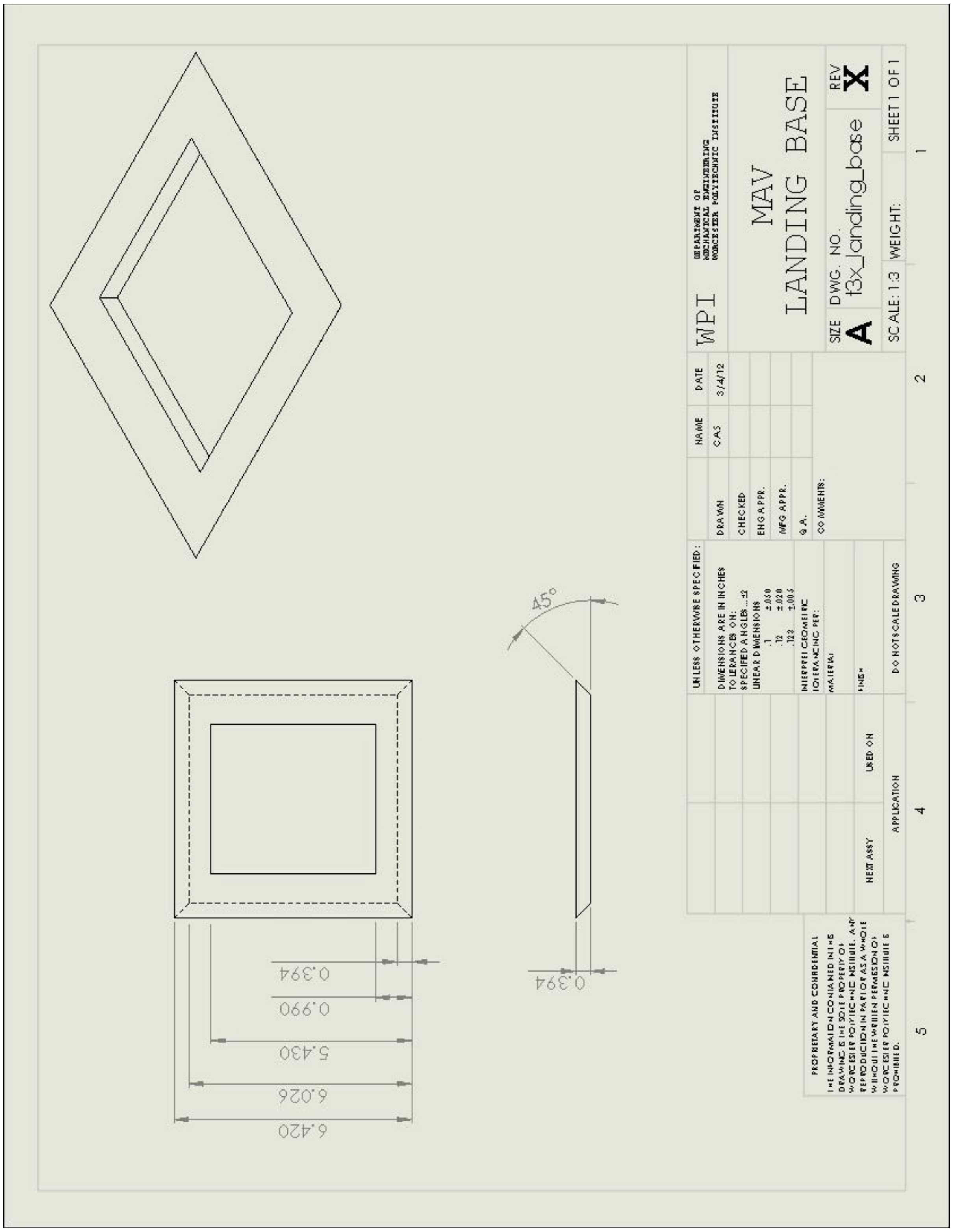



NOTE: ALL FILLETS MADE AT 0.1" RADIUS
Ø0.150
0.900
0.921
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFIED ANGLES ... ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
NAME
CAS
DATE
3/4/12
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV
MOTOR MOUNT
SIZE
A
DWG. NO.
t2x_motor_mount
REV
X
SCALE: 1:1
WEIGHT:
SHEET 2 OF 2
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
1
2
3
4
5

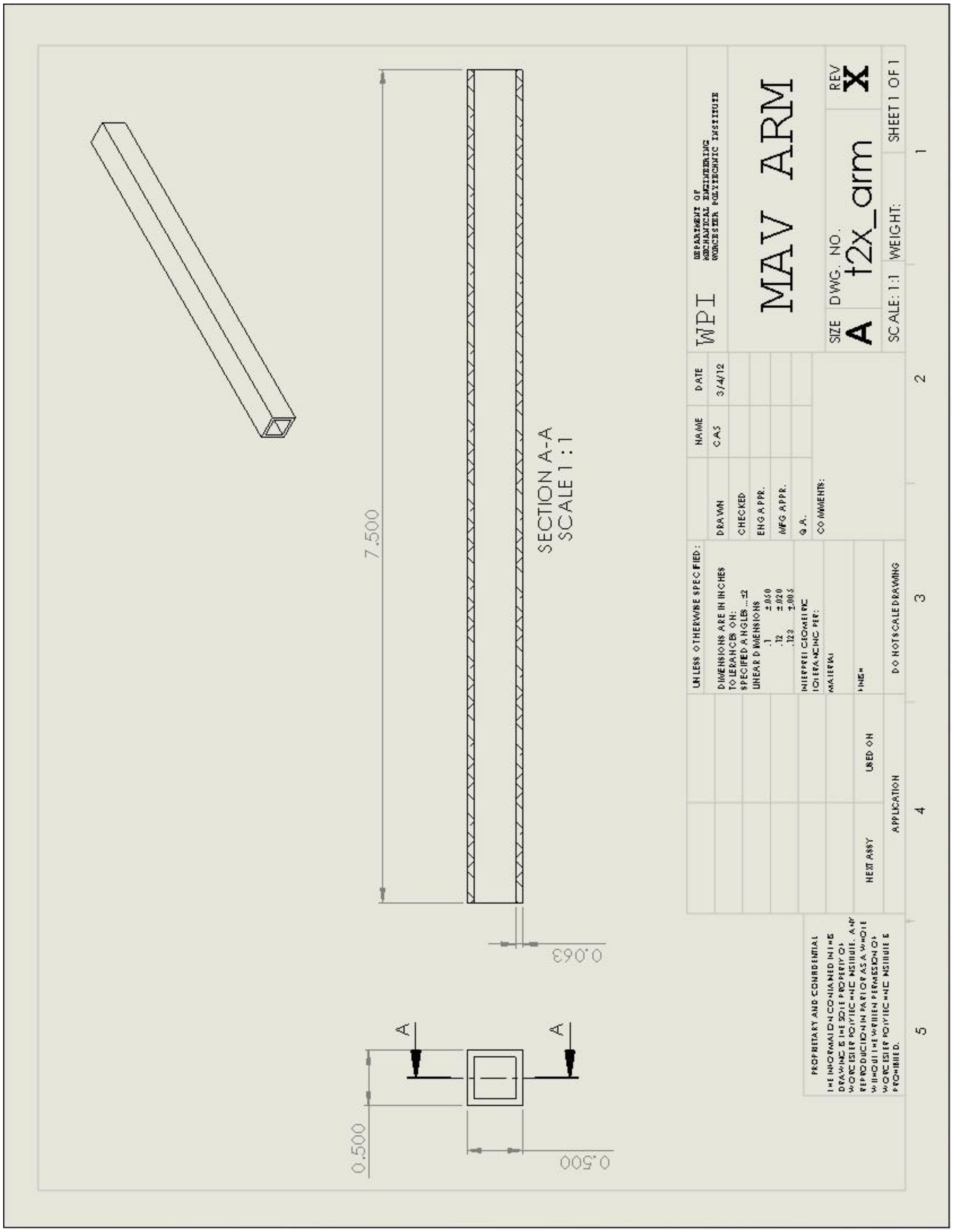
MAV Leg Spar


0.815
0.500
0.876
0.600
0.285
Φ0.020
0.280
0.530
2.966
3.216
0.157
0.315
0.815
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV LEG SPAR
SIZE A
DWG. NO. t3x_leg_spar
REV X
SCALE: 1:1.5
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN CAS 3/4/12
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFIED ANGLES ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
1
2
3
4
5

MAV Landing Base


6.420
6.026
5.430
0.990
0.394
0.394
45°
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
SPECIFIED ANGLES ±2
LINEAR DIMENSIONS
.1 ±.050
.12 ±.020
.123 ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
NAME
DATE
CAS
3/4/12
WPI
DEPARTMENT OF MECHANICAL ENGINEERING WORCESTER POLYTECHNIC INSTITUTE
MAV LANDING BASE
SIZE A
DWG. NO. f3x_landing_base
REV X
SCALE: 1:3
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.
1
2
3
4
5

MAV Arm



7.500
SECTION A-A
SCALE 1 : 1
0.063
0.500
0.500
A
A
WPI
MAV ARM
SIZE A
DWG. NO. t2x_arm
REV X
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
DRAWN CAS 3/4/12
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL

Appendix H: Instructions for MAV Use

To control the MAV, commands can be sent over serial. After connecting to a paired XBee at a baud rate of 52600, commands are sent line by line using a standard format. Every command begins with one character designating the command followed by up to four digits for the commands numerical value if applicable. A list of commands can be found in Table 9.

Table 9: MAV Commands

Command	Value?	Description
f		Begins full flight control
r		Sets MAV to "ready" mode for flight
k		Kills all motors
m		Enables testing of individual motors
s		Spins all four motors to 3000 RPM
S		Spins all four motors to 3500 RPM
l		Lands the MAV
a	X	Sets the desired altitude in CM
R	X	Sets the desired roll in degrees
P	X	Sets the desired pitch in degrees
Y	X	Sets the desired yaw in degrees
g	X	Sets the implied "weight" of the quadrotor
F		Resets the motor controller Arduinos
1	X	Spins motor 1 to the desired RPM
2	X	Spins motor 2 to the desired RPM
3	X	Spins motor 3 to the desired RPM
4	X	Spins motor 4 to the desired RPM

Appendix I: Unmanned Ground Vehicle Code

```
/*
John Pearsall

Used to test the speed controller or
calibrate the controller.

Main loop used for UGV.

*/

const int lowestPin = 2;
const int highestPin = 13;
const int switchUp = 22;
const int switchDown = 23;
const int UP = 150;
const int DOWN = 200;
const int STOP = 185;
String commandString, command;
int pin;

String getCommand() {

Serial.println("getCommand");
commandString = " ";
if(Serial.available() > 0){
commandString[0] = Serial.read();
}
Serial.println(commandString);
return commandString;
}

void setup() {
Serial.begin(115200);
pinMode(lowestPin,OUTPUT);
pinMode(switchUp,INPUT);
pinMode(switchDown,INPUT);
}

void loop() {
analogWrite(lowestPin, STOP);

command = getCommand();

if (command == "u"){
Serial.println("Recieved Up");

pin = digitalRead(switchUp);

while (pin == LOW){

analogWrite(lowestPin, UP);
pin = digitalRead(switchUp);
delay(10);
}
analogWrite(lowestPin, STOP);
Serial.println("Stopped");
}

if (command == "d"){
Serial.println("Recieved Down");

pin = digitalRead(switchDown);

while (pin == LOW){

analogWrite(lowestPin, DOWN);
pin = digitalRead(switchDown);
delay(10);
}
analogWrite(lowestPin, STOP);
Serial.println("Stopped");
}
}
```

Appendix J: Quad-rotor Flight Code

ArduPilot Software

```
#include <AP_ADC.h>
#include <Wire.h>

#define CALIBRATE_IMU 0    //If activated,
calibrates IMU zeros on startup
#define     ACCELERATION_COMPENSATION     1
//Compensates  for  acceleration  in  DCM
correction
#define USE_GYRO_CONTROL 1  //If activated,
uses gyro output in attitude control

#define  TALKOVERUSB  0    //Selects  which
serial channels to transmit over
#define TALKOVERXBEE 1

#define  PRINT_BREAK  1    //Selects  which
values to print each cycle
#define PRINT_IMUZEROS 0
#define PRINT_IMUSTATE 0
#define PRINT_EULERANGLES 1
#define PRINT_OMEGA 0
#define PRINT_DCM 0
#define PRINT_NORMERROR 0
#define PRINT_ACCELCOMP 0
#define PRINT_GRAVITYREFERENCE 0
#define PRINT_OMEGACORRECTION 0
#define PRINT_ATTITUDEDESIRED 0
#define PRINT_PIDERRORS 0
#define PRINT_PIDOUTPUT 0
#define PRINT_LOCALERROR 0
#define PRINT_ULTRASONICALTITUDE 0
#define PRINT_POSITIONVECTOR 1
#define PRINT_AT 0
#define PRINT_AM 0
#define PRINT_ACCELEROMETEROFFSETS 0
#define PRINT_MEASUREDACCELERATION 0
#define PRINT_LOCALACCELERATION 0
#define PRINT_MOTORSPEEDS 1

#define TODEG(r)      ((r)               *
57.2957795131)/*  convert  radians  to  degrees
*/
#define TORAD(d)      ((d)  /  57.2957795131)
/* convert to radians */

#define  GYROMAX  2000     //Gyro  input
parameters
#define GYROMIN .05
#define GYROSMOOTHVAL .1

#define  MAXHZ  100   //Maximum  ArduPilot
refresh rate
#define RPMLIMIT 6000  //Maximum rotor RPM
#define  MGMASTER  14.7  //Quadrotor  mass  in
newtons
#define MGSTART 14.7
#define MGMIN 12

AP_ADC_ADS7844 adc;
float MG;  //Current weight of quadrotor in
newtons
unsigned  long  time  =  0,  lastTime  =  0,
startTime = 0;
unsigned long lastSonarUpdate = 0;
unsigned long lastOmegaCorrection = 0;
unsigned long lastDCMNormalization = 0;
float dt = 0;
String stateString;
String lastStateString;

int channels[6] = {
2,1,0,5,4,6};  //omegaX,  omegaY,  omegaZ,  X,
Y, Z
char signs[6] = {
1,-1,1,1,-1,-1};

float IMUzeros[6];
float raws[6];
float gravityReference[3];
float gravityReferenceNorm[3];
float gyroVoltage = 3.3;
float gyroSensitivity = .002;  //Our gyro is
7mV/deg/sec
float ultrasonicAltitude;
float ultrasonicMinimum = 0.22; //Ultrasonic
kicks in at .22m

float IMUstate[6];
float filterVal = .5;
float omega[3] = {
0,0,0};
float omegaCorrection[3] = {
0,0,0};
float omegaICorrection[3] = {
0,0,0};
float omegaCorrectionWeight = .1;
float omegaKi = .05, omegaKp = 15;
float DCM[3][3] = {
{
1,0,0        }
,
{
0,1,0        }
,
{
0,0,1        }
};

float At[3] = {
0,0,0};
float At_1[3] = {
0,0,0};
float At_desired[3] = {
0,0,0};
float Am[3] = {
0,0,0};
float Au[3] = {
0,0,0};
float ea[3] = {
0,0,0};
float St[3] = {
0,0,0};
float Vt[3] = {
```

```
0,0,0};
float Vt_1[3] = {
0,0,0};
float Vt_desired[3] = {
0,0,0};
float Va[3] = {
0,0,0};
float Pt[3] = {
0,0,0};
float Pt_1[3] = {
0,0,0};
float Pt_desired[3] = {
0,0,0};
float Za = 0;
float Vu = 0;
float ez = 0;
float ev = 0;
float Cz = 0;
float Cv = 0;

float error[3] = {
0,0,0
};
float localError[3] = {
0,0,0
};
float lastError[3] = {
0,0,0
};
float intError[3]  = {
0,0,0
};
float lastdError[3] = {
0,0,0
};
float dError[3] = {
0,0,0
};
float ddError[3] = {
0,0,0
};

float localLateralAcceleration[2] = {
0,0};
float lastLocalLateralAcceleration[2] = {
0,0};
float lateralAccelerationError[2] = {
0,0};
float dLateralAcceleration[2] = {
0,0};
float lateralAccelerationUpdate[2] = {
0,0};
float intLateralAccelerationError[2] = {
0,0};

float altitude = 0;
float lastAltitude = 0;
float dAltitude = 0;
float desiredAltitude = 0;

int motorSpeeds[4] = {
0,0,0,0
};
float attitude[3] = {  //Psi, Theta, Phi
0,0,0
};
float attitudeDesired[3] = {
0,0,0
};
float u[4];
float forces[4] = {
0,0,0,0
};

void setup()
{
stateString = "STARTING";
beginWireless();
adc.Init();
Wire.begin();
zeroIMU();
startTime = millis();
}

void loop()
{
lastStateString = stateString;
commandCheck(); //Check serial for new input
if(time < (millis() - (1000 / MAXHZ))){
timeUpdate();
pollIMU();
DCMupdate();
positionUpdate();
wirelessPrint("Time = ");
wirelessPrint((long)millis());
wirelessPrint(", State = ");
wirelessPrintln(stateString);

//The following IF statements function as a
state machine

if(stateString.equals("STARTING"))
stateString = "READYING";

else      if(stateString.equals("READYING")){
//Start state READYING
onGroundReset();
if(((time - startTime) > 2500)){
reinitialize();
stateString = "READY";
}
}  //End state READYING

else         if(stateString.equals("READY")){
//Start state READY
for(char a = 0; a < 4; a++){   //READY mode
performs state estimation
motorSpeeds[a]  =  0;    //without  performing
flight control
}
writeSpeeds();
onGroundReset();
resetMotorControllers();
MG = MGSTART;
}  //End state READY

else          if(stateString.equals("KILL")){
//Start state KILL
for(char a = 0; a < 4; a++){   //KILL mode
turns off all motors
motorSpeeds[a] = 0;
}
writeSpeeds();
zeroIMU();
reinitialize();
onGroundReset();
}  //End state KILL

else       if(stateString.equals("SPINUP1")){
//Start state SPINUP1
for(char a = 0; a < 4; a++){   //The SPINUP
modes spin the motors to
motorSpeeds[a] = 3000;  //prepare for flight
}
writeSpeeds();
attitudeDesired[2] = attitude[2];
}

else       if(stateString.equals("SPINUP2")){
//Start state SPINUP2
for(char a = 0; a < 4; a++){
motorSpeeds[a] = 3500;
}
writeSpeeds();
attitudeDesired[2] = attitude[2];
}

else        if(stateString.equals("FLYING")){
//Start state FLYING
if(MG < MGMASTER) MG += (MGMASTER - MGSTART)
/ 30;
Pt_desired[2]   =   -1.5;      //FLYING   mode
performs all flight control calculations
accelerationControl();
attitudeControl();
rpmCalculator();
writeSpeeds();
if(abs(attitude[0]) > 45 || abs(attitude[1])
> 45) stateString = "KILL";
} //End state FLYING

Else    if(stateString.equals("DROP")){
//Start state DROP
MG = MGMIN; //DROP mode lands the quadrotor
Pt_desired[2] = 0;
Pt[2] = 0;
accelerationControl();
attitudeControl();
rpmCalculator();
writeSpeeds();
if(abs(attitude[0]) > 45 || abs(attitude[1])
> 45) stateString = "KILL";
} //End state FLYING

else   if   (stateString.equals("MOTORTEST")){
//Start state MOTORTEST
writeSpeeds();
}  //End state MOTORTEST

//delay(20);
#if PRINT_BREAK
for   (char   i   =   0;   i   <   5;   i++)
wirelessPrintln();
#endif
}
}

void  timeUpdate()     //Updates  the  time
variables for other functions
{
lastTime = time;
time = millis();
dt = (float)(time - lastTime) / 1000;
}

void  reinitialize()   //Resets  all  flight
control and attitude variables
{
for(char a = 0; a < 4; a++){
omega[a] = 0;
omegaCorrection[a] = 0;
omegaICorrection[a] = 0;
At[a] = 0;
At_1[a] = 0;
Vt[a] = 0;
Vt_1[a] = 0;
Pt[a] = 0;
Pt_1[a] = 0;
St[a] = 0;
error[a] = 0;
localError[a] = 0;
lastError[a] = 0;
intError[a]  = 0;
lastdError[a] = 0;
dError[a] = 0;
ddError[a] = 0;
}
for(char a = 0; a < 4; a++){
forces[a] = 0;
}
altitude = 0;
lastAltitude = 0;
dAltitude = 0;
desiredAltitude = 0;

for(char  a  =  0;  a  <  3;  a++)  for(char  b  =  0;
b < 3; b++) DCM[a][b] = 0;
DCM[0][0] = 1;
DCM[1][1] = 1;
DCM[2][2] = 1;
}

float az1 = 4; //3.25
float az2 = 2.5;
float ax1 = 0;
float ax2 = 0;
float ay1 = 0;
float ay2 = 0;
float zsat = 2;

float lateralAcceleration[2] = {
0,0};
float lateralVelocity[2] = {
0,0};

void    accelerationControl()       //Performs
altitude and lateral acceleration control
{  //Later acceleration control never fully
implemented
lastAltitude  =  altitude;   //Updates  state
variables for control
altitude = Pt[2];
dAltitude = (altitude - lastAltitude) / dt;
desiredAltitude = Pt_desired[2];

lateralAcceleration[0]      =      Am[0]      *
cos(TORAD(attitude[2]))      -     Am[1]      *
sin(TORAD(attitude[2]));
lateralAcceleration[1]      =      Am[0]      *
sin(TORAD(attitude[2]))      +     Am[1]      *
cos(TORAD(attitude[2]));
lateralVelocity[0]       =       Vt[0]       *
cos(TORAD(attitude[2]))      -     Vt[1]      *
sin(TORAD(attitude[2]));
lateralVelocity[1]       =       Vt[0]       *
sin(TORAD(attitude[2]))      +     Vt[1]      *
cos(TORAD(attitude[2]));

float   mzdd   =   az2   *   dAltitude   +   az2   *
(altitude - desiredAltitude);

mzdd = constrain(mzdd, -zsat, zsat);
```

```
u[3]      =      (mzdd      +      MG)      /
(cos(TORAD(attitude[0]))                    *
cos(TORAD(attitude[1])));


attitudeDesired[1]                          =
TODEG(asin(constrain((ax1                   *
lateralAcceleration[0]      +      ax2      *
(lateralVelocity[0]  -  Vt_desired[0]))/u[3],
-1, 1)));
attitudeDesired[0] = TODEG(asin(constrain((-
ay1   *   lateralAcceleration[1]   -   ay2   *
(lateralVelocity[1]                         -
Vt_desired[1]))/(u[3]*cos(TORAD(attitudeDesi
red[1]))), -1, 1)));
}

float a1attitude[3] = { //Kd
.15,.15,0//.15
};
float a2attitude[3] = { //Kp
1.4,1.4,0 //1
};
float a3attitude[3] = { //Kdd
.14,.14,0//.12
};

void attitudeControl()   //Performs attitude
P-D-DD control routine
{
errorUpdate();
attitudePIDController();
}

void errorUpdate()   //Updates error terms
for attitude control
{
#if PRINT_ATTITUDEDESIRED
wirelessPrintln("Attitude Desired = ");
printVec4(attitudeDesired);
#endif
#if PRINT_PIDERRORS
wirelessPrintln("Errors = ");
#endif
for(char i = 0; i < 3; i++){
lastError[i] = localError[i];
error[i] = attitudeDesired[i] - attitude[i];
}

if(error[2] > 180) error[2] -= 360;   //Yaw
+-180 degree wrap-around
if(error[2] < -180) error[2] += 360;

int directionRoll, directionPitch;
if(error[0] > 0) directionRoll = 1;
else directionRoll = -1;
if(error[1] > 0) directionPitch = 1;
else directionPitch = -1;

localError[0] = error[0];
localError[1] = error[1];
localError[2] = error[2];
         for(char  i  =  0;  i  <  3;  i++)
lastdError[i] = dError[i];

#if USE_GYRO_CONTROL
dError[0] = TODEG(IMUstate[0]) * -1;
dError[1] = TODEG(IMUstate[1]) * -1;
dError[2] = TODEG(IMUstate[2]) * -1;
#endif
for(char i = 0; i < 3; i++){
#if !USE_GYRO_CONTROL
dError[i] = (localError[i] - lastError[i]) /
dt;
#endif
ddError[i]  =  (dError[i]  -  lastdError[i])  /
dt;
intError[i] += localError[i] * dt;
#if PRINT_PIDERRORS
wirelessPrint(localError[i]);
wirelessPrint(" ");
wirelessPrint(intError[i]);
wirelessPrint(" ");
wirelessPrint(dError[i]);
wirelessPrint(" ");
wirelessPrint(ddError[i]);
wirelessPrintln(" ");
#endif
}


#if PRINT_LOCALERROR
wirelessPrint("Local error = ");
printVec4(localError);
#endif
}

void  attitudePIDController()    //Calculates
P-D-DD control output
{
float m1 = 80;  //Saturation level variables
float m2 = .5 * m1;
float m3 = .5 * m2;

//u[0] = constrain(a1attitude[0] * dError[0]
+ constrain(a2attitude[0] * localError[0] +
constrain(a3attitude[0]  *  ddError[0],  -m3,
m3), -m2, m2), -m1, m1);
//u[1] = constrain(a1attitude[1] * dError[1]
+ constrain(a2attitude[1] * localError[1] +
constrain(a3attitude[1]  *  ddError[1],  -m3,
m3), -m2, m2), -m1, m1);
//u[2] = constrain(a1attitude[2] * dError[2]
+ constrain(a2attitude[2] * localError[2] +
constrain(a3attitude[2]  *  ddError[2],  -m3,
m3), -m2, m2), -m1, m1);

u[0]  =  constrain(a3attitude[0]  * ddError[0]
+   constrain(a1attitude[0]   *   dError[0]   +
constrain(a2attitude[0]  *  localError[0],  -
m3, m3), -m2, m2), -m1, m1);
u[1]  =  constrain(a3attitude[1]  * ddError[1]
+   constrain(a1attitude[1]   *   dError[1]   +
constrain(a2attitude[1]  *  localError[1],  -
m3, m3), -m2, m2), -m1, m1);
u[2]  =  constrain(a3attitude[2]  * ddError[2]
+   constrain(a1attitude[2]   *   dError[2]   +
constrain(a2attitude[2]  *  localError[2],  -
m3, m3), -m2, m2), -m1, m1);

#if PRINT_PIDOUTPUT
wirelessPrint("PID Output = ");
printVec4(u);
#endif
}

#define NORMTHRESHOLD .01

float DCMX[3];
float DCMY[3];
float DCMZ[3];
float normError = 0;

void  DCMupdate()   //Updates  the  Direction
Cosine Matrix
{   //For  details,  see  DCM  IMU  Theory  by
Premerlani and Bizard
omegaUpdate();

float omegadtDCM[3][3] = {   //Builds update
DCM from gyro data
{
1,   -omega[2]   *   dt,   omega[1]   *   dt
}
,
{
omega[2]   *   dt,   1,   -omega[0]   *   dt
}
,
{
-omega[1]   *   dt,   omega[0]   *   dt,   1
}
};

float temp[3][3];
for (char i = 0; i < 3; i++)
for (char j = 0; j < 3; j++){
temp[i][j] = 0;
for (char k = 0; k < 3; k++){
temp[i][j]       +=       (DCM[i][k]       *
omegadtDCM[k][j]);
}
};
for (char i = 0; i < 3; i++)
for  (char  j  =  0;  j  <  3;  j++)  DCM[i][j]  =
temp[i][j];

DCMvecUpdate();
normError = dotProduct(DCMX, DCMZ);
normalizeDCM();

omegaCorrectionUpdate();

#if PRINT_DCM
print3x3(DCM);
#endif

updateEuler();

#if PRINT_EULERANGLES
wirelessPrint("Euler angles = ");
printVec(attitude);
#endif
}

void omegaUpdate()
{
for(char i = 0; i < 3; i++){
omega[i] = IMUstate[i] + omegaCorrection[i];
}
#if PRINT_OMEGA
wirelessPrint("Omega = ");
printVec(omega);
#endif
}

void DCMvecUpdate()
{
for (char i = 0; i < 3; i++){
DCMX[i] = DCM[i][0];
DCMY[i] = DCM[i][1];
DCMZ[i] = DCM[i][2];
}
}

void normalizeDCM() //Normalizes DCM vectors
to ensure orthogonality
{
#if PRINT_NORMERROR
wirelessPrint("Normalization Error = ");
wirelessPrintln(normError);
#endif

float Xorth[3] = {
DCMX[0] - ((normError/2)*DCMZ[0]),
DCMX[1] - ((normError/2)*DCMZ[1]),
DCMX[2] - ((normError/2)*DCMZ[2])
};
//printVec(Xorth);
float Zorth[3] = {
DCMZ[0] - ((normError/2)*DCMX[0]),
DCMZ[1] - ((normError/2)*DCMX[1]),
DCMZ[2] - ((normError/2)*DCMX[2])
};
float Yorth[3] = {
(Zorth[1]   *   Xorth[2])   -   (Zorth[2]   *
Xorth[1]),
(Zorth[2]   *   Xorth[0])   -   (Zorth[0]   *
Xorth[2]),
(Zorth[0]   *   Xorth[1])   -   (Zorth[1]   *
Xorth[0])
};

float mag[3] = {
1/sqrt(Xorth[0]  *  Xorth[0]  +  Xorth[1]  *
Xorth[1] + Xorth[2] * Xorth[2]),
1/sqrt(Yorth[0]  *  Yorth[0]  +  Yorth[1]  *
Yorth[1] + Yorth[2] * Yorth[2]),
1/sqrt(Zorth[0]  *  Zorth[0]  +  Zorth[1]  *
Zorth[1] + Zorth[2] * Zorth[2])
};

float Xnorm[3] = {
mag[0] * Xorth[0],
mag[0] * Xorth[1],
mag[0] * Xorth[2]
};
float Ynorm[3] = {
mag[1] * Yorth[0],
mag[1] * Yorth[1],
mag[1] * Yorth[2]
};
float Znorm[3] = {
mag[2] * Zorth[0],
mag[2] * Zorth[1],
mag[2] * Zorth[2]
};

for (char i = 0; i < 3; i++){
DCM[i][0] = Xnorm[i];
DCM[i][1] = Ynorm[i];
DCM[i][2] = Znorm[i];
};

DCMvecUpdate();

}

void      omegaCorrectionUpdate()      //Uses
accelerometer data to correct DCM drift
{
#if PRINT_GRAVITYREFERENCE
wirelessPrint("Gravity Reference = ");
printVec(gravityReference);
#endif

float          gravityReferenceMag          =
(sqrt(gravityReference[0]                   *
gravityReference[0]
+ gravityReference[1] * gravityReference[1]
+          gravityReference[2]              *
gravityReference[2]));

gravityReferenceNorm[0]                     =
gravityReference[0]                         *
cos(TORAD(attitude[2]))                     -
gravityReference[1]                         *
sin(TORAD(attitude[2]));
gravityReferenceNorm[1]                     =
gravityReference[0]                         *
sin(TORAD(attitude[2]))                     +
gravityReference[1]                         *
cos(TORAD(attitude[2]));
gravityReferenceNorm[2]                     =
gravityReference[2];

for   (char   i   =   0;   i   <   3;   i++)
gravityReferenceNorm[i]                     =
gravityReferenceNorm[i]                     /
gravityReferenceMag;
//for   (char   i   =   0;   i   <   3;   i++)
gravityReferenceNorm[i]                     =
gravityReferenceNorm[i]   *   (DCM[i][0]    +
DCM[i][0] + DCM[i][2]);

float RPCorrectionPlane[] = {
(DCMZ[1]    *    gravityReferenceNorm[2])   -
(DCMZ[2] * gravityReferenceNorm[1]),
(DCMZ[2]    *    gravityReferenceNorm[0])   -
(DCMZ[0] * gravityReferenceNorm[2]),
(DCMZ[0]    *    gravityReferenceNorm[1])   -
(DCMZ[1] * gravityReferenceNorm[0])  };

if(attitude[2] > 90 || attitude[2] < -90){
RPCorrectionPlane[0]          =             -
RPCorrectionPlane[0];
RPCorrectionPlane[1]          =             -
RPCorrectionPlane[1];
RPCorrectionPlane[2]          =             -
RPCorrectionPlane[2];
}

float accelerationCompensation = 1;
#if ACCELERATION_COMPENSATION
if       (gravityReferenceMag       >       1)
accelerationCompensation                    =
pow(1/gravityReferenceMag, 4);
else        accelerationCompensation        =
pow(gravityReferenceMag, 4);
#endif

#if PRINT_ACCELCOMP
wirelessPrint("AccelComp = ");

wirelessPrintln(accelerationCompensation);
#endif

for (char i = 0; i < 3; i++){
RPCorrectionPlane[i]  =  RPCorrectionPlane[i]
*           omegaCorrectionWeight           *
accelerationCompensation;         //Converts
RPCorrectionPlane into TotalCorrection
omegaICorrection[i]  =  omegaICorrection[i]  +
omegaKi * dt * RPCorrectionPlane[i];
omegaCorrection[i]      =      omegaKp      *
RPCorrectionPlane[i] + omegaICorrection[i];
}

#if PRINT_OMEGACORRECTION
wirelessPrint("OmegaCorrection = ");
printVec(omegaCorrection);
#endif
}

void  updateEuler()   //Updates  euler  angles
based off DCM
{
attitude[0] = TODEG(asin(DCM[2][1]));
attitude[1] = TODEG(-asin(DCM[2][0]));
attitude[2]                                 =
TODEG(atan2(DCM[1][0],DCM[0][0]));
}

float  dotProduct  (float  v1[3],  float  v2[3])
//Calculates vector dot product
{
float temp = 0;
for(char i = 0; i < 3; i++) temp += (v1[i] *
v2[i]);
return temp;
}

void print3x3 (float m[3][3])
{
wirelessPrint(m[0][0]);
wirelessPrint(" ");
wirelessPrint(m[0][1]);
wirelessPrint(" ");
wirelessPrint(m[0][2]);
wirelessPrintln(" ");
wirelessPrint(m[1][0]);
wirelessPrint(" ");
wirelessPrint(m[1][1]);
wirelessPrint(" ");
wirelessPrint(m[1][2]);
wirelessPrintln(" ");
wirelessPrint(m[2][0]);
wirelessPrint(" ");
wirelessPrint(m[2][1]);
wirelessPrint(" ");
wirelessPrint(m[2][2]);
wirelessPrintln(" ");
}

void printVec (float v[3]){
wirelessPrint(v[0]);
wirelessPrint(" ");
wirelessPrint(v[1]);
wirelessPrint(" ");
wirelessPrint(v[2]);
wirelessPrintln(" ");
}

void printVec4 (float v[4]){
wirelessPrint(v[0]);
wirelessPrint(" ");
wirelessPrint(v[1]);
wirelessPrint(" ");
wirelessPrint(v[2]);
wirelessPrint(" ");
wirelessPrint(v[3]);
wirelessPrintln(" ");
}
```

```
void zeroIMU()  //Sets IMU zero values
{
#if CALIBRATE_IMU
for (char i = 0; i < 6; i++){
adc.Ch(channels[i]);
} //Throwaway first values
delay(500);
for (char i = 0; i < 6; i++){
IMUzeros[i] = adc.Ch(channels[i]);
}
delay(100);
for (char i = 0; i < 3; i++){
IMUzeros[i] += adc.Ch(channels[i]);
IMUzeros[i] = IMUzeros[i] / 2;
}
delay(100);
for (char i = 0; i < 3; i++){
IMUzeros[i] = (2*IMUzeros[i]/3)
+ (adc.Ch(channels[i]) / 3);
}
#endif

#if !CALIBRATE_IMU
IMUzeros[0] = 1644;
IMUzeros[1] = 1600;
IMUzeros[2] = 1662;
IMUzeros[3] = 2021;
IMUzeros[4] = 2037;
IMUzeros[5] = 1687;
#endif

#if PRINT_IMUZEROS
wirelessPrint("IMU Zeros = ");
for(char i = 0; i < 6; i++){
wirelessPrint(IMUzeros[i]);
wirelessPrint(" ");
}
wirelessPrintln();
#endif
}

void pollIMU()  //Polls data from 6-axis IMU
{
for (char i = 0; i < 6; i++){
raws[i] = adc.Ch(channels[i]);
raws[i] -= IMUzeros[i];
}

//      IMUstate[0]   =   smooth(((raws[0]   *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[0] / 90), filterVal, IMUstate[0]);
//      IMUstate[1]   =   smooth(((raws[1]   *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[1] / 90), filterVal, IMUstate[1]);
//      IMUstate[2]   =   smooth(((raws[2]   *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[2] / 90), filterVal, IMUstate[2]);
IMUstate[0]    =    smooth(TORAD((raws[0]    *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[0]), GYROSMOOTHVAL, IMUstate[0]);
IMUstate[1]    =    smooth(TORAD((raws[1]    *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[1]), GYROSMOOTHVAL, IMUstate[0]);
IMUstate[2]    =    smooth(TORAD((raws[2]    *
gyroVoltage  /  (4095  *  gyroSensitivity))  *
signs[2]), GYROSMOOTHVAL, IMUstate[0]);
IMUstate[3] = ((raws[3]) / 418) * signs[3];
IMUstate[4] = ((raws[4]) / 418) * signs[4];
IMUstate[5] = ((raws[5]) / 418) * signs[5] +
1;
for (char i = 0; i < 3; i++)
{
if(abs(IMUstate[i]) > GYROMAX)
IMUstate[i] = 0;
if(abs(IMUstate[i]) < GYROMIN)
IMUstate[i] = 0;
}

gravityReference[0] = IMUstate[3];
gravityReference[1] = IMUstate[4];
gravityReference[2] = IMUstate[5];

#if PRINT_IMUSTATE
printIMUstate();
#endif
}

void  printIMUstate()   //Prints  current  IMU
values
{
wirelessPrint("IMU state = ");
for (char i = 0; i < 6; i++){
wirelessPrint(IMUstate[i]);
wirelessPrint(" ");
}
wirelessPrintln();
}

void  pollUltrasonic(){   //Reads  and  scales
data from ultrasonic rangefinder
float conversion = 0.0117219;
float offset = .0812063;
int sensorValue = analogRead(A7);

ultrasonicAltitude                          =
smooth((conversion*(float)sensorValue)      +
offset, .7, ultrasonicAltitude);
}

void positionUpdate()
{
updateT_1();
estimateStep();   //Estimates  position  based
on accelerometer data

if(time  >  (lastSonarUpdate  +  20)){   //Sets
alitude from ultrasonic data
pollUltrasonic();
if  (ultrasonicAltitude  >  ultrasonicMinimum)
Pt[2] = -ultrasonicAltitude;
else Pt[2] = -ultrasonicMinimum;
if (ultrasonicAltitude > 3) Pt[2] = -3;
lastSonarUpdate = time;
Pt[2]  =  Pt[2]  *  cos(TORAD(attitude[0]))  *
cos(TORAD(attitude[1]));
}

#if PRINT_ULTRASONICALTITUDE
wirelessPrint("Ultrasonic Altitude = ");

wirelessPrintln(ultrasonicAltitude);
#endif

//AscalingStep();

#if PRINT_POSITIONVECTOR
wirelessPrint("Position = ");
printVec(Pt);
#endif
#if PRINT_ACCELEROMETEROFFSETS
wirelessPrint("Accelerometer Offsets = ");
printVec(St);
#endif
}

void ultrasonicPositionUpdate()
{
float Kz = 1.02329;
float Kv = 1.02329;

//ultrasound outputs upward Z while global Z
is downward
ez = abs((Pt[2] + ultrasonicAltitude));

//Cz = pow(Kz, -1 * 100 * ez);
Cz = .5;

Pt[2] = (Cz * -1 * ultrasonicAltitude) + ((1
- Cz) * Pt[2]);

Vu = (Pt[2] - Pt_1[2]) / dt;
ev = abs(Vt[2] - Vu);

//Cv = pow(Kv, -1 * 100 * ev);
Cv = .7;


Vt[2] = (Cv * Vu) + ((1-Cv) * Vt[2]);

}

void updateT_1()
{
for(char i = 0; i < 3; i++){
Vt_1[i] = Vt[i];
At_1[i] = At[i];
Pt_1[i] = Pt[i];
}
}

void onGroundReset()
{
for(char a = 0; a < 3; a++){
Vt[a] = 0;
At[a] = 0;
}
Pt[2] = 0;
}

void estimateStep()
{
measuredAccelerationUpdate();
for (char i = 0; i < 3; i++){
At[i] = Am[i] + St[i];
Vt[i] = (At_1[i] + At[i])*dt/2 + Vt_1[i];
Pt[i] = (Vt_1[i] + Vt[i])*dt/2 + Pt_1[i];
}
#if PRINT_AT
wirelessPrint("A(t) = ");
printVec(At);
#endif
#if PRINT_AM
wirelessPrint("A(m) = ");
printVec(Am);
#endif
}

void AscalingStep()
{
float Ks[3] = {
0,0,0    };
for(char i = 0; i < 3; i++){
Au[i] = (Vt[i] - Vt_1[i]) / dt;
ea[i] = ((Am[i] + St[i]) - Au[i]);
St[i] = St[i] - (Ks[i] * ea[i]);
}
}

void  measuredAccelerationUpdate()  //Rotates
accelerometer data into global frame
{

Am[0] = (gravityReference[0] * DCM[0][0]
+ gravityReference[1] * DCM[1][0]
+ gravityReference[2] * DCM[2][0])*9.8;
Am[1] = (gravityReference[0] * DCM[0][1]
+ gravityReference[1] * DCM[1][1]
+ gravityReference[2] * DCM[2][1])*9.8;
Am[2] = (gravityReference[0] * DCM[0][2]
+ gravityReference[1] * DCM[1][2]
+ gravityReference[2] * DCM[2][2] - 1)*9.8;

#if PRINT_MEASUREDACCELERATION
wirelessPrint("Measured Acceleration = ");
printVec(Am);
#endif
}

void  rpmCalculator()   //Calculates  desired
motor speeds from control outputs
{
float L = 15.56, Ctau = .0209;

forces[0]  =  -u[0]/(4*L)  +  u[1]/(4*L)  +
u[2]/(4*Ctau) + u[3]/4; // F = C^-1 * U
forces[1]  =   u[0]/(4*L)  +  u[1]/(4*L)  -
u[2]/(4*Ctau) + u[3]/4;
forces[2]  =   u[0]/(4*L)  -  u[1]/(4*L)  +
u[2]/(4*Ctau) + u[3]/4;
forces[3]  =  -u[0]/(4*L)  -  u[1]/(4*L)  -
u[2]/(4*Ctau) + u[3]/4;


for (char i = 0; i < 4; i++){

motorSpeeds[i] = (54.988 * pow(forces[i],3)
- 662.66 * pow(forces[i],2)
+ 2963.7 * forces[i]); //Equation to convert
from thrust to RPM

motorSpeeds[i]  =  constrain(motorSpeeds[i],
0, RPMLIMIT);  //Limits to top speed

}
}


float  smooth(float  data,  float  filterVal,
float smoothedVal){
if  (filterVal > 1){       //  check  to  make
sure param's are within range
filterVal = .99;
}
else if (filterVal <= 0){
filterVal = 0;
}

smoothedVal  =  (data  *  (1  -  filterVal))  +
(smoothedVal  *  filterVal);

return smoothedVal;
}
```

```
String commandString;
int commandLength = 0;

void beginWireless()
{
#if TALKOVERUSB
Serial.begin(57600);
#endif
#if TALKOVERXBEE
Serial3.begin(57600);
#endif
}

void commandCheck()
{
commandString = "     ";

commandLength = 0;
#if TALKOVERUSB
while(Serial.available()>0){
commandString[commandLength]                  =
(char)Serial.read();
commandLength++;
}
#endif
#if TALKOVERXBEE
while(Serial3.available()>0){
commandString[commandLength]                  =
(char)Serial3.read();
commandLength++;
}
#endif

if (commandLength > 0){
wirelessPrint("Command received! = ");
wirelessPrintln(commandString);
if  (commandString[0]  ==  'f')  stateString  =
"FLYING";
if  (commandString[0]  ==  'r')  stateString  =
"READY";
if  (commandString[0]  ==  'k')  stateString  =
"KILL";
if  (commandString[0]  ==  'm')  stateString  =
"MOTORTEST";
if  (commandString[0]  ==  's')  stateString  =
"SPINUP1";
if  (commandString[0]  ==  'S')  stateString  =
"SPINUP2";
if  (commandString[0]  ==  'l')  stateString  =
"DROP";

if (commandString[0] == 'a') Pt_desired[2] =
(commandNumber() / 100);
if       (commandString[0]       ==       'R')
attitudeDesired[0] = commandNumber();
if       (commandString[0]       ==       'P')
attitudeDesired[1] = commandNumber();
if       (commandString[0]       ==       'Y')
attitudeDesired[2] = commandNumber();
if   (commandString[0]   ==   'g')   MG   =
commandNumber();
if       (commandString[0]       ==       'F')
resetMotorControllers();

if         (stateString.equals("MOTORTEST")){
//Begin commands for MOTORTEST
if(commandString[0] == '1') motorSpeeds[0] =
commandNumber();
if(commandString[0] == '2') motorSpeeds[1] =
commandNumber();
if(commandString[0] == '3') motorSpeeds[2] =
commandNumber();
if(commandString[0] == '4') motorSpeeds[3] =
commandNumber();
}
}
}

int commandNumber()
{
int result = 0;
for (char i = 1; i < commandLength; i++){
char *l;
*l = commandString[i];
result = (result * 10) + atoi(l);
}
wirelessPrint("commandNumber = ");
wirelessPrintln(result);
return result;
}


void wirelessPrint(float val)
{
#if TALKOVERUSB
Serial.print(val);
#endif
#if TALKOVERXBEE
Serial3.print(val);
#endif
}

void wirelessPrint(int val)
{
#if TALKOVERUSB
Serial.print(val);
#endif
#if TALKOVERXBEE
Serial3.print(val);
#endif
}

void wirelessPrint(long val)
{
#if TALKOVERUSB
Serial.print(val);
#endif
#if TALKOVERXBEE
Serial3.print(val);
#endif
}

void wirelessPrint(double val)
{
#if TALKOVERUSB
Serial.print(val);
#endif
#if TALKOVERXBEE
Serial3.print(val);
#endif
}

void wirelessPrint(String val)
{
#if TALKOVERUSB
Serial.print(val);
#endif
#if TALKOVERXBEE
Serial3.print(val);
#endif
}
```

```
void wirelessPrintln(float val)
{
#if TALKOVERUSB
Serial.println(val);
#endif
#if TALKOVERXBEE
Serial3.println(val);
#endif
}

void wirelessPrintln(int val)
{
#if TALKOVERUSB
Serial.println(val);
#endif
#if TALKOVERXBEE
Serial3.println(val);
#endif
}

void wirelessPrintln(long val)
{
#if TALKOVERUSB
Serial.println(val);
#endif
#if TALKOVERXBEE
Serial3.println(val);
#endif
}

void wirelessPrintln(double val)
{
#if TALKOVERUSB
Serial.println(val);
#endif
#if TALKOVERXBEE
Serial3.println(val);
#endif
}

void wirelessPrintln(String val)
{
#if TALKOVERUSB
Serial.println(val);
#endif
#if TALKOVERXBEE
Serial3.println(val);
#endif
}

void wirelessPrintln()
{
#if TALKOVERUSB
Serial.println();
#endif
#if TALKOVERXBEE
Serial3.println();
#endif
}

void writeSpeeds(){  //Sends motor speeds to
Arduino Minis

#if PRINT_MOTORSPEEDS
wirelessPrint("Speeds = ");
for(char i = 0; i < 4; i++){
wirelessPrint(motorSpeeds[i]);
wirelessPrint(", ");
}
wirelessPrintln();
#endif
//wirelessPrint("Sending speeds");

Wire.beginTransmission(2);
String valString = String(motorSpeeds[0]);
Wire.send((int)valString.length());
for(char i = 0; i < valString.length(); i++)
Wire.send(valString[i]);
valString = String(motorSpeeds[1]);
Wire.send((int)valString.length());
for(char i = 0; i < valString.length(); i++)
Wire.send(valString[i]);
Wire.endTransmission();

Wire.beginTransmission(3);
valString = String(motorSpeeds[2]);
Wire.send((int)valString.length());
for(char i = 0; i < valString.length(); i++)
Wire.send(valString[i]);
valString = String(motorSpeeds[3]);
Wire.send((int)valString.length());
for(char i = 0; i < valString.length(); i++)
Wire.send(valString[i]);
Wire.endTransmission();

wirelessPrintln("Sent speeds");
}

void   resetMotorControllers(){      //Sends
signal to reset the motor speed controllers
Wire.beginTransmission(2);
Wire.send('f');
Wire.endTransmission();
Wire.beginTransmission(3);
Wire.send('f');
Wire.endTransmission();
}
```

Arduino Pro Mini Software

```
#include <avr/interrupt.h>
#include <Servo.h>
#include <Wire.h>

#define COMMANDSFROMPILOT 1

#define I2CDEVICEID 2 //Controllor 1 = 2,
Controller 2 = 3
#define RPMCALCSTEPS 20   //The length of
time over which RPM is averaged
#define TIMESTEP 15 //is approximately
RPMCALCSTEPS * TIMESTEP

int motorPins[2] = {
10,11
};
int hallEffectPorts[2] = {
0,1
};
int pwmOffsets[2] = {
140,140
};
double a1offset = .0003; //.00015
double a1slope = .00000007; //.0000000566
double a1[2] = {
a1offset, a1offset
};
double a2offset = .24;
double a2slope = -.00003; //3
double a2[2] = {
a2offset, a2offset
};

volatile unsigned long   counter[2] = {
0,0};
unsigned long countertime = 1;
unsigned long oldcountertime = 0;
unsigned long receiveTime = 0;
double rpm[2] = {
0,0};
double oldrpm[2] = {
0,0};
int rpmData[2][RPMCALCSTEPS];
char rpmCalcIndex = 0;
float motorVals[2];
int x;
char p, *l;
boolean fault = false;

int desiredRPM[2];
long error[2], oldError[2];
float dt = 1;

void setup() {
for(char i = 0; i < 2; i++){
pinMode(hallEffectPorts[i], INPUT);
pinMode(motorPins[i], OUTPUT);
for(char j = 0; j < RPMCALCSTEPS; j++)
rpmData[i][j] = 0;
motorVals[i] = pwmOffsets[i];
volatile unsigned long counter[] = {
0,0       };
}
attachInterrupt(hallEffectPorts[0],
counter0Inc, CHANGE);
attachInterrupt(hallEffectPorts[1],
counter1Inc, CHANGE);
Serial.begin(57600);
Wire.begin(I2CDEVICEID);
Wire.onReceive(received);

float motorVals[] = {
pwmOffsets[0], pwmOffsets[1]        };
}

void loop() {
countertime = millis();
if (countertime > oldcountertime +
TIMESTEP){
if(Serial.available()>0){
desiredRPM[0] = 0;
while(Serial.available()>0)
{
p = Serial.read();
*l = p;
desiredRPM[0] = desiredRPM[0] * 10 +
atoi(l);

}
if(desiredRPM[0] > 7000) desiredRPM[0] =
7000;
desiredRPM[1] = desiredRPM[0];
}

if(desiredRPM[0] == 0){
motorVals[0] = pwmOffsets[0];
motorVals[1] = pwmOffsets[1];
}

#if COMMANDSFROMPILOT
if (countertime > receiveTime + 250){
fault = true;
Serial.println("Communication fault!");
}
#endif

rpmCalc();

for(char i = 0; i < 2; i++){
//if((oldrpm[i] > 0) && (rpm[i] == 0) &&
(desiredRPM[i] != 0)) fault = true;
}

errorUpdate();
gainUpdate();
pidUpdate();

if(fault) int motorVals[] = {
0,0           };

analogWrite(motorPins[0],
(int)motorVals[0]);
analogWrite(motorPins[1],
(int)motorVals[1]);

Serial.print("RPM0 = ");
Serial.print(rpm[0]);
Serial.print(", RPM1 = ");
Serial.print(rpm[1]);
//  Serial.print(", Motor0Val = ");
//  Serial.print(motorVals[0]);
//  Serial.print(", Motor1Val = ");
//  Serial.print(motorVals[1]);
Serial.print(", desiredRPM0 = ");
Serial.print(desiredRPM[0]);
Serial.print(", desiredRPM1 = ");
Serial.println(desiredRPM[1]);

oldcountertime = countertime;
}
}


void pidUpdate()
{
dt = (countertime - oldcountertime);
for(char i = 0; i < 2; i++){

float increment = a1[i] * error[i] - a2[i] *
(rpm[i] - oldrpm[i]) / dt;
motorVals[i] += increment;
motorVals[i]    =    constrain(motorVals[i],
pwmOffsets[i], 254);
}
}

void counter0Inc()
{
counter[0]++;
}

void counter1Inc()
{
counter[1]++;
}

void errorUpdate()
{
for (char i = 0; i < 2; i++){
oldError[i] = error[i];
error[i] = desiredRPM[i] - rpm[i];
}
}

void gainUpdate()
{
for(char i = 0; i < 2; i++){
a1[i] = a1offset + a1slope * rpm[i];
a2[i] = a2offset + a2slope * rpm[i];
if(a2[i] < 0) a2[i] = 0;

if(abs(error[i]) < 100){
a1[i] = 0;
//a2[i] = 0;
}
}
}
void received(int numBytes){
desiredRPM[0] = 0;
desiredRPM[1] = 0;

char nextByte = Wire.read();
byte valSize = nextByte;
if (nextByte == 'f') fault = false;
else{
for(char i = 0; i < valSize; i++){
nextByte = (char) Wire.read();
desiredRPM[0] = desiredRPM[0] * 10;
desiredRPM[0] += atoi(&nextByte);
}

nextByte = Wire.read();
valSize = nextByte;
for(char i = 0; i < valSize; i++){
char nextByte = (char) Wire.read();
desiredRPM[1] = desiredRPM[1] * 10;
desiredRPM[1] += atoi(&nextByte);
}
}


receiveTime = millis();
}
void rpmCalc()
{
float dt = (countertime - oldcountertime);
for(char j = 0; j < 2; j++){
oldrpm[j] = rpm[j];
float newrpm = ((60000 * counter[j] / dt) /
2);
rpm[j]    =    newrpm    /    RPMCALCSTEPS    +
(RPMCALCSTEPS    -    1)    *    oldrpm[j]    /
RPMCALCSTEPS;
counter[j] = 0;
}


rpmCalcIndex++;
if(rpmCalcIndex       ==       RPMCALCSTEPS)
rpmCalcIndex = 0;
}
```