2013

Design, Fabrication, and Testing of a Hopper Spacecraft Simulator

Evan Phillip Krell Mucasey
Lehigh University

Design, Fabrication, and Testing of a Hopper Spacecraft Simulator

by

Evan Phillip Krell Mucasey

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering

Lehigh University

May 2013

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

_____

Date

_____

Thesis Advisor

_____

Chairperson of Department

I would like to first thank Dr. Terry Hart, who originally gave me the opportunity and inspiration to pursue this project for a Master's Thesis. His continual guidance helped me clear the developmental obstacles that would have road-blocked along the way.

Next, I would like to thank Dr. Joachim Grenestedt, without whom I would never have developed the level of engineering and 'first-principle' though required to achieve the goals of this project.

I would like to thank Anthony Dzaba and Andrew Abraham for help with the initial development of the first generation vehicle. Also, thank you to Andrew Papazian and Nick Tashjian for helping to develop the thrust test bench, and with whose help around the shop, with the addition of Amos Ambler, allowed for the completion of fabrication of the second generation vehicle in time for flight testing. Thank you also to Melissa Dye who helped develop the software associated with the real time data acquisition system and aided during flight testing.

Lastly, I would like to thank my family for giving me the opportunity to pursue my passion for an education and a career. Without their love and support, I would never have gotten the necessary foundation that has allowed me to get to this point in my life.

Contents

List of Tables

List of Figures

ABSTRACT

A robust test bed is needed to facilitate future development of guidance, navigation, and control software for future vehicles capable of vertical takeoff and landings. Specifically, this work aims to develop both a hardware and software simulator that can be used for future flight software development for extraplanetary vehicles. To achieve the program requirements of a high thrust to weight ratio with large payload capability, the vehicle is designed to have a novel combination of electric motors and a micro jet engine is used to act as the propulsion elements.

The spacecraft simulator underwent several iterations of hardware development using different materials and fabrication methods. The final design used a combination of carbon fiber and fiberglass that was cured under vaccuum to serve as the frame of the vehicle which provided a strong, lightweight platform for all flight components and future payloads.

The vehicle also uses an open source software development platform, Arduino, to serve as the initial flight computer and has onboard accelerometers, gyroscopes, and magnetometers to sense the vehicles attitude. To prevent instability due to noise, a polynomial kalman filter was designed and this fed the sensed angles and rates into a robust attitude controller which autonomously control the vehicle' s yaw, pitch, and roll angles.

In addition to the hardware development of the vehicle itself, both a software simulation and a real time data acquisition interface was written in MATLAB/SIMULINK so that real flight data could be taken and then correlated to the simulation to prove the accuracy of the analytical model.

In result, the full scale vehicle was designed and flown outside of the lab environment and data showed that the software model accurately predicted the flight dynamics of the vehicle.

1 Background

Space exploration has been a possibility for the last fifty years. It has helped boost the state of technology, expanded the understanding of science, and provided the framework for the inspiration to pursue challenging goals. It still remains a relevent topic, as spacecraft are travelling further from the earth in the hope of discovering new concepts.

Currently, the state of the art design of robotic planetary vehicles consists of a rover type of vehicle that is constrained to travel on the ground.

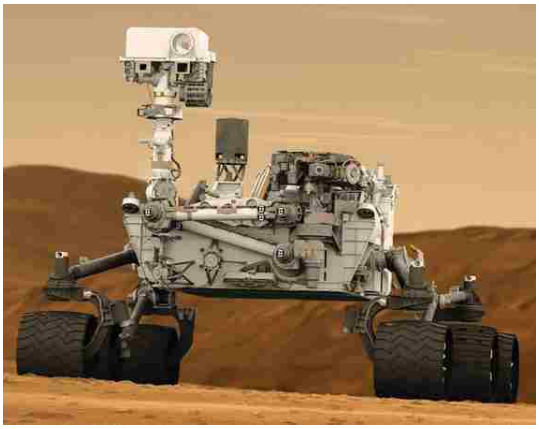

Figure 1: Mars Curiousity Rover

This type of design necesitates travel on the ground which is inherently constraining. Mission design consists of specific locations on the planet that allow the rover to traverse the ground, free of obstructions such as boulders, mountains, or craters. The mission usually does not leave a designated zone alocated to experiments.

The work contained within this thesis designs, fabricates, and tests a testbed for an innovative design of a planetary lander. Instead of conventional rovers, this new type of 'hopper' spacecraft will be able to re-ignite its engines once on the planet to then hop to different locations. If proven robust, this design would open up future mission architectures to explore many different locations on the planet, and can even be combined with conventional designs to accomplish mission goals.

The idea for a spacecraft re-igniting its engines once landed on a planetary body is not a new one. NASA designed a mission to study how to safely land on the moon prior to the Apollo program's success. One of these missions, Surveyor 6 in 1967, performed the only 'hop' in space history. The hop consisted of a 2.5 m movement from an original position.

Figure 2: Surveyor 6

As this type of design also opens up further risk in exploration, the design of a robust testbed is necessary to learn the constraints involved in the guidance, navigation, and control of a hopping spacecraft. This testbed could also be used to learn how to interface with other flight components including flight computers and inertial navigation systems as well as how to design the software associated with a particular mission, especially when combined with an analytical model and software simulation.

The purpose of this thesis work is to provide the capability of testing these types of systems at Lehigh University and produce an initial solution to the control problem so that flight hardware as well as guidance and navigation algorithms can be tested in the future.

1.1 Requirements Definition

Lehigh University is currently working with Penn State towards developing a vehicle to compete in the Google Lunar X-Prize. This is a competition that awards a prize to the first team to launch a spacecraft to the moon, land, then travel 500 meters on the surface while transmitting back data. To facilitate development, Lehigh's spacecraft simulator will be designed

with similar geometry to the Penn State Lunar Lion vehicle.

The requirements for the simulator are then:

1. Minimum thrust to weight ratio of 3:1
2. 5 kg payload capability while maintaining the minimum thrust to weight ratio
3. 3 actuator locations providing attitude and position control for the vehicle
4. Real time data acquisition capability for off line data post-processing
5. Low cost components

There is no known low cost solution that satisfies each of these requirements, so custom hardware and software will need to be developed to provide each of these capabilities into a single, robust system.

2 Structure Design and Evolution

An important consideration for the design of the hopper spacecraft simulator was to maintain the low-cost requirement. In addition, Lehigh University does not have the facilities to support flight propulsion options. These two facts led to the decision to design a vehicle that resembles a quadrotor. Quadrotors are very common types of vertical take off and landing vehicles that are used in industries from law enforcement to aerial photography. The concept is scalable such that there exist many variations in size as well as geometry.

2.1 Quadrotor Operation

The classical quadrotor uses differential thrust from four actuators to control its attitude and position. The actuators are usually electric motors with a rigid propeller attached to each shaft. The pitch of the propellers is fixed, so the angular velocity of the motors are modulated to achieve different thrust levels. The motors are usually arranged in a 'plus' configuration.

Figure 3: Classical Quadrotor Design

A common way to describe the vehicle's attitude in space is to define roll, pitch, and yaw angles about the vehicle's axes. These angles are controlled by the differential thrust levels of the motors, and consequently, the change in pitch and roll causes the vehicle to translate in position.

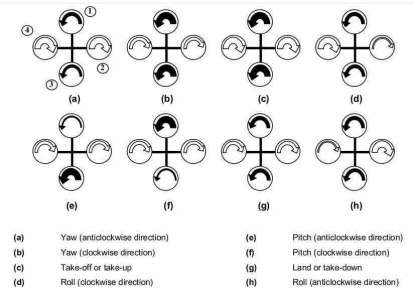

Figure 4: Quadrotor Attitude Control

2.2 First Generation

Prior to this thesis work, an initial concept for the spacecraft hopper simulator had been fabricated. As an investment had already been made into the design and construction of this vehicle, initial testing and development of the flight control software was performed on this first generation.

While this design used the classic quadrotor geometry, it was novel in that each actuator could tilt with respect to the vehicle body. This gave each actuator an extra degree of freedom and allowed for the de-coupling of position and attitude. This means that the vehicle could control its position simply by tilting the actuators and not the vehicle body itself.

Figure 5: First Generation Hopper Spacecraft Simulator CAD Drawing

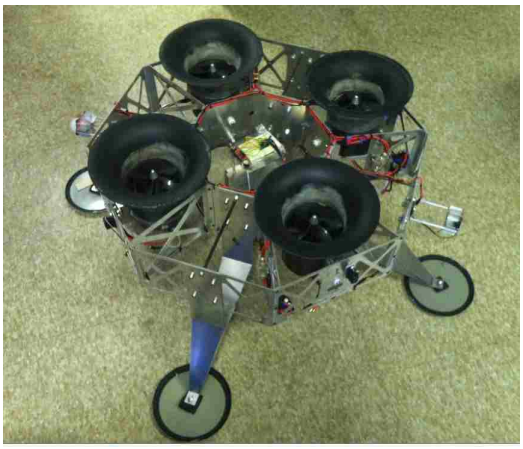

Figure 6: First Generation Hopper Spacecraft Simulator

The vehicle uses four ducted fans that produce approximately 4.5 kg of thrust each. Associated with each fan is a 37 volt lithium-polymer battery. These four batteries are wired in parallel to give the vehicle more energy for longer flight times. The chassis consists of two concentric welded aluminum hexagons that housed all flight components and landing gear by mounting to its inner and outer walls.

2.2.1 Thrust to Weight Ratio

While the desired thrust to weight ratio of the spacecraft simulator is 3:1, the first generation simulator was designed to achieve a ratio 1.25:1, as the vehicle investigated using thrust vector control instead of attitude changes to translate. This produces a challange for the control software, as saturation in the actuators becomes an issue. Saturation occurs when the actuators are at the limit of its thrust capabilities, and if the vehicle needs almost all of its thrust to counteract gravity, there is very little margin left for rejecting disturbances and controlling the vehicle.

In order to meet the required thrust to weight ratio, the rest of the vehicle's mass would have to be significantly reduced. The mass of the propulsion system including motors, propellers, and batteries is 8.44 kg with a total thrust capability of 19.5 kg. Based off these values, it can be seen that the ratio of 3:1 is impossible to achieve, as the total vehicle mass would have to be only 6.5 kg. Further, the design of the aluminum frame is inadequate for meeting these

requirements. The load paths required high mass for adequate strength and stiffness.

2.2.2 Propulsion System

The most significant drawback of the first generation was the propulsion system. The ducted fans and batteries had low reliability. Constant maintanence was required in order to keep these fans in working order, and in addition, several improvements were also made including:

1. Adding an intake cowling
2. Re-design of the cowling mounting
3. Machine a groove in the motor shaft as a fail-safe for set screw friction failure
4. Added fillets to the carbon fiber mounts to lower stresses
5. Replacing all set screws that kept the propeller on the motor shaft
6. Moved the batteries to the center of the vehicle to reduce the total vehicle inertia

During operation the motor experiences very high-frequency, low-amplitude vibration. This leads to several problems. First, the hardware the fan uses to stay together eventually loosens. Repeated tightening of the set screws and DC motor mounting bolts is required. There are very tight distance tolerances between the propeller and the fan body so this operation takes a significant amount of time to perform. This has also led to significant damage to the inner carbon fiber skin of the fan body and also damage to the propeller blade tips.

These powerful electric fans have a very large power requirement. At full thrust, the system is capable of drawing 12.8kW of power. Due to the poor thrust to weight ratio, this power draw occurs regularly. The large on-board lithium polymer batteries used to achieve the power requirements have a mass of approximately 4.2 kg total and produce very high currents. On one occasion, ohmic heating caused a solder joint to melt and all engines lost power in the middle of a test. Also, if mishandled, the leads to the batteries are capable of providing a strong enough arc current to weld aluminum and steel; the aluminum frame shows the result of several accidental shorts.

2.2.3 Testing

Testing and implementation of a controller has also proven difficult because of the propulsion system. With the current laboratory setup, the system is limited to short duration tests because of the battery life. Also, the very long battery charge time makes it impossible to perform several tests in a single day. This makes tuning the controller very difficult, and progress during testing is consistently delayed due to the loss of battery power. To facilitate our controller tuning, a system requires either a capability for long duration testing, or the ability to quickly re-gain flight capabilities. Our experience indicates that one of these two options is absolutely critical.

In addition, the large vehicle geometry and mass necessitated a test stand to constrain the position while the software for the attitude controller was tuned. Unfortunately, it was difficult to mount the vehicle at its center of mass so the test stand introduced unwanted dynamics into the attitude controller. This made it difficult to correlated balanced flight on the test stand to balanced free flight.

The combination of the issues listed above prevented furthering development on the first generation simulator. It was decided to use the lessons learned to start over for the next generation of simulator.

2.3 Transition to Next Generation

To design the next generation vehicle, the requirements outlined in section **1** were re-visited in order to ensure the system designed could satisfy each. Several high level changes could be immediatly implemented for the future vehicle.

2.3.1 Frame Design

The chassis of the next generation would be fabricated from composite materials. This a good option when considering strength and stiffness to weight ratios. In addition, Lehigh University has the infrastructure for and a background in working with these materials. Coupled with better load path design, a much more efficient frame could be designed and fabricated. Also,

these materials are very easy to repair so the time in between test flights and free flight crashes of the vehicle would be drastically reduced.

2.3.2 Propulsion Selection

A better propulsion system could be chosen for the application. A trade study was conducted in which several options were considered including; the current electric ducted fan technology, lighter than air systems, solid fuels, hobby grade motors and propellers, and hobbyist micro jet turbines. Due to the inefficiencies of lighter than air systems and the dangerous handling requirements of solid fuels, both were excluded from further analysis.

The detailed trade study for different propulsion options can be found in section **A.** Based on the requirements of a 3:1 thrust to weight ratio and a 4-5 kg payload capability, two leading candidates for propulsion were found to be hobby grade motors and propellers and the hobbyist micro jet turbines.

There is a very distinct tradeoff between these two options. The motors and propellers have a very quick time response to inputs. This is an important consideration in a control system, as time lag adversely affects the stability of the system. However, the total thrust from this technology is much lower than that of the turbines. The first generation confronted the issues with scaling this type of technology. Micro turbines have the opposite effects: high thrust level but a slow time response.

The decision was made to use a combination of these two systems to achieve the above requirements. The motors and propellers would provide the fast time response required for the control system and a micro turbine would provide enough thrust to reach the desired accelerations.

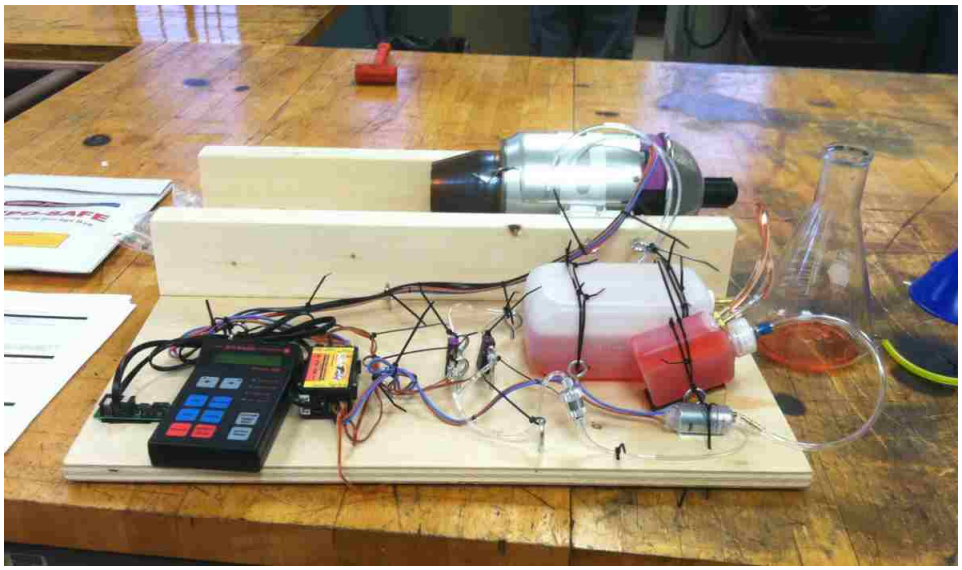

Figure 7: JetCat P200 Jet Turbine Test Fixture

2.4 Proof of Concept Vehicle

While many important considerations were found by working with the first generation simulator, there were still many unknowns of how to design a VTOL vehicle from the ground up. Many of these pertained to the flight control software, as it was very difficult to test different control algorithms and gains with the short flight times and unknown test stand dynamics.

Before spending resources to build the full scale second generation vehicle, a proof of concept vehicle was designed and built. The reason for this was that the proof of concept enabled the ability to test all steps in the design process as would be accomplished on the full scale version:

1. Structure design and construction techniques
2. Sensor selection
3. Actuator selection and characterization
4. Filtering techniques
5. Controller and electrical interfaces
6. Communication and data logging

7. Testing techniques

This proved to be invaluable in the success of the project.

A simple frame was constructed from fiberglass, epoxy and foam for a core material. The fiberglass was brought to single skin so that the motors can be direclty bolted to the frame. Carbon stiffeners were added to the single skin areas of the frame.

Figure 8: Proof of Concept Frame Construction

Figure 9: Carbon Stiffeners and Completed Frame

The vehicle was then populated with cheap components and outfitted so that development could commence on the flight control software.

Figure 10: Populated Proof of Concept Vehicle

2.5 Second Generation Simulator

While software development took place on the proof of concept vehicle, parallel design and fabrication was accomplished on the second generation spacecraft simulator. To better keep track of all system components and the mass of the vehicle, a master equipment list was constructed. This allowed for fast high level iterations of components and allocations of mass to ensure the mass budget was never exceeded.

Evan Mucasey				
Lehigh University				
HexaHopper V2				
Master Equipment List				
		Mass (kg)		Quantity
AXI 2826/12 Motors		0.18	1.08	6
Castle 50A ESC		0.022	0.132	6
5Cell 2700 mAh Battery		0.3	0.9	3
VN-100 Rugged AHRS		0.015	0.015	1
Arduino Mega		0.03	0.03	1
PCB for Connections		0.045	0.045	1
Structure		1.5	1.5	1
Wires and Connectors		0.3	0.3	1
JetCat P200		2.8	2.8	1
Fuel Mass		0.75	0.75	1
Subtotal No Jet			4.002	
Subtotal Yes Jet			7.552	
PAYLOAD			5	
TOTALS			**12.552**	
Thrust to Weight Ratio			**3.11881777**	
Thrust to Weight JetCat Out Full Payload			**1.24282983**	
Thrust to Weight No Payload No JetCat			**3.89805097**	

Table 1: Master Equipment List

Three scenarios were considered to ensure the thrust-to-weight ratio satisfied the requirements. First in a nominal configuration, with motors and propellers, the turbine are attached to the vehicle, as well as 5 kg of payload, it can be seen that the thrust to weight ratio is 3.12:1. Next, leaving out the turbine with associated components, and payload, the thrust to weight

ratio is 3.90:1. These two nominal configurations cover all regimes of flight. If required, more fuel can be addeded or larger batteries can be swapped to increase flight time, but decrease the thrust to weight ratio.

However, if the vehicle experiences a turbine failure with 5 kg of payload and full fuel, the system was designed to enter into a fail-safe mode where the thrust to weight ratio is not nominal, but is enough to safely land the vehicle without further damage. This thrust to weight ratio is 1.24:1, similar to that of the first generation simulator.

The frame was designed so that all flight components for stabilization and control could be mounted on the underside of the vehicle. This reserves surface area on the top for future payloads. In addition, the central circle has a circular pattern of six bolt holes to eventually house the turbine at the geometric center of the vehicle. The highest mass components will eventually be the payload, so the frame was designed to support the loading of the 5 kg distributed about the top surface.

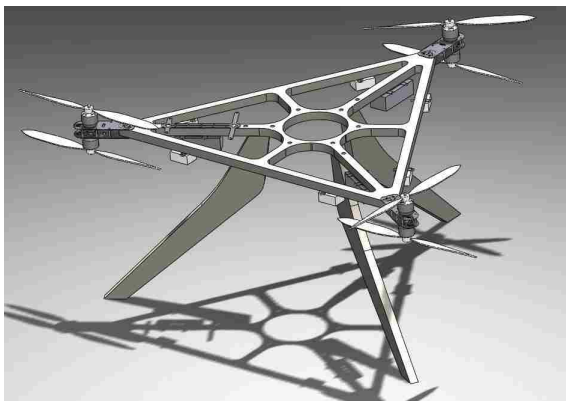

Figure 11: Second Generation Simulator CAD Drawing

Similar methods of fabrication as the proof of concept vehicle were used to construct the frame. Initially, A large triangle was rough cut from a rigid PVC foam called Divinycell H80. This core material was chosen for its excellent of shock and impact resistance as well as low water absorption and high compressive and shear strength.

Next, unidirectinal carbon and fiberglass tape was wet-layed with MGS epoxy while the same

epoxy was mixed with micro glass spheres and brushed onto the foam. This was to ensure that surface pores were filled so that the carbon would bond to the maximum surface area possible. Finally, a vacuum was pulled over the entire part until fully cured to ensure any extra epoxy was removed and the wet fibers stayed adhered to the core throughout the cure process. The fully cured part was then cut on a waterjet to remove extra material.

Figure 12: Vacuum Cure of Frame

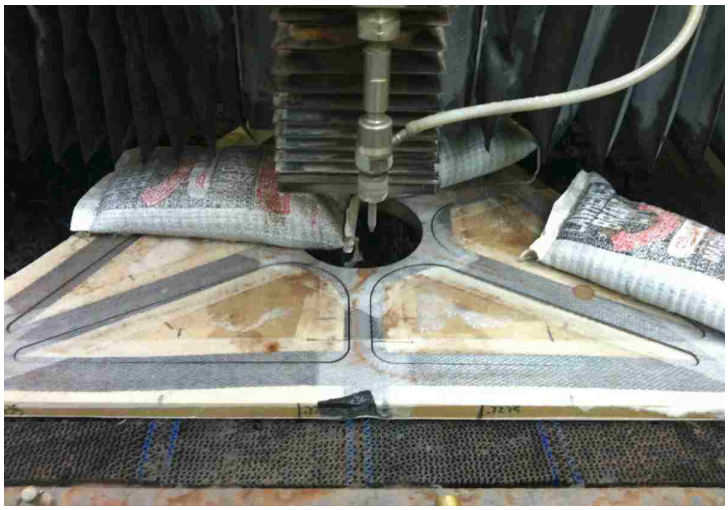

Figure 13: Frame Cut on Water-Jet

Aluminum tabs to house the motors were cut on the waterjet then bonded onto the frame. These pairs of plates were then bolted together with aluminum standoffs.

Figure 14: Motor Tab Bonding

Finally, the top of the frame could be populated with flight componenets and wires.

Figure 15: Populated Frame

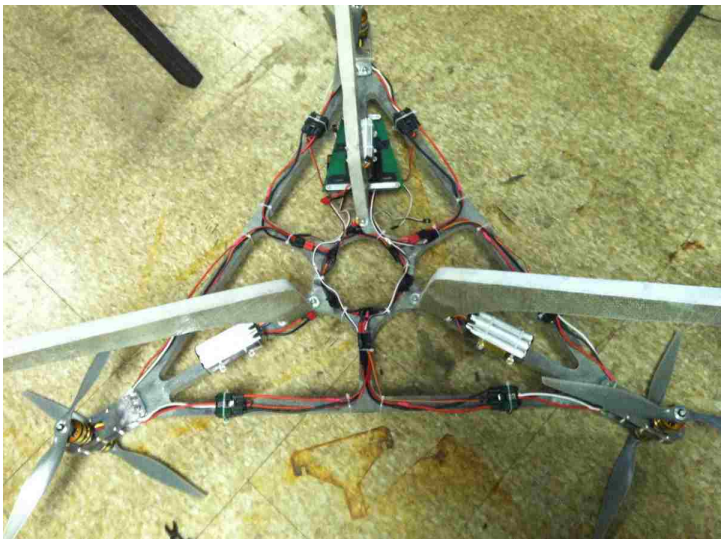

Figure 16: Populated Frame Bottom

It can be seen that the mass at this point in the construction process is approximatly 2.5 kg. According to the MEL in table 1, there is 3.102 kg alloted to this flight configuration. This leaves an extra 0.6 kg for landing legs and extra electrical connectors. The mass of the vehicle remained under-budget during construction.

Figure 17: Final V2 Hopper Spacecraft Simulator

2.6 Flight Testing and Test Stands

It was found that the design of the test stand is critical to ensuring the gains found through testing also correlate to free flight stability. Essentially, the purpose of the test stand was to simulate free flight as close as possible, and not introduce unwanted dynamics while testing the controller.

Initially, a three DOF test stand was constructed to assist in the testing of the first generation simulator. The mass of the first generation simulator, 18 kg, necessitated a strong, rigid platform for constraining the position of the vehicle while the attitude was constrolled. The 3 DOF's were achived through the use of a universal joint mounted on top of a shaft that was housed within two bearings. The structure itself was designed to be heavy enough that the vehicle's accelerations would not be able to move the stand.

Figure 18: 3-DOF Test Stand

Figure 19: Universal Joint on Shaft

Also included in the design of the test stand was the ability to constrain any of the three of the DOF's such that a single axis on the vehicle could be isolated.

As the mass of the universal joint became a significant percentage of the proof of concept vehicle as well as the second generation simulator, this test stand could not be used to test the flight control software. For the POC simulator, different methods mounting the vehicle to a test stand were attempted.

Figure 20: Quadrotor Cage

Ultimately, the best option was to drill a large hole in the geometric center of the vehicle and slide this over a smaller diameter shaft. This allowed the vehicle to hover and even stabilize in place, while constraining its position without the use of a joint.

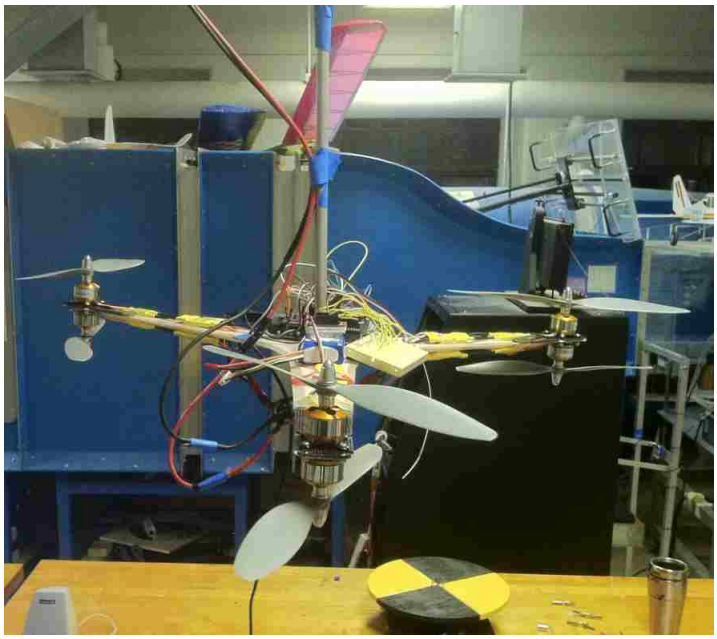

Figure 21: Proof of Concept Vehicle Pole Testing

The second generation simulator used a similar, albeit crued, concept, but the test stand served its purpose perfectly. The design of this test stand, coupled with the development of the software simulations and proof of concept vehicle, allowed for the successful test flight of

the second generation vehicle less than 24 hours after construction completed.

3 Equations of Motion

The first step to controlling a flying system is understanding the dynamics. For this, we can establish two coordinate frames. The first frame is an inertial frame of reference and will be called the planetary frame. This remains fixed in space and provides the reference for vehicle position information. The second frame is a non-inertial reference frame that is attached to the vehicle as it translates and rotates within the inertial reference frame.

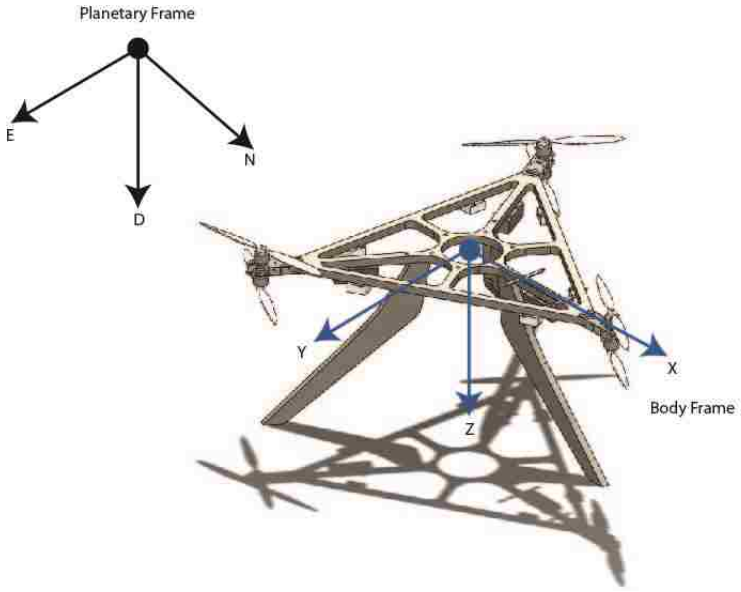

Figure 22: Planetary and Body Reference Frames

The sequence of rotations about different axes are called euler angles. It is important to note that these angles are not always about the planetary frame's axes. In fact, only the first rotation is about the planetary frame's axis and then each subsequent rotation is about an axis in a newly created coordinate frame. Hence, the sequence of rotation is unique in that a different sequence will produce a different description of attitude. As can be seen from figure **22** we have defined these rotation angles as:

$$\phi \triangleq roll$$
$$\theta \triangleq pitch$$
$$\psi \triangleq yaw$$

We will need to be specific throughout the derivation when we refer to the reference frame in

which the action is taking place. For our system, it will eventually be convenient to describe the translations in the planetary frame, while we describe the rotations in the body frame. While this may seem inconsistent, this is done because of how our sensors operate. Our attitude control system has sensors that read body frame information, the euler angles, and the controller describes desired body frame torques from this information. Meanwhile, our position information will come from a GPS which is capable of giving planetary information. This combination separates the attitude controller from the position dynamics, which eases computation on board and is easier to visualize.

To derive the equations of motion for our system, we will use the well known Newton-Euler equations as in [9, 3, 2]. These are used to describe the motion of a rigid body that is both translating and rotating. We will begin by formulating the dynamics in the body frame.

$$\begin{bmatrix} F^B \\ T^B \end{bmatrix} = \begin{bmatrix} mI & 0 \\ 0 & I_v \end{bmatrix} \begin{bmatrix} \ddot{\zeta}^B \\ \ddot{\eta}^B \end{bmatrix} + \begin{bmatrix} \dot{\eta}^B \times m\dot{\zeta}^B \\ \dot{\eta}^B \times I_v\dot{\eta}^B \end{bmatrix} \tag{1}$$

The superscript B in (1) shows the representation in the body frame. For the derivation, we will assume that the origin of the body frame resides at the center of mass of the vehicle, and the body frame axes are aligned with the principle axes of inertia. Also, with:

$$\zeta^B = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$$

$$\eta^B = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$$

$$m = vehicle\, mass$$

$$I_v = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

$$F^B = F_A^B - G^B = \begin{bmatrix} mgs(\theta) & -mgc(\theta)s(\phi) & -mgc(\theta)s(\phi) + F_Z \end{bmatrix}^T \tag{2}$$

$$T^B = \begin{bmatrix} T_\phi & T_\theta & T_\psi \end{bmatrix}^T$$

Our actuators are fixed to the body, so they are only capable of providing a force, F_A^B , in the Z body frame direction. In addition, the contribution of the gravitational vector in the body frame is:

$$G^B = R^{-1}G^P$$

Where R is the rotation matrix from the planetary frame to the body frame with $c(\epsilon) = cos(\epsilon)$ and $s(\epsilon) = sin(\epsilon)$

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

The addition of the cross products in the newton-euler formulation arise from the fact that our body frame is accelerating with respect to the planetary frame, and contains the coriolis and centripetal effects of the dynamics. See appendix **B** to see a detailed derivation of the Newton-Euler equation as well as the rotation matrix.

$$\dot{\eta}^B \times m\dot{\zeta}^B = m \begin{bmatrix} \dot{\theta}\dot{Z} - \dot{\psi}\dot{Y} \\ \dot{\psi}\dot{X} - \dot{\phi}\dot{Z} \\ \dot{\phi}\dot{Y} - \dot{\theta}\dot{X} \end{bmatrix} \tag{3}$$

$$\dot{\eta}^B \times I_v\dot{\eta}^B = \begin{bmatrix} \dot{\theta}\dot{\psi}(I_{ZZ} - I_{YY}) \\ \dot{\psi}\dot{\phi}(I_{XX} - I_{ZZ}) \\ \dot{\phi}\dot{\theta}(I_{YY} - I_{XX}) \end{bmatrix} \tag{4}$$

Hence when (3) and (4) are substituted into (1) with the previous know values, the following system of equations for the body frame:

$$
\begin{aligned}
\ddot{X} &= \left(\dot{\psi}\dot{Y} - \dot{\theta}\dot{Z}\right) + gs\left(\theta\right) \\
\ddot{Y} &= \left(\dot{\phi}\dot{Z} - \dot{\psi}\dot{X}\right) - gc\left(\theta\right)s\left(\phi\right) \\
\ddot{Z} &= \left(\dot{\theta}\dot{X} - \dot{\phi}\dot{Y}\right) - gc\left(\theta\right)s\left(\phi\right) + \tfrac{F_Z}{m} \\
\ddot{\phi} &= \tfrac{\dot{\theta}\dot{\psi}(I_{YY} - I_{ZZ})}{I_{XX}} + \tfrac{T_\phi}{I_{XX}} \\
\ddot{\theta} &= \tfrac{\dot{\psi}\dot{\phi}(I_{ZZ} - I_{XX})}{I_{YY}} + \tfrac{T_\theta}{I_{YY}} \\
\ddot{\psi} &= \tfrac{\dot{\phi}\dot{\theta}(I_{XX} - I_{YY})}{I_{ZZ}} + \tfrac{T_\psi}{I_{ZZ}}
\end{aligned}
\tag{5}
$$

We can see that in (5), our angular accelerations are linearly related to T_ϕ, T_θ, and T_ψ. These are the torques that are defined by our attitude controller. This is the reason it is convenient to define the angular accelerations in the body frame, as we see the direct effect from our control torques.

However, when looking at the body fixed translational accelerations in (5) , we see that our control force, F_Z is only apparent in $\ddot{Z}$. We do see the coupling effect due to $\dot{Z}$, $\dot{\phi}$, and $\dot{\theta}$ in $\ddot{X}$ and $\ddot{Y}$. This shows that it is still possible to achieve accelerations in the X and Y directions, but in an indirect fashion from F_Z because of the definition of forces in (2). This gives rise to the idea of formulating the position dynamics in such a way that our control force $F_Z$directly affects each of the accelerations. This is accomplished if we resolve the body frame actuator forces into the planetary frame by:

$$
F^P = RF_A^B - G^P = R\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = \begin{bmatrix} \left(c\left(\phi\right)s\left(\theta\right)c\left(\psi\right) + s\left(\phi\right)s\left(\psi\right)\right)F_z \\ \left(c\left(\phi\right)s\left(\theta\right)s\left(\psi\right) - s\left(\phi\right)c\left(\psi\right)\right)F_z \\ \left(c\left(\phi\right)c\left(\theta\right)\right)F_z - mg \end{bmatrix} \tag{6}
$$

Then, since our planetary fixed frame is inertial, it does not experience any coriolis or centripital effects so the contribution from the cross product is 0. This then leads to the following

revised Newton-Euler equations of motion:

$$\begin{bmatrix} F^P \\ T^B \end{bmatrix} = \begin{bmatrix} mI & 0 \\ 0 & I_v \end{bmatrix} \begin{bmatrix} \ddot{\zeta}^P \\ \ddot{\eta}^B \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\eta}^B \times I_v \dot{\eta}^B \end{bmatrix} \tag{7}$$

Or written as a system of equations when (6) in combination with (4) is placed in (7),

$$\begin{aligned} \ddot{N} &= \left[c\left(\phi\right) s\left(\theta\right) c\left(\psi\right) + s\left(\phi\right) s\left(\psi\right)\right] \tfrac{F_Z}{m} \\ \ddot{E} &= \left[c\left(\phi\right) s\left(\theta\right) s\left(\psi\right) - s\left(\phi\right) c\left(\psi\right)\right] \tfrac{F_Z}{m} \\ \ddot{D} &= -g + \left[c\left(\phi\right) c\left(\theta\right)\right] \tfrac{F_Z}{m} \\ \ddot{\phi} &= \tfrac{\dot{\theta}\dot{\psi}(I_{YY} - I_{ZZ})}{I_{XX}} + \tfrac{T_\phi}{I_{XX}} \\ \ddot{\theta} &= \tfrac{\dot{\psi}\dot{\phi}(I_{ZZ} - I_{XX})}{I_{YY}} + \tfrac{T_\theta}{I_{YY}} \\ \ddot{\psi} &= \tfrac{\dot{\phi}\dot{\theta}(I_{XX} - I_{YY})}{I_{ZZ}} + \tfrac{T_\psi}{I_{ZZ}} \end{aligned} \tag{8}$$

By deriving the equations of motion in this combined frame of reference, we have allowed for the design and simulation of the attitude controller that is independent of the position dynamics, and we also have provided a convenient way of showing the position dynamics in the inertial frame with a direct effect from $F_A^B$and the coupling of the vehicle attitude.

4 Controller

Now that it is understood how the vehicle moves in its respective reference frames, we can begin to design a controller that stabilizes the system in each frame. At first, open loop simulations will show the double integrator plant is unstable in attitude and then a proposed controller will be shown to stabilize the system in a closed loop. A position controller will then be designed in simulation.

The first step in this system is to ensure that there is a robust attitude control system. Because our system's attitude is coupled to position, as can be seen from the equations of motion, we must roll, pitch, or yaw in order to get to a different position. To do this, the position controller will define a specific attitude orientation to which the attitude control system must quickly and precicely re-orient itself. Thus, the first step for control is ensuring a robust attitude control system.

When designing a controller for this type of application, it is important to look at what sensors are at the disposal of the system itself. Usually, the more sensors, the more controllable the system, as the more layers of control can be applied. In our case, we will have the capability of sensing angles and rates in all three axes. This means that we will have the ability to sense $\phi_R, \dot{\phi}_R, \theta_R, \dot{\theta}_R, \varphi_R, \dot{\varphi}_R$. The subscript 'R' is used to denote that these are the raw values for each angle and angle rate. We will later see that we can substantially reduce mechanical and electrical noise by filtering each of these states with a polynomial Kalman filter.

There are many types of controllers that have been shown to work for double integrator type plants. It was shown in [1] that a model independent PD controller is a suitale, robust replacement for more complicated non-linear controllers. Hence, the controller that will be used on board the vehicle will be a variation of a PD controller. In [5], a PD controller was developed for attitude control that showed good stability characteristics for quadrotors, but the simulations did not include time delays in the analytical model. A step further can be seen in [4] where a cascaded control law is used.

A cascaded control law consists of two control loops, both containing a specific setpoint. The

outer, master loop, creates the set point for the inner, slave loop. In the case of the spacecraft simulator, the master loop stabilizes around an angle, while the slave loop stabilizes around an angular rate. This type of controller is faciliated by the fact that the chosen sensors onboard are capable of providing measurments for each loop, the current angle and angular rate. As stated and shown in **[6]**, cascade control has been shown to improve performance by suppressing the effect of disturbances on the master loop objective as well as reducing the sensitivity of the master loop to large gain variations in the slave loop. This allows larger gains to be used in the slave loop, and when the system has large time delays, allows for quicker response times over single loop control systems.

For this application, a P-PD cascaded controller has been chosen. Coupled with open loop system identifications of the actuators, it will be shown to provide a very robust and simple method for controlling the vehicle.

4.1 Real World Considerations

When designing a controller that is to be applied in the real world, it is important to include two important factors in the dynamic simulations. These are time delays and noise. Without these in simulation, the operational system may not exhibit the same stability as the system simulation shows. Worse, these could cause the operational system to be inherently unstable.

4.1.1 Time Delay

Time delay is a term that covers a broad range of issues in controlling unstable systems. Specifically for our system, time delay occurs at each step in the control loop as can be seen in figure 23.

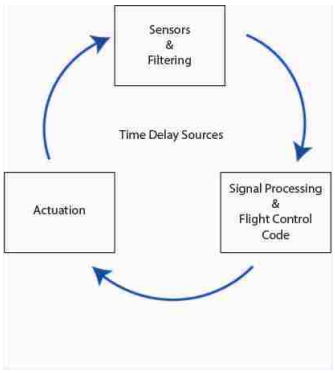

Figure 23: Time Delay Sources

Beginning with sensors, raw sensor data itself does not experience significant delay, but the issue arises as filters are placed in the control path. It can be difficult to apply a controller with the noise from low cost sensors so filters are almost always a necessary burden. If designed correctly, the delay associated with this filter can be orders of magnitude less than the dominating inertial delays of the system.

Next, and possibly the least significant, is the processing delay in the flight computer. Issues may arise when using many electrical interfaces and long programs or floating point numbers, but usually sourcing the right components can overcome these delays.

Finally, and the most significant, is the delay associated with the actuators. Up until this point in the loop, most of the processes have been dominated by electrical phenomenon, but now inertial effects enter into the equation. For our system the process is:

1. The flight computer gives the electronic speed controller (ESC) an input signal associated with a desired force
2. The ESC then interprets this signal and enables output on a different circuit to the motor
3. The motor spins the propeller to achieve the desired change in force

In fact, these effects are so important that a thrust test stand was developed to test for a combination of motor, propeller, and voltage that has the smallest time delay and still enables

the system to satisfy the high thrust to weight ratio. This will be explained in detail in section **6**.

While some of these delay sources may be insignificant compared to others, it is useful to note all sources for delay, as this aids in debugging the physical system.

4.1.2 Noise

The second real world consideration is noise. This can either be mechanial or electrical, but both directly affect the sensed measurements. Several possibilities exist to reduce noise. The first is a software or hardware filter in line with the sensor itself. For this system, a polynomial Kalman filter was used and will be demonstrated later in section **5.**

Mechanical noise in our system can be reduced by mounting components on vibration isolaters, as well as ensuring that all propellers are mass balanced around the axis of rotation. Mechanical noise is inevitable but usually not detrimental to the system.

4.2 Open Loop Simulation

Before we apply our controller, we want to check the open loop dynamics to confirm that the simulink model and equations of motion provide the correct, unstable baseline response. The input we will be using to judge system response is the step change. This simulates a desired change in attitude.

This specific system deals with a double integrator model. This means that the dynamics are such that our desired state is related to an input via two time derivatives, as can be seen from the dynamic equations in the previous section.

Using the dynamic equations, a full non-linear attitude simulation was designed in SIMULINK.

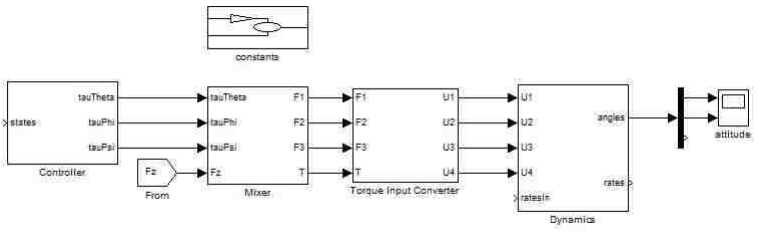

Figure 24: Non-Linear Open Loop Simulation

The open loop step response of $\theta$and ϕ are shown below:

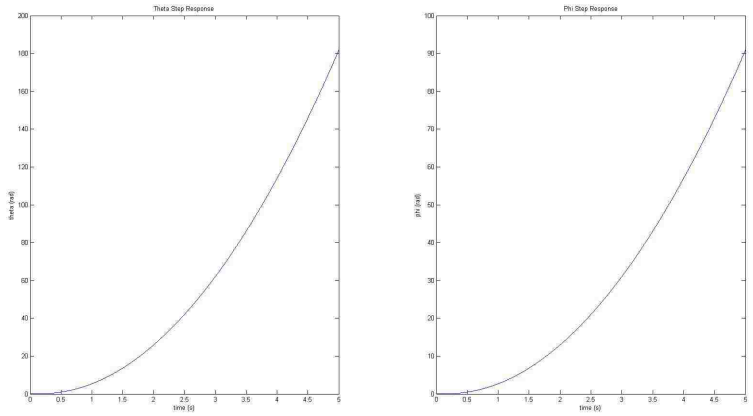

Figure 25: Non-Linear Open Loop Step Response

We can begin to analyze the stability of the system if we simplify our system, for test purposes and clarity, to one axis and linearize about the horizontal level mark. Doing this allows us to use the root locus design technique.

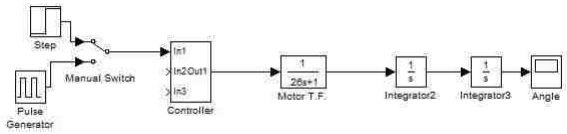

Figure 26: Linear Open Loop Simulation

The transfer function in the frequency domain for the system shown, including a pole for the time delay transfer function can be shown to be:

$$L(s) = \frac{1}{s^2(\tau s + 1)} \tag{9}$$

Where τ is the measured time constant of the system.

The corresponding root locus plot then is shown to be:

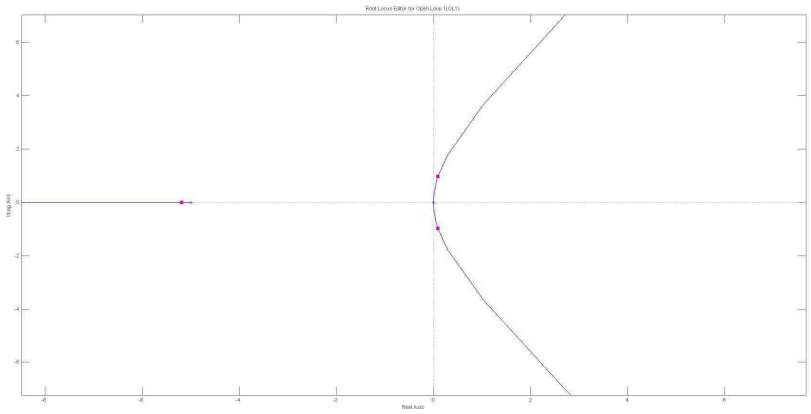

Figure 27: Root Locus of Linear Open Loop System

We see that the poles, designated with an x in figure 27, occur at the time constant of the motor and twice at the origin of the complex plane, as these are where (9) tend to infinity. The motor time constant was arbitrariliy chosen to be 0.2 for this simulation, but the slower poles are the integrators and these tend to dominate the system. We can also see from the branches of the root locus in figure 27 that no possible gain will give this system stability, as there is no part that reaches back into the negative side of the real axis.

4.3 Roll and Pitch Controller Design

Our sensors enable us to feed back all of the angles and rates involved in our attitude controller, these being $\phi_R, \dot{\phi}_R, \theta_R, \dot{\theta}_R, \varphi_R, \dot{\varphi}_R$. We can then use this information to stabilize our system by controlling around the error between a desired state and the actual state. A common form of a control law that will be used in our application is the Proportional and Derivative control.

A desired acceleration or torque can be defined to be a function of emperically found, math-

ematical weights to the error in angle and error in angle rate and a function of the errors themselves. Hence for example, ϕ gives

$$T_\phi = f\left(k_{\phi_e}, k_{\dot{\phi}_e}, \phi_e, \dot{\phi}_e\right) \tag{10}$$

Where:

$$\phi_e = \phi_D - \phi \tag{11}$$

$$\dot{\phi}_e = \dot{\phi}_D - \dot{\phi} \tag{12}$$

The 'D' subscript denotes the desired state. For ϕ_D this is trivially the desired roll angle prescribed by either the pilot or a position controller. For $\dot{\phi}_D$ however, it is possible to go a step further and consider our desired rate to be:

$$\dot{\phi}_D = f\left(k_{\phi_e}, \phi_e\right) = k_{\phi_e}\left(\phi_D - \phi\right) \tag{13}$$

Such that when (10) is written out to include (11), (12), and (13):

$$T_\phi = k_{\dot{\phi}_e}\left(k_{\phi_e}\left(\phi_D - \phi\right) - \dot{\phi}\right) \tag{14}$$

The same controller can be extended to θ such that:

$$T_\theta = k_{\dot{\theta}_e}\left(k_{\theta_e}\left(\theta_D - \theta\right) - \dot{\theta}\right) \tag{15}$$

(14) and (15) provide the essence of the cascaded state controller. The error of one state defines the error of the next state which is a derivative of the first.

4.3.1 P-P Controller Simulation

While we have mathematical gains on both the angle and angle rate, this is not however yet considered a P-D controller. Since we have a sensor for both ϕ and $\dot{\phi}$ we are not physically taking the derivative of any of the signals. This is instead a P-P controller.

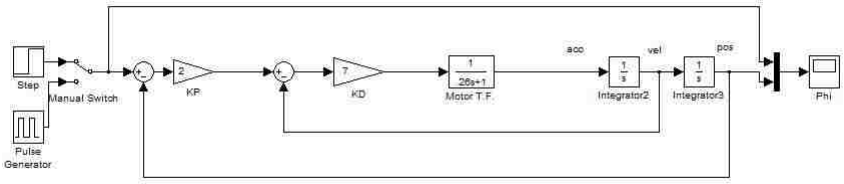

Figure 28: Linear P-P Controller Simulation

A typical trade-off in the step response of the system is between speed and oscillatory behavior; the faster the system, the more oscillatory. We can see both phenomena in the following responses in figures 29 and 30:

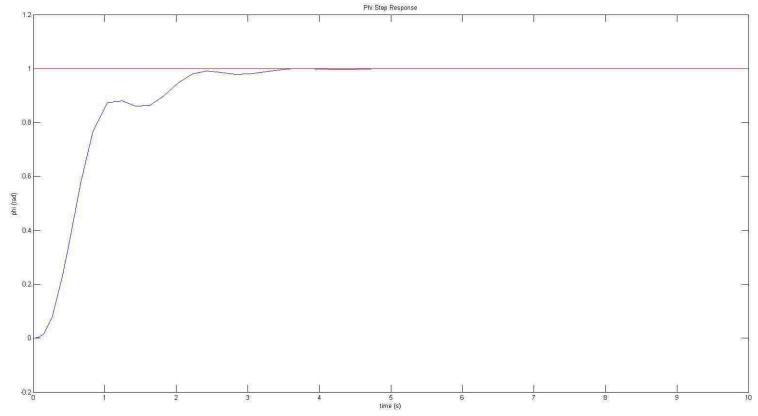

Figure 29: Linear P-P Controller Slow Step Response

Or if a faster response is required:

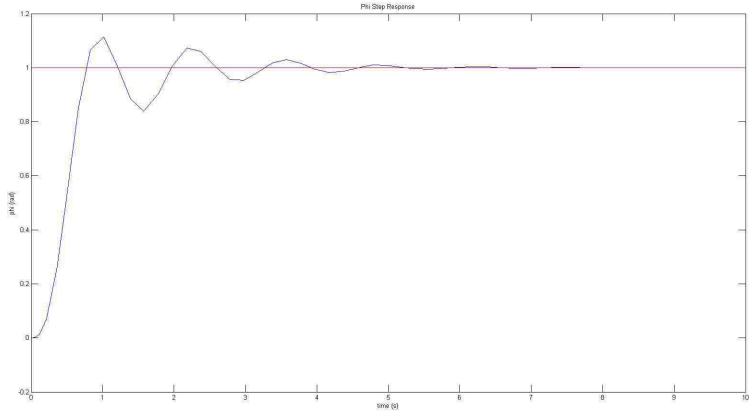

Figure 30: Linear P-P Controller Fast Step Response

To understand this stability, we can again look at the root locus of this system of which the transfer function can be found to be:

$$L(s) = \frac{k_{\phi_e} k_{\dot{\phi}_e}}{\tau s^3 + s^2 + k_{\dot{\phi}_e} s + k_{\phi_e} k_{\dot{\phi}_e}} \tag{16}$$

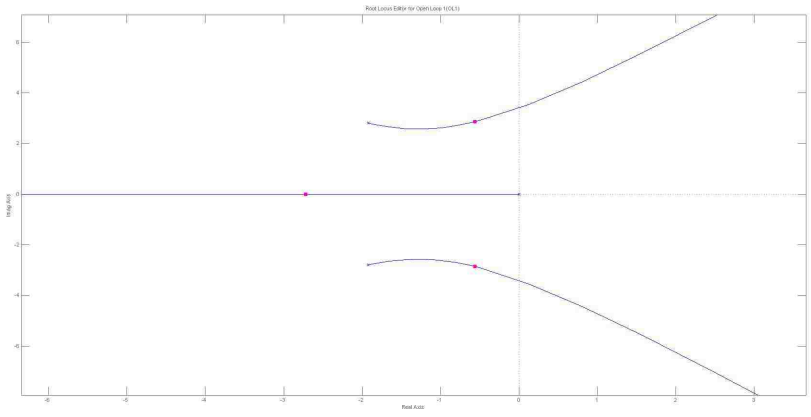

Figure 31: Root Locus of P-P Controller

It can be seen that the addition of feedback has brought the branches in figure 31 into the negative side of the real axis, but they are still very close to the complex axis. This indicates that stability is possible, but explains the oscillatory behavior observed in the simulations. Again, the poles seen on the locus are from the system time delay transfer function as well as

the change from the double pole at the origin to the complex pair.

4.3.2 P-PD Controller Simulation

After running the P-P controller simulations, it was clear that a derivative term needed to be added to achieve a faster response. This was added to the inner loop such that the new input is now a result of a P-PD controller. There is a proportional gain on the angle and both a proportional and derivative gain on the angle rate:

$$T_\phi = k_{\dot{\phi}_e}\left(k_{\phi_e}\left(\phi_D - \phi\right) - \dot{\phi}\right) + k_{\frac{d\dot{\phi}_e}{dt}}\frac{d\dot{\phi}_e}{dt}\left(k_{\phi_e}\left(\phi_D - \phi\right) - \dot{\phi}\right) \tag{17}$$

And for θ:

$$T_\theta = k_{\dot{\theta}_e}\left(k_{\theta_e}\left(\theta_D - \theta\right) - \dot{\theta}\right) + k_{\frac{d\dot{\theta}_e}{dt}}\frac{d\dot{\theta}_e}{dt}\left(k_{\theta_e}\left(\theta_D - \theta\right) - \dot{\theta}\right) \tag{18}$$

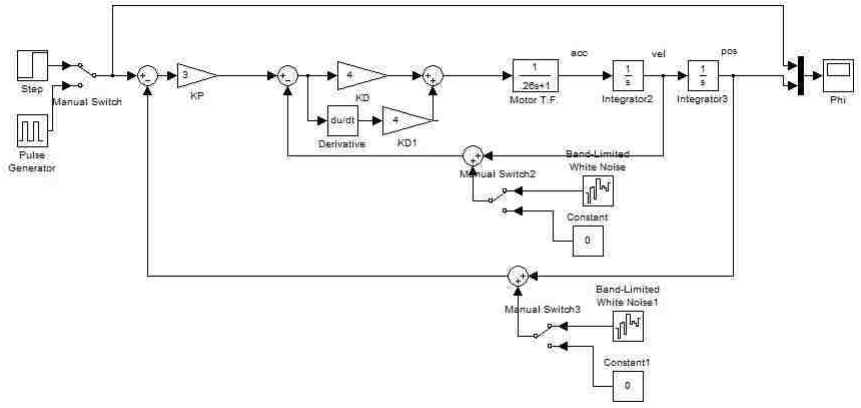

Figure 32: Linear P-PD Controller Simulation

We can see in figure 33 that the P-PD controller achieves a much faster and much less oscillatory response than the P-P:

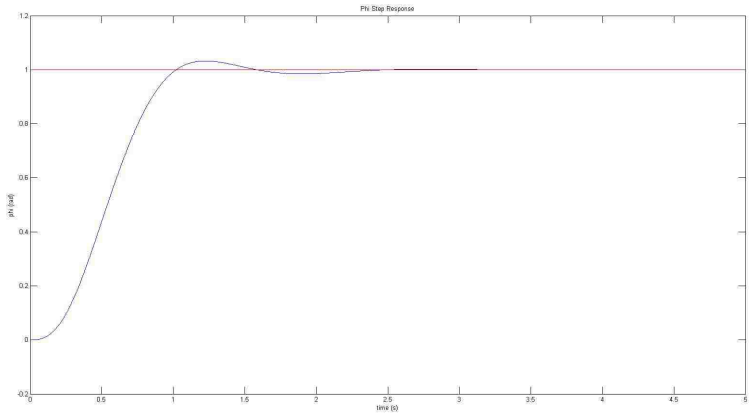

Figure 33: Linear P-PD Controller Step Response

And when looking at the root locus of the transfer function:

$$L(s) = \frac{k_{\phi_e} k_{\frac{d\dot{\theta}_e}{dt}} s + k_{\phi_e} k_{\dot{\phi}_e}}{\tau s^3 + \left(k_{\frac{d\dot{\theta}_e}{dt}} + 1\right) s^2 + \left(k_{\phi_e} k_{\frac{d\dot{\theta}_e}{dt}} + k_{\dot{\phi}_e}\right) s + k_{\phi_e} k_{\dot{\phi}_e}} \tag{19}$$

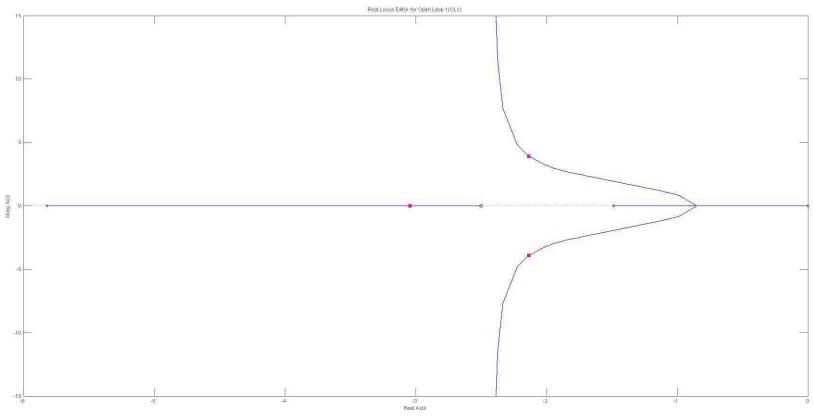

Figure 34: Root Locus of P-PD Controller

From figure 34, it is seen that the addition of the zero has drastically improved our step response. Specifically, it has dragged the branches of the root locus far into the negative real side, thus ensuring stability over a wide range of gain values. It is important to note that these simulations are with arbitrary values for the time constant as well as gains. The actual flight data will be compared to simulation later in section **8**.

When a step function is applied to this system, the derivative term in the inner control loop causes the input to spike towards infinity. Limits will be applied to the controller so that the actuators will saturate, but this does not cause any instability, as the control loop runs at 180Hz and can quickly overcome the saturation. In essence, the saturated signal gets taken out quickly.

4.4 Non-Linear Attitude Control Simulation

Now that a controller has been designed with the root locus and linear techniques, we can apply it to a non-linear, full three-axis attitude simulation to get a better understanding of response in the body fixed frame.

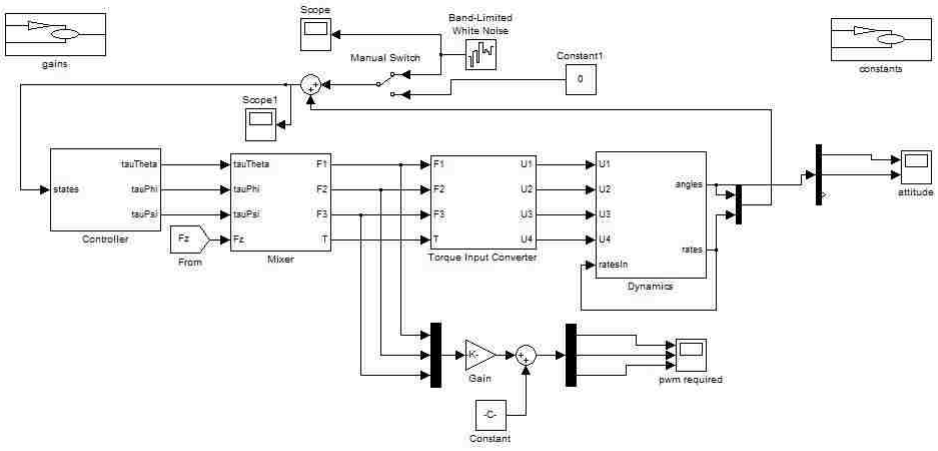

Figure 35: Non-Linear Closed Loop Simulation

This non-linear simulation in figure 35 also includes all real world effects, including time delay, noise, and also applies saturation limits on the amount of force the actuators can produce. The values eventually used in simulation are taken from actual measured values from flight hardware.

The last block in the simulation includes all the dynamic equations that were derived in the previous section. Each of the MATLAB function blocks contains a separate equation of motion for roll, pitch, and yaw.

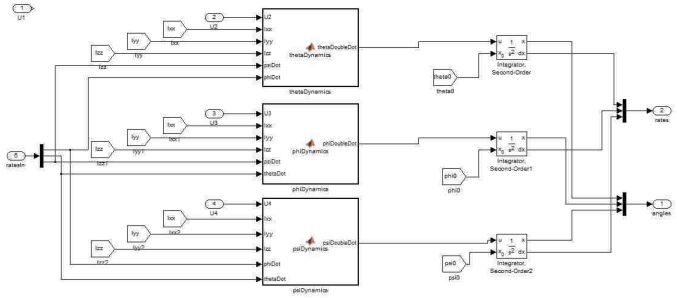

Figure 36: Non-Linear Dynamics Subsystem

This subsystem solves for all the angles and rates, then feeds these back to the controller subsystem. During the feedback path, realistic values of noise are added to each of the signals.

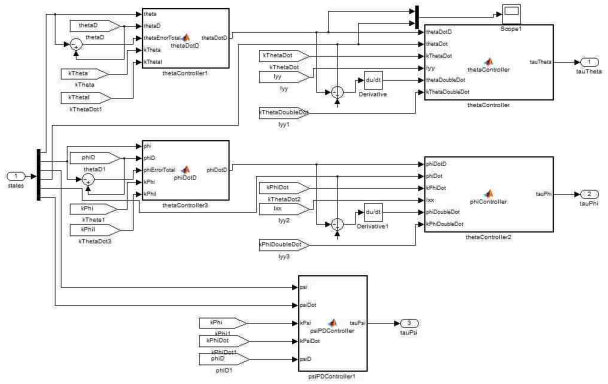

Figure 37: P-PD Controller Subsystem

Here, is all of the control laws that were derived for the P-PD controller. The simulation also has the capability of changing each of the gains to tune the controller. The output of this subsystem are the desired torques for each axis. These are the torques that were defined in (17) and (18).

The next two blocks take the desired torques and, based on the geometry of the vehicle, define forces for each actuator set. This is also where the system time delays are included.

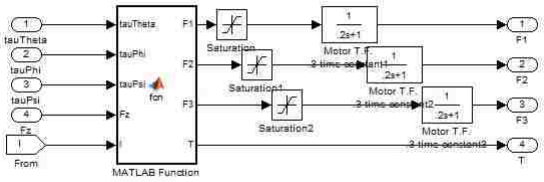

Figure 38: Mixer Subsystem

These forces defined by the controller are then sent to a graph so that the actuator effort can be monitored in simulation. They are also sent to the torque input converter block to be converted back to a useful torque for the non-linear dynamics block. This is essentially the inverse of the mixer subsystem.

Finally, in figure 39 we see the step response in our non-linear dynamics simulation is as expected:

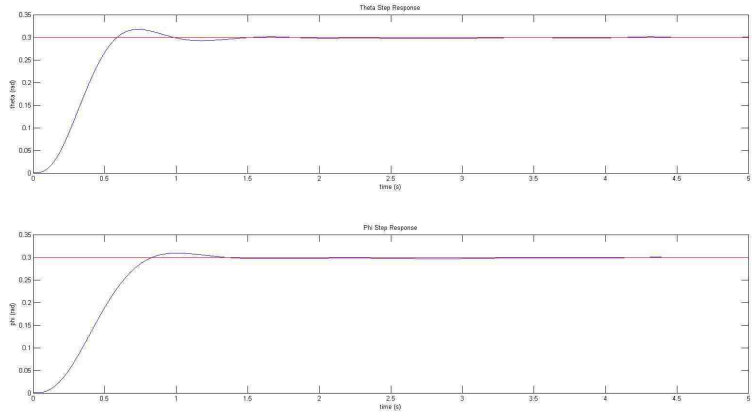

Figure 39: Non-Linear Closed Loop Step Response

4.5 Yaw Controller Design

The yaw controller on the vehicle can be less robust than that of the pitch and roll controllers. This is because the vehicle will be able to pitch and roll to reach its final destination, leaving the yaw controller to stabilize the vehicle around a constant, zero rate reference. To accomplish this, a P-D controller using rate feedback was defined and implemented as:

$$T_\psi = k_{\dot{\psi}}\dot{\psi}_e + k_{\frac{d\dot{\psi}_e}{dt}}\frac{d\dot{\psi}_e}{dt} \tag{20}$$

The vehicle controls yaw position and rate by applying a differential thrust to the top and bottom rotors. Since they are spinning in opposite directions, the torque about the z axis generated from the top rotors will be different than that from the bottom, which will cause the vehicle to yaw. The gains for this controller were tuned to output a decimal that represented a percentage of force that will be distributed to the top and bottom rotors, so that the total force perscribed by the pitch and roll controllers will always be attained by the group, but the distribution between the top and bottom rotors will be used to control yaw.

4.6 Digitization

All of the control laws discussed above are designed for an analog system. Our flight computer runs off a certain clock cycle so it is inherently digital. However, it will be assumed that since the sample rate of the flight computer is much faster than the natural frequency of the system, the control laws written can be in the analog form, and difference equations are not necessary for implementation. This assumption is proven in section 8. The digital system can be seen graphed with the analog system and there is no delay associated with the fast sample rate.

5 Sensor Filter

As stated in section **4,** measurements coming from the sensors are subjected to two forms of noise, mechanical and electrical. Without the use of a filter, the noise could cause the system to be unstable. It is essential that the angles and rates fed into the control system are clean signals so that the flight control software can define the proper forces to the motors. At first, a simple resistive-capacitive (RC) hardware filter was tested in series with each of the signals, but as this was found to be ineffective for the system, a polynomial Kalman filter (PKF) was implemented.

5.1 RC Filter

RC filters provide a simple method for allowing only low frequency signals pass through a circuit while blocking higher frequency signals.

Figure 40: RC Circuit

Summing the currents, the governing equation for this circuit in the time domain can be found to be:

$$C\frac{dV}{dt} + \frac{V}{R} = 0 \tag{21}$$

$$V(t) = V_0 e^{-\frac{t}{RC}} \tag{22}$$

Or, converting to the laplace domain, the transfer function is:

$$H(s) = \frac{1}{1 + RCs} \tag{23}$$

Which contains a single pole at $s = -\frac{1}{RC}$. The value of RC can be tuned to have a different response and cutoff frequency depending on the amount of noise from the filter. The tradeoff then becomes the amount of filtering, cutoff frequency, versus time lag for the filtered value to reach the actual value.

To better understand what the cuttoff frequency should be for our application, data can be taken from the sensors then a discrete fourier transform (DST) can be performed to isolate the

frequencies of noise. Data was taken at 180 hertz (Hz) and the following plots were created with the motors running on the simulator.

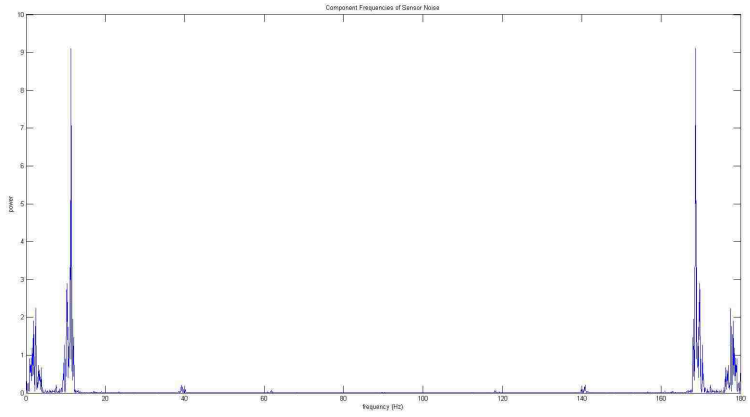

Figure 41: DFT of Sensor Noise

From this plot in figure 41, we see that the noise has frequency components in both the high and low frequency range, so it can be concluded that a simple low pass filter would have to have a very low cutoff frequency. This resulted in large amounts of time lag which is unacceptable for this type of control system.

5.2 Polynomial Kalman Filter

After testing the simple RC filter, a more robust approach was taken and a PKF was designed and implemented as in [8]. This is a recursive filter that estimates the actual states of a system from noisy measurements. Recursive means that the filter only needs distinct data points and not necessarily a history of the data. This is convenient for applied system, as it does not have large memory requirements.

The PKF takes information from the sensors and integrates this with a model of the system itself as well as knowledge of the noise experienced to produce an estimate of the system. Refer to appendix **C** for the derivation of the filter and recursive equations. For simplification, this section will show the derivation of the filter for only one axis, i.e. pitch and pitch rate. The same derivation can also apply to roll and roll rate. The equation for the PKF is:

$$\hat{X}_k = \varphi_k \hat{X}_{k-1} + G_k u_{k-1} + K_k \left(Z_k - H \left(\varphi_k \hat{X}_{k-1} - G_k u_{k-1} \right) \right) \tag{24}$$

Where,

- $\hat{X}_k$ is the estimated states at the current time step,
- $\hat{X}_{k-1}$ is the estimated states at the previous time step,
- φ_k is the discrete state transition matrix,
- G_k is the control matrix,
- u_{k-1} is the input matrix,
- K_k is the Kalman gain matrix,
- Z_k is the measurement matrix, and
- H is the observation matrix

In our system, $G_k = 0$ because the states X_k are not directly controlled by the inputs, and the filter only involves measurments and models of the system. This leaves:

$$\hat{X}_k = \varphi_k \hat{X}_{k-1} + K_k \left(Z_k - H \varphi_k \hat{X}_{k-1} \right) \tag{25}$$

With the Riccati equations:

$$M_k = \varphi_k P_{k-1} \varphi_k^T + Q_k \tag{26}$$

$$K_k = M_k H^T \left(H M_k H^T + R_k \right)^{-1} \tag{27}$$

$$P_k = \left(I - K_k H \right) M_k \tag{28}$$

Where,

- M_k is the covariance representing errors in the state estimates before the update,
- P_k is the covariance representing errors in the state estimates after the update,

- Q_k is the matrix of scalars representing process noise, and
- R_k is the matrix of scalars representing the measurement noise

Specifically, our state transition matrix in (25) shows that one signal is the derivative of the next, or:

$$\varphi_k = \begin{bmatrix} 1 & Ts \\ 0 & 1 \end{bmatrix} \tag{29}$$

And our process noise matrix is:

$$Q_k = \int_0^{Ts} \varphi_k(\tau) Q \varphi_k^T(\tau) d\tau = \begin{bmatrix} \frac{Ts(Ts^2+3)c}{3} & \frac{Ts^2c}{2} \\ \frac{Ts^2c}{2} & Tsc \end{bmatrix} \tag{30}$$

Where Ts is the sampling time and Q is a diagonal matrix of scalars c. Also, our measurment noise matrix $R_k$is a diagonal matrix of scalars m. Both c and m are tunable scalar constants that represent the process and measurement noise respectively.

Similar tradoffs between filtering and time delay exist with the PKF so the values of c and m that matched well with the system through emperical testing were found to be .25 and 1 respectively. The raw signals are compared to the filtered signals in the plots below. The code associated with the PKF is shown in appendix **D.**

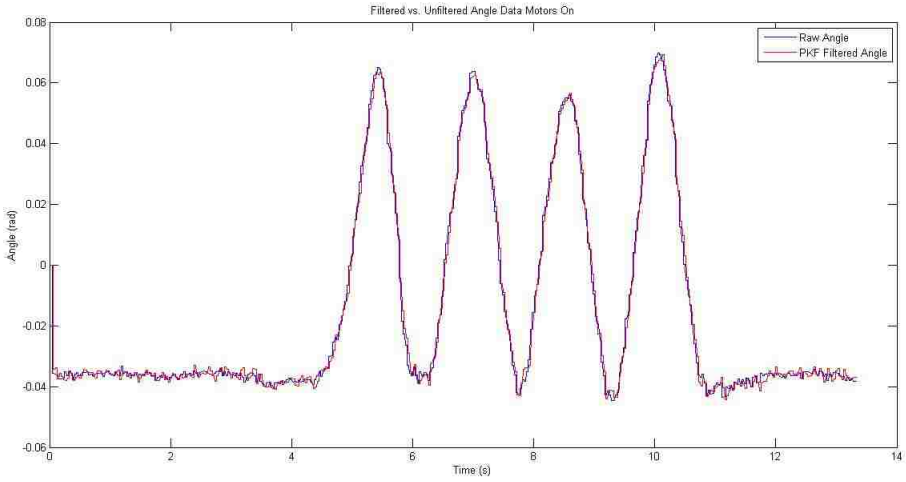

Figure 42: Comparison of PKF vs. Unfiltered Angle Measurement

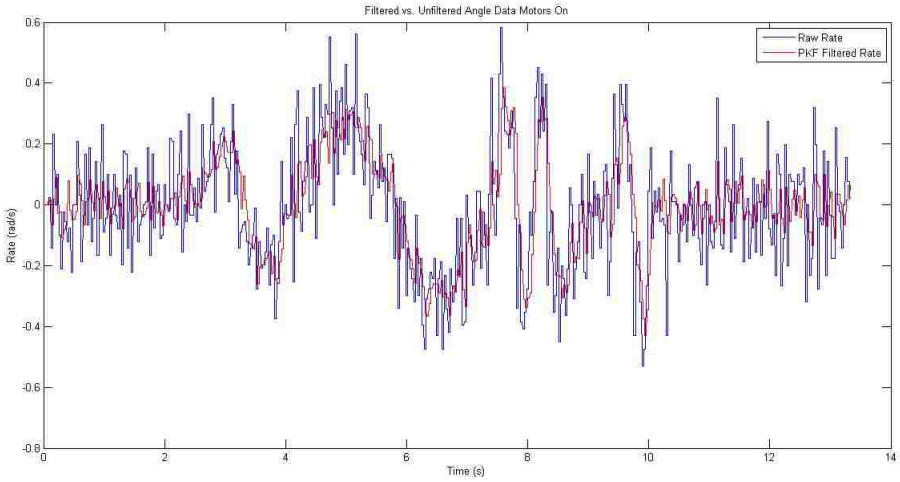

Figure 43: Comparison of PKF vs. Unfiltered Angle Rate Measurement

From these plots, it can be seen that the designed and tuned PKF does not greatly affect the internal filter of the VN-100 for the angular measurement, but provides a quick, smooth response to the noisy data of the angular rate measurement.

6 Thrust Test Stand

6.1 Motivation

A very distinct difference between simulation and an applied system deals with the observation of how our actuators respond to inputs. Introductory simulations assume that for a given input, the actuator produces a perfect response. This means that there is no time delay from input to output and that the forces prescribed by the controller is precisely achieved by the actuators. In reality, this is impossible because of real world processing effects in the software and inertial effects in the hardware. To diminish this effect, a thrust test stand was designed such that both of these metrics could be identified and measured. In essence, this test stand allows for the open loop system identification of the actuators. This accomplishes the following goals:

1. Static characterization of thrust vs. input
2. Transient response to an input
3. Comparison of different actuators based upon the previous metrics and a unique thrust vs. power consumption curve

6.1.1 Static Characterization

It is important to note the specifics of our real world input. While the controller is designed to prescribe a force required from our actuators, our real world input is much farther upstream from this force. In fact, we are actually providing a PWM signal to an electronic speed controller, which then has software that further interprets this as a point in a range from zero to full throttle. Only then is current applied to the brushless motor. Models of these systems have been created and simulated, but a more emperical approach was chosen. The static characterization will provide the conversion from a desired force to a desired PWM signal. A second order emperical model of this relationship will be found from the test data of our actuators. This relationship is specific to the flight computer, electronic speed controllers, motors, propellers, and voltages that are selected.

It would, in fact, be possible to design a multi-rotor platform and controller without giving

the system any knowledge of the actuator set. This is because the gains we choose in our controller could be capable of stabilizing the system for a specific thrust level. If the thrust deviates form this nominal level, the second order relationship of the input to output prevents the same stability characteristics from these distinct thrust levels. In other words, since the actuators are non-linear, the tuned accelerations for a specific thrust level will not provide the same response at a different level. We counteract this by providing the controller with the knowledge of the static characterization found on our thrust test bench, so that the amount of force output is always known. Thus, our accelerations can be predictable and precisely achieved across the entire range of throttle levels.

6.1.2 Transient Response

The transient response of our actuators can be described in terms of the idea of a time constant. When provided with a step input, the time constant is the physical time it takes the system to reach approximatly 63% of its final value. This is derived from the exponential decay of the error in our step response. While the controller is not told anything of the specific time constant that is found, the value is used in simulation to design a more robust controller. In general, systems with smaller time constants are easier to control, and in fact the best time constant possible is zero. The higher the time constant, the more time lag in our system and our step response gets slower with the designed controllers.

6.1.3 Comparison of Different Actuators

The knowledge of these two previous phenomena as well as the desire for the most power efficient actuator set as possible allows us to develop a testing strategy that leads to the choice of a specific motor and propeller combination. Since there are many variables involved along the actuator chain, testing was done with fixed speed controllers and flight computer while the motors, propellers, and voltages were variable. For each actuator set, a curve of thrust vs. power consumption was created so that each combination could be compared.

6.2 Test Stand Design

6.2.1 Sensors

In order to fulfill the goals for actuator testing, the test stand needed to be outfitted with a load cell as well as voltage and current sensors. These in conjunction with MATLAB's data acquisition toolbox will give the data necessary to make a proper choice for actuators, based on the 3 metrics above. The s-beam load cell was selected and wired to interface with a National Instruments 9237 data acquisition signal conditioning module.

The power sensors were found from Eagle Tree Systems and selected for their standard electrical interfaces as well as software package that allows for easy data logging and real time visualization.

6.2.2 Hardware and Construction

The structure of the test bench needed to be designed in order to facilitate the accuracy and repeatability of the measurements. This requires minimal deflections and slop throughout the entire frame and mounting interfaces such that all deflections are small with respect to those occuring within the load cell. An indirect mounting approach was chosen in order to minimize the mechanically induced noise going into the load cell.

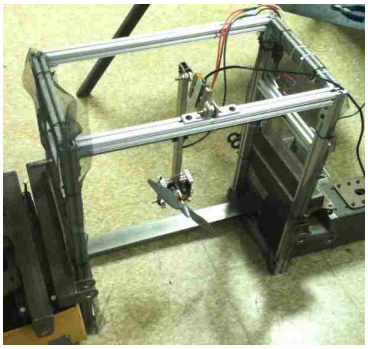

Figure 44: Thrust Test Stand Frame

The load cell was mounted with rod ends on both sides to ensure that only the axial direction is constrained. This minimizes effects due to off axis loading when stressing the load cell.

Figure 45: Load Cell Mount

Finally, the power sensors were mounted on the side along with all other necessary hardware components. This allows for quick changing of wiring and debugging.

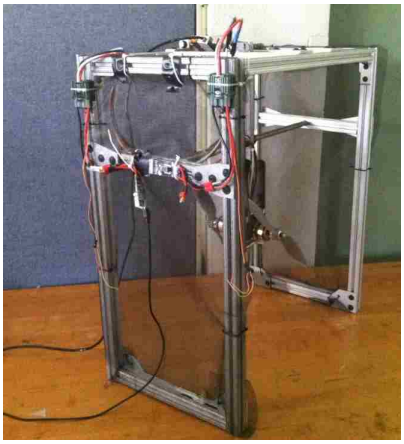

Figure 46: Power Sensors on Thrust Test Stand

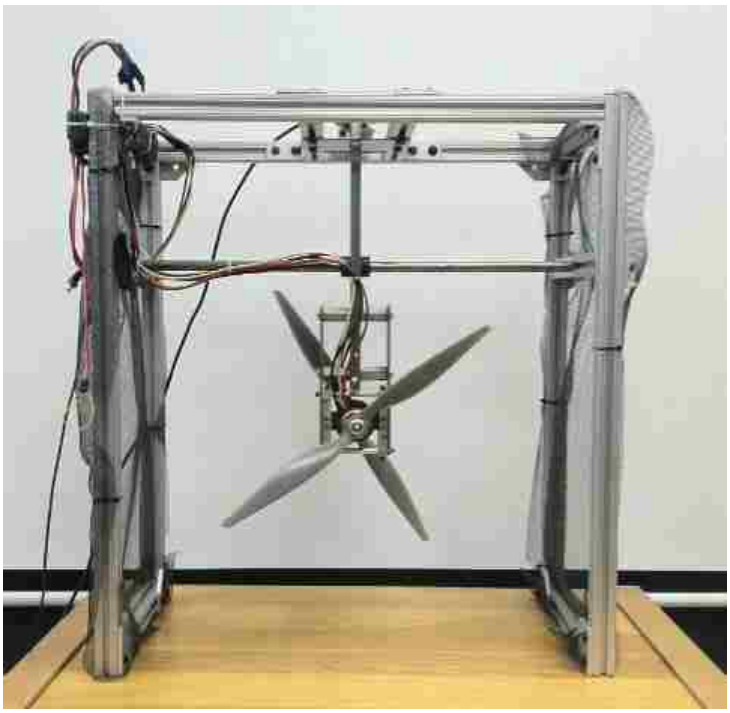

Figure 47: Final Assembly of Thrust Test Stand

6.3 Actuator Test Matrix

The following depicts the testing program on the thrust test stand. Four standard voltages were tested with different motor and propeller combinations. The combinations tested were chosen based upon the specific motor capabilities that were found on the manufacturer's website.

At first, only one motor and propeller were tested at a time. This avoids the contra-rotating setup and was done to prevent high initial cost in purchasing more equipment, as well as to maintain a reasonably sized test matrix. While the results of the testing cannot be linearly scaled to two motors, it does give an adequate idea of efficiencies and performance so that a combination can be chosen.

Motor	Propellers
Orbit 15-20	APC 13x8
	APC 14x8.5
	APC 15x4
BP 2826	APC 13x8
	APC 14x4.7
AXI 2826	APC 14x4.7
	APC 15x4
	APC 16x8

Table 2: 11.1V Test Matrix

Motor	Propellers
Orbit 15-20	APC 13x8
	APC 14x8.5
	APC 15x4
	APC 16x8
BP 2826	APC 13x8
	APC 14x4.7
AXI 2826	APC 14x4.7
	APC 15x4
	APC 16x8

Table 3: 14.8V Test Matrix

Motor	Propellers
Orbit 15-20	APC 13x8
	APC 14x8.5
	APC 15x4
	APC 16x8
AXI 2826	APC 14x4.7
	APC 15x4
	APC 15x10
	APC 16x8
AXI 4120	APC 14x8.5
	APC 15x4
	APC 16x8
	XOAR 16x8

Table 4: 18.5V Test Matrix

Motor	Propellers
AXI 4120	APC 14x8.5
	APC 15x4
	APC 16x8
	XOAR 16x8

Table 5: 22.2V Test Matrix

6.4 Load Cell Calibration

The stand was turned on its side and loaded with known masses to produce the following calibration plot:

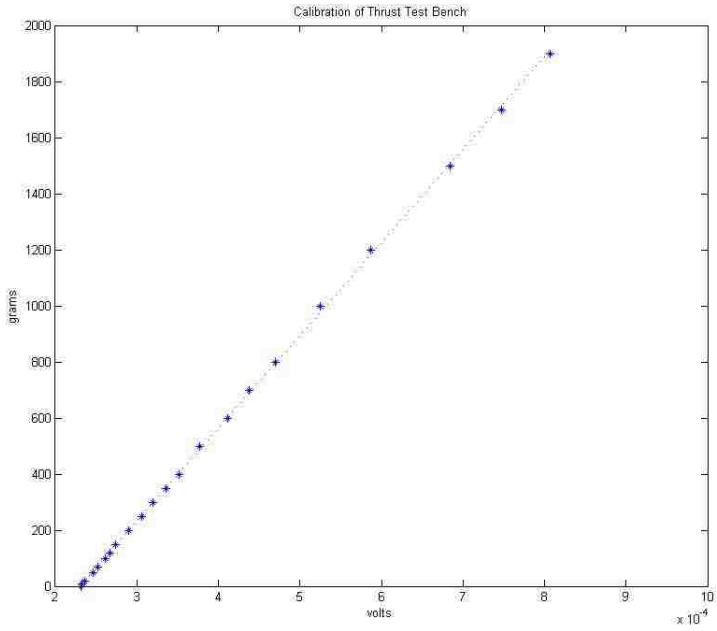

Figure 48: Thrust Test Stand Calibration

From these results, the slope was found in figure 48 to be:

$$3,311.121 \ \frac{g}{mv}$$

6.5 Test Data

6.5.1 Static Thrust

The first test conducted from the test matrix was for static thrust. This data was then used to filter out some combinations that failed to satisfy the requirement for high static thrust with respect to other combinations.

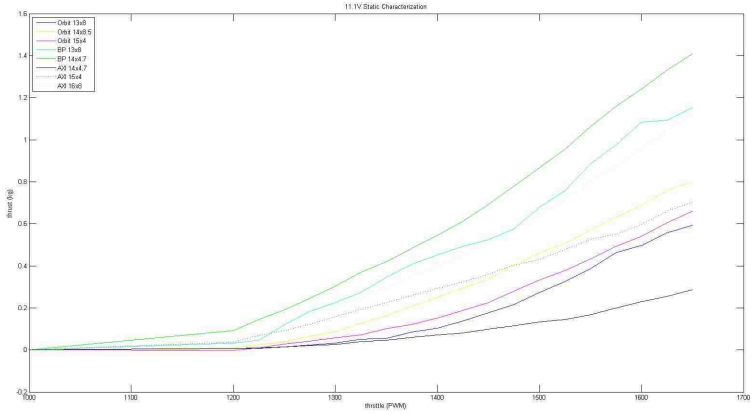

Figure 49: 11.1V Test Matrix Static Thrust

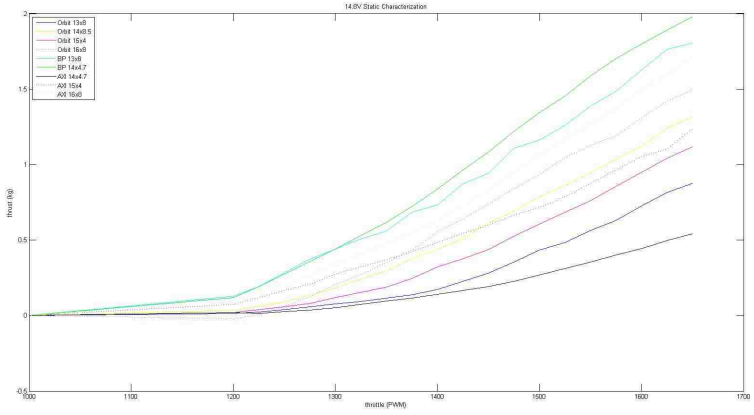

Figure 50: 14.8V Test Matrix Static Thrust

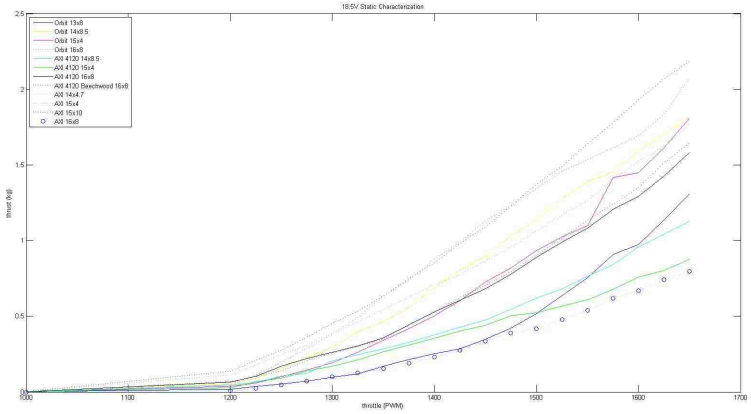

Figure 51: 18.5V Test Matrix Static Thrust

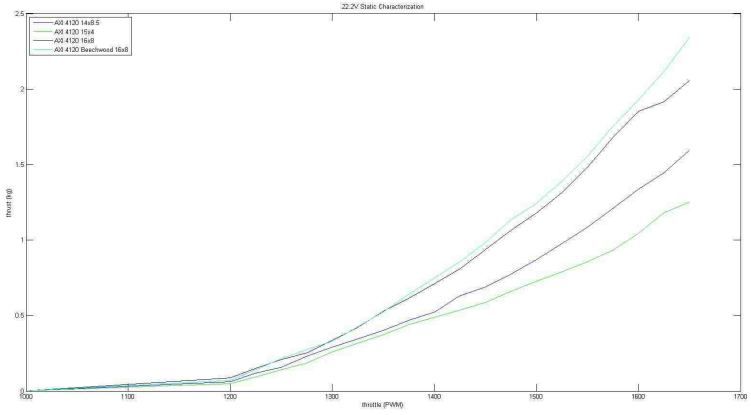

Figure 52: 22.2V Test Matrix Static Thrust

6.5.2 Transient Analysis

The transient analysis was conducted such that a step change was applied to the system, and the time response was recorded with MATLAB. The value of the step change was held constant throughout the test so that each combination could be compared against a baseline.

A MATLAB program was written to apply a finite impulse response smoothing algorithm to the raw data. This is essentially a moving average and the matlab code can be seen in appendix E.

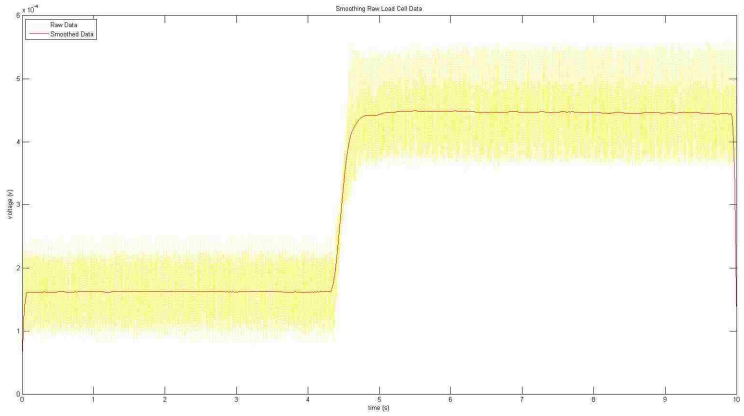

Figure 53: Results of Smoothing Algorithm

After applying the smoothing algorithm, the derivative of the signal was checked for a certain value of step change. This marked the beginning of the step response and its time was recorded. The initial and final value of the data was then marked so that a value of the time constant could be predicted. The entire data array was then searched for this value and its time was recorded. The difference in the two times is the time constant on the system. This process was repeated for each motor and propeller combination.

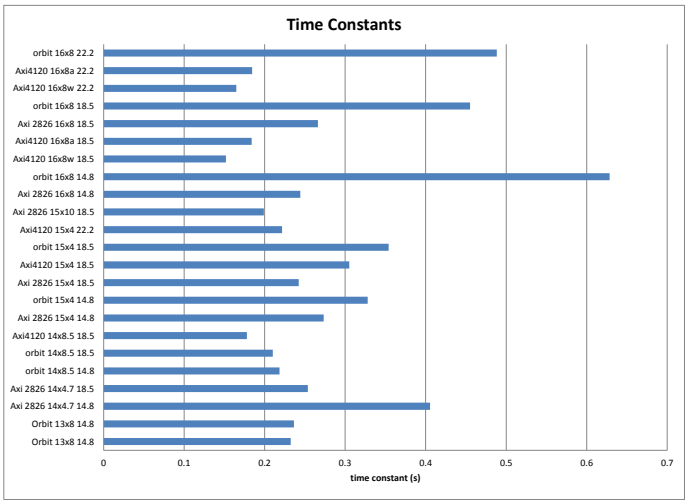

Figure 54: Transient Analysis

The transient data analysis shown in figure 54 did not show a trend or correlation between voltage, motor, and propeller. However, many of the values were similar in magnitude.

Several conclusions can be drawn from this data. First, the testing method and apparatus was not suited well for this type of data extraction. There could have been both static and kinetic friction within the apparatus that prevented the load cell from reading small changes in thrust. However, this method led to the conclusion that many of the combinations have about the same value in the time constant.

6.5.3 Thrust vs. Power

The third and final relationship for testing was thrust vs. power. The test matrix was further filtered to expedite the post-processing.

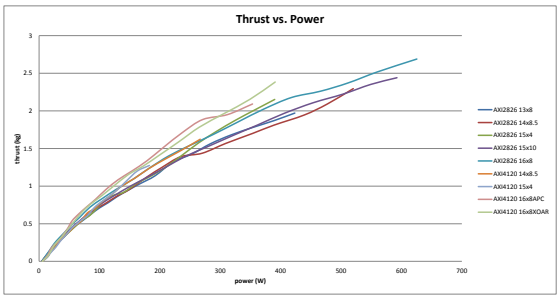

Figure 55: Thrust vs. Power Comparison

6.6 Actuator Selection

In conjunction with practical limitations, we can use this data to aid in the selection of an actuator set.

6.6.1 Propellers

First, two different sources for propellers were tested. The first, APC, is a well known RC propeller manufacture and its products are widlely available in the diameters and pitches required. The second, XOAR, manufactures a very high quality beechwood propeller with a smooth surface finish. If these propellers were selected, the reverse pitch would have necessitated a custom fabricated propeller which increases cost.

The data for the 16x8 propellers showed a slight advantage to the XOAR propeller, but this was not considered significant enough to take precedent over the sourcing issues.

6.6.2 Motors

Similar sourcing issues occured with the motors. While AXI motors are widely available, Orbits are much harder to find. Again the data did not show significant advantages of one over the other so the choice for motor was narrowed to either the AXI 2826 or the AXI 4120. As can be seen from the data above, the BP 2826 did not have the same performance as the AXI's.

6.6.3 Combination Selection

Finally, the selection of the AXI 2826 with a 15x10 propeller running at 18.5V was selected. This took into account its fast time constant, high static thrust, and sourcability. Another selection could have been the same motor and voltage but with the 16x8 propeller. This outperformed the 15x10 in the thrust vs. power relationship, but was had a slower time constant.

6.7 Contra-Rotating Actuator Characterization

Now that the actuator set has been chosen, it was possible to use the same test stand to statically and dynamically characterize the contra-rotating pair.

6.7.1 Static Characterization

The static thrust characterization of the contra-rotating pair of actuators shows that you can not assume a linear combination effect of both motors and propellers. This is the relationship that will be used in the controller to convert from a desired force, to a desired PWM signal.

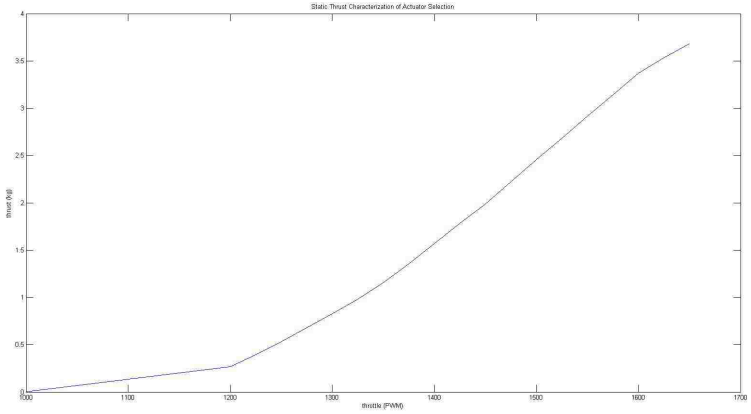

Figure 56: Static Characterization of Contra-Rotating Actuator Selection

6.7.2 Transient Analysis

We can see that the transient data for the contra-rotating pair is consistent with the single motor and propeller. The time constant for this test was .188 seconds.

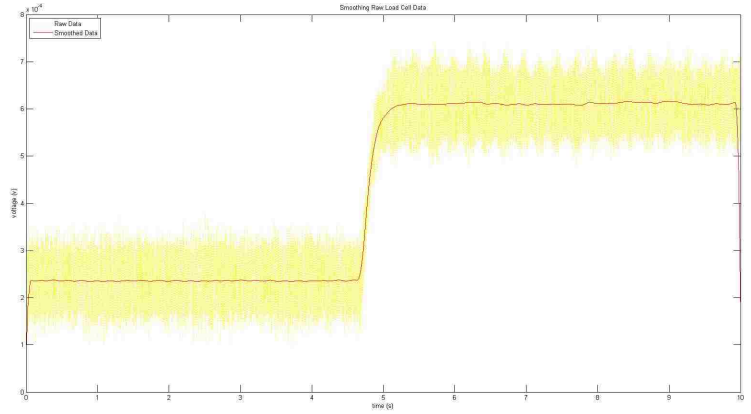

Figure 57: Transient Analysis of Contra-Rotating Actuator Selection

7 Flight Electronics and Communication

7.1 On-Board Electronics

7.1.1 Flight Computer

In order for the simulator to serve as an autonomous flight system, a flight computer needs to be present on board to serve as the central processing location for all information in the flight control software. There exists many options of computers on the market. In keeping with the low cost requirement, a very popular computer are the Arduino microcontroller boards.

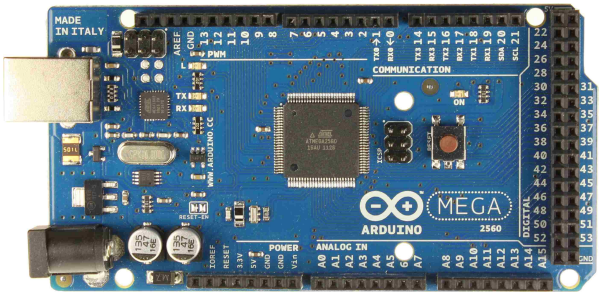

Figure 58: Arduino Mega

Arduino is a user programmable, open-source hardware and software platform. There are several reasons which make the Arduino a suitable choice for this application. Specifically, the Arduino Mega board was chosen as it is designed around an ATmega2560 microcontroller. The programming language is a derivative of C and the programming environment is very user friendly. There is also a large online community that can be used for reference when coding.

Despite being a powerful processor, the chip also has the ability to interface with other 4 other devices with its universal asynchronous receiver/transmitters (UART).This allows for further development of onboard autonomy and communication. In addition, there are also internal PWM outputs that can be accessed directly through the use of the Arduino integrated development environment. The combination of these features makes the Arduino Mega a great choice for initial development of the simulator.

7.1.2 Inertial Sensors

The inertial sensors are the components that provide the flight computer with the vehicle's orientation in real time. Low cost, lightweight sensors usually consist of micro-electromechanical systems (MEMS) technology. Specifically, the attitude and heading reference system (AHRS) will use a combination of MEMS accelerometers, gyroscopes, and magnetometers. While the data from these types of sensors are more susceptible to noise than high cost, commercial systems, MEMS sensors in conjunction with a PKF provide an excellent estimate of orientation.

After trying several options of low cost analog sensors, the VN-100 AHRS was purchased because of the integrated onboard processor and 9 DOF sensing capability. The VN-100 also has a very small footprint and a mass of only 13 g. Also included is generic filter software to help reduce the inherent noise in MEMS sensors.

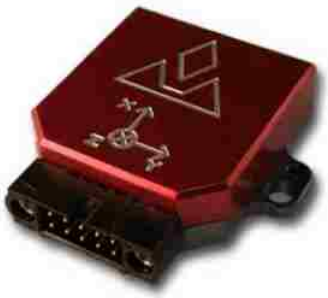

Figure 59: VN-100 AHRS

The sensor is able to communicate one of the Arduino Mega's 4 UART lines, but a software library was developed in order for the Arduino to understand the string of characters sent from the VN-100. The library code can be seen in appendix **??**.

Inside the casing of the VN-100 are 3-axis accelerometers, 3-axis gyroscopes, and 3-axis magnetometers. From the datasheet, these are capable of measuring ± 8 g, ± 500 deg/s, and 2.5 gauss respectively. Combined with a filter these values are adequate for providing an estimate for the simulators orientation.

7.1.3 Electronic Speed Controller

The motors on spacecraft simulator rotate at a certain angular rate based on the voltage applied to the terminals. If the motors were connected directly to the battery, this would cause a constate angular rotation rate. Since the simulator requires constant modulation of rotation rate to change the thrust of the motor and propeller group, this type of connection is unnacceptable for the application. Instead, an ESC is connected in series between the battery and the motor to allow the motor to rotate at desired speeds.

Figure 60: Phoenix Ice 50 ESC

The Arduino interfaces with the ESC by sending a pulse width modulation (PWM) signal. PWM is a method for getting analog results using digital signals. At a fixed frequency of 50 Hz, the Arduino sends a digital square wave to the ESC. In this fixed period of 20 milliseconds, the ratio of time that the signal is at a high voltage compared with the time it is at low voltage is called the duty cycle and is what is used to tell the ESC to rotate the motor at different angular rates.

In the ESC, this PWM signal is then used generate timing analog signals to send to power metal-oxide field effect transistors (MOSFETS) which serve as switches for the high current going to the motor. When the MOSFETS receive a voltage at its gate pin, it opens a circuit to allow current to flow threw two other isolated power pins which connect the battery to the motor. This still is an on and off signal, so to vary the speed of the motor, the ESC takes the on and off signal and applies it to the MOSFETS at a very fast rate. Since the motor is not capable of starting and stopping at these fast rates because of inertial effects, By doing so,

the motor receives more or less power which in turn causes it to continuously rotate at slower and faster speeds because inertial effects prevent starting and stopping at these high rates.

In addition, software within the ESC can cause very different responses to PWM input changes. There is internal filtering and timing signal generation that can be changed to allow for faster implementation of signals being sent to the MOSFETS. The above Castle Phoenix Ice 50 ESC was chosen because of the different firmware options available. The software could be tuned to produce faster resonse signals which is important for quick throttle modulation and control.

7.1.4 Wireless Capability

During development, the pilot will be sending flight commands to control the vehicle in real time. These include desired throttle levels and attitude angles to control position. The vehicle must also send back important flight data for offline analysis, so a robust wireless method of communicating with the spacecraft simulator during flight is necessary. Compatible with the Arduino serial library is the Xbee module.

Figure 61: Xbee Pro Module

These low cost, lightweight modules communicate over radio frequency and can interface directly with the Arduino Mega through UART. Unfortunately, these modules are limited to half-duplex operations so they are not able to send and receive at the same time. Since the vehicle is both receiving flight instructions and sending flight telemetry, two modules will be used to avoid overloading the serial lines of either. One module will be dedicated to receiving

information on a UART and the other will be dedicated to sending data on a different UART.

7.1.5 Flight Board

A after prototyping with solderless breadboards, a printed circuit board (PCB) was designed so that the electrical connections between all flight components could be contained on a single board without the concern for the loosing of wires due to flight vibration or other environmental issues.

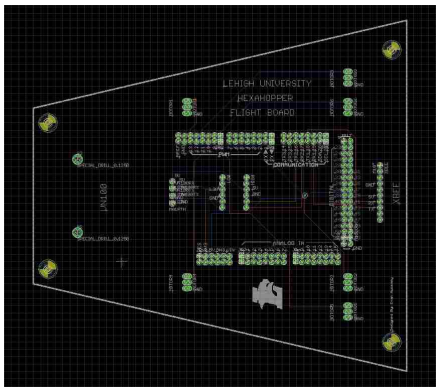

Figure 62: Flight Board Wire Diagram

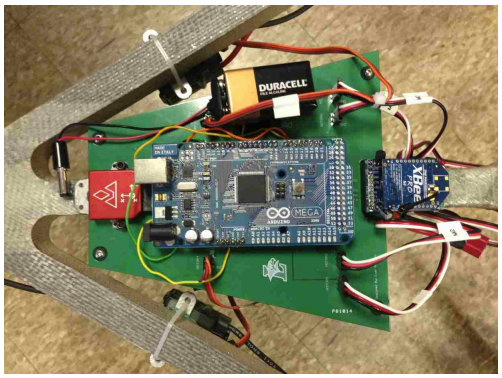

Figure 63: Flight Board

7.2 Base Station

As stated above, the spacecraft simulator is both receiving and sending signals for control and data acquisition. A base station was designed to serve as the interface at the other end of the communication line. There are several design requirements for the base station:

1. Interface with the RC receiver signals and send flight commands to the vehicle with minimal delay and precision loss
2. Provide the ability for real time controller tuning
3. Interface with data acquisition software for real time flight visualization and data recording

The base station is designed with similar hardware to that of the vehicle so that interfaces can be simplified. Hence, the base station uses an additional Arduino Mega for processing and two additional Xbee modules for sending and receiving information. The code for the base station can be found in appendix **D**.

The first requirement is satisfied by connecting the RC receiver to the interrupt pins on the Arduino Mega. Code interrupts provide a convenient way of monitoring certain tasks in the background of a main loop while not slowing down the loop frequency. In this application, the main base station code runs and will be interrupted when important informaiton from the RC receiver is sent to the Arduino. The RC receiver sends the Arduino PWM signals representing variation in the flight control commands so the high signal triggers an interrupt that contains an internal timer used to keep track of time between the high interrupt and when the code is interrupted again when the state of the pin changes to low. The result of the twice interrupted code is the duty cycle of the PWM signal sent from the RC receiver. There are four PWM signals sent which include, throttle level, as well as the three desired attitude angles for the vehicle.

Interrupts do not allow for a constant loop frequency, as they can occur at any point within the main loop of code. Since it is convenient for the flight control software to maintain a constant frequency, the RC receiver and interrupt code was placed on the base station which isolates the vehicle from this phenomenon.

Second, real time controller parameter tuning is accomplished through the use of nine slider potentiometers. These are capable of producing a variable resistance which produce a variable voltage drop. The Arduino has the capability of reading analog signals and converting them

to digital values. The nine potentiometers can be used to adjust the gains of the current controller design, but also can be used for any future expansion and development on the vehicle.

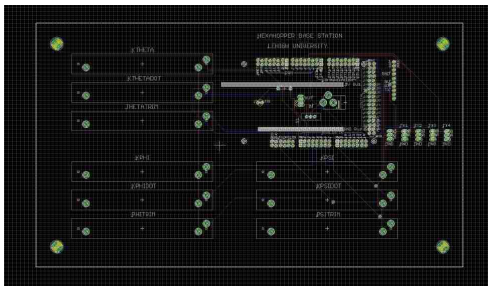

Figure 64: Base Station Wire Diagram

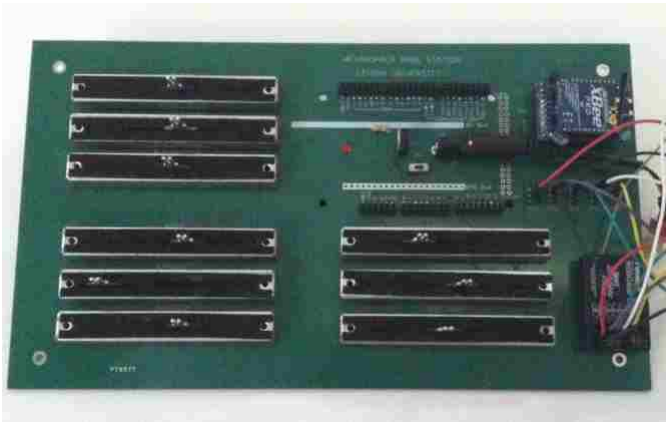

Figure 65: Base Station

The four flight commands and nine potentiometer signals are then combined into a representative string of 13 bytes that are sent over Xbee to the vehicle where the bytes are parsed and mapped to real useful values.

Finally, the third requirement is satisfied with the use of MATLAB/SIMULINK. The communication is again accomplished through the Arduino UART and a custom parse function within simulink. Only bytes can be sent over serial, so the decimal points representing the flight telemetry must first be converted to a combination of whole numbers within a range of 0-255 and then re-assembled within simulink to be displayed as the original decimal point values. To do so, each digit was separated and sent as an individual byte with a number range

of 0-9 with the sign of the number represented as either 15 or 16 for positive and negative. The precision of each number sent over serial was kept to four decimal places. This operation and communication protocol as well as the full base station code can be seen in appendix **D**.

The resulting serial communication architecture between the base station, vehicle, and data acquisition software is shown below:

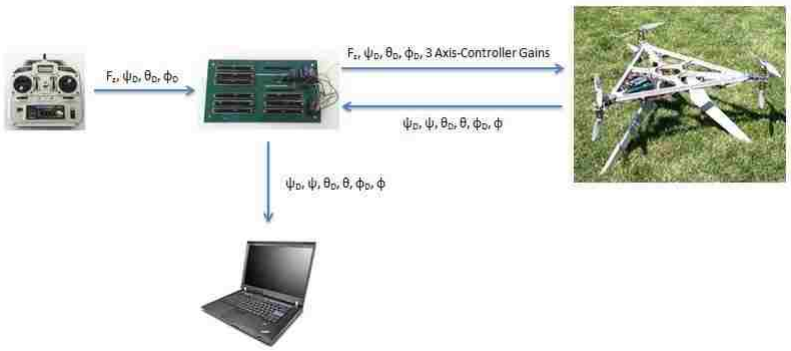

Figure 66: Spacecraft Simulator Communication Architecture

8 Flight Data

The real time data acquisition was designed such that attitude information from the vehicle can be sent to MATLAB wirelessly during flight. As the Xbee modules are not full-duplex capable, a separate Xbee module was placed on the vehicle to prevent serial buffer overload issues on the serial line already receiving gains and desired attitude values from the base station. This second Xbee on the vehicle communicates with a separate Arduino attached to the computer running MATLAB. It can be seen that if no data acquisition is required, it is easy to remove the extra hardware and software from the system.

To facilitate post-processing, the data acquisition system does not continuously take data from the vehicle, but is triggered to begin upon user request. Further, the system will only take data while the motors are running on the vehicle. The result is a compact bundle of information that the user can plot on the computer directly without having to sift through a large amount of data. The code for the data acquistion system can be found integrated into the flight code in appendices **D** and E.

The tests conducted outside the lab environment consisted of responses to the step change in the desired pitch and roll angles. This data was then compared to simulations to prove the analytical model derived in section **4** as well as the actuator parameters found in section 6 using the thrust test bench.

8.1 Data Comparison

Figure **67**shows actual flight data taken during a desired pitch step response.

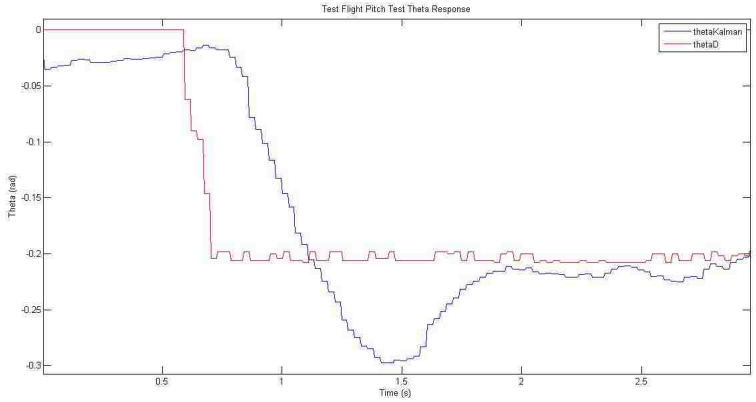

Figure 67: Pitch Step Response Flight Data

Next, to have an accurate analytical model, the time constant found from the transient analysis of mounting two motors and propellers on the thrust test bench was placed into the model. Figure 68depicts the transient plot from the thrust test:

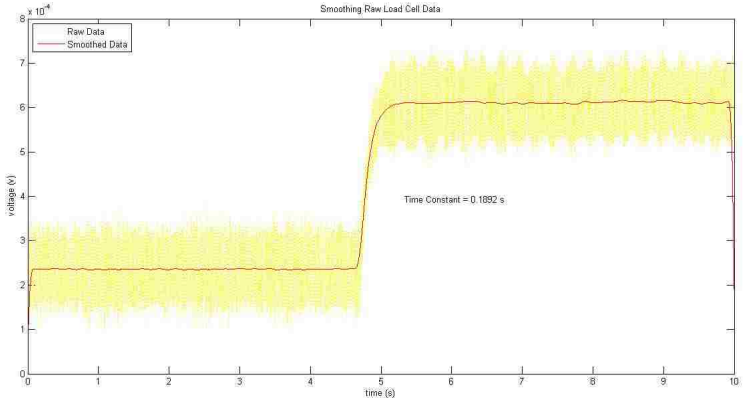

Figure 68: Transient Thrust Analysis

The time constant found from this test was approximately 0.1892 seconds. This was then placed in the root locus simulation and finally, using the actual values of gains on the flight vehicle, figure **69** was produced that depicts the simulated pitch step response:And when plotted on the same graph as the actual flight data. It can be seen that the analytical model closely resembles the response of the actual vehicle:

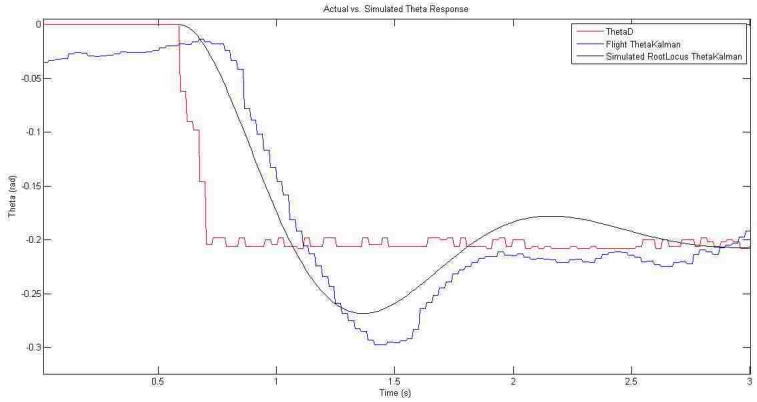

Figure 69: Actual vs. Simulated Pitch Step Response

It is also important to note than when discretized, the system still exhibits the same stability because of the very fast sample time. The control loops on board the vehicle run at approximately 180 Hz while the INS updates at 50Hz.

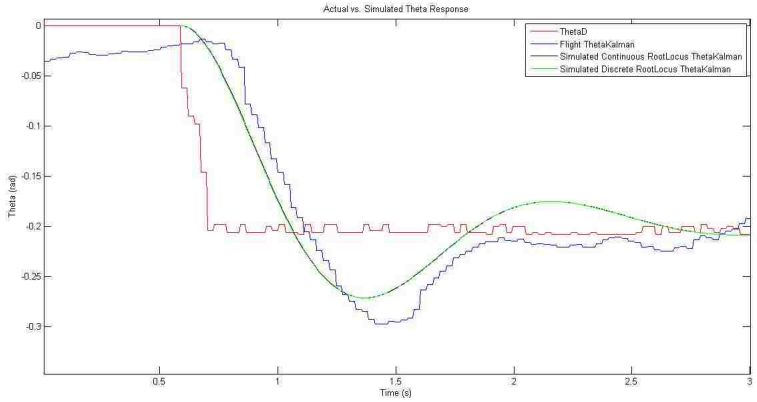

Figure 70: Actual vs. Simulated Pitch Step Response With Discretization

The greater divisions in the flight data seen in figure **70** are due to the fact that the vehicle only sends data to MATLAB every third loop. This is again to prevent overloading the serial buffer on the Xbee modules and the Arduinos.

It can be seen that the analytical model generally predicts flight performance well in terms of overshoot, rise time, as well as final settling value. The discrepency between the analytical model and the actual flight data could be due to several factors. First, the data was taken when

outside conditions were such that wind gusts and velocities greatly affected flight performance, and the analytical model does not take wind into account. Second, the time constant as well as inertia values of the vehicle itself might not be precise, thereby affecting the transient response. And finally, the system identification performed on the actuators was accomplished using a constant voltage of 18.5V. The charge on the batteries can reach 21V and drop to approximately 17V. If a model for the battery is introduced, then the predicted thrust would more closely match the actual.

The transfer function for the roll axis is the same as the pitch axis, hence, the analytical model for the other axis of the vehicle has also been proven.

Because the model has been proven with empirical data, guidance and navigation algorithms can first be tested in software with great confidence using the attitude control transfer functions in section **4.**

9 Future Work

9.1 Altitude Control

Now that the attitude control system has been shown to be effective and robust, the next step is to implement a closed loop altitude control system. As the vehicle changes attitude during flight, the thrust vector changes such that the vertical force in the z direction is changed. Without altitude control, a change in attitude will result in a drop in altitude. To compensate, the pilot must increase the overall thrust on the vehicle.

There are several options for sensing altitude outside of the lab environment. A reliable method of sensing is a barometric altimeter. The limiting factor for this type of sensor is the resolution in height. Ultimately however, the pressure sensor can be fused with accelerometer or gps data to provide a more precise measurement of altitude.

In order to select a specific pressure sensor, a relationship between pressure and temperature was found to be:

$$P_h = P_0 e^{\frac{-gh}{RT}}$$

Where $P_h$is the pressure at height h, $P_0$is the pressure at the zero level, R is the gas constant, g is the gravitational constant, and T is the temperature. If used in conjunction with a temperature sensor, the pressure sensor will be able to obtain a temperature compensated pressure reading and using this analytical relationship, the height h can be found. Also, if given the resolution of the sensor, the resolution in height can be found. Currently, the MS5611 barometric pressure sensor was found to have the best resolution for the price and was purchased. It also has the ability to interface with the Arduino via an I2C line.

9.2 Position Control

Further, in order to implement guidance and navigation algorithms outside of the lab environment, it is necessary to sense X and Y position. While it is trivial to use a GPS module and project the latitude and longitude onto a local N-E-D frame, the resolution is usually poor. The GPS readings can also be fused with accelerometer data to provide a better estimate of the vehicle's position. Currently, an Arduino program has been written that takes raw GPS data and does the vector projection to get changes in North and East position. The next step is to fuse this information with a Kalman filter.

Once the position is reliably estimated outside of the lab environment using GPS and inertial data, then both the attitude and position of the vehicle will be sensed and guidance and navigation algorithms can finally be implemented and tested.

9.3 Jet Turbine

At this point, the jet turbine has been assembled on a test fixture and tested outside. Once the closed loop altitude control has been tested and finalized, the jet can be assembled and mounted on the vehicle. Future work will have to be accomplished to mix the thrust of the jet with the thrust from the attitude control electric motors to ensure that altitude control is realized. Since the time constant of the turbine is much higher than that of the electric

motors, the jet turbine will only be modulated when a change in attitude is requested. All vehicle accelerations will be accomplished using the electric motors, as the sole function of the turbine is to counter-act a selected amount of the vehicle's mass.

Further, a test stand has been designed and a load cell purchased, to perform a system identification on the jet turbine, similar to the electric motors. This emperical relationship will be useful for ensuring the amount of thrust out of the turbine accurately represents the requested amount from the flight control software.

References

[1] Stephen McGilvray Abdelhamid Tayebi. Attitude stabilization of a vtol quadrotor aircraft. *IEEE Transactions on Control*, 14:562–571, 2006.

[2] S. Bouabdallah. Design and control of an indoor micro quadrotor. In *IEEE International Conference on Robotics and Automation, Proceedings*, 2004.

[3] Tommaso Bresciani. Modelling, identification and control of a quadrotor helicopter. Master's thesis, Lund University, 2008.

[4] R. Czyba G. Szafranski. Different approaches of pid control uav type quadrotor. *Proceedings of the International Micro Air Vehicles*, pages 70–75, 2011.

[5] Hakan Temeltas I. Can Dikmen, Aydemir Arisoy. Attitude control of a quadrotor. *IEEE*, pages 722–729, 2009.

[6] Evanghelos Zafiriou Manfred Morari. *Robust Process Control*. Prentice Hall, 1989.

[7] P. Garcia R. Lozano P. Castillo, P. Albertos. Simple real-time attitude stabilization of a quad-rotor aircraft with bounded signals. IEEE Conference on Decision and Control, 2006.

[8] Howard Musoff Paul Zarchan. *Fundamentals of Kalman Filtering: A Practical Approach*. American Institute of Aeronautics and Astronautics, 2005.

[9] Eric Stoneking. Newton-euler dynamic equations of motion for a multi-body spacecraft. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2007.

A Turbine Selection

Several different options of turbines were compared against electric ducted fans. Eventually, the ducted fans were taken out of the trade study because of the poorer thrust to weight ratio, as well as the tendancy to cause a static torque on the vehicle due to drag and inertial effects. A 7th rotor mounted in the geometric middle of the vehicle would unbalance the yaw characteristics and would necessitate a static torque from the 3 of the other propellers in the oppoosite direction. Turbines do not suffer from this problem, as the flow out of the compressor is axial in nature.

Each of the turbines in the trade study use kerosene for fuel and consist of a single stage, meaning they each have a single row of stationary vanes and a single row of rotating vanes in the compressor. Below is a graph of the thrust to weight ratio for the turbine and associated componenets.

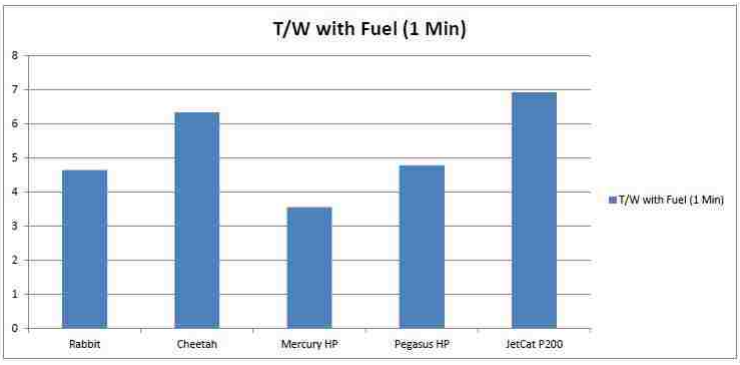

Figure 71: Turbine Thrust to Weight Comparison

Both the cheetah and the P200 stand out as candidates in the thrust to weight ratio, but the maximum thrust of the P200 far exceeds that of the cheeta.

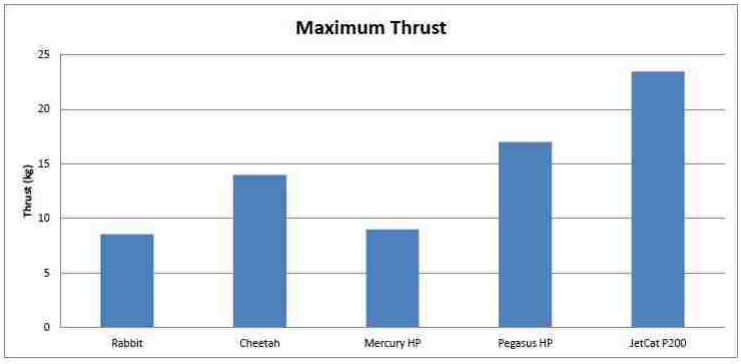

Figure 72: Turbine Maximum Thrust

This information coupled with the fact that the manufacturer of the P200 holds the largest market share in the hobbyist domain for turbines, allowed the decision to be made to select the JetCat P200 turbine for this application.

B Rotation Matrices and Newton-Euler Equation

A derivation of the rotation matrix and Newton-Euler equations used in section 3 is provided here.

B.1 Rotation Matrix

As stated in section **3**, the euler angles are a sequence of three rotations about axes. Each rotation produces a new coordinate frame, which causes the rotations to be unique in sequence.

The first is a rotation about the D axis and call this yaw, ψ.

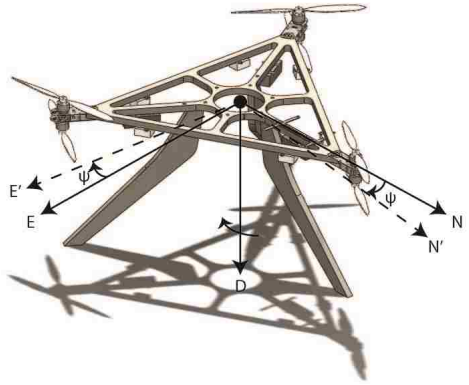

Figure 73: Yaw Rotation

$$R(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{31}$$

Next, a rotation about the E axis is called pitch, θ.

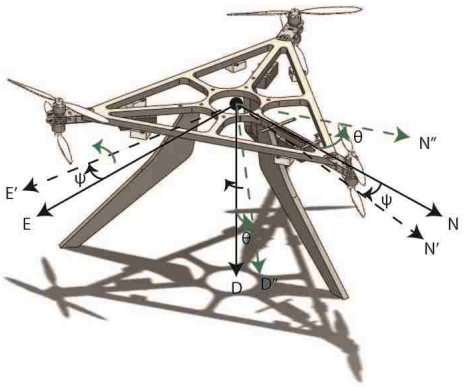

Figure 74: Pitch Rotation

$$R(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \tag{32}$$

Finally, the third rotation about the N axis is called roll, ϕ.

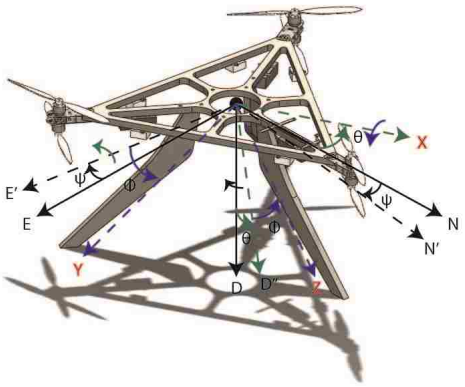

Figure 75: Roll Rotation

$$R(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix} \tag{33}$$

When (31), (32), and (33) are multiplied together in this sequence, we get the 3D rotation matrix used in **SECTION 349852:**

$$R = R(\psi)\, R(\theta)\, R(\phi)$$

$$R = \begin{bmatrix} c(\theta)\, c(\psi) & s(\phi)\, s(\theta)\, c(\psi) - c(\phi)\, c(\psi) & c(\phi)\, s(\theta)\, c(\psi) + s(\phi)\, s(\psi) \\ c(\theta)\, c(\psi) & s(\phi)\, s(\theta)\, c(\psi) + c(\phi)\, c(\psi) & c(\phi)\, s(\theta)\, s(\psi) - s(\phi)\, c(\psi) \\ -s(\theta) & s(\phi)\, c(\theta) & c(\phi)\, c(\theta) \end{bmatrix}$$

B.2 Newton-Euler Equations

We can use the knowledge of the rotation matrix and newton's second law of motion to describe the dynamics of a rigid body. The first part of the matrix equation is the translational portion.

$$F^P = m\ddot{\zeta}^P \tag{34}$$

Because Euler's first law states that force can be expressed in terms of the time rate of change of momentum, we can show (34) to be:

$$RF^B = m\dot{\widehat{R\dot{\zeta}^B}} \tag{35}$$

Then using the chain rule:

$$RF^B = m\left(R\ddot{\zeta}^B + \dot{R}\dot{\zeta}^B\right)$$

$$RF^B = mR\left(\ddot{\zeta}^B + \dot{\eta}^B \times \dot{\zeta}^B\right) \tag{36}$$

We can then write the final form of (34) to be:

$$F^B = m\left(\ddot{\zeta}^B + \dot{\eta}^B \times \dot{\zeta}^B\right) \tag{37}$$

Simillarly for rotation, we can show:

$$T^P = I_v\ddot{\eta}^P \tag{38}$$

Leads to:

$$T^B = I_V \ddot{\eta}^B + \dot{\eta}^B \times I_V \dot{\eta}^B \tag{39}$$

C Polynomial Kalman Filter Derivation

Starting with the system in section 5 but renumbered for convenience:

$$\hat{X}_k = \varphi_k \hat{X}_{k-1} + K_k \left(Z_k - H\varphi_k \hat{X}_{k-1} \right) \tag{40}$$

Where,

- $\hat{X}_k$ is the estimated states at the current time step,
- $\hat{X}_{k-1}$ is the estimated states at the previous time step,
- φ_k is the discrete state transition matrix,
- K_k is the Kalman gain matrix,
- Z_k is the measurement matrix, and
- H is the observation matrix

The measurement equation can be used to substitute for Z_k:

$$Z_k = HX_k + V_k \tag{41}$$

Then define the error in the estimate as:

$$\widetilde{X}_k = X_k - \hat{X}_k = X_k - \varphi_k \hat{X}_{k-1} + K_k \left(HX_k + V_k - H\varphi_k \hat{X}_{k-1} \right) \tag{42}$$

The current state of the system can be defined as:

$$X_k = \varphi_k X_{k-1} + W_k \tag{43}$$

Then, when (41), (42), and (43) are placed in (40):

$$\widetilde{X}_k = \varphi_k X_{k-1} + W_k - \varphi_k \hat{X}_{k-1} + K_k \left(H\varphi_k X_{k-1} + HW_k + V_k - H\varphi_k \hat{X}_{k-1} \right) \tag{44}$$

Then reducing (44) and defining the previous estimate error as:

$$\widetilde{X}_{k-1} = X_{k-1} - \hat{X}_{k-1} \tag{45}$$

Gives:

$$\widetilde{X}_k = (1 - K_k H)\,\widetilde{X}_{k-1}\varphi_k + (1 - K_k H)\,W_k - K_k V_k \tag{46}$$

If the following are defined and we square both sides of (46):

$$P_k = E\left(\widetilde{X}_k \widetilde{X}_k^T\right)$$

$$Q_k = E\left(W_k W_k^T\right)$$

$$R_k = E\left(V_k V_k^T\right)$$

$$P_k = (1 - K_k H)\left(\varphi_k P_{k-1} \varphi_k^T + Q_k\right)(1 - K_k H)^T + K_k R_k K_k^T \tag{47}$$

Then defining:

$$M_k = \varphi_k P_{k-1} \varphi_k^T + Q_k \tag{48}$$

and substituting (48) into (47) gives:

$$P_k = (1 - K_k H)\,M_k\,(1 - K_k H)^T + K_k R_k K_k^T \tag{49}$$

To find the Kalman gain K_k that minimizes the variance of the error in the estimate, take the partial derivative of (49) and equate to 0:

$$\frac{\partial P_k}{\partial K_k} = 0 = -2\,(1 - K_k H)\,M_k H + 2 K_k R_k \tag{50}$$

Then, rearranging (50), the optimal gain is given by:

$$K_k = M_k H\left(H M_k H^T + R_k\right)^{-1} \tag{51}$$

If (51) is then substituted into (49):

$$P_k = \frac{R_k K_k}{H} \tag{52}$$

And then inverting (51) and substituting into (52) gives:

$$K_k R_k = M_k H - H M_k H^T K_k \tag{53}$$

$$P_k = (1 - K_k H)\, M_k \tag{54}$$

D Arduino Code

D.1 Flight Vehicle Code

```
//HexaHopper CODE
//Author: Evan Mucasey
#include <Servo.h>
#include <VN100.h>
//////////////*********************///////////////
//////////////VARIABLE DEFINITIONS///////////////
//////////////*********************///////////////
//Pins #define pinAccX 0
#define pinAccY 1
#define pinGyroX 2
#define pinGyroY 3
#define pinVRef1 4
#define ledPin 13
//Actuator Instances
Servo motor1; //top
Servo motor2; //top
Servo motor3; //top
Servo motor4; //bottom
Servo motor5; //bottom
Servo motor6; //bottom

//Loop Timing Variables
unsigned long startLoop;
unsigned long lastLoop;
unsigned long dtLoop;
unsigned long stdLoop=5500;
//Kalman Filter, Sensor, and State Variables
float accX;
float accY;
float gyroX;
float gyroY;
float vRef1;
float thetaRaw;
float thetaDotRaw;
float thetaKalman;
```

```
float thetaDotKalman;
float thetaDotD;
float thetaDoubleDot;
float phiRaw;
float phiDotRaw;
float phiKalman;
float phiDotKalman;
float phiDotD;
float phiDoubleDot;
float psiRaw;
float psiDotRaw;
float ST[2][2];
float STTrans[2][2];
float PT[2][2];
float PP[2][2];
float MkT[2][2];
float MkP[2][2];
float KkT[2][2];
float KkP[2][2];
float Qk[2][2];
float Rk[2][2];
float KgCT;
float KgCP;
float p=.2; //coefficient for process noise
float m=1.0; //coefficient for measurment noise
float Ts=stdLoop/1000000.0; //sampling time

//Controller Variables
float thetaDotError;
float thetaDotErrorDot;
float thetaDotErrorOld;
float thetaErrorTotal=0;
float phiDotError;
float phiDotErrorDot;
float phiDotErrorOld;
float phiErrorTotal=0;
float psiDotError;
float psiDotErrorDot;
float psiDotErrorOld;
```

```
75 float psiCoeff;
76
77 //Xbee Communication and RC Transmitter Variables
78 int readdata;
79 boolean started=false;
80 boolean ended=false;
81 int packet[13];
82 float Fz;
83 float thetaD;
84 float phiD;
85 float psiDotD;
86 byte throttleRx;
87 byte phiRx;
88 byte thetaRx;
89 byte psiRx;
90 byte kThetaRx;
91 byte kThetaDotRx;
92 byte thetaTrimRx;
93 byte kPhiRx;
94 byte kPhiDotRx;
95 byte phiTrimRx;
96 byte kPsiRx;
97 byte kPsiDotRx;
98 byte psiTrimRx;
99
100 //Gains
101 float kTheta;
102 float kThetaDot;
103 float kThetaDotErrorDot=.5;
104 float thetaTrim;
105 float kThetaI=0.00;
106 float kPhi;
107 float kPhiDot;
108 float kPhiDotErrorDot=.45;
109 float phiTrim;
110 float kPhiI=0.00;
111 float kPsi;
112 float kPsiDot;
113 float kPsiDotErrorDot=0;
```

```
float psiTrim;

//Analytical Moment Inputs
float tauPsi = 0;
float tauTheta = 0;
float tauPhi = 0;
float totErrorPsi=0;

//Governer Variables
long gov=100; //maximum throttle allowed (percent)
long idle=1000;
//idle command to motors in microseconds
long redline=1650; //maximum thrust command to motors in microseconds
long ceiling; //maximum throttle to be mapped from idle and redline using gov
   variable

//Force Arrays
float f[3] = {0, 0, 0};  //motor thrust calculated using mixer
int pwm[6] = {0, 0, 0, 0, 0, 0};  //pwm signal sent to motors using labview
  and excel data

float L=0.5;  //distance from center of motors to center of HexaHopper
float Ixx=0.241296166;
float Iyy=0.243896259;
float Izz=0.456738918;
int i=0;
boolean PauseProgram=true;

//Data Acquisition Variables
byte startSend=55;
byte endSend=77;
byte sign;
byte count=0;
byte digits[4];

////////////////**********////////////////
////////////////*******SETUP*******////////////////
////////////////**********////////////////
void setup(){
```

```
151 Serial.begin(57600);
152 Serial.flush();
153 Serial1.begin(57600);
154 Serial1.flush();
155 Serial3.begin(57600);
156 Serial3.flush();
157
158 motor1.attach(7);
159 motor2.attach(12);
160 motor3.attach(11);
161 motor4.attach(10);
162 motor5.attach(9);
163 motor6.attach(8);
164
165 //Initialize VN_100 Rugged
166 IMU.Init();
167 Serial2.flush();
168
169 //Define maximum pwm signal that can be sent to motor
170 ceiling = map(gov, 0, 100, idle, redline);
171 //primer();
172
173 //State Transition Matrix
174 ST[0][0]=1;
175 ST[0][1]=Ts;
176 ST[1][0]=0;
177 ST[1][1]=1;
178
179 //Transpose of State Transition Matrix
180 STTrans[0][0]=1;
181 STTrans[0][1]=0;
182 STTrans[1][0]=Ts;
183 STTrans[1][1]=1;
184
185 //Initial Covariance Matrix for Theta
186 PT[0][0]=1;
187 PT[0][1]=0;
188 PT[1][0]=0;
189 PT[1][1]=1;
```

```
190
191 //Initial Covariance Matrix for Theta
192 PP[0][0]=1;
193 PP[0][1]=0;
194 PP[1][0]=0;
195 PP[1][1]=1;
196
197 //Qk Matrix solved from integral equation (constant)
198 Qk[0][0]=p*(Ts*Ts*Ts + 3)/3;
199 Qk[0][1]=p*Ts*Ts/2;
200 Qk[1][0]=p*Ts*Ts/2;
201 Qk[1][1]=p*Ts;
202
203 //Rk Matrix (constant)
204 Rk[0][0]=m;
205 Rk[0][1]=0;
206 Rk[1][0]=0;
207 Rk[1][1]=m;
208 }
209
210 ////////////////**********////////////////
211 ////////////////********LOOP********////////////////
212 ////////////////**********////////////////
213 void loop(){
214 getDesired();
215 primer();
216 kalmanAttitudeEstimates();
217
218 //P_PD Controllers
219 //Desired Rates from Cascade definition
220 thetaDotD = kTheta*(thetaD - thetaKalman - thetaTrim);
221 phiDotD = kPhi*(phiD - phiKalman - phiTrim);
222
223 //Integrated Errors
224 thetaErrorTotal = thetaErrorTotal + (thetaD - thetaKalman)*(lastLoop*.000001)
    ;
225 phiErrorTotal = phiErrorTotal + (phiD - phiKalman)*(lastLoop*.000001);
226
227 //Create Errors in Rates
```

```
228 thetaDotError = thetaDotD - thetaDotKalman;
229 phiDotError = phiDotD - phiDotKalman;
230 psiDotError = psiDotD - psiDotRaw;
231
232 //Derivative of Rate Errors
233 thetaDotErrorDot = (thetaDotError - thetaDotErrorOld)/(lastLoop*.000001);
234 phiDotErrorDot = (phiDotError - phiDotErrorOld)/(lastLoop*.000001);
235 psiDotErrorDot = (psiDotError - psiDotErrorOld)/(lastLoop*.000001);
236
237 //Cascade Control Law
238 tauTheta = (kThetaDot*thetaDotError + kThetaDotErrorDot*thetaDotErrorDot +
      thetaErrorTotal*kThetaI)*Iyy;
239 tauPhi = (kPhiDot*phiDotError + kPhiDotErrorDot*phiDotErrorDot +
      phiErrorTotal*kPhiI)*Ixx;
240 psiCoeff = (kPsiDot*psiDotError + kPsiDotErrorDot*psiDotErrorDot);//)*Izz;
241
242 //Define Old Errors for Derivative
243 thetaDotErrorOld = thetaDotError;
244 phiDotErrorOld = phiDotError;
245 psiDotErrorOld = psiDotError;
246
247 mixer(Fz, tauTheta, tauPhi); // calculate motor thrust from analytical moment
       matrix inversion
248 converter();
249 governer(); // limit throttle % on each motor to value given by variable 'gov
      '
250 actuator(); // set motor speeds
251
252 //Send Data to Arduino Data AQ then to MATLAB
253 if(count==4){
254   Serial3.write(startSend);
255   sendDigits(thetaKalman);
256   sendDigits(thetaD);
257   sendDigits(phiKalman);
258   sendDigits(phiD);
259   Serial3.write(endSend);
260   count=0;
261 }
262 count++;
```

```
263
264 //Loop Timing Control
265 dtLoop=micros()-startLoop;
266 if (dtLoop<stdLoop){delayMicroseconds(stdLoop-dtLoop);}
267 lastLoop=micros()-startLoop;
268 startLoop=micros();
269 }
270
271 ////////////*********************///////////////
272 ////////////*****FUNCTIONS******///////////////
273 ////////////*********************///////////////
274 void getDesired(){
275 while (Serial1.available()>5){
276   readdata=Serial1.read();
277   if (readdata==254){
278     started=true;
279     ended=false;
280   }
281   else if (readdata==255){
282     ended=true;
283     break;
284   }
285   else{
286     packet[i]=readdata;
287     i++;
288   }
289 }
290
291 if(started&&ended){
292   throttleRx=packet[0];
293   psiRx=packet[1];
294   thetaRx=packet[2];
295   phiRx=packet[3];
296   kThetaRx=packet[4];
297   kThetaDotRx=packet[5];
298   thetaTrimRx=packet[6];
299   kPhiRx=packet[7];
300   kPhiDotRx=packet[8];
301   phiTrimRx=packet[9];
```

```
302   kPsiRx=packet[10];
303   kPsiDotRx=packet[11];
304   psiTrimRx=packet[12];
305
306   Fz=floatMap(float(throttleRx),0.0,250.0,0.0,90.0);
307   thetaD=floatMap(float(thetaRx),0,250,-0.25,0.25);
308   phiD=floatMap(float(phiRx),0,250,-0.25,0.25);
309   psiDotD=floatMap(float(psiRx),0,250,-.5,.5);
310   kTheta=floatMap(float(kThetaRx),0,250,0,10);
311   kThetaDot=floatMap(float(kThetaDotRx),0,250,0,15);
312   thetaTrim=floatMap(float(thetaTrimRx),0,250,-0.75,0.75);
313   kPhi=floatMap(float(kPhiRx),0,250,0,10);
314   kPhiDot=floatMap(float(kPhiDotRx),0,250,0,15);
315   phiTrim=floatMap(float(phiTrimRx),0,250,-0.5,0.5);
316   kPsi=floatMap(float(kPsiRx),0,250,0,10);
317   kPsiDot=floatMap(float(kPsiDotRx),0.0,250.0,0.0,.3);
318   psiTrim=floatMap(float(psiTrimRx),0.0,250.0,-.5,.5);
319   psiCoeff=psiTrim+psiControl;
320
321 //When appropriate gains are found through tuning put them here:
322   kTheta=7.0;
323   kThetaDot=2.5;
324   thetaTrim=-.11;
325
326   kPhi=7.0;
327   kPhiDot=2.5;
328   phiTrim=0;
329
330   kPsiDot=.3;
331   psiTrim=-.5;
332   started=false;
333   ended=false;
334   i=0;
335   }
336 }
337
338 void primer(){
339   if (Fz<=7){
340     PauseProgram=true;
```

```
341     Serial1.flush();
342   }
343   while (PauseProgram==true){
344     motor1.writeMicroseconds(1000);
345     motor2.writeMicroseconds(1000);
346     motor3.writeMicroseconds(1000);
347     motor4.writeMicroseconds(1000);
348     motor5.writeMicroseconds(1000);
349     motor6.writeMicroseconds(1000);
350     thetaErrorTotal = 0;
351     phiErrorTotal = 0;
352     Serial.println("Program Paused");
353     getDesired();
354     if (Fz>7){
355       PauseProgram=false;
356       Serial1.flush();
357       break;
358     }
359   }
360 }
361
362 void kalmanAttitudeEstimates(){
363 //Ricatti Equations
364 MkT[0][0]=PT[0][0] + Ts*PT[1][0] + Ts*PT[1][0] + Ts*Ts*PT[1][1] + Qk[0][0];
365 MkT[0][1]=PT[0][1] + Ts*PT[1][1] + Qk[0][1];
366 MkT[1][0]=PT[1][0] + Ts*PT[1][1] + Qk[1][0];
367 MkT[1][1]=PT[1][1] + Qk[1][1];
368
369 MkP[0][0]=PP[0][0] + Ts*PP[1][0] + Ts*PP[1][0] + Ts*Ts*PP[1][1] + Qk[0][0];
370 MkP[0][1]=PP[0][1] + Ts*PP[1][1] + Qk[0][1];
371 MkP[1][0]=PP[1][0] + Ts*PP[1][1] + Qk[1][0];
372 MkP[1][1]=PP[1][1] + Qk[1][1];
373
374 //Kalman Gain Inverse Constant Denominator
375 KgCT=MkT[0][0]*MkT[1][1] - MkT[0][1]*MkT[1][0] + MkT[0][0]*m + MkT[1][1]*m +
      m*m;
376 KgCP=MkP[0][0]*MkP[1][1] - MkP[0][1]*MkP[1][0] + MkP[0][0]*m + MkP[1][1]*m +
      m*m;
377
```

```
//Kalman Gain Matrix
KkT[0][0]=(MkT[0][0]*(MkT[1][1] + m) - MkT[0][1]*MkT[1][0])/KgCT;
KkT[0][1]=(MkT[0][1]*(MkT[0][0] + m) - MkT[0][0]*MkT[0][1])/KgCT;
KkT[1][0]=(MkT[1][0]*(MkT[1][1] + m) - MkT[1][0]*MkT[1][1])/KgCT;
KkT[1][1]=(MkT[1][1]*(MkT[0][0] + m) - MkT[0][1]*MkT[1][0])/KgCT;

KkP[0][0]=(MkP[0][0]*(MkP[1][1] + m) - MkP[0][1]*MkP[1][0])/KgCP;
KkP[0][1]=(MkP[0][1]*(MkP[0][0] + m) - MkP[0][0]*MkP[0][1])/KgCP;
KkP[1][0]=(MkP[1][0]*(MkP[1][1] + m) - MkP[1][0]*MkP[1][1])/KgCP;
KkP[1][1]=(MkP[1][1]*(MkP[0][0] + m) - MkP[0][1]*MkP[1][0])/KgCP;

//Covariance Matrices
PT[0][0]= -MkT[1][0]*KkT[0][1] - MkT[0][0]*(KkT[0][0] - 1);
PT[0][1]= -MkT[1][1]*KkT[0][1] - MkT[0][1]*(KkT[0][0] - 1);
PT[1][0]= -MkT[0][0]*KkT[1][0] - MkT[1][0]*(KkT[1][1] - 1);
PT[0][0]= -MkT[0][1]*KkT[1][0] - MkT[1][1]*(KkT[1][1] - 1);

PP[0][0]= -MkP[1][0]*KkP[0][1] - MkP[0][0]*(KkP[0][0] - 1);
PP[0][1]= -MkP[1][1]*KkP[0][1] - MkP[0][1]*(KkP[0][0] - 1);
PP[1][0]= -MkP[0][0]*KkP[1][0] - MkP[1][0]*(KkP[1][1] - 1);
PP[0][0]= -MkP[0][1]*KkP[1][0] - MkP[1][1]*(KkP[1][1] - 1);
getSensorData();

//Calculate State Estimates
thetaKalman=ST[0][0]*thetaKalman + ST[0][1]*thetaDotKalman - KkT[0][0]*(ST
  [0][0]*thetaKalman + ST[0][1]*thetaDotKalman - thetaRaw) - KkT[0][1]*(ST
  [1][0]*thetaKalman + ST[1][1]*thetaDotKalman - thetaDotRaw);
thetaDotKalman=ST[1][0]*thetaKalman + ST[1][1]*thetaDotKalman - KkT[1][0]*(ST
  [0][0]*thetaKalman + ST[0][1]*thetaDotKalman - thetaRaw) - KkT[1][1]*(ST
  [1][0]*thetaKalman + ST[1][1]*thetaDotKalman - thetaDotRaw);    phiKalman=
  ST[0][0]*phiKalman + ST[0][1]*phiDotKalman - KkP[0][0]*(ST[0][0]*phiKalman
  + ST[0][1]*phiDotKalman - phiRaw) - KkP[0][1]*(ST[1][0]*phiKalman + ST
  [1][1]*phiDotKalman - phiDotRaw);    phiDotKalman=ST[1][0]*phiKalman + ST
  [1][1]*phiDotKalman - KkP[1][0]*(ST[0][0]*phiKalman + ST[0][1]*phiDotKalman
   - phiRaw) - KkP[1][1]*(ST[1][0]*phiKalman + ST[1][1]*phiDotKalman -
  phiDotRaw);     }

void getSensorData(){
IMU.Read();
```

```
407 if (IMU.NewData){   //If there is new data on the serial line
408   thetaRaw = IMU.pitch * 0.0174532925;
409   phiRaw = (IMU.roll * 0.0174532925)*-1;
410   psiRaw = (IMU.yaw - 12) * 0.0174532925;
411   if (psiRaw <0){psiRaw = psiRaw + 6.28318531;} // 0->6.238 radians for 0-360
        degrees
412
413   thetaDotRaw=IMU.Angular1;
414   phiDotRaw=IMU.Angular0*-1;
415   psiDotRaw=IMU.Angular2;
416   IMU.NewData = 0;           // We have read the data
417   }
418 }
419
420 void mixer(float fz, float tTheta, float tPhi){
421 f[0] = fz/3  + (tTheta)/(L);  //F14
422 f[1] = fz/3  - tTheta/(2*L) - 1.73205*tPhi/(3*L); //F25
423 f[2] = fz/3  - tTheta/(2*L) + 1.73205*tPhi/(3*L); //F36
424 long j;
425 for(j = 0; j < 3; j++){if(f[j]<0) f[j]=0;}
426 }
427
428 void converter(){
429 pwm[0] = -0.198*f[0]*(.5 + psiCoeff)*f[0]*(.5 + psiCoeff) + 25.46*f[0]*(.5 +
      psiCoeff) + 1191.0;  //15x10 quadratic fit
430 pwm[1] = -0.198*f[1]*(.5 + psiCoeff)*f[1]*(.5 + psiCoeff) + 25.46*f[1]*(.5 +
      psiCoeff) + 1191.0;
431 pwm[2] = -0.198*f[2]*(.5 + psiCoeff)*f[2]*(.5 + psiCoeff) + 25.46*f[2]*(.5 +
      psiCoeff) + 1191.0;
432 pwm[3] = -0.198*f[0]*(.5 - psiCoeff)*f[0]*(.5 - psiCoeff) + 25.46*f[0]*(.5 -
      psiCoeff) + 1191.0;
433 pwm[4] = -0.198*f[1]*(.5 - psiCoeff)*f[1]*(.5 - psiCoeff) + 25.46*f[1]*(.5 -
      psiCoeff) + 1191.0;
434 pwm[5] = -0.198*f[2]*(.5 - psiCoeff)*f[2]*(.5 - psiCoeff) + 25.46*f[2]*(.5 -
      psiCoeff) + 1191.0;
435 }
436
437 void governer(){
438 long i;
```

```
for (i = 0; i < 6; i++){     // Saturation Alert
  if (pwm[i] > (ceiling - 10))
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
  pwm[i] = constrain(pwm[i],idle,ceiling);
  }
}
void actuator(){
motor1.writeMicroseconds(pwm[0]);
motor2.writeMicroseconds(pwm[1]);
motor3.writeMicroseconds(pwm[2]);
motor4.writeMicroseconds(pwm[3]);
motor5.writeMicroseconds(pwm[4]);
motor6.writeMicroseconds(pwm[5]);
}

void sendDigits(float number){
if (number<0){sign=15;}
else {sign=16;}
int numberInt=abs(number*10000.0);
for (int i=0;i<4;i++){
  digits[i]=numberInt % 10;
  numberInt/=10;
}
Serial3.write(sign);
Serial3.write(digits[3]);
Serial3.write(digits[2]);
Serial3.write(digits[1]);
Serial3.write(digits[0]);
}

float floatMap(float x, float inMin, float inMax, float outMin, float outMax)
  {
return (x-inMin)*(outMax-outMin)/(inMax-inMin)+outMin;
}
```

D.2 Base Station Code

```
1  //Code for HexaHopper Base Station (accept and send reference commands)
2  //RC transmitter talks to TX arduino/xbee and sends over wireless to //RX
     arduino/Xbee on HexaHopper
3  //Uses interrupts for reading PWM from RC transmitter
4  //Evan Mucasey
5  //7/19/12
6
7  #define pinKThetaA 0
8  #define pinKThetaDotA 1
9  #define pinThetaTrim 2
10 #define pinKPhiA 3
11 #define pinKPhiDotA 4
12 #define pinPhiTrim 5
13 #define pinKPsiA 6
14 #define pinKPsiDotA 8
15 #define pinPsiTrimA 7
16 int psiOffset=0;
17
18 //read PWM signals from an RC reciever and convert
19 byte throttleRxPin = 2;  //interrupt 0 pin
20 byte psiRxPin = 3; //interrupt 1 pin
21 byte thetaRxPin = 20;  //interrupt 3 pin
22 byte phiRxPin = 21;  //interrupt 2 pin
23
24 volatile int throttlePwmRx;  // store RC signal pulse length
25 byte throttlePwm;  // mapped value to be between 1000-1600
26 volatile int psiPwmRx;
27 byte psiPwm;
28 volatile int thetaPwmRx;
29 byte thetaPwm;
30 volatile int phiPwmRx;
31 byte phiPwm;
32 volatile int throttleStart;
33 int throttleReady;
34 volatile int psiStart;
35 int psiReady;
36 volatile int thetaStart;
```

```
int thetaReady;
volatile int phiStart;
int phiReady;

int kThetaA;
int kThetaDotA;
int thetaTrimA;
int kPhiA;
int kPhiDotA;
int phiTrimA;
int kPsiA;
int kPsiDotA;
int psiTrimA;

byte kTheta;
byte kThetaDot;
byte thetaTrim;
byte kPhi;
byte kPhiDot;
byte phiTrim;
byte kPsi;
byte kPsiDot;
byte psiTrim;

float startSend=555;
float thetaKalman;
float phiKalman;
float psiDotRaw;
float thetaD;
float phiD;
float psiDotD;
float assembledNumber;
byte *arrayOfBytes;
float packet[3];
byte readData[4];
boolean started=false;
boolean started2=false;
boolean ended=false;
boolean ended2=false;
```

```
int i=0;
int k=0;

byte sign;
byte digits[4];
float thetaSign;
float phiSign;
float psiDotSign;
byte inByte;

void setup() {
Serial.begin(57600);
Serial1.begin(57600);
Serial2.begin(19200);
Serial.flush();
Serial1.flush();
Serial2.flush();

//PWM inputs from RC receiver
pinMode(throttleRxPin, INPUT);
pinMode(psiRxPin, INPUT);
pinMode(thetaRxPin, INPUT);
pinMode(phiRxPin, INPUT);

attachInterrupt(0, getThrottlePwmRx, CHANGE);
attachInterrupt(1, getpsiPwmRx, CHANGE);
attachInterrupt(3, getthetaPwmRx, CHANGE);
attachInterrupt(2, getphiPwmRx, CHANGE);
}

void getThrottlePwmRx() {
// did the pin change to high or low?
if (digitalRead( throttleRxPin ) == HIGH)   {
  // store the current micros() value
  throttleStart = micros();
}
else{
// Pin transitioned low, calculate the duration of the pulse
```

```
114 throttlePwmRx = micros() - throttleStart; // may glitch during timer wrap-
      around
115 // Set flag for main loop to process the pulse
116 throttleReady = true;
117 }
118 }
119
120 void getpsiPwmRx() {
121 if (digitalRead( psiRxPin ) == HIGH)   {
122 psiStart = micros();
123 }
124 else   {
125 psiPwmRx = micros() - psiStart;
126 psiReady = true;
127 }
128 }
129
130 void getthetaPwmRx() {
131 if (digitalRead( thetaRxPin ) == HIGH)   {
132 thetaStart = micros();
133 }
134 else{
135 thetaPwmRx = micros() - thetaStart;
136 thetaReady = true;
137 }
138 }
139
140 void getphiPwmRx() {
141 if (digitalRead( phiRxPin ) == HIGH)   {
142 phiStart = micros();
143 }
144 else{
145 phiPwmRx = micros() - phiStart;
146 phiReady = true;
147 }
148 }
149
150 void loop() {
151 if (throttleReady) {
```

```
152   throttleReady = false;
153   if (throttlePwmRx<1000) throttlePwmRx=1000;
154   else if (throttlePwmRx>2000) throttlePwmRx=2000;
155   throttlePwm = map(throttlePwmRx, 1000, 2000, 0, 250);
156 }
157
158 if (psiReady) {
159   psiReady = false;
160   if (psiPwmRx<1000) psiPwmRx=1000;
161   else if (psiPwmRx>2000) psiPwmRx=2000;
162   psiPwm = map(psiPwmRx, 1000, 2000, 0, 250);
163   if (psiPwm>=115 && psiPwm<=135) psiPwm=125;
164 }
165
166 if (thetaReady) {
167   thetaReady = false;
168   if (thetaPwmRx<1000) thetaPwmRx=1000;
169   else if (thetaPwmRx>2000) thetaPwmRx=2000;
170   thetaPwm = map(thetaPwmRx, 1000, 2000, 0, 250);
171   if (thetaPwm>=115 && thetaPwm<=135) thetaPwm=125;
172 }
173
174 if (phiReady) {
175   phiReady = false;
176   if (phiPwmRx<1000) phiPwmRx=1000;
177   else if (phiPwmRx>2000) phiPwmRx=2000;
178   phiPwm = map(phiPwmRx, 1000, 2000, 0, 250);
179   if (phiPwm>=120 && phiPwm<=130) phiPwm=125;
180 }
181
182 thetaD=floatMap(float(thetaPwm),0,250,-0.25,0.25);
183 phiD=floatMap(float(phiPwm),0,250,-0.25,0.25);
184 psiDotD=floatMap(float(psiPwm),0,250,-.5,.5);
185
186 //Map to real world values:
187 Fz=floatMap(throttlePwm,1000,1650,0,27);
188 phi=floatMap(phiPwm,0,250,-1*phiMax,phiMax);
189 theta=floatMap(thetaPwm,0,250,-1*thetaMax,thetaMax);
190
```

```
kThetaA=analogRead(pinKThetaA);
kThetaDotA=analogRead(pinKThetaDotA);
thetaTrimA=analogRead(pinThetaTrim);
kPhiA=analogRead(pinKPhiA);
kPhiDotA=analogRead(pinKPhiDotA);
phiTrimA=analogRead(pinPhiTrim);
kPsiA=analogRead(pinKPsiA);
kPsiDotA=analogRead(pinKPsiDotA);
psiTrimA=analogRead(pinPsiTrimA);
kTheta=map(kThetaA,0,1023,0,250);
kThetaDot=map(kThetaDotA,0,1023,0,250);
thetaTrim=map(thetaTrimA,0,1023,0,250);
kPhi=map(kPhiA,0,1023,0,250);
kPhiDot=map(kPhiDotA,0,1023,0,250);
phiTrim=map(phiTrimA,0,1023,0,250);
kPsi=map(kPsiA,0,1023,0,250);
kPsiDot=map(kPsiDotA,0,1023,0,250);
psiTrim=map(psiTrimA,0,1023,0,250);

Serial1.write(254);
Serial1.write(throttlePwm);
Serial1.write(psiPwm);
Serial1.write(thetaPwm);
Serial1.write(phiPwm);
Serial1.write(kTheta);
Serial1.write(kThetaDot);
Serial1.write(thetaTrim);
Serial1.write(kPhi);
Serial1.write(kPhiDot);
Serial1.write(phiTrim);
Serial1.write(kPsi);
Serial1.write(kPsiDot);
Serial1.write(psiTrim);
Serial1.write(255);
delay(10);

getAttitudeData();
}
void getAttitudeData(){
```

```
while(Serial2.available()>15){
  inByte=Serial2.read();
  if (inByte==55){
    started=true;
    ended=false;
  }
  else if (inByte==77){
    ended=true;
    if(started&&ended){
      thetaKalman=thetaSign*(float(packet[0]/10.0) + float(packet[1]/100.0) +
         float(packet[2]/1000.0) + float(packet[3]/10000.0));
      phiKalman=phiSign*(float(packet[4]/10.0) + float(packet[5]/100.0) +
        float(packet[6]/1000.0) + float(packet[7]/10000.0));
      psiDotRaw=psiDotSign*(float(packet[8]/10.0) + float(packet[9]/100.0) +
        float(packet[10]/1000.0) + float(packet[11]/10000.0));
      started=false;
      ended=false;
     i=0;
     k=0;
     Serial2.flush();
    break;
    }
  }
  else if (started&&ended==false){
    if(i==0){
      if(inByte==16){thetaSign=1; i++;}
      else if (inByte==15) {thetaSign=-1; i++;}
    }
    else if (i==5){
      if(inByte==16){phiSign=1; i++;}
     else if (inByte==15) {phiSign=-1; i++;}
    }
    else if (i==10){
      if(inByte==16){psiDotSign=1; i++;}
      else if (inByte==15) {psiDotSign=-1; i++;}
    }
    else{packet[k]=inByte;k++;i++;}
    }
  }
```

```
266 }
267
268 void FloatSend(float FloatVar){
269   arrayOfBytes = (byte*)&FloatVar;
270   Serial.write(arrayOfBytes[0]);
271   Serial.write(arrayOfBytes[1]);
272   Serial.write(arrayOfBytes[2]);
273   Serial.write(arrayOfBytes[3]);
274 }
275
276 float floatMap(float x, float inMin, float inMax, float outMin, float outMax)
    {
277 return (x-inMin)*(outMax-outMin)/(inMax-inMin)+outMin;
278 }
```

D.3 Data Acquisition Code

```
//Data Acquisition Arduino Code
//Evan Mucasey
//4/1/2013

byte *arrayOfBytes;
float startSend=555;
float thetaKalman;
float thetaD;
float phiKalman;
float phiD;
float psiDotRaw;
float psiDotD;
float assembledNumber;

byte readData[4];
float packet[16];
boolean started=false;
boolean started2=false;
boolean ended=false;
boolean ended2=false;

int i=0;
int k=0;

float Fz;
byte testArray[4];
int n=0;
byte inByte;
int j=0;
float inFloat;
byte sign;
byte digits[4];

float thetaSign;
float thetaDSign;
float phiSign;
float phiDSign;
```

```
38
39 unsigned long startLoop;
40 unsigned long lastLoop;
41 unsigned long dtLoop;
42 unsigned long stdLoop=6666;
43
44 byte zero = 0;
45
46 void setup() {
47 Serial.begin(57600);
48 Serial1.begin(57600);
49 Serial.flush();
50 Serial1.flush();
51 }
52
53 void loop() {
54 getAttitudeData();
55
56 dtLoop=micros()-startLoop;   if (dtLoop<stdLoop){delayMicroseconds(stdLoop-
     dtLoop);}
57 lastLoop=micros()-startLoop;
58 startLoop=micros();
59 }
60
61 void getAttitudeData(){
62 while(Serial1.available()>15){
63   inByte=Serial1.read();
64   if (inByte==55){
65     started=true;
66     ended=false;
67   }
68   else if (inByte==77){
69     ended=true;
70     if(started&&ended){
71       thetaKalman=thetaSign*(float(packet[0]/10.0) + float(packet[1]/100.0) +
           float(packet[2]/1000.0) + float(packet[3]/10000.0));
72       thetaD=thetaDSign*(float(packet[4]/10.0) + float(packet[5]/100.0) +
          float(packet[6]/1000.0) + float(packet[7]/10000.0));
```

```
      phiKalman=phiSign*(float(packet[8]/10.0) + float(packet[9]/100.0) +
        float(packet[10]/1000.0) + float(packet[11]/10000.0));
      phiD=phiDSign*(float(packet[12]/10.0) + float(packet[13]/100.0) + float
        (packet[14]/1000.0) + float(packet[15]/10000.0));
     FloatSend(startSend);
      FloatSend(thetaKalman);
      FloatSend(thetaD);
      FloatSend(phiKalman);
      FloatSend(phiD);
      started=false;
      ended=false;
      i=0;
      k=0;
      break;
    }
  }
  else if (started&&ended==false){
    if(i==0){
      if(inByte==16){thetaSign=1; i++;}
      else if (inByte==15) {thetaSign=-1; i++;}
    }
    else if (i==5){
      if(inByte==16){thetaDSign=1; i++;}
      else if (inByte==15) {thetaDSign=-1; i++;}
    }
    else if (i==10){
      if(inByte==16){phiSign=1; i++;}
      else if (inByte==15) {phiSign=-1; i++;}
    }
    else if (i==15){
      if(inByte==16){phiDSign=1; i++;}
      else if (inByte==15) {phiDSign=-1; i++;}
    }
    else{packet[k]=inByte;k++;i++;}
    }
  }
}
void FloatSend(float FloatVar){
arrayOfBytes = (byte*)&FloatVar;
```

```
110 for (int j=0;j++;j<4){
111   if(arrayOfBytes[j]==0){arrayOfBytes[j]=zero;}
112 }
113 Serial.write(arrayOfBytes[0]);
114 Serial.write(arrayOfBytes[1]);
115 Serial.write(arrayOfBytes[2]);
116 Serial.write(arrayOfBytes[3]);
117 }
```

E MATLAB Code

E.1 Flight Data Acquisition Code

```
%Matlab File to Take Data from HexaHopper in Real Time
%Evan Mucasey
%Melissa Dye

%% Clear Data Space and Assign Variables
clear all
clc
freq=.0055;
numSamples=1000;
pitchCount=1;
rollCount=1;
time=[1*freq:1*freq:numSamples*freq];

%% Establish Serial Object
s = serial('COM5');
set(s,'BaudRate',57600);
fopen(s);
flushinput(s);
flushoutput(s);

%% Take Data Pitch
flushinput(s);
for i=1:numSamples
  u=fread(s,1,'single')
  [thetaKalman(i), thetaD(i), phiKalman(i), phiD(i)]=parse(u);
end

pThetaKalman(pitchCount,:)=thetaKalman(:);
pThetaD(pitchCount,:)=thetaD(:);
pPhiKalman(pitchCount,:)=phiKalman(:);
pPhiD(pitchCount,:)=phiD(:);

save('pThetaKalmanData','pThetaKalman')
save('pThetaDData','pThetaD')
save('pPhiKalmanData','pPhiKalman')
```

```
save('pPhiDData','pPhiD')

figure(1) plot(time,thetaKalman(count,:),'b-',time,thetaD(count,:),'r-')
title('Test Flight Pitch Test Theta Response');
xlabel('Time (s)')
ylabel('Theta (rad)')
legend('thetaKalman','thetaD')

figure(2)
plot(time, phiKalman(count,:),'b-',time,phiD(count,:),'r-')
title(' Test Flight Pitch Test Phi Response');
xlabel('Time (s)')
ylabel('Phi (rad)')
legend('phiKalman','phiD')

pitchCount=pitchCount+1;

%% Clear Port
fclose(s)
delete(s)
clear s

%% Take Data Roll
flushinput(s)
for i=1:numSamples
  u=fread(s,1,'single')
  [thetaKalman(i), thetaD(i), phiKalman(i), phiD(i)]=parse(u);
end

rThetaKalman(rollCount,:)=thetaKalman(:);
rThetaD(rollCount,:)=thetaD(:);
rPhiKalman(rollCount,:)=phiKalman(:);
rPhiD(rollCount,:)=phiD(:);

save('rThetaKalmanData','rThetaKalman')
save('rThetaDData','rThetaD')
save('rPhiKalmanData','rPhiKalman')
save('rPhiDData','rPhiD')

```

```
figure(3)
plot(time,thetaKalman(count,:),'b-',time,thetaD(count,:),'r-')
title('Test Flight Roll Test Theta Response');
xlabel('Time (s)')
ylabel('Theta (rad)')
legend('thetaKalman','thetaD')

figure(4)
plot(time, phiKalman(count,:),'b-',time,phiD(count,:),'r-')
title(' Test Flight Roll Test Phi Response');
xlabel('Time (s)')
ylabel('Phi (rad)')
legend('phiKalman','phiD')

rollCount=rollCount+1;
```

E.2 Data Smoothing and Time Constant Code

```
%Script for data analysis and for finding the time constant of the transient
  data
%Evan Mucasey
%6/20/12

clear
clc
s=input('data:')
t=input('time:')
voltage=s.data;
time=t.time;

span=1000;
window=ones(span,1)/span;
smoothedVoltage=convn(voltage,window,'same');
spanA=500;
windowA=ones(spanA,1)/spanA;
smoothedVoltageA=convn(smoothedVoltage,windowA,'same');
force=smoothedVoltageA*33672*0.453592;

%% Find Initial and Final Values and the time constant value
for i=1:25000
  initialValueArray(i)=force(i+499);
end initialValue=mean(initialValueArray);
for j=1:15000
  finalValueArray(j)=force(j+69999);
end
finalValue=mean(finalValueArray);
stepChange=finalValue-initialValue;
tauValue=.632*stepChange+initialValue;

%% Take derivative of signal where the step can be changed below
derivative=gradient(force,.0001);

%% find the time when the derivative is greater than a certain value ignoring
   first erroneous values
for l=5000:35000
```

```
  derivativeInitial(l)=derivative(l);
end
maxDerivativeInitial=max(derivativeInitial)+.1; %with buffer...
for m=5000:80000
  derivativeCheck=derivative(m);
    if derivativeCheck>maxDerivativeInitial
      timeStart=m;
      break
    end
end
%% Use the derivative time value found above to start the count to find where
   the force is at tauValue
for n=timeStart:1:80000
  forceCheck=force(n);
  if forceCheck>=tauValue
    timeStop=n;
    break
  end
end
tau=(timeStop-timeStart)/10000

figure(1)
plot(time,voltage,'y:',time,smoothedVoltageA,'r-')
xlabel('time (s)')
ylabel('voltage (v)')
title('Smoothing Raw Load Cell Data')
legend('Raw Data','Smoothed Data','location','NorthWest')
h=gtext('Time Constant = 0.1892 s');
```

Evan Mucasey has worked on several aerospace related projects throughout his undergraduate and graduate career at Lehigh University. His long term undergraduate project consisted of designing and fabricating an innovative craft to break the land speed record for a wind powered vehicle. Along with working towards the completion of this Master's Thesis, Mucasey has interned for Excalibur Almaz and SpaceX and plans to pursue a career in the aerospace industry.

Mucasey also received his private pilot license in 2009 and has been greatly enjoying flying over Texas and Pennsylvania, giving aerial tours to friends and family. Some other hobbies include studying guitar music and training in mixed martial arts.