Guidance, Navigation and Control System of a Hopper Spacecraft Simulator

By

Yu Hin Hau (Billy)

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering and Mechanics

Lehigh University

May 2015

CERTIFICATE OF APPROVAL

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

_____

Date

_____

Thesis Advisor, Dr. Terry J. Hart

_____

MEM Department Chair, Dr. Gary Harlow

ACKNOWLEDGMENT

I would like to express my most sincere thanks to Professor Terry Hart for his continuous support and providing many opportunities for me to develop skills both technically and professionally during my undergraduate and graduate studies at Lehigh University. The stories he tell about his experience flying as an Air Force fighter pilot and flying in space as an astronaut really inspire me. Having an astronaut as a professor and advisor make it seems that space is really not that far out of reach. Without his guidance, I would not have the necessary skills and experiences to pursue my dream in working at NASA for space exploration.

Furthermore, I would like to thank Dr. Andrew Abraham for being my mentor when I first joined the Space Hopper project in the summer of sophomore year. Without his leadership, I wouldn't have been exposed to the field of electronics, which enable me to lead development of the software and avionics systems today. Even though I was clueless back then and had no idea what's going on, he still patiently explain and let me experiment to learn. Later in senior year, he was my T.A. for the Spacecraft Systems Engineering course and taught me how to use AGI STK to plan orbital trajectory for interplanetary travel. This came in very handy now that I work in the space exploration industry.

Along that note, I would like to thank my managers, mentors, and colleagues at NASA Glenn Research Center for the opportunity to work at such a prestigious agency even though my formal academic background doesn't really match the job description. They are extremely patient in helping me to develop skills in electrical and electronics engineering. The projects that I was assigned to really help strengthen my weaknesses.

Next, I would like to thanks my partner in the Hopper Simulator project. Brian Wisniewski is a great friend and it was a pleasure to work with him even though we often disagree with each other. I enjoyed our time traveling to Penn State and the Mid-Atlantic Space Grant Consortium in Williamsburg to present our research. His knowledge and expertise in mechanical design contribute much to the project's success.

I would also like to thanks our undergraduate research student Chen Xi Liu for her assistance to the project in the past several semesters. She contributed a lot to the team by helping to calibrate lab equipment, collect test data, solder electronic circuits and repair the QuadX after crashes. She even went on to learn and implemented her own PID controller on the Single Axis Motor Balancer to verify that our understanding of the control law is logically correct.

In addition, credits should be given to Tian Li and Joachim Amoah for programming the ArduDAQ and the next generation ground control station in preparation for autonomous flight.

Special thanks must be given to Zachary Rambo who worked with me to start up the Lehigh Aerospace Club again in the beginning of sophomore year after the previous officers graduated without plans to maintain the club. He has a great passion in aviation and taught me a lot about RC aircraft systems. Without the projects that we did in the Aerospace Club, I wouldn't have the practical hands-on experience in designing and building RC aircrafts.

Shout out to Mr. Warwick, Mr. Bauer and Mr. Hawkins from Council Rock High School South. They play a very important role in developing my STEM career through a range of extracurricular activities such as design build fly a 5 ft. tall rocket and going to Science on Saturday seminars at Princeton Plasma Physics Lab. Furthermore, I must thank the Internet open source community whose members selflessly share their knowledge and skills freely to make the world a better place. Without individuals like them, I wouldn't be capable of doing most of what I know today.

Last but not least, I must thank my family for all that they have done to make me who I am today. Pertaining to aerospace alone, I thank my parents, Frank and Alice Hau for repeatedly bringing me to Kai Tak Airport when I was young because I was fascinated to see how those giant machines magically take off and soar freely in the sky. We used to go to the space museum in Tsim Sha Tsui all the time just because they got a mini staging rocket exhibition that look exactly like the Saturn V in Apollo 13. When I was around 8, we got to see the real thing by traveling to the US and visit Kennedy Space Center. Eventually, my parents decided to leave their business behind and immigrate to the US so that I can go chase after my then not-so-possible dream of working for NASA. Now thinking back, they really did a lot for me. As for my younger brothers, Morris and Sam, like all younger brothers, they are quite annoying. Though now in college, I actually miss them bugging me and keeping me company. I really enjoy those moments we spent together, going through challenges and face them together. We are all different, each of us have our strength and weaknesses. When we are working hand in hands, we can achieve great things.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The space hopper simulator project drew its origin from a partnership with Penn State University to compete in the Google Lunar XPRIZE competition. Lehigh University is tasked with the exploring the guidance, navigation and control (GN&C) system of the hopper spacecraft. To simulate the dynamics and flight behavior of the concept, Earth-based multirotor flying platforms were developed with the end goal of executing the hopping maneuver.

The overall project has been ongoing for more than 5 years and went through several major revisions to fix flaws discovered in the previous design. As older students graduate and new teams are form, knowledge and experience are lost in the process. Due to the time it take to relearn and redesign the simulators, the project progress only get as far as achieving radio controlled flight. The current and 3rd generation development team aims to change that by developing both the hardware and software using modular design.

With modular design, the manufacturing, repair and modification process for the multirotor speed up significantly. The damaged component can be replaced with little effort. In addition to the hardware advantages, the software modules enable concurrent development of both a PID and a Fuzzy Logic based flight control system using similar avionics and software architecture. Since the flight operating system function by linking the various software modules, individual modules can easily be swapped to test different control laws, electronic devices, etc. The software modules are also capable of being reused in other applications, such as running the thrust test stand and logging data with the wireless ground station.

In theory and simulation, the GN&C system is quite simple. The hopping guidance trajectory can be generated by a set of linear and trigonometric equations. The trajectory can be optimized by

minimizing the total energy consumption at the end of the hopping maneuver. The navigational data can be collected from the GPS and localized for the cascade PID controllers to achieve the desired trajectory. In the ideal world, everything is simple and easy.

In the real world, a range of problems arise during implementation. Factors such as time delay and noises significantly impact the performance of the control system, making stable aggressive tuning very difficult to achieve. In an attempt to improve the condition, a number of digital filters such as the moving average filter and the Kalman filter were explored. In addition, every sub-system was analyzed in depth to optimize for speed. This resulted in 3 major revisions in changing flight computer and programming languages.

Even though the main topic of this research is the guidance, navigation and control system, the project quickly expanded into a systems engineering problem. Everything must work well together in order for the aircraft to achieve stable flight.

Chapter 1: PROJECT BACKGROUND OVERVIEW

The foundation of the Hopper Spacecraft Simulator project drew its origin from the Google Lunar XPRIZE Competition. Lehigh University and Pennsylvania State University formed a partnership in collaborating in the mission of sending a robot to the moon. Lehigh University is tasked the guidance, navigation and control system while the Penn State Applied Research Lab and students in the Penn State Lunar Lion team lead development in the remainder of the spacecraft along with logistics such as public relation and fundraising.

GOOGLE LUNAR XPRIZE COMPETITION

The Google Lunar XPRIZE (GLXP) Competition is an international space technology competition organized by the X Prize foundation and sponsored by Google. The competition was announced on September 13, 2007. The mission is for privately funded teams to safely land a robot on the Moon, maneuver 500 meters across the Lunar surface and transmit data such as images and video back to Earth. There are additional awards for spacecraft that travel greater than 5000 meters, capture images of legacy Apollo program hardware, surviving a lunar night, etc.

The Lunar Lion team along with Lehigh University decided to enter the competition and conducted a mission analysis with the COMPASS team at NASA Glenn Research Center (GRC) to investigate the space hopper concept. The Collaborative Modeling for Parametric Assessment of Space Systems (COMPASS) is an interdisciplinary team that drew engineers from various divisions and branches across GRC. The team initially layout the overall architecture of the spacecraft design and mission, including areas such as mechanical system, thermal, propulsion, power, GN&C and communications. The cost, mass, power and delta V budget are iterated as the design changes to meet the mission specifications.

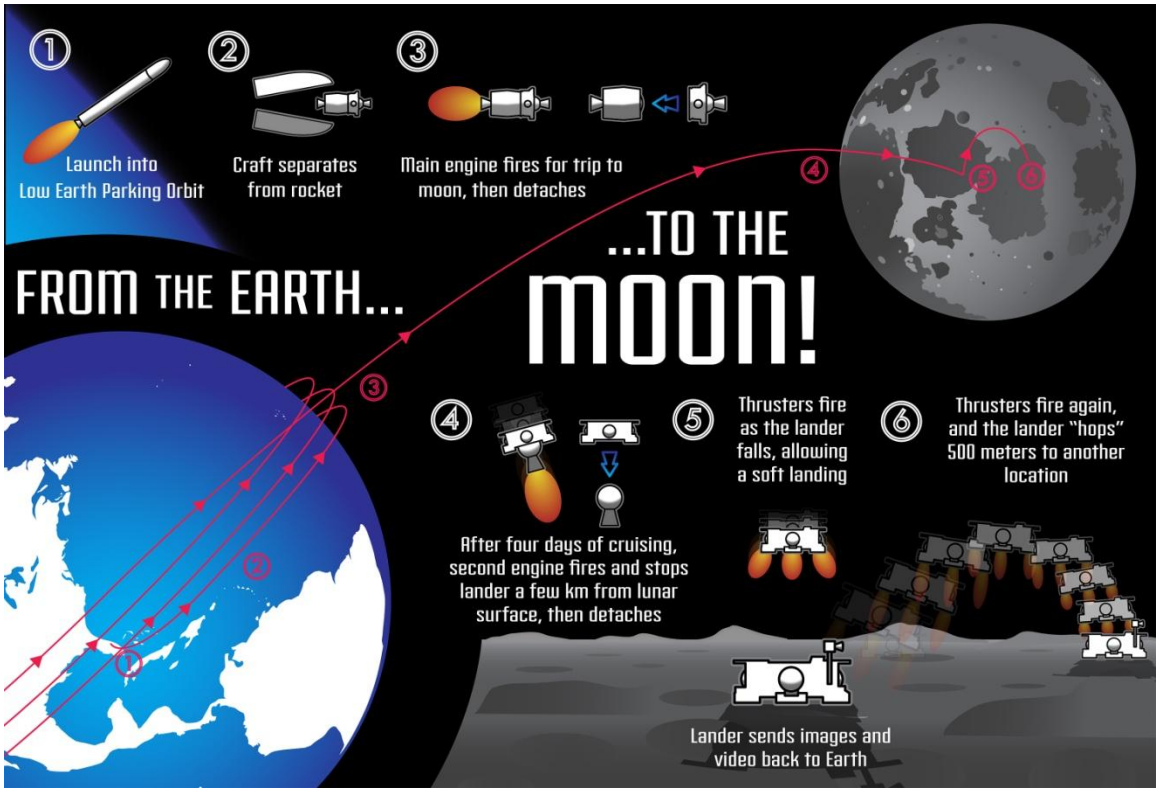


FIGURE 1-1: LUNAR LION MISSION CONCEPT OF OPERATION

The Lunar Lion mission concept of operations (ConOps) is illustrated in Figure 1-1 based on the collaborative studies with COMPASS. In general, the spacecraft will be lifted into a parking orbit by a launch vehicle, possibly SpaceX Falcon 9, before using the main engine from the transfer vehicle to enter a lunar transfer orbit. After cursing for 4 days, the space hopper's main engine along with the thrusters will be used for a powered descends onto the lunar surface. After capturing and transfer some visual data back to Earth, the space hopper will execute a hop to its destination 500 meters away using its thrusters.

There are many robots currently exploration the solar systems. Almost all of the mobile robotic platforms take the form of rovers. A disadvantage of rovers is that they can easily become damaged or stuck on the surface due to rocks, uneven terrain and soft soil. A famous example is

NASA's Spirit rover, which ended its mission in around 7 years while its identical twin rover, Opportunity, is still going on to explore Mars. The hopper spacecraft overcome this problem by flying over the rough terrain and going directly to the points of interest. The tradeoff is that due to the lack of thick atmospheres in celestial bodies like the Moon and Mars, propulsion can only be done with thrusters. As oppose to the renewable electric energy, which can be recharged from solar panels, thruster consume the limited amount of fuel onboard the spacecraft. Hence, only a limited number of maneuvers can be executed. For the GLXP, this is a perfect choice since it only requires the spacecraft to do one maneuver to a location 500 meters away.

1ST GENERATION HOPPER SIMULATOR

At its current stage, the hopper spacecraft simulator project went through 3 major revisions. The previous thesis and reports resulted from this project have detail documentation on technical matters such as the target specifications and requirements of each revision. Therefore, the section here will only briefly discuss the history and overview of each generation.

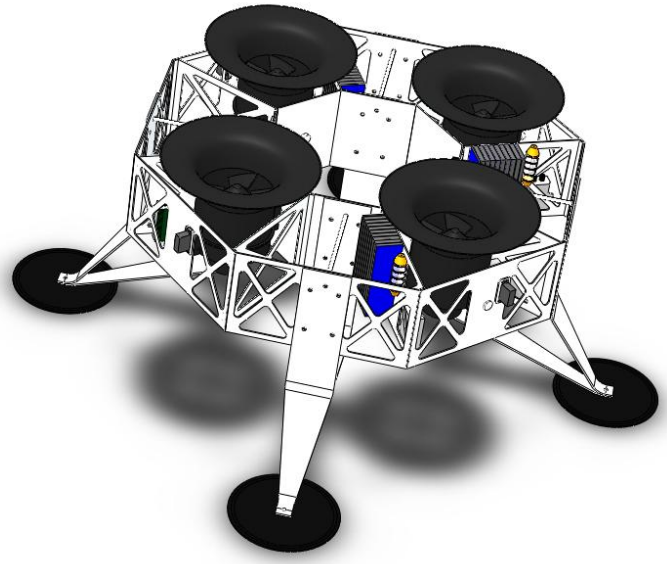


FIGURE 1-2: 1ST GENERATION HOPPER SIMULATOR CAD MODEL

The first generation hopper spacecraft simulator, as shown in Figure 1-2, began as part of an undergraduate student Integrated Product Development course project. The hardware and project was later transferred to Andrew Abraham, a PhD student studying orbital mechanics, who led a team consisted of several graduate and undergraduate students including Evan Mucasey, Anthony Dzaba, Zachary Rambo, Nick Graber and myself. The project provided great cross-disciplinary research opportunities for students for the simulator involve not only mechanical system, but also electrical, power and software development as well.

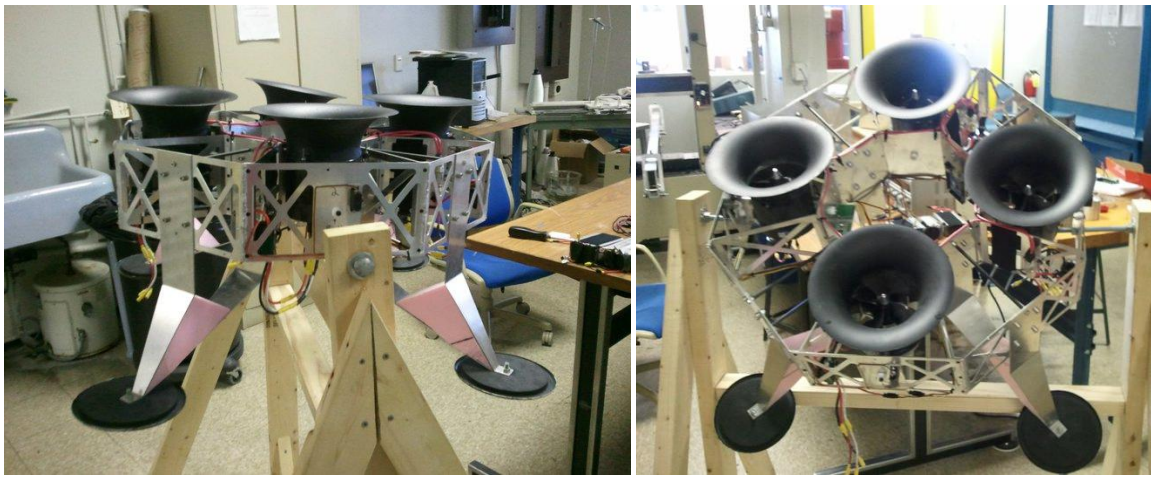


FIGURE 1-3: 1ST GENERATION HOPPER SIMULATOR ON SINGLE-AXIS TEST RIG

As shown in Figure 1-2, the first generation hopper was manufactured out of aluminum. The four ducted fan were also partially manufactured in-house with carbon fiber and provide approximately 4.5 kg of thrust each. The system is powered by four 37 volts lithium polymer (LiPo) batteries wired in parallel and weigh approximately 22.5 kg in total.

One of the unique designs in the first generation hopper simulator is its thrust vectoring system. Each ducted fan is connected to a servo, allowing it to rotate in a single axis. Theoretically, the thrust vectoring system could allow the aircraft to maneuver while maintaining level flight. The

level flight capability could be advantageous in situations like capturing image and video. The thrust vectoring system later became part of Anthony Dzaba's PhD dissertation research.

One of the major design flaws of the first generation hopper is the thrust to weight ratio. The four motor can only output a maximum thrust of roughly 18 kg, however, the total aircraft weigh around 22.5 kg. Therefore, the thrust to weight ratio is only about 1.25:1. Along with thrust vectoring, this cause the actuator to saturate as the overall system loose controllability.

Another problem is the vibration caused by the ducted fan. Since the fans are partially manufactured by students, the qualities were not the best. Some of the fan blades were not balance correctly hence while operating sometime cut into the side wall. In one case during the test, the vibration was so great that the top of the ducted fan broke off and flew across the lab.

The power consumed by the fans was also enormous. The system is regularly drawing the maximum 12.8 kW of power, due to the low thrust to weight ratio. This requires the batteries to have high capacity and output at a dangerously high current at around 345 Amps in total. There are several incidents where solder joints were melted and accident shorts causing power failure. Due to the many design flaws of the first generation hopper, the second generation hoppers were created.

2ND GENERATION HOPPER SIMULATOR

The second generation hopper simulator was an effort to improve the previous generation led by Evan Mucasey along with other graduate and undergraduate students including Melissa Dye, Nick Tashjian and Andrew Papazian.

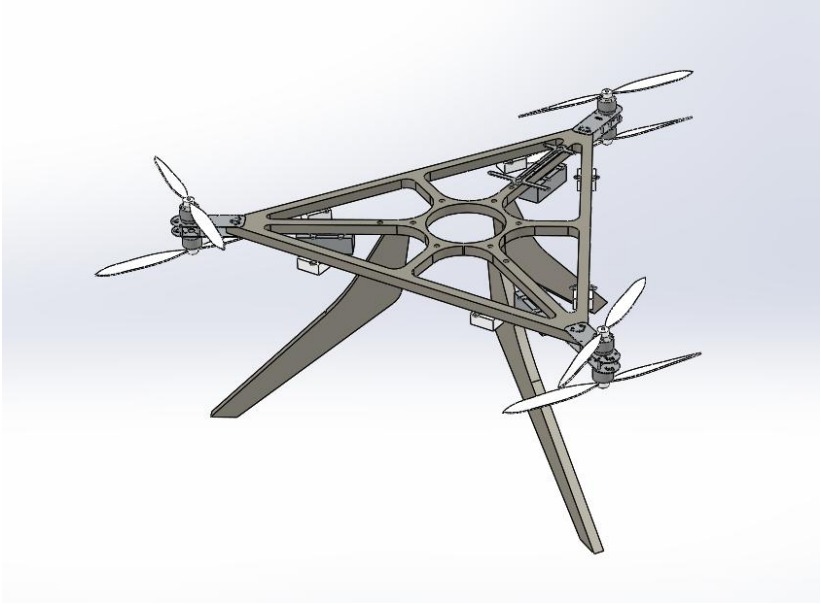


FIGURE 1-4: 2ND GENERATION HOPPER SIMULATOR CAD MODEL

In order to address the first generation's thrust to weight ratio problem, the second generation hopper was constructed primarily with composite material in order to reduce the overall mass. The propulsion system was also transitioned from a 4 tilt-rotor design to a 3-point fixed rotor design to reduce system complexity and reflect development changes in Penn State's hopper spacecraft. As shown in Figure 1-3, the rotors are set up in 3 groups of 2 counter rotating motors. This allow more authority in yaw control and overall provide more thrust.

In order to test the new flight configuration and gain experience in the manufacturing the hopper with composite material, a proof-of-concept hexacopter was constructed. This prototype simulator later became the primary experimental platform during the early stages of the current project team for learning the basics of controls and system integration. A picture of the 2nd generation prototype was included in Figure 1-4 in the modified test rig.

FIGURE 1-5: 2ND GENERATION PROOF-OF-CONCEPT HOPPER SIMULATOR

The manufacturing process for the 2nd generation simulators was rather complicated and time consuming. This involved cutting the primary structure out of a rigid PVC foam, while wet-laying unidirectional carbon and fiberglass tape with epoxy. In addition, the epoxy was mixed with micro glass spheres and brushed into the foam to ensure that the surface was filled. Finally, the entire structure was vacuum cured. An image of the completed 2nd generation hopper simulator can be found in Figure 1-5.

The overall mass of the hexacopter was around 4 kg. The 6 AXL 2826/12 motors from the aircraft are able individually output a thrust of 2.6 kg, combining to a total of 15.6 kg of thrust. This yield a 3.90:1 thrust to weight ratio.

In order for the flying platform to carry additional payloads, the hexacopter is designed so that a JetCat P200 gas turbine engine can be attached to the center of aircraft as seen in the circular cutaway in Figure 1-5. The P200 jet engine weigh around 2.8 kg and can output 23.5 kg of thrust. Along with 5 kg of additional scientific payload, the 2nd generation hopper was estimated to have a combine thrust to weight ratio of 3.12:1.

FIGURE 1-6: 2ND GENERATION HOPPER SIMULATOR

In the end of the development phase, the second generation simulator was a success in that controlled flight was achieved via radio command with reasonable flight time. However, additional problems arise even though design flaws in the first generation hopper were mostly corrected.

The composite materials used in the hexacopter were rather rigid and can easily be damaged. As the material starting to wear, repair became very difficult due to the lengthy manufacturing process and that the primary structure was essentially one single part. Even if one of the arms was damaged, then it will be necessary to manufacture an entirely new frame. In addition, due to the design's high center of mass, it was easy for the aircraft to tumble over or damage its landing struts during landing. Unfortunately, the jet engine was also never used in flight due to the difficulties in implementation and the hazards of carrying jet fuel onboard in a lab and outdoor campus test.

3RD GENERATION HOPPER SIMULATOR

The third and current generation project team was led by Brian Wisniewski and myself. The undergraduate members include Chen Xi Liu, Joachim Amoah, and Trevor Hayes. The team's goal is to redesign the simulators with "modular" components in both hardware, electronics and software so that they can easily be repair, replace and reuse. The flight configuration had reverted back to the four thruster configuration due to Penn State's development changes, hence, the Quad+ (Figure 1-6) and QuadX flying platform were created. In addition, the team had begun development in the guidance and navigation system, hoping to achieve the hopping maneuver at the end of the research.

While Brian Wisniewski focused on developing enhanced landing struts and investigating the fuzzy logic control system for the Quad+ platform, I focused on developing the avionics systems and implementing the commonly used PID controller on the QuadX platform. The modular software architecture allow similar avionics and flight operating systems be used on the two flying platforms, swapping only the Attitude Controller module and other customary settings.

FIGURE 1-7: 3RD GENERATION HOPPER SIMULATORS (QUAD+)

Chapter 2: SPACE HOPPER SIMULATOR

MULTIROTOR FLYING PLATFORM

In order to physically test the guidance, navigation and control (GNC) system for the space hopper, an Earth based simulator is necessary. The simulator should be safe, low cost and have similar dynamics. Hence, the multirotor flying platforms are created.

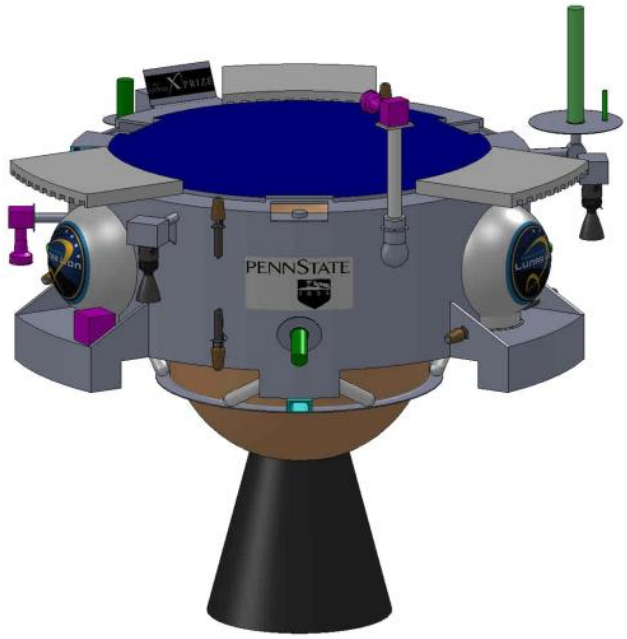


FIGURE 2-1: PENN STATE LUNAR LION HOPPER SPACECRAFT DESIGN

The 3-thrusters configuration of the Penn State Lunar Lion Hopper Spacecraft is illustrated in Figure 2-1. In an overview, the spacecraft is consisted of a main rocket engine for a powered descends to the lunar surface. There are 3 sets reaction control thrusters along the body of the spacecraft to control attitude and the hopping maneuver from the landing site to another point of interest.

The hopper simulator flying platforms simulate the space hopper by taking the form of a multirotor aircraft. The thrusters are simulated by brushless motors and propellers for their similar pulsing operation control. The main rocket engine is simulated by a jet turbine. However, it is later determined that using a jet engine to test control design is not feasible for the difficulty in implementation and hazardous nature.

FLIGHT CONFIGURATIONS

The original design of the hopper spacecraft is consisted of 4 sets of thrusters. The design is later revised to a 3-set configuration but revert back to a 4-set design in the Fall of 2013. Several multirotor flight configurations are illustrated in Figure 2-2 to reflect the changes in design. Through the implementation of a Flight Mixer, it is possible to develop the GNC system independent of the changes flight configuration.

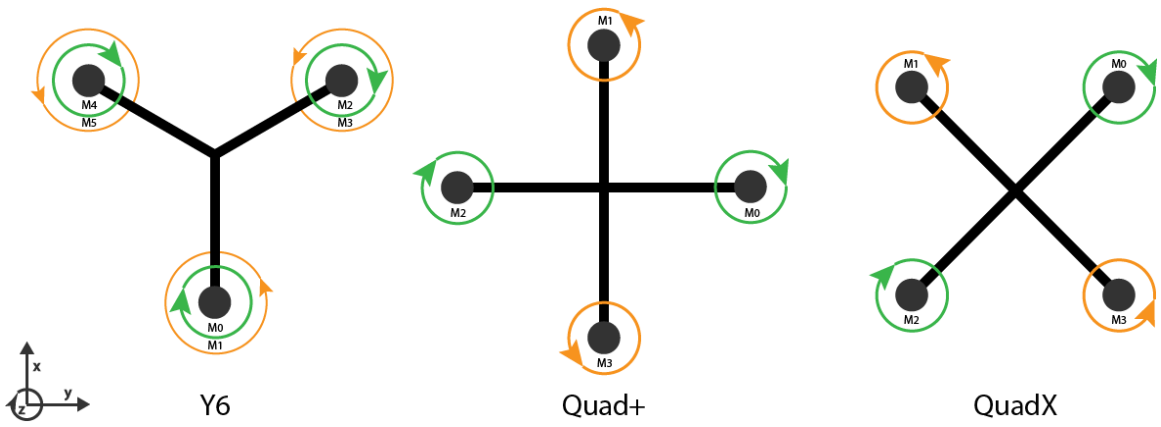


FIGURE 2-2: HOPPER SIMULATOR FLIGHT CONFIGURATIONS

Using the coordinate frame illustrated on the bottom left corner of Figure 2-2, one can define the roll, pitch and yaw motion on the flying platforms. With the right hand rule denoting positive direction, the motion around the x axis is defined as roll. On the other hand, motion around the y axis is pitch and the motion around the z axis is yaw.

The 2nd generation flying platform was an Y6 flight configuration. Even though the focus of the 3rd generation simulators are Quad configurations, some work was done on the 2nd generation prototype simulator before the new flying platforms were designed and built.

The Y6 flight configuration is consisted 3 sets of 2 counter rotating motors and props. From top view, the motors on top (M0, M2, M4) are rotating clockwise (CW) while the motors on the bottom (M1, M3, M5) are rotating counter-clockwise (CCW). Assuming that the motor blow into the page, it can be seen that M4 and M5 contribute to a positive roll motion while M2 and M3 contribute to a negative roll motion. As for pitch, M2, M3, M4 and M5 all contribute to a positive pitch while only M0 and M1 contribute to a negative pitch. In general, CW motors contribute to a positive yaw motion while the CCW motors contribute to a negative yaw motion.

The Quad+ is relatively simpler with only a pair of motors, M0 and M2, controlling roll. A separate pair, M1 and M3, are used to control pitch. Therefore, the roll and pitch motion are not coupled by the motors.

The QuadX configuration is a bit more complicated since the roll and pitch motions are coupled like the Y6. Positive roll is achieved by increasing thrusts in M1 and M2 while decreasing thrust in M0 and M3. On the other hand, positive pitch is achieved by increasing thrusts in M0 and M1 while decreasing thrust in M2 and M3.

In comparison, it is expected that the Y6 configuration will produce better dynamics response due to the additional thrusts and torque produce by the 2 extra actuators. However, the motor mount is more difficult to design due to the two counter-rotating motors at each arm. While the Quad configurations are less capable, they are simpler to design and manufacture. The Quad+ is probably the simplest flight configuration and easiest to control. However, QuadX is more suitable for situations like adding onboard photography. The front of the aircraft will have plenty of space to allow a wide view free of obstruction.

Flight Dynamics

Inertial and Body Frame

The derivation of the equations of motion starts with the definition of coordinate frames. The inertial frame along with the resulting body frame after going through the yaw, pitch and roll rotation is illustrated in Figure 2-3.

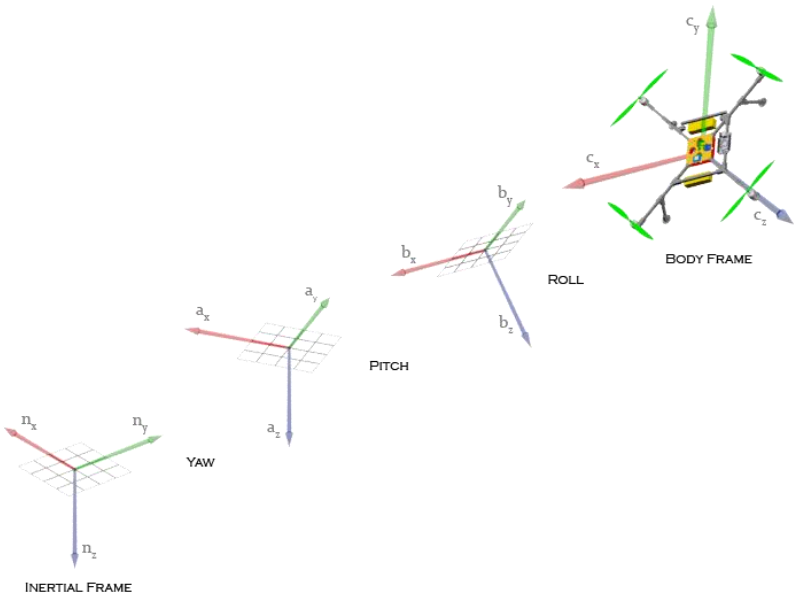


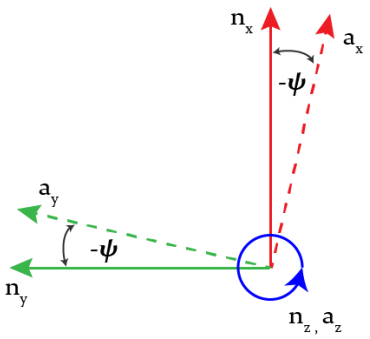
FIGURE 2-3: EULER YAW-PITCH-ROLL ROTATION

The inertial frame is a coordinate frame others referenced to. In an Earth-based aircraft, it most sense to fix the inertial frame relative to Earth's surface while aligning its unit vectors $\hat{n}_x$, $\hat{n}_y$, $\hat{n}_z$ toward North, East and the center of Earth respectively. The body frame is the coordinate frame that is fixed on the aircraft with $\hat{c}_x$, $\hat{c}_y$, $\hat{c}_z$ aligns with the aircraft's front, right and bottom.

In many cases, it is necessary to relate the body frame to the inertial frame. For example, calculating how gravity is acting on the aircraft. It is difficult to directly relate the body frame to the inertial frame in full 3D orientation. Fortunately, through the use of Euler's rotation theorem, the attitude can be broken down in 3 separate simple rotation matrices and later multiply together to obtain the full 3D rotation matrix.

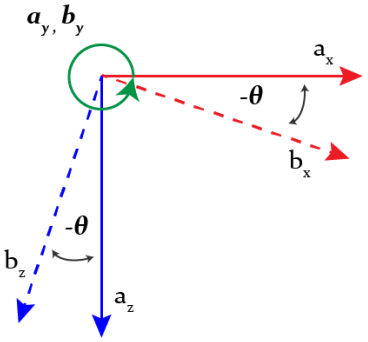
In aerospace, it is common practice to express attitude in order of yaw, pitch, then roll with respect to the inertial frame. This is sometime referred to as the Euler ZYX or the Euler YPR rotation. The 3 simple rotations frame for Figure 2-3 are illustrated below along with rotation matrices. Note that since the rotation in Figure 2-3 are in the negative direction. Hence, the acute angle between the two frames is negative. The angle used in the matrices should be the negative angle. The trigonometric identities, $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$, are used to simplify the matrices.

Yaw Rotation Matrix (ψ)



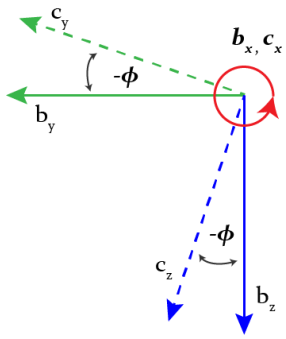
${}^{a}C^{n}$	$\hat{n}_x$	$\hat{n}_y$	$\hat{n}_z$
$\hat{a}_x$	$\cos(\psi)$	$\sin(\psi)$	0
$\hat{a}_y$	$-\sin(\psi)$	$\cos(\psi)$	0
$\hat{a}_x$	0	0	1

Pitch Rotation Matrix (θ)



${}^{b}C^{a}$	$\hat{a}_x$	$\hat{a}_y$	$\hat{a}_z$
$\hat{b}_x$	$\cos(\theta)$	0	$-\sin(\theta)$
$\hat{b}_y$	0	1	0
$\hat{b}_x$	$\sin(\theta)$	0	$\cos(\theta)$

Roll Rotation Matrix (ϕ)



${}^{c}C^{b}$	$\hat{b}_x$	$\hat{b}_y$	$\hat{b}_z$
$\hat{c}_x$	1	0	$0)$
$\hat{c}_y$	0	$\cos(\phi)$	$\sin(\phi)$
$\hat{c}_x$	0	$-\sin(\phi)$	$\cos(\phi)$

Using the simple rotation matrices above the body frame c can be related to the inertial frame n with the following equation to produce a 3D Euler rotation matrix:

$$ {}^{c}C^{n} = {}^{c}C^{b} * {}^{b}C^{a} * {}^{a}C^{n} $$

After multiplying the simple matrices together the full 3D rotation matrix is obtained. Due to the number of terms involve in the matrix, the *cos* and *sin* operators are simplified to *c* and *s*.

$$ {}^{c}C^{n} = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ c\psi s\theta s\phi - c\phi s\psi & c\phi c\psi + s\theta s\phi s\psi & c\theta s\phi \\ s\phi s\psi + c\phi c\psi s\theta & c\phi s\theta s\psi - c\psi s\phi & c\theta c\phi \end{bmatrix} $$

The ${}^{c}C^{n}$ rotation matrix can be used to transform a vector from inertia frame to the body frame.

$$\begin{bmatrix} \hat{c}_x \\ \hat{c}_y \\ \hat{c}_z \end{bmatrix} = \quad {}^{c}C^{n} * \begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \end{bmatrix}$$

If it is desired to transform a vector from body frame to inertia frame, then the ${}^{c}C^{n}$ matrix can be transposed obtain to the ${}^{n}C^{c}$ matrix.

$${}^{n}C^{c} = \quad {}^{c}C^{n^T}$$

$$\begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \end{bmatrix} = \quad {}^{n}C^{c} * \begin{bmatrix} \hat{c}_x \\ \hat{c}_y \\ \hat{c}_z \end{bmatrix}$$

FREE BODY DIAGRAM

The free body diagram of the QuadX flying platform is illustrated in Figure 2-4. The primary forces exerted internally by the aircraft are the thrusts generated by the four motors. The differences in these thrust forces along with yaw axis torque generated by the propellers will eventually determine the 3 axis torques acting upon the aircraft's center of mass. Externally, gravitation force is acting on the aircraft. In order to simulate lunar environments, external forces such as atmospheric drag and wind are disregarded.

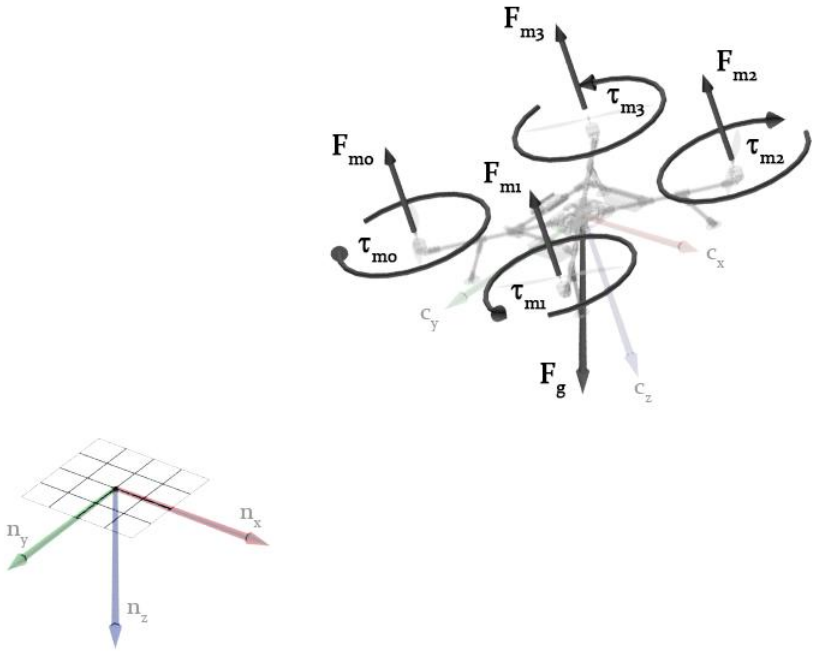


FIGURE 2-4: QUADX SPACE HOPPER SIMULATOR FREE BODY DIAGRAM

EQUATIONS OF MOTION

The translation and rotational dynamic equations of motion for the space hopper simulator is derived using the Newton-Euler method with respect to the body frame *c*.

$$\begin{bmatrix}\mathbf{F}\\ \boldsymbol{\tau}\end{bmatrix}_c = \begin{bmatrix}m\mathbf{I} & 0\\ 0 & \mathbf{I_{cm}}\end{bmatrix}_c \begin{bmatrix}\mathbf{a_{cm}}\\ \boldsymbol{\alpha}\end{bmatrix}_c + \begin{bmatrix}0\\ \boldsymbol{\omega} \times \mathbf{I_{cm}}\boldsymbol{\omega}\end{bmatrix}_c$$

In the equation above, the force vector $\boldsymbol{F}$ is the net force acting on the flying platform, this is consisted of the thrust generated by the motors and the aircraft's weight. The torque vector $\boldsymbol{\tau}$ are the roll, pitch and yaw torque resulted from the different thrust forces generated by each motor. The variable m is the aircraft's mass while the matrix $\boldsymbol{I}$ is an identity matrix. The inertia matrix $\boldsymbol{I_{cm}}$ contain the 3 principal inertia with respect to the simulator's center of mass. The vector $\boldsymbol{a_{cm}}$ contain the translation accelerations in the body frame while the vector α contain the rotational accelerations. Finally, the vector $\boldsymbol{\omega}$ is the rotation velocity of the aircraft.

ROTATIONAL DYNAMICS

To simplify the derivation process, the Newton-Euler is divided into the translational and rotational portion. Translational Euler's equation is as follow:

$$\boldsymbol{\tau_c} = \boldsymbol{I_{cm}\alpha_c} + \boldsymbol{\omega_c} \times \boldsymbol{I_{cm}\omega_c}$$

With variables defined as:

$$\boldsymbol{\tau_c} = [\tau_\phi \quad \tau_\theta \quad \tau_\psi]_c^{\mathrm{T}}$$

$$\boldsymbol{I_{cm}} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}_c$$

$$\boldsymbol{\alpha_c} = [\ddot{\phi}_c \quad \ddot{\theta}_c \quad \ddot{\psi}_c]_c^{\mathrm{T}}$$

$$\boldsymbol{\omega_c} = [\dot{\phi}_c \quad \dot{\theta}_c \quad \dot{\psi}_c]_c^{\mathrm{T}}$$

Rearranging the equation and substituting the variables into Euler's equation:

$$\boldsymbol{I_{cm}\alpha_c} = \boldsymbol{\tau_c} - \boldsymbol{\omega_c} \times \boldsymbol{I_{cm}\omega_c}$$

$$\boldsymbol{I_{cm}\alpha_c} = \boldsymbol{\tau_c} - \boldsymbol{\omega_c} \times \boldsymbol{I_{cm}\omega_c}$$

$$\boldsymbol{\alpha_c} = \boldsymbol{I_{cm}^{-1}}\,(\boldsymbol{\tau_c} - \boldsymbol{\omega_c} \times \boldsymbol{I_{cm}\omega_c})$$

$$\begin{bmatrix} \ddot{\phi}_c \\ \ddot{\theta}_c \\ \ddot{\psi}_c \end{bmatrix}_c = \begin{bmatrix} {}^1/_{I_{xx}} & 0 & 0 \\ 0 & {}^1/_{I_{yy}} & 0 \\ 0 & 0 & {}^1/_{I_{zz}} \end{bmatrix}_c * \left(\begin{bmatrix} \tau_{\phi_c} \\ \tau_{\theta_c} \\ \tau_{\psi_c} \end{bmatrix}_c - \begin{bmatrix} \dot{\phi}_c \\ \dot{\theta}_c \\ \dot{\psi}_c \end{bmatrix}_c \times \begin{bmatrix} I_{xx}\dot{\phi}_c \\ I_{yy}\dot{\theta}_c \\ I_{zz}\dot{\psi}_c \end{bmatrix}_c \right)$$

Solving the cross product:

$$\begin{bmatrix}\dot{\phi}_c\\ \dot{\theta}_c\\ \dot{\psi}_c\end{bmatrix}_c \times \begin{bmatrix}I_{xx}\dot{\phi}_c\\ I_{yy}\dot{\theta}_c\\ I_{zz}\dot{\psi}_c\end{bmatrix}_c = \begin{vmatrix}\hat{c}_x & \hat{c}_y & \hat{c}_z\\ \dot{\phi}_c & \dot{\theta}_c & \dot{\psi}_c\\ I_{xx}\dot{\phi}_c & I_{yy}\dot{\theta}_c & I_{zz}\dot{\psi}_c\end{vmatrix}$$

$$= \left(I_{zz} - I_{yy}\right)\dot{\theta}_c\dot{\psi}_c\,\hat{c}_x - \left(I_{zz} - I_{xx}\right)\dot{\phi}_c\dot{\psi}_c\,\hat{c}_y + \left(I_{yy} - I_{xx}\right)\dot{\phi}_c\dot{\theta}_c\,\hat{c}_z$$

And finally plugging the cross product result back into the rearranged equation, the rotational dynamics equations are obtained in the body frame:

$$\boxed{\ddot{\phi}_c = \frac{\tau_{\phi_c}}{I_{xx}} + \frac{I_{yy} - I_{zz}}{I_{xx}}\dot{\theta}_c\dot{\psi}_c}$$

$$\boxed{\ddot{\theta}_c = \frac{\tau_{\theta_c}}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}}\dot{\phi}_c\dot{\psi}_c}$$

$$\boxed{\ddot{\psi}_c = \frac{\tau_{\psi_c}}{I_{zz}} + \frac{I_{xx} - I_{yy}}{I_{zz}}\dot{\phi}_c\dot{\theta}_c}$$

The main drivers for the rotational acceleration in the equations above are torques in the roll, pitch and yaw direction. As discussed in earlier sections, different flight configurations will result in a different combination of motors and prop rotation scheme in order to achieve those torque. Since the focus of this thesis is the QuadX flight simulator, the torques for the QuadX flight configuration will be derived below. However, this portion can easily be modified, hence separating the development of GNC and the flight configuration.

Roll (+)		Roll (-)	
$Motor\ 2$	$Motor\ 3$ F_{m3}	$Motor\ 0$ F_{m0}	$Motor\ 1$ F_{m1}

Pitch (+)		Pitch (-)	
$Motor\ 1$	$Motor\ 2$ F_{m2}	$Motor\ 0$ F_{m0}	$Motor\ 3$ F_{m3}

Yaw (+)		Yaw (-)	
$Motor\ 0$ τ_{m0}	$Motor\ 2$ τ_{m2}	$Motor\ 1$ τ_{m1}	$Motor\ 3$ τ_{m3}

TABLE 2-1: QUADX FLIGHT CONFIGURATION MOTOR FORCE AND TORQUE CONTRIBUTION TABLE

Using the data in Table 2-1, the torque for the roll, pitch and yaw axis can be calculated. Let the variable l_x and l_y be the distance of the motors away from the center of mass in the $\hat{c}_x$ and $\hat{c}_y$ direction respectively.

$$\tau_{\phi_c} = l_y\ [(F_{m2} + F_{m3}) - (F_{m0} + F_{m1})]$$

$$\tau_{\theta_c} = l_x\ [(F_{m1} + F_{m1}) - (F_{m0} + F_{m3})]$$

$$\tau_{\psi_c} = \tau_{m0} + \tau_{m1} + \tau_{m2} + \tau_{m3}$$

TRANSLATIONAL DYNAMICS

Newton's equation for the translational dynamics is as follow:

$$\boldsymbol{F_c} = m\, \boldsymbol{I}\, \boldsymbol{a_{cm}}$$

Let's take a look at the force acting on the quadcopter in the Free Body Diagram in Figure 2-4. The forces in the simplified model are the motor thrust in the $-\hat{c}_z$ direction and the gravitation force in the $\hat{n}_z$ direction. Let T_c be the total thrust in the body frame:

$$T_c = F_{m0} + F_{m1} + F_{m2} + F_{m3}$$

$$\boldsymbol{F_c} = \begin{bmatrix} 0 \\ 0 \\ -T_c \end{bmatrix}_c + \begin{bmatrix} 0 \\ 0 \\ F_g \end{bmatrix}_n$$

Using the rotation matrix ${}^{c}C^{n}$, the total acting force in the body frame is:

$$\boldsymbol{F_c} = \begin{bmatrix} 0 \\ 0 \\ -T_c \end{bmatrix}_c + {}^{c}C^{n} \begin{bmatrix} 0 \\ 0 \\ F_g \end{bmatrix}_n$$

$$\boldsymbol{F_c} = \begin{bmatrix} 0 \\ 0 \\ -T_c \end{bmatrix}_c + \begin{bmatrix} -F_g s\theta \\ F_g c\theta s\phi \\ F_g c\theta c\phi \end{bmatrix}_c$$

$$\boldsymbol{F_c} = \begin{bmatrix} -mgs\theta \\ mgc\theta s\phi \\ mgc\theta c\phi - T_c \end{bmatrix}_c$$

Let x_c , y_c , z_c be the displacements in the $\hat{c}_x$, $\hat{c}_y$, $\hat{c}_z$ direction respectively:

$$\begin{bmatrix} -mgs\theta \\ mgc\theta s\phi \\ mgc\theta c\phi - T_c \end{bmatrix}_c = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix} \begin{bmatrix} \ddot{x}_c \\ \ddot{y}_c \\ \ddot{z}_c \end{bmatrix}_c$$

Then, solving the equation for the translational acceleration:

$$\boxed{\ddot{x}_c = -gs\theta}$$

$$\boxed{\ddot{y}_c = gc\theta s\phi}$$

$$\boxed{\ddot{z}_c = gc\theta c\phi - \frac{T_c}{m}}$$

CONVERT DYNAMICS EQUATIONS TO INERTIAL FRAME

The equations of motion derived in the previous sections are all in the body frame. In order to for them to be useful, the equations must be expressed in the inertial frame. This can be achieved by multiplying the two vectors with the ${}^{n}C^{c}$ rotation matrix. Since the inertia frame is aligned with the North, East and Down direction, the displacement in those directions are defined as N, E, D respectively. For the rotational dynamics, the flight angle should be expressed respective to the ground level, but roll don't necessary have to be along the N axis and the pitch don't necessary have to be along the E axis. Therefore, the yaw frame a should be used instead.

$$\boxed{\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix}_a = {}^{a}C^{c}\begin{bmatrix}\ddot{\phi}_c\\ \ddot{\theta}_c\\ \ddot{\psi}_c\end{bmatrix}_c}$$

$$\boxed{\begin{bmatrix}\ddot{N}\\ \ddot{E}\\ \ddot{D}\end{bmatrix}_n = {}^{n}C^{c}\begin{bmatrix}\ddot{x}_c\\ \ddot{y}_c\\ \ddot{z}_c\end{bmatrix}_c}$$

Note that the angles inside the rotation matrix are in the inertia frame. To simplify implementation in simulation, the angles used in the rotation matrix can be replaced with the angle of that from the previous process loop.

Chapter 3: HARDWARE

QUADX HOPPER SIMULATOR DESIGN OVERVIEW

The QuadX flying platform was designed and built with spare time after work hours during the summer internship of 2014. With limited tools and resource as constraints, the QuadX frame design was consisted mostly of off-the-shelf injection molded joints, carbon fiber struts and two custom laser cut central hub plates. The top plate of the hub was manufactured out of semi-transparent acrylic so that the electronics in between the hub are visible. The bottom plate was manufactured out of plywood for strength. In the design, there was a hole that goes through the center of the aircraft. This would allow the QuadX be mounted on an indoor test rig. The overall dimension of the QuadX simulator was around 550 mm x 550 mm x 110 mm. A CAD model of the QuadX simulator was illustrated in Figure 3-1.

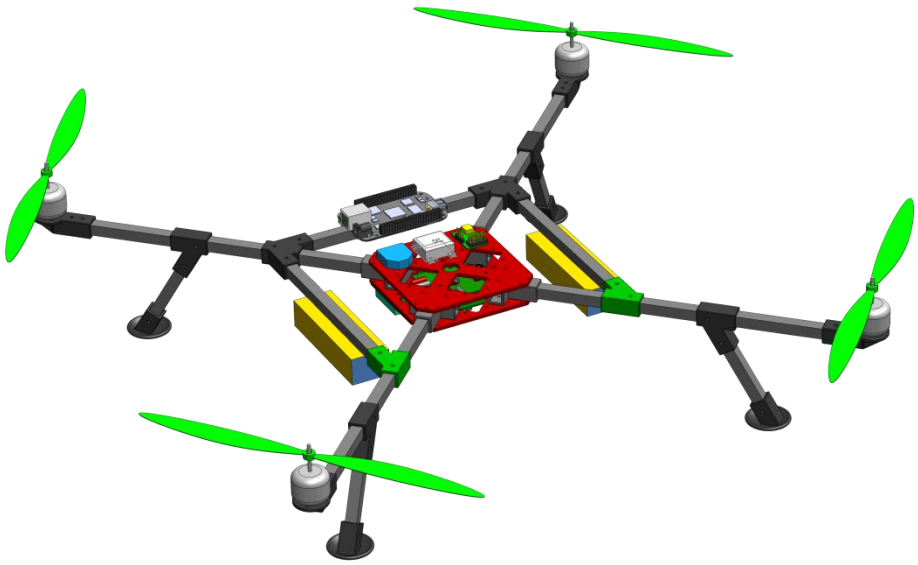


FIGURE 3-1: QUADX HOPPER SPACECRAFT SIMULATOR CAD MODEL

The QuadX flying platform is designed with the cross flying configuration. Each motor contribute to the rolling, pitching and yawing movement of the quadcopter. To reduce complexity in GN&C design, the QuadX is designed to be symmetric along the $\hat{c}_x$ axis with sensors and power circuits mounted on the hub and the batteries mounted on the two side struts. The frame in the $\hat{c}_y$ axis, however, was not symmetric due to the lack of a mounting strut in the front. This was to allow the mounting of extra hardware such as a camera in later stages of project. Since the flight computer was mounted in the back strut, the overall center of mass is shifted back. To overcome this, the two batteries are shifted toward the front as oppose to the center line.

The 100 mm landing struts were designed to be short to keep a low overall center of mass for the aircraft. They were mounted on a $60°$ joint with respect to each of the 4 arms and at a significant distance away from the center. This would allow the quadcopter to touch down on the correct orientation even if the landing attitude was not level. Dimensioned drawing of the quadcopter will be illustrated in Appendix A.

Assembly / Parts		Quantity	Weight (g)	Unit Cost ($)	Total Weight (g)	Total Cost ($)
Frame					**436.5**	**$ 75.90**
	Carbon Fiber Stuct 300 mm	4	19.8	$ 1.92	79.2	$ 7.68
	Carbon Fiber Stuct 200 mm	3	13.2	$ 1.28	39.6	$ 3.84
	Carbon Fiber Stuct 100 mm	4	6.6	$ 0.64	26.4	$ 2.56
	Rotorbits Motor Mount	4	8.9	$ 2.28	35.6	$ 9.12
	Rotorbits Landing Pad	4	10.8	$ 2.22	43.2	$ 8.88
	Rotobits 60 Deg Connector	2	9.9	$ 2.22	19.8	$ 4.44
	Rotorbits 45 Deg Connector	4	10.1	$ 2.77	40.4	$ 11.08
	Rotorbits 45 Deg Y Connector	2	20	$ 2.58	40	$ 5.16
	Rotorbits T Joint	4	14	$ 2.77	56	$ 11.08
	Central Mounting Hub	2	19	$ 6.03	19	$ 12.06
	Misc	1			37.3	$ -
Power and Propulsion					**1117.2**	**$ 160.01**
	NTM Prop Drive 28-36 750KV / 265W (USA Warehouse)	4	87	$ 16.15	348	$ 64.60
	NTM Prop Drive 28 Series Accessory Pack (US Warehouse)	4	13.8	$ 2.09	55.2	$ 8.36
	12x4.5 Props (4)	1	48	$ 4.36	48	$ 4.36
	Turnigy Multistar 30 Amp Multi-rotor Brushless ESC 2-4S (USA Wareho	4	29	$ 9.95	116	$ 39.80
	ZIPPY Compact 2200mAh 4S 35C Lipo Pack (USA Warehouse)	2	242	$ 19.35	484	$ 38.70
	Power Distribution Hub	1	29	$ 4.19	29	$ 4.19
	Turnigy 7.5A UBEC	1	37	$ 17.75	37	$ 17.75
Avionics					**85.9**	**$ 288.32**
	Arduino Due	1	25	$ 35.00	25	$ 35.00
	QuadX FlightOS PCB Shield	1	14	$ 11.52	14	$ 11.52
	UM7 - LT Orientation Sensor	1	6	$ 129.95	6	$ 129.95
	XBee Pro S1 60mW	1	8.4	$ 37.95	8.4	$ 37.95
	Adafruit Ultimate GPS Breakout	1	8.5	$ 39.95	8.5	$ 39.95
	MB1240 Ultrasonic Sensor	1	6	$ 24.95	6	$ 24.95
	RF Receiever	1	18	$ 9.00	18	$ 9.00
TOTAL					1639.6	$ 524.23

TABLE 3-1: QUADX SIMULATOR BILL OF MATERIALS

The entire design for the QuadX was modular and consisted mostly of easily obtainable parts. Because of that, post-crash repairs were often fast and easy, swapping only the damaged components. With the exception of electronics and the propulsion system, the parts for the frame were relatively inexpensive. The QuadX's bill of materials is illustrated in Table 3-1.

PROPULSION SYSTEMS

The QuadX hardware was broken three primary subsystems: the frame, avionics, propulsion and power system. From Table 3-1, it could be seen that the actual structure of the aircraft is relatively light, weighing only 436.5 g. The avionics were the lightest, weighing only 85.9 g. The propulsion and power system, however, was the heaviest, weighing 1117.2 g.

LITHIUM POLYMER BATTERY

Out of all the components in a multirotor aircraft, the battery is probably the heaviest component. The type of battery used in the modern aviation and space industry are lithium based. Whereas Lithium-Ion (Li-ion) batteries are commonly used in a professional settings, RC hobbies usually use Lithium Polymer (LiPo) batteries for the advantage of lighter weight and flexible dimension options for the same energy capacity. However a major disadvantage is that LiPo usually come in a soft pack and could be dangerous if mishandled. In any case, these batteries have higher energy density and discharge rate over aged technology such as Nickel-Metal Hydride (NiHM).

A LiPo battery is commonly consisted of multiple LiPo cells connected in series. A typical LiPo cell has a nominal voltage of 3.7V with full charge voltage at 4.2V and full discharge voltage of 3.0V. One must be careful with the cell voltage since a voltage higher or lower than those limits will

damage the battery, causing it to become unstable and could become a fire risk. A cell that drops below 3.0V will not be charged by ordinary LiPo charger and must be replaced or revived by brute force.

Generally, LiPo batteries are rated by the cell count, capacity and discharge rate. Typically, low power devices like a cell phone use 1 cell battery while higher power devices like a quadcopter would take a 3 or 4 cell batteries. As mentioned earlier, the cells are wired in series. Hence, a 3-cell LiPo would have a nominal voltage of 11.1V, since individual cells are 3.7V each. Consider that the battery is consisted of 3 cells wired in series, these batteries are often called 3S batteries. Similar, a 4S battery would have a nominal voltage of 14.8V. Wiring scheme for a 3S LiPo is illustrated in Figure 3-2. As seen in the wiring scheme, the left most plug is the main power line. This is to deliver power to the system. The plug in the middle is used for monitoring voltage in each individual cell. This plug would also go into a cell balancer during charging to ensure that all cells are charged evenly.

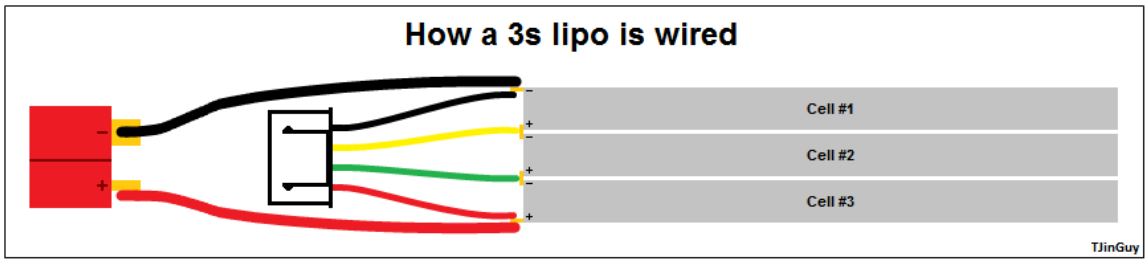


FIGURE 3-2: WIRING SCHEME FOR A 3 CELL LITHIUM POLYMER BATTERY [1]

The next property of a LiPo battery is the energy capacity. This is commonly expressed in milliamp-hours, mAh. In another word, a 2200 mAh battery could theoretically supply 2.2 A of current for an hour. This is also called discharging at a rate of 1C.

The discharge rate is another important property of LiPo battery. The variable C is related to the capacity of the battery. For example, a 2200 mAh battery would have a C = 2.20 A, while a 450 mAh battery would have C = 0.45 A. A discharge rate of 1C as mentioned earlier would discharge the entire charge of the battery in an hour, for a 2200 mAh battery, that would be 2.20 A per hour. A discharge rate of 2C would allow the battery be discharged in 0.5 hour, at a current of 4.40 A. Transmitter batteries are usually rated at a low discharge, such as 1.5C. On the other hand, the main battery for a quadcopter would commonly have a constant rating of around 25C and a burst of 35C to support a maximum constant current draw of 55A and a brief current draw of 77A.

It is always better to use a battery with higher discharge rating, since discharging too fast in a lower quality battery could lead to the battery becoming unstable. At the same time, care should be taken in charging the battery. Commonly, it is recommended to charge the battery at the rate of 1C to ensure that each cell is charged evenly. However, as the technology mature, some batteries support higher charge rate with the use of a cell balancer.

MOTOR AND PROPELLER

The brushless DC motors are somewhat of a standard in the RC industry. These motors usually have several characteristics such as KV value, maximum rated current and power flow in additional to the normal dimensional factors. The KV is a motor's rpm constant. It is defined as a motor's rotational speed under no load condition per volt applied to it. For example, an 1100 KV motor operating at nominal voltage with a 3S LiPo will spin at a speed of $1100\frac{RPM}{V} * 11.1V = 12210\ RPM$. In reality with the load of a propeller, depending on diameter and pitch, the motor will likely saturate before reaching that speed. The rated maximum current and power are easy

to understand. That the maximum amount of current or powers the motor can draw before damage occur. Holding the RPM constant, usually the larger the propeller diameter and pitch, the more trust will be produced. However, at the same time, more power is required to spin the propeller and hence higher current draw.

FIGURE 3-3: NTM 28-36 750KV BRUSHLESS DC MOTOR

In multirotor aircraft there are two main types of propellers, the standard tractor and pusher props. Essentially, they have the same flight characteristic except the fact that one spin clockwise to produce a downward force while the other spin counter-clockwise to produce a downward force. A set of counter-rotating propellers are illustrated in Figure 3-4. To make a brushless DC motor to spin the other direction, one can simply switch 2 of the 3 connections into the motor as shown in Figure 3-3. In general, an equal number of tractor and pusher props will be used in a multirotor. This configuration is beneficial since it will generally balance out the yawing torque created by the motion of the propellers.

FIGURE 3-4: A SET OF COUNTER-ROTATING 12X4.5 APC PROPELLERS

Other than the direction, propellers are also defined by their diameter, pitch and material. The diameter of a propeller is the distance across a circle drawn by the propeller's tips. On the other hand, the pitch is the distance that the propeller would travel in 1 revolution if it was drive like a screw. In another word, this is how much the blades bend from the center. The material of the propeller defines certain properties, such as deflection and durability. Usually propellers made with soft material like thin plastic, they are usually more durable because of their ability to flex. However, this somewhat hurt the control performance of the aircraft, since more will be required to achieve a certain target thrust. On the other hand, a harder propeller like one made with carbon fiber are brittle, but very hard. This allows it to change thrust output much faster and improve the response time.

There are multiple combinations for a motor and propeller to achieve a certain thrust. Assume that V_0 is the propeller forward airspeed in m/s, d is the propeller diameter in inches, $pitch$ is the propeller pitch in inches, RPM is the rotation speed in revolutions per minute and *F* is thrust in N. The equation to estimate thrust is as follow [2]:

$$F = 4.392399E-8 * RPM * \frac{d^{3.5}}{\sqrt{pitch}}(4.23333E-4 * RPM * pitch - V_0) \tag{3.1}$$

Assuming that the motor is spinning at 6,500 rpm with a propeller diameter of 12 inch and a pitch of 6 inch, or 12x6 prop, the static thrust of the combination is 11.5 N or 1174 g. In vertical takeoff aircraft, it is often advantageous to express thrust in terms of mass so that it would be apparently how much mass the vehicle can lift. In order to achieve the same thrust with an 11x7 and 10x7 prop, the rotational speed would have to be 7,277 rpm and 8,598 rpm respectively.

The power required to drive the motor and props with these parameters are described by the following equation using the experimental prop constant and power factor from Table 3-2 [3]:

$$Power = Prop\ Constant * Thousands\ of\ RPM^{Power\ Factor} \quad (3.2)$$

Prop Size	Prop Constant	Power Factor
10x7	0.223	3.20
11x7	0.301	3.20
12x6	0.322	3.20

TABLE 3-2: APC ELECTRIC "E" SERIES PROPELLER CONSTANTS [4]

Using the equation and constants from Table 3-2 for the APC Electric "E" Series propellers, the estimated power draw for the 12x6, 11x7, 10x7 propellers to produce 1174 g of thrust are 129 W, 173 W and 218 W respectively. Hence, it is obvious that the power draw is inversely proportional to the propeller diameter.

In summary, to produce the same amount of thrust, a larger diameter propeller requires lower RPM than a smaller propeller. At the same time, a larger propeller also requires less power. Typically, a lower KV motor is more efficient anyway. Therefore the most energy efficient combination should be a large propeller with a lower KV motor. However, a higher cell count LiPo is often required to drive a lower KV motor to a higher speed so that more thrust can be produced.

There are cases when a small propeller high KV motor combination is desired. This is mostly for aerobatics maneuver. Smaller propellers can change thrust a lot quicker than larger propeller due to the momentum differences. Therefore for flights that require fast response time like

aerobatics, higher KV motor would be necessary. In addition, lower cell count battery could be used hence make the multirotor more maneuverable.

ELECTRONIC SPEED CONTROLLER

The electronic speed controller (ESC) is used to control the speed of a DC brushless motor. As shown in Figure 3-5, there are two power input lines that go into the ESC. The red is positive voltage and the black is the ground from the power source. It is important that the polarities are not switch, or else the ESC will be damaged. The other ends are three output power and signal lines that are connected to the three pins on the motor. The order of which these pin are connected is not important, for switching any two of them will simply result in the motor spinning the other direction. Some testing are required in order to get the tractor and pusher propeller to spin the correct direction with the motor.

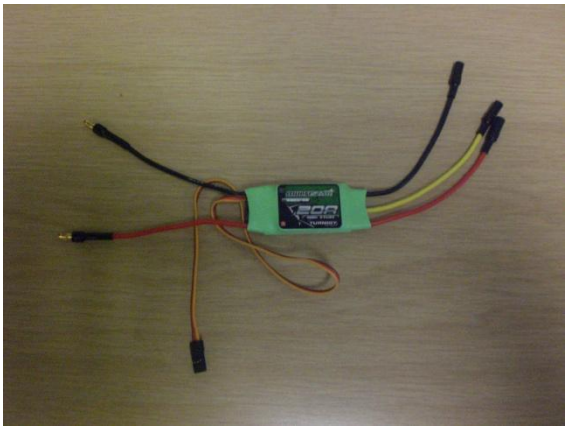


FIGURE 3-5: MULTISTAR 20 AMP 2-4S ESC

The ESCs are rated by the type maximum amount of current and voltage input they can handle. Usually, this is the current in Amp and the LiPo cell count. The ESC in Figure 3-5 is a MultiStar 20 Amp 2-4S ESC. This essentially say that the ESC can properly handle power input from a 2 to 4

cell LiPo battery at maximum current draw of 20 A. Usually when selecting ESC, a current rating higher than the motor current rating is desired.

The ESC also includes a battery eliminator circuit (BEC) that is used to provide power to the flight computer and the RF receiver. The BEC output is usually around 5V and provides several amps of current depending on specific hardware specs. The BEC output is the red and brown line in the servo wire as shown in Figure 3-5. The yellow line is a signal line that takes pulse-width modulation. In a later section, there will be discussion regarding the pulse-width modulation and how to control an ESC from a microcontroller perspective. Essentially, the ESC work by turning the electromagnet on and off depending on the location of the rotor. The pulse-width modulation signal determines how long these electromagnet are on and hence increasing or decreasing the rotation speed.

Propulsion and Power Hardware Selection

In selecting components for the propulsion system, the thrust to weight ratio must be considered. In order for a vertical takeoff vehicle to lift off from the ground, a thrust to weight ratio larger than 1 is required. However, extra thrust margin must be reserved so that the control system can balance the aircraft. Therefore, a ratio of around 1.50 is required at the minimum for controllable flight. For stable flight, then the ratio should be around 2 to 3. For aerobatic flight where the aircraft is required to make steep and fast maneuvers, than a ratio greater than 3 is required.

The Space Hopper is a scientific research spacecraft, its primary purpose is to collect visual data. Therefore it is more important that the platform achieve stable flight rather than aerobatic

flight. Though consider that a hopping maneuver is required, the target thrust to weight ratio should be around 2.50.

In the end, the propulsion hardware selected are four NTM 28-36 Prop Drive 28-36 750KV 265W motors matched with two pairs of 12x4.5 counter-rotating propellers. The motors are driven by four MultiStar 30A ESC along with two ZIPPY Compact 2200 mAh 4S 35C LiPo batteries. The two batteries are connected in parallel into a power distribution board that distributes the power into the four ESCs.

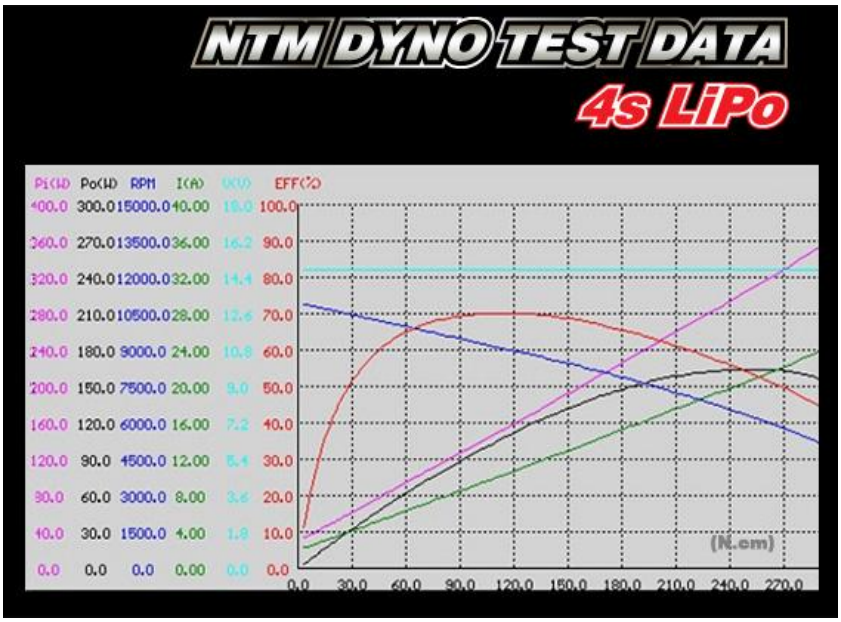


FIGURE 3-6: NTM 28-36 750KV 265W MOTOR DYNAMOMETER TEST DATA [5]

From the datasheet, the NTM 750KV motors are rated to draw up to 18A of current and has a maximum power of 265 W. Looking at the dynamometer test data provided by the manufacturer in Figure 3-6, the maximum output power with a 4S battery is around 165 W.

Since not test data are available for the 12x4.5 propeller the values are interpolated from the APC sport series which yield prop constant of 0.545 and a power factor of 3.0. Plugging these

values into Equation 3.2 along with the maximum output power of 165W with a 4S power supply, the maximum rotational speed is calculated to be around 6,715 rpm. Using equation 3.1, the estimated thrust for each motor is around 1085 g, with a total thrust of 4340 g. This number is relatively close to the experimental result gathered during motor thrust system identification. When compare to the totally system's mass of 1640 g, this set up would yield a thrust to weight ratio of 2.65.

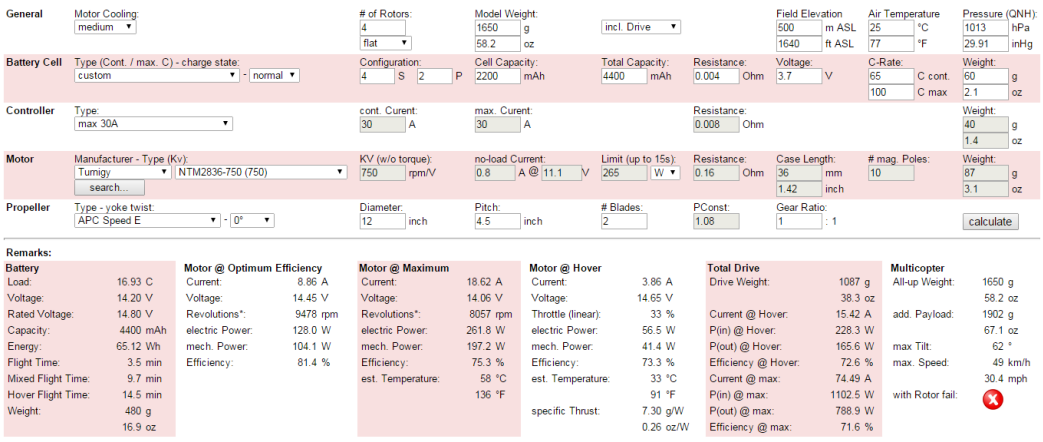


FIGURE 3-7: QUADX PROPULSION AND POWER PERFORMANCE CALCULATION RESULT

The selected components were entered into the xcopterCalc, a multirotor propulsion calculator from eCalc as shown in Figure 3-7. The hovering throttle was estimated to be at 33%, hence yielding a thrust to weight ratio of 3. However, this is assumed that at maximum, the motor is drawing 261.8W of electrical power at 18.62A producing 197.2W of mechanical power, hence spinning the 12x4.5 prop at a speed of 8057 rpm. This in totally draw a maximum of 74.48A from the two batteries, hence divided into 37.24A for each battery. Since the capacity of each battery is 2200 mAh, a maximum of 16.93C is drawn from the battery. In reality, the motor output performance saturate before reaching 18A of current, therefore, it is not possible to produce 197.2W of mechanical power as indicated. Hence, the thrust to weight ratio of 3 is not realistic. However, from the calculator, it was estimated that the aircraft have a flight time of 3.5 min and

a hovering time of 14.5 min. This brings the mixed flight time to 9.7 min. While not ideal, this is acceptable for GNC research and development.

To ensure that avionics have enough power, a dedicated universal battery eliminator circuit (UBEC) is used to provide power to the flight computer. For that, a Turnigy 7.5A UBEC is used to output regulated 5V power. The flight computer's onboard voltage regulator in turn maintains power at 3.3V for all electronics including the RF receiver, XBee radio and various sensors.

Chapter 4: Embedded System Overview

Computer Mathematics

Binary System

The binary system is the fundamental building block of digital electronics. The binary system is chosen because it is simple as each digit only has 2 states, "on" and "off". Computations can be done by manipulating the electronics "switches" in a circuit. In modern days, these switches are implemented as silicon transistors. As manufacturing technology advances, the size of transistors decrease and more transistors are able to be packed into a processor. This allows higher numbers of calculation to be done in a given amount of time.

The binary system is also sometime refers to as the base 2 system. Each digit in a binary number is call a bit, and every 8 bits make up a byte. As mentioned earlier, each bit only have two states, "on" and "off", which represent "1" and "0".

Converting Binary to Decimal

To further understand the binary system, let's consider a binary number with 4 bits. The right most bit of the number is the least significant bit (LSB) while the left most bit of the number is the most significant bit (MSB). From the LSB to MSB, the number is indexed as 0 to 3 respectively. This index is important as it relate the numerical value of that bit back in decimal, or base 10, through the following equation:

$$Base\ 10\ Value = 2^{Index}$$

Hence, the LSB represent a value of $2^0 = 1$ in base 10. On the other hand, the 2nd bit represent a value of $2^1 = 2$. Therefore, the 4 bits in order from LSB to MSB represents a value of 1, 2, 4 and 8 in base 10.

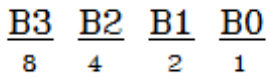


FIGURE 4-1: 4-BIT BINARY NUMBER

Let's assume that B0, B1, B2 and B3 represent the represent the state of each digital in the binary number from LSB to MSB as shown in Figure 4-1. The following equation is used to convert the 4-bit binary number to decimal:

$$decimal\ value = \boldsymbol{B3} * 2^3 + \boldsymbol{B2} * 2^2 + \boldsymbol{B1} * 2^1 + \boldsymbol{B0} * 2^0$$
$$= \boldsymbol{B3} * 8 + \boldsymbol{B2} * 4 + \boldsymbol{B1} * 2 + \boldsymbol{B0} * 1$$

A list example is provided in Table 4-1. As demonstrated, conversion from binary to decimal is not too difficult given that value for each bit is known. Essentially, add up all the bits that have a "1" in it. This method of conversion can also be extended to binary that's a byte or higher in length. To things into perspective, in an Arduino, the size of an integer value is 16 bits, or 2 bytes. In an unsigned int, the memory allocated to the number can range from 0 to a maximum of $2^{16} - 1 = 65535$. In a signed int, the minimum is -32768 and the maximum is 32767. Methods to express negative number in binary will be discussed later.

Binary	Decimal
0011	0+0+2+1 = 3
0101	0+4+0+1 = 5
1010	8+0+2+0 = 10
0101 0001	0+64+0+16+0+0+0+1 = 81
1111 1111	128+64+32+16+8+4+2+1 = 255

TABLE 4-1: EXAMPLES OF CONVERTING BINARY NUMBER TO DECIMAL FORM

CONVERTING DECIMAL TO BINARY

Converting a decimal number to its binary form is a bit more difficult. One can do the conversion by manually selecting bits as "1" or "0" to make up a decimal number. However, that is not very methodical and could be tedious for larger values. A better method is repeatedly a number and its Quotient by two and the remainders is the binary number. This is just illustrated with an example. Consider that it is desired to convert a number, "38", from decimal to binary.

Division	Quotient	Remainder
38/2	19	0
19/2	9	1
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1

TABLE 4-2: DECIMAL TO BINARY CONVERSION USING DIVISION BY 2 METHOD

Procedures in converting "38" into binary form are provided in Table 4-2. As seen in the example, the original number is divided by two with its remainder written in a separate column. This process is repeated with the quotient until a quotient of 0 is reached. In the remainder column, the first remainder is the LSB while the last remainder is the MSB of the binary number. Hence, "38" in decimal is converted to "B100110" in binary. In order to allocate enough memory for this number, the integer must at least be 1 byte in size. Hence "38" in the computer will be "B 0010 0110". Note that when writing binary number a prefix of "B" is added to the number to denote that it is a binary number.

BINARY ARITHMETIC

Addition in binary is relatively similar to that of base 10 addition. Instead of carrying over to the next decimal place when a number is larger or equals to 10, in binary arithmetic, the carry over take place when a value larger than 1 is reached. For example, consider computing the sum of 9 and 3:

```
   1001
+  0011
------
   1100
```

The result is "12", that's relatively easy. Now let's take a look at another example, calculating the sum of 9 and 9 and storing that into a 4-bit integer variable like before.

```
   1001
+  1001
------
  10010
------
   0010
```

And in the end, the result stored in the 4-bit int variable will be "2". Formally, the result is an overflow. Obviously 9 + 9 = 18 and not 2. However, since the memory allocated to a 4-bit int variable is only 4 bits, any bits exceed 4th bits will be rejected. Hence, the end result is "2".

Overflow errors can be catastrophic if precautions were not taken. On June 4, 1996, the Ariane 5 rocket from the European Space Agency lifted off for its maiden flight from Kourou, French Guiana. The launch vehicle exploded 40 seconds into flight due to failures in the Guidance, Navigation and Control system. The root cause of the failure was a software integer overflow error in the Inertial Reference System. The overflow happened when the computer was trying to optimize memory allocation and assign the 64-bit floating point horizontal velocity value to a 16-bit signed integer variable. This caused the system to shut down and switch to the backup system. Unfortunately, the identical redundant backup system also suffered from the same problem [6].

TWO'S COMPLEMENT: DECIMAL TO BINARY

Most digital system uses two's complement to represent a signed integer in binary. In another word, this system is used to represent a negative number For example, let's consider expressing the number "-21" in 8-bit 2's complement. First, express "21" in binary:

0001 0101

Then, all the digits are inverted, or toggle:

1110 1010

Finally, add one to the number and "-21" in binary is:

1110 1011

Note that the MSB of the binary number designate whether a number is positive or negative. If it is "1" then it is negative, on the other hand, if it is "0", then it is positive.

Two's Complement: Binary to Decimal

Let's take a look at another example to convert a binary number in 8-bit 2's complement back to decimal. Let's consider the binary number:

1011 0100

The MSB is "1", therefore, this number is negative. First, toggle all number:

0100 1011

Then add one to the number:

0100 1100

Hence, converting this number back to decimal, the number is "-76".

Binary Arithmetic with Signed Number

Subtraction in binary is simplified to adding a number of the opposite sign. Using 2's complement, the difference of subtracting 21 from 10 is equivalent to the sum of -21 and 10:

```
   0000 1010
+  1110 1011
-------------
   1111 0101
```

```
    0000 1010
+   0000 0001
-----------------------
(-)  0000 1011
```

The result after the addition is "B1111 0101". Since the highest bit is a "1", then the number is negative. After going through the process of toggling all bits and adding a value of "1", the result is "-11" in decimal form.

Overflow errors can happen in adding signed number too. The detection method is relatively easy. When the sum of two positive numbers is a negative number, then an overflow occurs. Similarly, when the sum of two negative numbers is a positive number, an overflow also occurs. When adding a positive and negative number, an overflow cannot occur. Another way to look at this is that if the highest bits of two numbers are the same, then the sum of the two numbers must have the same value in the highest bit or else an overflow happened.

BITWISE OPERATION

Since binary number are consisted of 1s and 0s, logical operators call easily be applied and are useful at times. Some common bitwise operators are NOT, AND, OR and XOR.

The bitwise NOT operator is also sometime also called the complement operator. It performs logical negation on each bit like the toggle action in 2's complement. The bitwise operator is usually represented by a "~" in programming languages like C++. Note that this differ than the logical not "!" operator.

NOT 1001 = 0110

The bitwise AND operator is represented by "&" and it perform a logical AND check among 2 binary strings. The result of two bits is true if both bits are true. The AND operator is especially useful when trying to apply a bit mask to a binary string. For example, getting the lower 4 bits of a byte:

```
        0000 1111
AND     1001 0101
-----------------------
        0000 0101
```

The bitwise OR operator is represented by "|" and it perform a logical OR check amount 2 binary strings. The result of two bits is true if one or both of them are true. For example:

```
        0000 1111
OR      1001 0101
-----------------------------
        1001 1111
```

The bitwise XOR operator stand for logical exclusive OR and is represented by "^". The result of two bits is true if and only if one of them is true. For example:

```
        1001 1111
XOR     1001 0101
-----------------------------
        0000 1010
```

LOGICAL BIT SHIFT

Recall that the maximum value for a byte of unsigned int is 255. Consider that in serial communication, which will be discussed in later section, data are communicated in form of encoded bytes. If it is desired to transfer a number greater than 255 from a device to another, than the only way to do this efficiently is to split the number into multiple bytes. For example, it is desired to transmit a 16-bit unsigned integer of value "18015":

$$0100\ 0110\ 0101\ 1111$$

The first step is to designate a high byte that is consisted of only the highest byte value of the integer. This can be accomplished by using the bit shift operator. By shifting the value 8 places to the right, value of the highest byte is now located in the lower byte with 0s replacing the high bytes. By assigning this number to an unsigned int, the high byte is calculated.

$$High\ Byte\ =\ 0100\ 0110\ 0101\ 1111\ >>\ 8\ =\ \cancel{0000}\ \cancel{0000}\ 0100\ 0110$$

After calculating the high byte, it is time to calculate the low byte. This can easily be accomplished by applying a mask to keep only the lowest byte using the AND operator.

$$Low\ Byte = 0000\ 0000\ 1111\ 1111\ \&\ 0100\ 0110\ 0101\ 1111 =\ \cancel{0000}\ \cancel{0000}\ 0101\ 1111$$

The high and low bytes can then be transmitted through the serial link. Then, in order to calculate the value from the high and low bytes, the reverse process can be applied:

$$Value = (High\ Byte \ll 8) +\ Low\ Byte$$
$$= 0100\ 0110\ 0000\ 0000 + 0000\ 0000\ 0101\ 1111$$
$$= 0100\ 0110\ 0101\ 1111$$

It is again important to remember to allocate enough memory after reassembling the value or else an overflow error could occur.

Hexadecimal Representation

Since binary is often lengthy and cumbersome to refer to, it is common practice to refer to computer hardware or memory addresses with hexadecimal. A number in hex is denoted with the "0x" prefix. In hexadecimal, the number system is base 16. In another word, a single digital can have a value from 0 to F. In relation with decimal, A, B, C, D, E, F represent 10, 11, 12, 13, 14, 15 respectively.

Conversion from binary to hex is relatively easy. From decimal to hex is a bit more involve for it take the extra step of converting from decimal to binary. For example, convert number 45660 from binary to hex. First, the number should be expressed in binary and separated in groups of 4 bits.

$$1011\ 0010\ 0101\ 1100$$

$$11 \quad 2 \quad 5 \quad 12 \quad => \quad 0x\,B25C$$

The groups of 4-bits are then translate directly into hex or through a middle step of translating back to decimal then to hex if unfamiliar with binary math. Either way, the end result is the same, 0xB25C. The process can be reversed to obtain the binary string or decimal value.

Octal Representation

Another popular representation is octal, which is a base 8 numeric system. Octal is widely used in older computer system. In modern days, it is sometime used as shorthand in UNIX operating system representing file permission. Binary to octal conversion can be done by grouping bits in

groups of 3. Each digital has a range from 0 to 8. Since this is very similar to hexadecimal, relatively simple and less commonly used, an example will be omitted.

ASCII TABLE

The American Standard Code for Information Exchange (ASCII) is a widely used scheme to encode characters into binary integers. This allow text and symbols be represented in computer. The original ASCII standard take only 7 bits with 128 defined characters, 95 of which are printable and 33 are non-printable control characters. As technology progress, the 8th bit is used for the Extended ASCII Code for additional symbols. The original 7-bit table is illustrated in Figure 4-2.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

FIGURE 4-2: 7-BIT ASCII TABLE

In serial communication, whenever something is sent from the terminal or user input console, it is important to remember that the inputs are characters instead of actual numerical value. For example, "123" is a char array string consisted of '1', '2' and '3'. To properly represent the value

123, the character '{' should be used as represented in Figure 4-2. If the extra conversion step is not taken, this can easily result in an accident. For example, a command of '0' is sent to level the quadcopter's roll and pitch. This will result in commanding the quadcopter to tilt to an angle of 48 degrees, which is quite steep.

This confusion is usually not a problem is the user is out of the communication loop and only simple data type like an 8-bit int is transferred. The programmer can easily cast the arriving char as an int to retrieve the numeric value. If the data is bigger than a byte, than bit shifting operating will need to be done in addition to type casting. For floating point value, the programmer can scale the value by a predefined constant.

The easiest way to transmit data in a way readable to human, however, is also the most inefficient. The method is to convert everything to a string. And the device from the other end can run some process to convert the string back to a numeric value. For example, the value 123 can be transmitted as "123". However, as opposed just transmitting a byte for an 8-bit value, 3 bytes will needed to be transmitted for there are now 3 chars. Extra process will also needed to be run in order to retrieve the numerical value of each char and add them back to the correct decimal place. While this is very inefficient, it makes transmitting floating point data easier. Also, the stream is a lot easier to debug since it is readable and not in binary chars.

ELECTRONIC INTERFACES

There are many input and output interfaces within electronic devices. The input and output ports are used for the transfer of signal and data between devices. In order to successfully design and implement a control system for the hopper spacecraft simulator, one must first learn about how various devices in a system talk to each other. Before beginning, it is important to remember that for devices to communicate with each other, they must establish a "common ground". In another word, the ground pin of each device must be connected.

DIGITAL SIGNAL

Modern computers operate on the binary system where 1 represents ON and 0 represents OFF. Similarly, digital signal also only have 2 states: HIGH and LOW. In a 5V logic level device like the Arduino UNO, a digital HIGH signal is 5V and a digital LOW signal is 0V. On the other hand, 3.3V logic devices such as the Arduino DUE and the BeagleBone Black, digital HIGH are defined as 3.3V and a digital LOW is defined as 0V. Connecting a 5V and 3.3V logic device together without using a logic shift can cause damage to the electronics. In either case, digital signal can be used in many applications. For example, a microcontroller can output a HIGH or LOW signal to turn an LED on and off. Similarly, a microcontroller can also read digital signal from sensors like a switch to detect whether a button is pushed or not.

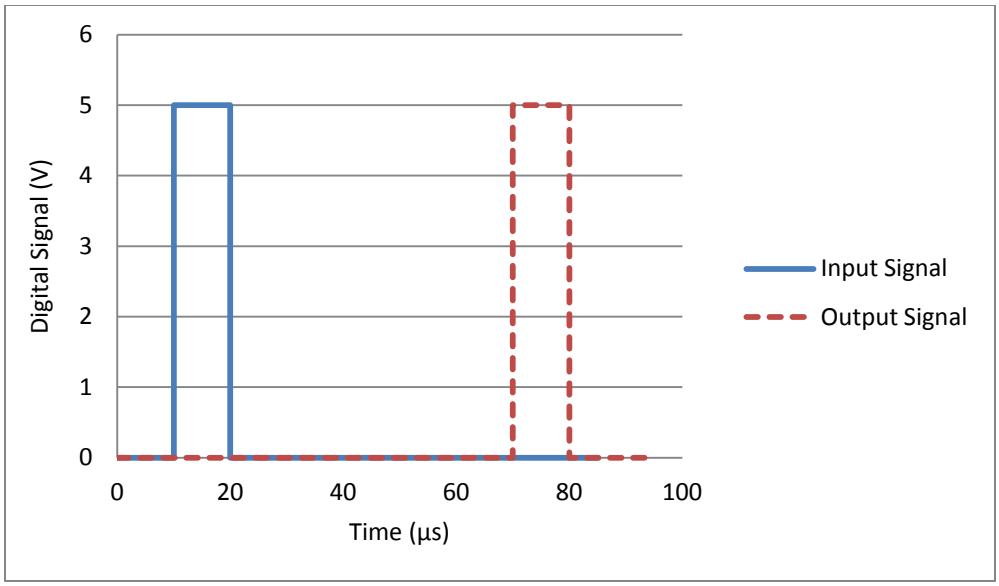


FIGURE 4-3: HC-SR04 ULTRASONIC SENSOR INPUT / OUTPUT SIGNAL

Figure 4-3 depicts a possible signal interchange between a 5V microcontroller and the HC-SR04 ultrasonic range sensor. The microcontroller first send out a 10 microseconds signal pulse to the sensor. The sensor then sends out an ultrasound pulse to the environment. When the sensor senses the returning signal, it outputs a digital HIGH signal pulse back to the microcontroller. In this example, the time between the input and output signal pulse is roughly 60 microseconds. Take into consideration that the ultrasound wave travels out and back, the time taken for the ultrasound wave to reach the closest object is actually 30 microseconds. Multiplying the traveling time with the speed of sound yield a distance of roughly 1 cm. In reality, the sensor will most likely not function in such close distance. However, this example demonstrates how digital signal could be used to exchange information between devices.

ANALOG SIGNAL

Naturally, the world is not so clear cut where something is either on or off, true or false. There are many possibilities. For example, a light bulb can have different lighting intensity and an electric motor can have various speeds. Analog devices such as a force gauge usually represent data in the form of continuous voltage signals. As oppose to digital signal where the voltage level can only be HIGH or LOW, analog voltage signal can have a range from 0V to as high as input power source voltage. Since most computers are digital devices, analog-to-digital converters (ADC) are often used to convert the analog into a form understandable by the computers. Some devices such as Arduino return a value from 0 to 1023 to represent the analog signal from 0V to the maximum voltage. Other devices return the numerical voltage level. Experimental calibration data for the Wind Tunnel Thrust Stand is depicted in Figure 4-4. The graph shows analog voltage signals returned by the load cell with respect to various loading weight.

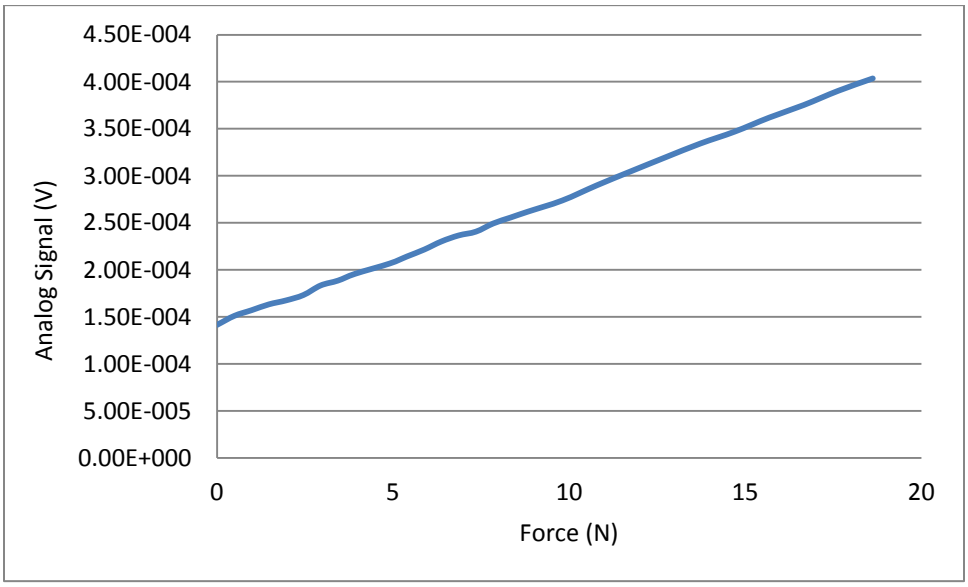


FIGURE 4-4: PIEZOELECTRIC LOAD CELL EXPERIMENTAL CALIBRATION DATA

PULSE-WIDTH MODULATION (PWM)

As discussed in the previous section, analog signal best represent real world properties. However, without a digital-to-analog converter (DAC), true analog signal with varying voltage level cannot be generated by a microcontroller. Most hobby grade electronics such as the Arduino UNO and the BeagleBone Black lack such a device. An alternative method, pulse-width modulation (PWM), could be used to emulate analog data in a digital signal through repeating square waves. PWM is generally used to control power delivery in devices like light-emitting diodes (LED), relays, transistors and electric motors. However, since the output is not a real analog signal, it doesn't work well in applications like audio.

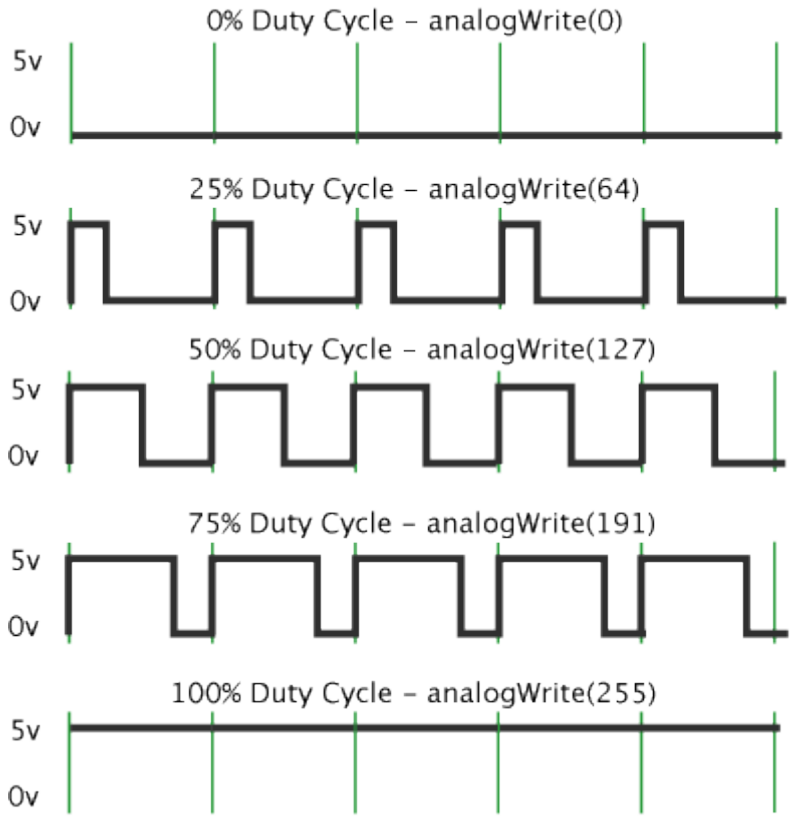


FIGURE 4-5: PWM SIGNAL FOR VARIOUS DUTY CYCLE [7]

The amount of power deliver to various electronic devices is control by PWM signal through duty cycle. The duty cycle is defined as the percentage of a period that the signal is active. In another word, the amount that a digital signal is in the ON state in a set period. The duty cycle of a signal can be calculated using the equation:

$$Duty\ Cycle = \frac{Time\ Active}{Period} \cdot 100\%$$

A plot of signal at various duty cycles is illustrated in Figure 4-5. The green vertical bar represents intervals in which the square wave repeats itself. When a duty cycle of 0% is generated, the signal remains at 0V during the entire period. Similar at 100% duty cycle, the signal remain at 5V the entire period. Overall, this is the same is an ordinary digital LOW and HIGH output. However, let's say that a signal of 25% duty is generated, the signal is active at 5V for 25% of the interval and then jump back to 0V for the remainder of the interval. It is important to point out that the active voltage of a PWM signal is determined by the logic voltage level of the device being used.

For example, an Arduino UNO is used to control the brightness of an LED through its digital pin. While ideally, true analog signal should be used to control the amount of current flow through the LED, given the device's limitation, PWM will be used. When a 75% duty cycle PWM signal is sent, the LED is on for 75% of the time in a given interval. If the frequency of the interval is slow, then as expected, the LED will appear to be blink. However, if the frequency is set faster than the human eye's refresh rate, then the LED will appear to be at 75% brightness.

COMMUNICATION PROTOCOL: UART

The Universal Asynchronous Receiver/Transmitter (UART) is one of the most popular electronic interfaces that are used in digital serial communication between more sophisticated devices. One of the benefits of serial communication interfaces like UART is that it allows the transmission of data in a simple and direct manner.

For example, let's say that a Microcontroller A want to relay a value of "168" to Microcontroller B. Using methods discussed so far, Microcontroller A will have to convert that value into an analog or PWM signal, Microcontroller B will then read and interpret that signal. Through serial interface, Microcontroller A will send the value in its binary form, "B10101000", directly to Microcontroller B one byte at a time expressed in bits. This uses transistor-transistor level (TTL) logic, meaning that 0 is logic LOW and 1 is logic HIGH, as mentioned before in the digital signal section. When the UART hardware in Microcontroller B receives the data in bits, it will assemble them into bytes and are accessible to the software for processing.

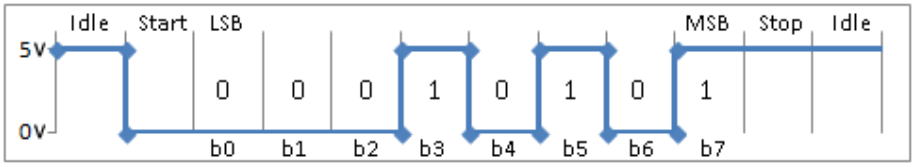


FIGURE 4-6: TTL SERIAL COMMUNICATION SIGNAL

Depending on the specific serial interface properties being used, the transmission would be encoded in binary digital signal similar to one as shown in Figure 4-6. The vertical bars in the figure represent divisions of bits in the signal frame. In each transmission signal block, the first bit represents the start bit, a logic LOW. The following 8 bits contain data up to 1 byte in size. This could be an ASCII character or a numerical value as discussed earlier. The data bits are organized with the least significant bit (LSB) first and most significant bit (MSB) last. For the

value "168" or "B10101000" in binary, the LSB is the right most bit. This is because the right most bit represent a value of 2^0 or 1 if it is active. On the other hand, the left most bit, or MSB, represent a value of 2^7 or 128 if it is active. Again, this will be explained in depth in the binary mathematics section. Because the signal block is set up LSB first, the data bits will appear to be arranged in reverse like "00010101". After the data bits come the end bit, which is a logic HIGH. Note that when the signal is at idle, the logic level is set at HIGH.

The UART interface defines a standard that allow compatible devices to connect and communication with each other. In the simplest set up, the UART interface requires the connection of the transmitter (TX) pin of one device to the receiver (RX) pin of another as shown in Figure 4-7. This will allow data to be transmitted from the first device to the second. If the same is applied from the second device to the first, 2 way communication link is established. As with all previous interfaces, the devices must have a common ground (GND) connection in order for signal to transfer. And optionally, a final connect is made with the voltage pin to supply power from a master and slave device.

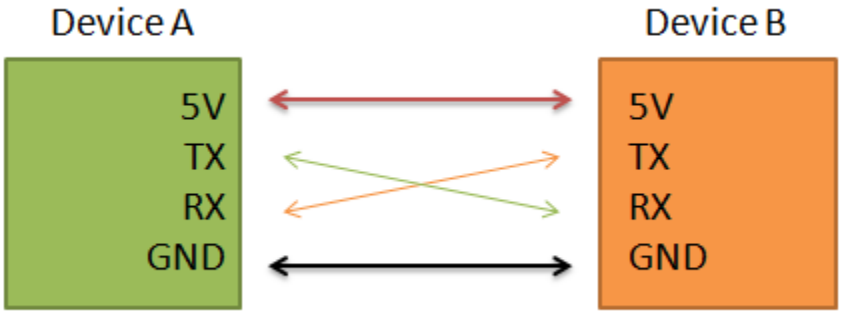


FIGURE 4-7: UART INTERFACE

There are several properties that define a UART connection. In order for two devices to communicate, their properties must match. One of the most important factor is the speed, or baud rate, of the connection. Since UART serial ports are asynchronous protocol, timing of when data are transmitted and received are essential. In telecommunication, baud is a unit for symbols or pulses transmitted per second. A symbol can represent one or more bits of data. Common standard baud rates used in consumer grade electronics are 9600, 19200, 38400, 57600, and 115200 bauds per second. With a slower baud rate, less data can be transmitted in a given amount of time and might introduce unwanted delays in a control loop. On the other hand, devices communicate at a high baud rate might suffer from data corruption problems. Therefore, other quality assurance method, like checksum, must be used to ensure that data received are accurate.

Some other properties of a UART connection are parity and stop bits. It is important to note that UART compatible devices can easily communication with a computer through an FTDI RS232 to Serial Port adapter. In this case, the connected device will show up on the computer as an opened Serial Port, allowing communication in a serial console like PuTTY.

COMMUNICATION PROTOCOL: I2C

While the UART is wide spread simple to implement to electronic systems, it is not ideal. Some of their disadvantages are slow data transfer rate, devices must agree on predefined properties and the inability to have a communication bus with multiple devices. One could argue that a work around could be implementing a multiplexer or using a microcontroller unit with multiple UART chips. However, there are more efficient communication protocols for this application. This section will introduce the Inter-Integrated Circuit (I2C) serial communication protocol.

The I2C serial communication protocol designed by Phillips in 1982. Unlike UART, I2C is a synchronous protocol where data are sent in sync with a clock signal. Without the overhead processing to keep track of timing, faster transfer rates are possible with less corrupted data. In addition, multiple devices can be attached to a communication bus. This feature reduces the number of hardware needed in a system and make wiring easier. While most modern controllers have dedicated hardware I2C support, the protocol can be implement in software through bit-banging in the digital signal pins.

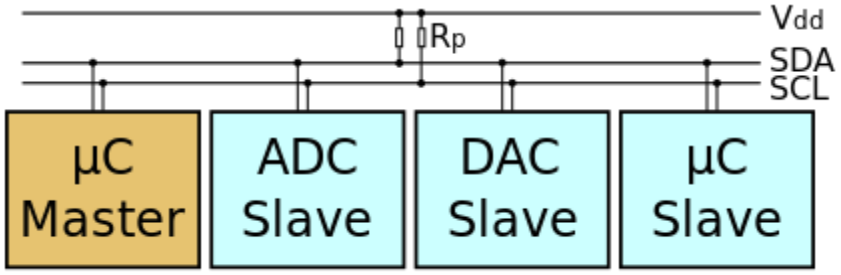


FIGURE 4-8: I2C COMMUNICATION BUS [8]

There are two wire that connects I2C devices, the signal data line (SDA) and the signal clock line (SCL). In addition, a common ground. Since I2C utilize an open-drain design in the signal line to pull the signal to the logic LOW state when needed, a pull-up resistor is require on each signal line to maintain the line at a logic HIGH state at idle. Figure 4-8 illustrated an I2C communication bus with a single master controller and several slave devices.

In an I2C bus, a master device generates the clock signal and initiates communication with slave devices through the data line. A slave device's role is to respond to a master's command when spoken to. Each slave device has a 7-bit address. With this address, the slave can determine whether the master is communicating with it or another device. In theory, the 7-bit address can support 128 devices. However, in reality only 8 devices of the same type can be attached. This is

because most manufacture uses the first 4 MSB to characterize device type, leaving only the last 3 LSB for custom addresses. For example, the LM75A digital temperature sensor has a 7-bit address formatted as "B1001XXX" in binary, where the last 3 bits are programmable by pull a respective 3 pins on the device to logic HIGH or logic LOW. Hence, there are 8 possible address ranging from "0x48" to "0x4F" in hexadecimal, or in binary, "B1001000" to "B1001111". To overcome this address limitation, an I2C multiplexer could be implemented.

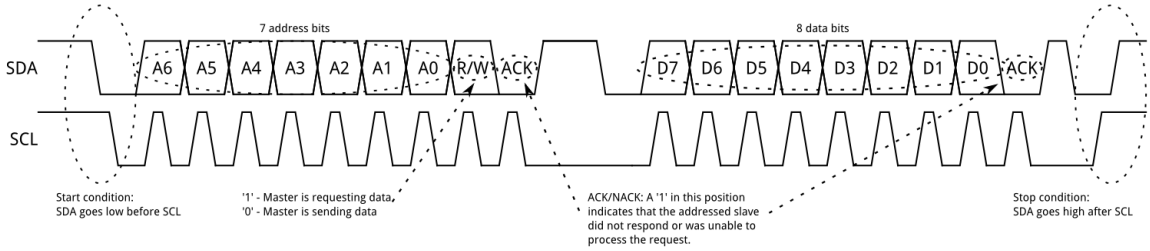


FIGURE 4-9: I2C COMMUNICATION MESSAGES [9]

Figure 4-9 illustrates a standard 7-bit addressed I2C communication message. The message is consisted of a 7-bit address frame along with an 8-bit data frame. As mentioned earlier, the I2C signal lines idle at logic HIGH. In the start of a communication, the master will pull the SDA line to logic LOW to notify all device that a transmission is about to begin. The master then generates a clock signal on the SCL line while outputting the 7-bit address MSB first through TTL logic levels in alignment with the clock pulse. This way the slave device can observe the logic level on the SDA line whenever a HIGH is read on the SCL line. After the 7-bit address, a read/write bit is sent with logic HIGH as read from slave and logic LOW and writes to slave. Following the R/W bit, an ACK acknowledgment bit is sent by the receiving device by pulling the SDA signal to LOW on the 9th clock pulse. If SDA is not pulled LOW, then this indicate an error in either the address or the slave device.

The address frame is followed by the data frame. Depending on whether the slave device is writing to master or the master is writing to slave, the transmitting device will start the frame by pulling SDA low like before. Master will then generate a clock pulse and a one byte data packet will be sent by the transmitting device as 8 bits. And finally the data frame end with an ACK bit sent by the device receiving data. The data frame repeats until all data are sent. Then a stop condition is generated by the master by shifting the SDA line from LOW to HIGH in the middle of a clock pulse HIGH in the SCL line.

Traditionally, the I2C protocol only support 7-bit addresses and low speed communication with peripherals in comparison to SPI. However, recent revisions allow the device to support 10-bit addresses and increase clock frequency from 100 kHz to 5 MHz, this allows much faster data rate.

COMMUNICATION INTERFACE: SPI

The Serial Peripheral Interface (SPI) is another synchronous serial communication interface developed by Motorola in 1985. In comparison to I2C, SPI support faster data transfer rate, therefore, it is often implemented in systems with sensor and memory storage devices like the Secure Digital (SD) card.

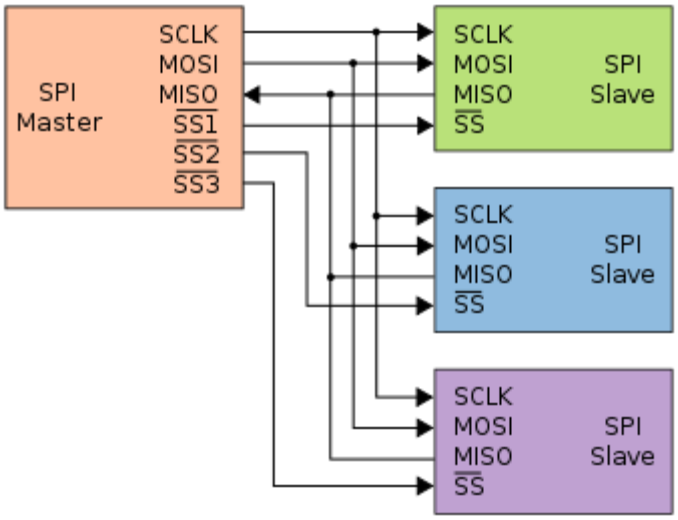


FIGURE 4-10: SPI BUS WITH 1 MASTER AND 3 SLAVE DEVICES [10]

SPI is a 4-wire serial bus in comparison to I2C and UART's 2-wire connection. Figure 4-10 illustrates a SPI bus that connects 1 Master and 3 slave devices. Like most synchronous communication protocol, a clock line (SCL) is required in SPI in order for the devices to be in sync while communicating. Unlike I2C where the SDA line is bidirectional between Master and Slave, SPI's data lines are unidirectional. The two data signal lines are Master Out Slave In (MOSI) and Master In Slave Out (MISO). The two lines are relatively self-explanatory. MOSI line is dedicated to data signal from Master to Slave. On the other hand, MISO is dedicated to data signal from Slave to Master. Finally, since SPI lack an addressing system like I2C, a Slave Select (SS) line is required so that the Slave device know that the Master device is talking to it. This could be seen as an advantage and disadvantage. The advantage is that SPI would not suffer the device number limitation from the 7-bit addresses that I2C have. However, for each slave device attach to the bus, the Master must provide a SS digital pin. Therefore, the device number is now limited by the number of pin the Master device have. In addition, wiring could get messy. This

problem could be overcome in some situations by connecting the device in a daisy chain configuration. However, this could make implementation more difficult.

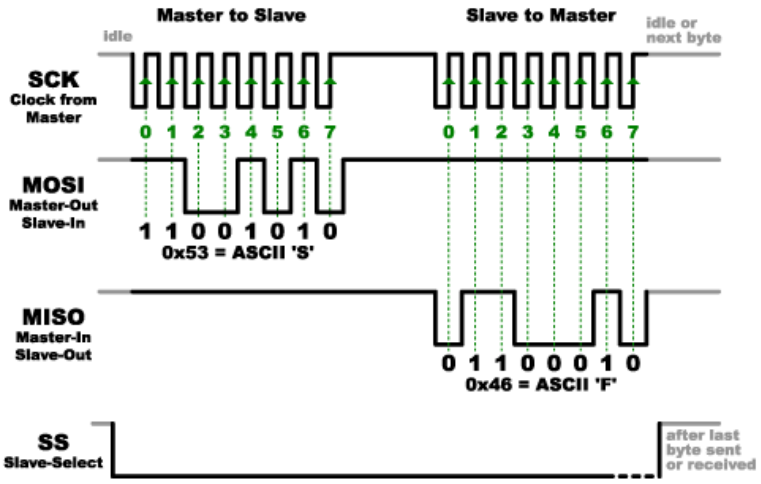


FIGURE 4-11: SPI COMMUNICATION MESSAGES [11]

The TLL signal in an SPI communication message is illustrated in Figure 4-11. Like I2C, the signal lines idle at logic HIGH. To start the communication procedure, the Master first pull the SS line to LOW and then generate a clock pulse. At the same time, it transmits data to the Slave device through TTL signal in the MOSI line. In Figure 4-11, an ASCII 'S' is transmitted in LSB first. Note that an SPI protocol can be set to MSB first or LSB first. The properties must match on both Master and Slave in order for a valid message to be transmitted. After the Master finish transmitting various bytes of data, it stop generate the clock pulse. If the Slave device is supposed to transmit something back, the Master will then generate another clock pulse after the pause. The Slave device will then transmit data to Master through the MISO line. After completion of the entire communication, Master will set the SS line to HIGH again.

Basic Digital Signal Processing (Digital Filter)

It is inevitable that the values provided by the sensors are inaccurate. The error in a reading is consisted of a systematic component and a random component. The systematic error is also known as the bias. This an error inherited in the system, an example of systematic error is the drift in a gyroscope. The random error is the random fluctuation in the readings. The errors are commonly referred to as noise.

While systematic error can usually be reduced by properly calibrating the sensor, the random error is usually the problem. Traditionally, RC circuit filters are used to process analog signals. However, electronics nowadays are usually digital. Hence the digital filters are needed. This section will discuss the moving average filter and the Kalman filter that is commonly used in digital signal processing.

Moving Average Filter

The moving average filter is one of the simplest and most commonly used signal smoothing filters. The basic concept of the filter is to smooth senor readings by computing the average using several previous readings. The mathematical equation is as follow:

$$\theta_{filtered} = \frac{\theta_t + \theta_{t-1} + \theta_{t-2} + \cdots + \theta_{t-(N-1)}}{N}$$

The variable N represents the number of terms being used in the filter while θ_t denote the sensor reading at current time. Furthermore, θ_{t-1} represent the previous reading and θ_{t-2} represent the reading before θ_{t-1}. Generally, the more terms used in a filter, the smoother the data become. However, that's a delay problem that will be discussed shortly. The best way to implement the moving average filter is by storing the sensor readings in an array.

At the beginning of each cycle, the array must be updated so that the new reading is placed at the top position while pushing all other elements down one memory slot. The last element of the array gets remove from the list.

For example, a 3 terms moving average filter is implemented below:

Update Phase

$$\theta_{t-2} = \theta_{t-1}$$

$$\theta_{t-1} = \theta_t$$

$$\theta_t = new\ sensor\ reading$$

Calculation Phase

$$\theta_{filtered} = \frac{\theta_t + \theta_{t-1} + \theta_{t-2}}{3}$$

The moving average filter use only past data points, as future data points are unknown. Hence, there will always be a delay of $\frac{N-1}{2}$ samples. So in the example, the signal is only 1 sample behind. If the sensor is taking data at 5 Hz, then the delay is 0.2 sec. However, let us assume that the signal is very noisy and that a filter of 10 terms is necessary. That's a delay of 4.5 samples, or 0.9 sec.

The delay caused by the averaging process can be manually adjusted by shifting the time scale. This is very useful when the filter is used in data post processing, such as interpreting signals from a noisy thrust stand. However, this is not an option for real time application like IMU filtering since time shifting is not possible.

KALMAN FILTER

The Kalman filter is a powerful estimator that was first implemented to estimate spacecraft trajectory for the Apollo missions. The Kalman filter compares model based prediction with measurements read by sensors overtime and provide a more accurate estimation. Since the Kalman filter is a model based estimator, it is able to estimate the state variables even when sensor measurements are missing. This was particularly useful during the Apollo age while the spacecraft is at the far side of the moon. There are many great literatures on the Internet regarding the theory of the Kalman Filter. Therefore, this subsection will only focus on basic implementation.

State Space Model

$$x_k = Ax_{k-1} + Bu_k + w_k$$

$$z_k = Cx_k + v_k$$

The Kalman filter predicts state variables using physics based state space model. For simple GPS location tracking, this could be basic kinematics equation. And for more advanced motion estimation, this would involve the equations of motion to fuse data gathered from multiple sensors. Since the Kalman filter use a state space physics model, the system must be linear.

In the equations above, x represent the state vectors, containing variables such as acceleration, velocity and displacements. The subscript k denote the current state while $k-1$ denote previous state. The variable u_k is the current system input, such as acceleration for motion tracking. The process noise is denoted by w_k. On the other hand, z_k is the current observation while v_k is the noise in measurement. The matrix A is called the state transition matrix, the current predicted state vector is resulted when it is multiple by the previous state vector. The

matrix B is the input matrix, it is used to map the input commands to the proper state variables. Finally, the matrix C is the observation matrix for selecting the state variables to be observed.

Predict Phase

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$$

$$P_k = AP_{k-1}A^T + Q_k$$

In the prediction phase, the state transition matrix is used along with the input command to predict the current state. The noise of the system is predicted as well with P_{k-1} being the previous error covariance matrix and P_k being the current error covariance matrix. The matrix Q is the process noise covariance matrix that's estimating the processing noise w_k. Note that the state vector $\hat{x}$ is the estimated state.

Update Phase

$$G_k = P_k C^T (CP_k C^T + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k + G_k(z_k - C\hat{x}_k)$$

$$P_k = (I - G_k C)P_k$$

In the update phase the Kalman gain, G_k is updated with the error covariance matrix, observation matrix and the measurement noise covariance matrix R that's estimating the measurement noise v_k. The sensor measurements are then compared with the predicted state variables. The differences are multiple by the Kalman gain and added to the predicted states to get an estimated state. Finally, error covariance matrix is updated. The estimated state combine information from motion the physics model and sensor measurements, therefore, it tends to be more accurate. Examples will be given in later section with application in IMU attitude estimation and GPS position tracking.

Chapter 5: Avionics

Flight System Architecture

The overall space hopper simulator flight system is broken into 5 subsystems. They are the power, propulsion, wireless communication, data acquisition and flight logging systems. The flight operating system (FlightOS) is in charge of controlling all the subsystems in order for the aircraft to fly. The flight system architecture is illustrated in Figure 5-1, showing the components in each subsystem and their respective communication interfaces.

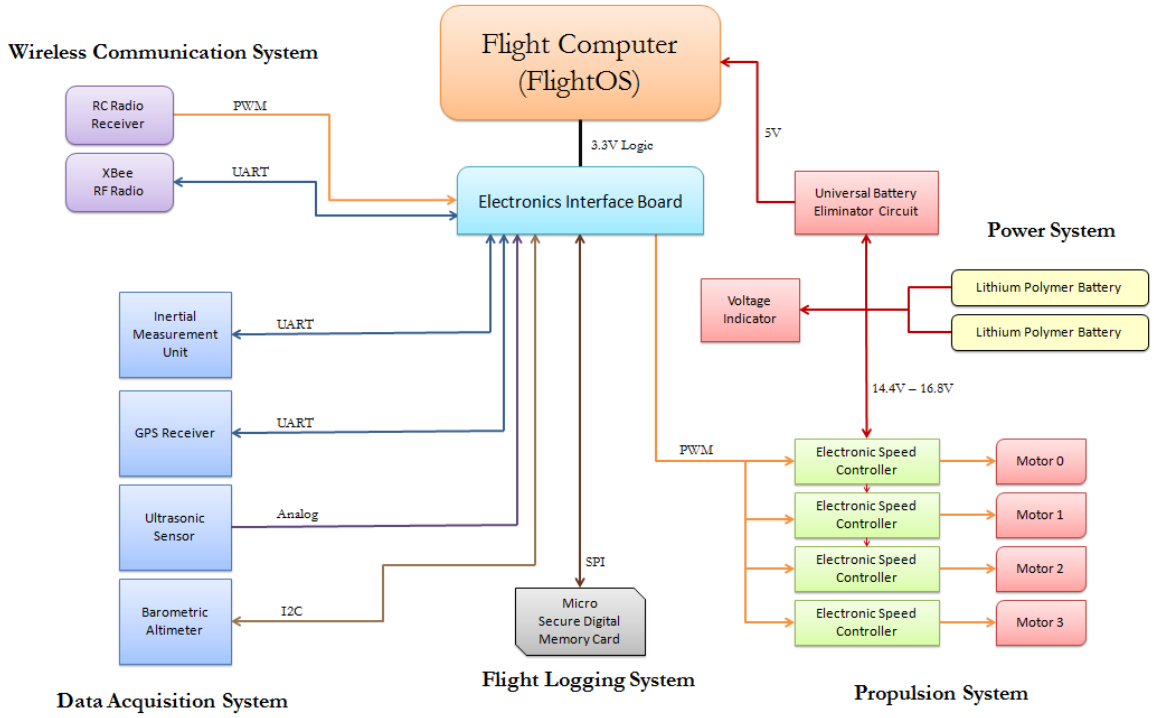


FIGURE 5-1: QUADX FLIGHT SYSTEM ARCHITECTURE

FLIGHT COMPUTER

The flight computer is the brain of the space hopper simulator. Its duty is to run the FlightOS and interact with all the subsystems. While traditional winged RC aircrafts are able to fly without a dedicated flight computer, it is almost impossible to manually control rotorcrafts due to that unstable nature. The flight computer is essential because the flight performance is depended on the speed of the system. The research went through three major hardware and software revisions in search for an ideal development and deployment platform.

In the Fall of 2013, it was decided that the guidance and navigation system could take a lot of processing power to run path-finding and optimization algorithm. In order to speed up software development, the flight system would be developed in MATLAB while running Ubuntu Linux. Therefore, the Intel D33217GKE "Golden Lake" Ivy Bridge i3 NUC was chosen as the flight computer. At a retail price of around $550, the computer was powered by an Intel Core i3-3217U central processing unit (CPU) that is clocked at 1.8 GHz. The motherboard was fitted with a 64 GB solid state drive (SSD) and 4 GB of DDR3 random access memory (RAM) to optimize computation performance. An image of the computer is illustrated in Figure 5-2.

FIGURE 5-2: INTEL D33217GKE "GOLDEN LAKE" NUC

Due to the limited electronic interfaces on the Intel NUC, all avionics were interfaced by universal serial bus (USB). Therefore the FTDI FT232RL USB to Serial breakout boards were required for all devices that utilize a UART connection, including the IMU and the wireless communication radio. In addition, the Pololu Micro Maestro 6-Channel USB Servo Controller was used to output PWM signals to control the motor. In order to interface with other electronics and detect LiPo voltage level, a real time flight data acquisition board was designed. The breadboard prototype is illustrated in Figure 5-3.

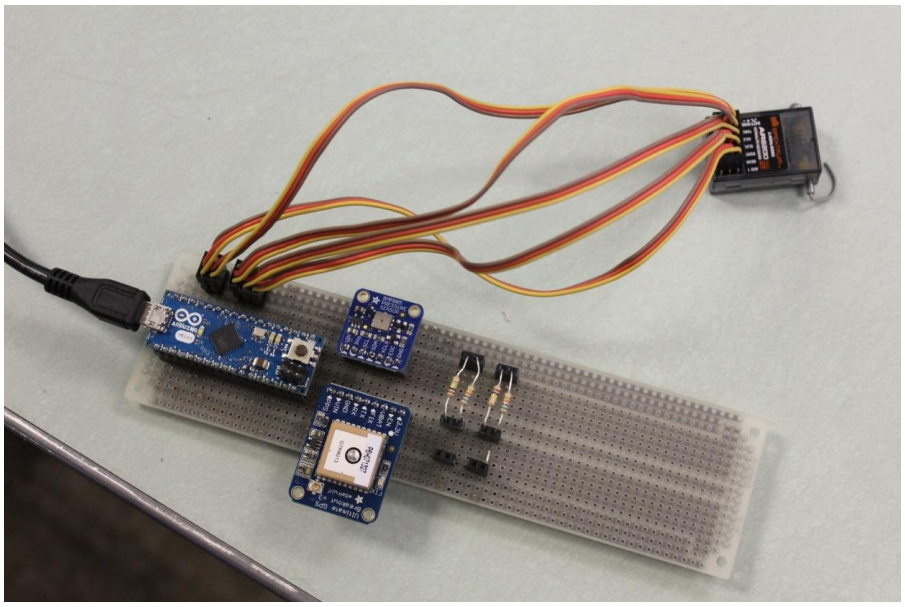


FIGURE 5-3: ARDUDAQ REAL-TIME FLIGHT DATA ACQUISITION SYSTEM PROTOTYPE

To extend the interface capability of the Intel NUC, an Arduino Micro was used to run the ArduDAQ, flight data acquisition system. An electronic schematic of the interface board can be found in Figure 5-4 and the printed circuit board (PCB) layout along with the actual printed board with electronics attached can be found in Figure 5-5.

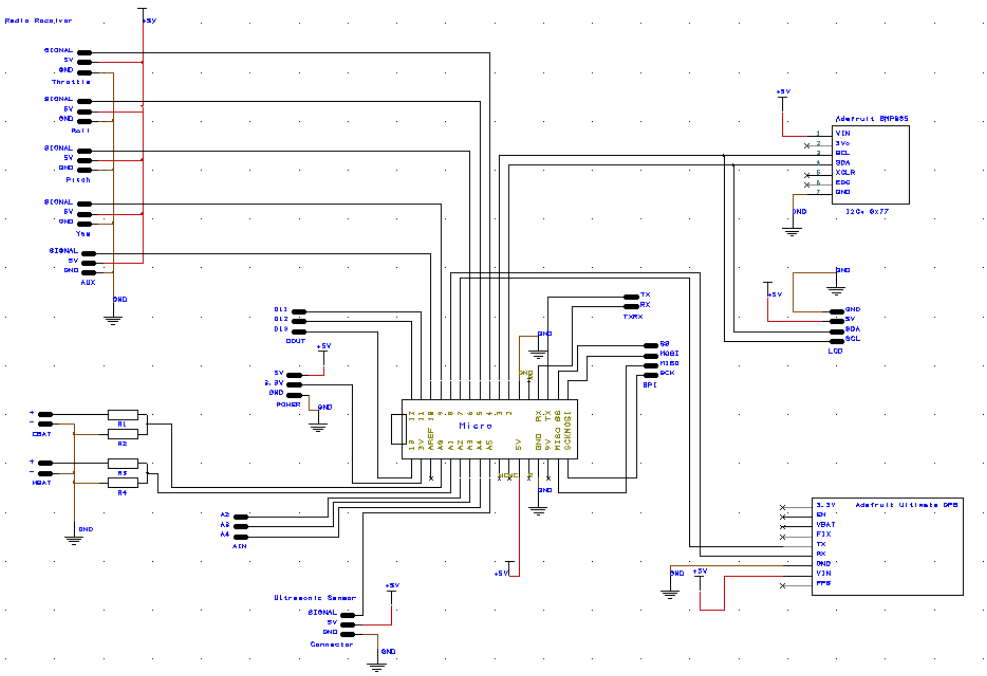


FIGURE 5-4: ARDUDAQ SCHEMATIC

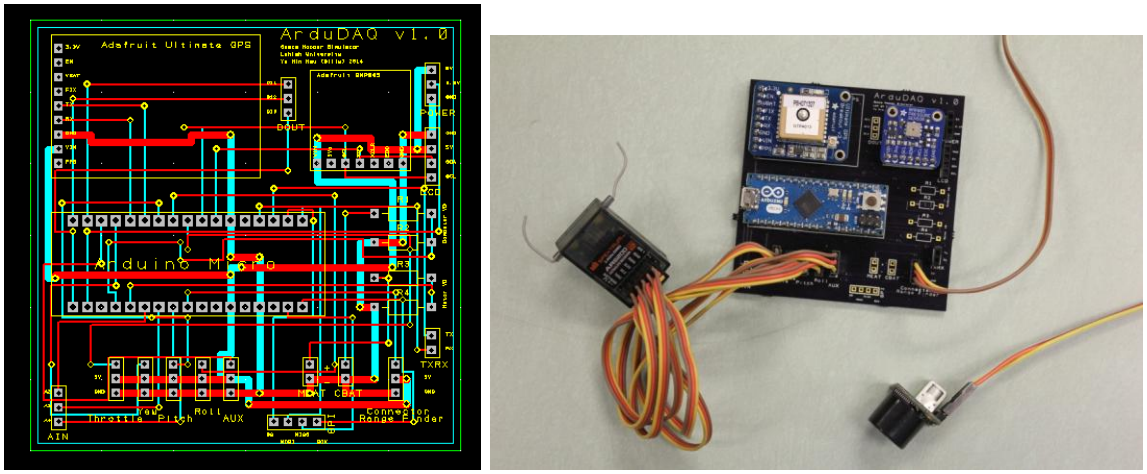


FIGURE 5-5: ARDUDAQ PCB LAYOUT AND PRINTED BOARD WITH SENSORS

Much progress was made in developing the flight system in MATLAB. By the beginning of Spring 2014, after replacing the avionic system of the 2nd generation prototype simulator, the team was able to conduct rig test on the hexacopter with the flight controlling the simulator but not physically mounted on it. Overall, satisfactory results were obtained using the PI-P controller while only connecting the IMU and the servo controller. However, problems arise when trying to prepare the platform for flight test along with other electronics. In summary, the overhead of running MATLAB along with the USB interface take a major toll on the software performance. Topics regarding software and hardware optimization will be discussed later. In addition software performance, there were several hardware reliability issues with the NUC that poses a risk of the machine failing during mid-flight. With the Intel NUC being such as expensive piece of hardware, the risk is unacceptable and decision was made to switch to a different development platform.

In late Spring of 2014, the BeagleBone Black Rev B was chosen as the new flight computer for the 3rd generation hopper spacecraft simulators and software development restarted using Python as the programming platform. Python since it require less overhead when compared to MATLAB, therefore yield better performance. In addition, it is not as difficult as C++ and also many a wide range of open source libraries available similar to MATLAB's toolboxes, allowing rapid software development.

The BeagleBone Black (BBB) is an open source single-board computer that features an AM335x ARM Cortex-A8 processor that's clocked at a speed of 1GHz. It is fitted with 4 GB of eMMC flash memory and 512 MB of DDR3 RAM. There are two 32-bit microcontrollers that partially allow direct interfacing with some external hardware through its general purpose output/input (GPIO)

pins that can be configure as both digital and analog. An image of the BBB was illustrated in Figure 5-6.

FIGURE 5-6: BEAGLEBONE BLACK IN PROTECTION CASE

The BBB's hardware specification was not as impressive as the Intel NUC. However, it performed remarkably well considering its $45 price tag, making it more suitable for flight testing without the fear for crashing. The BBB natively run on the Angstrom Linux distribution. This was later modified to Debian Linux for easier customization. In comparison, the data acquisition loop for retrieving radio PPM signal was benchmark at a frequency of 10 Hz in MATLAB with the Intel NUC, while using BBB and Python, the same loop achieve a speed of 70 Hz. On the other hand, the rig test control loop speed of around 115 Hz was achieved using the BBB as opposed to the NUC's speed of around 25 Hz. The speed up is mostly a result of software and hardware interface optimization.

The BeagleBone Black performs considerably better than the Intel NUC, and that the rig test performance improved, however, the control loop speed is still not ideal. While the PID controller was able to control the quadcopter with reasonable performance, the Fuzzy Logic controller was more sensitive to the time delay. Since the BBB doesn't have enough PWM

output pins and that it cannot read the RC radio directly, the data acquisition board and the external servo controller is still required. They became the bottleneck of the system when combine with the rest of the control system. Initially, it was hoped that parallel processing would solve the problem. However, a major performance drop still occurred. This slowdown was to the point that even the PID controller produce unstable response.

In December of 2014, optimization effort with the BeagleBone Black stopped and development of the FlightOS switched to the Arduino Due as the final flight computer and operating system revision. An image of the Arduino Due can be found in Figure 5-7.

FIGURE 5-7: ARDUINO DUE

The Arduino Due is another popular open source development platform that cost around $50. This microcontroller is different than the previous Arduino board in that it is powered by an Atmel SAM3X8E ARM Cortex-M3 32-bit CPU as opposed the traditional ATmega328 8-bit microcontroller. The new CPU run at 84 MHz a great improvement to the traditional Arduino operate at 16 MHz. The board has 96 KB of SRAM and 512 KB of flash memory. Like all previous

Arduino board, the Arduino Due is capable of analog, digital, PWM, UART, SPI and I2C interfaces. This eliminates the need of a strap-on data acquisition system, the major bottleneck in the previous flight computers. However, a downside of the Arduino Due is that it operate of 3.3V logic, hence, any voltage above such level will destroy the board.

The hardware specification of the Arduino Due is not at all comparable to the Intel NUC and the BeagleBone Black. However, due to the lack of a fully featured Linux operating system that adds significant overhead, the raw processing power of the hardware can be harnessed fully. The microcontroller only runs the FlightOS as developed by the team and no other background processes. Hence, all instructions were run at real time and minimal time delay. As a result the control loop received a significant speed boost to around 210 Hz. However, because of the minimalistic nature of the platform, scripting languages that required an interpreter such as Python, Java and MATLAB cannot be used. Therefore the FlightOS was developed purely in C++ and compiled to machine code so that it could be executed in the microcontroller. This added a layer of difficulty to the system software development since C++ is not as forgiving.

A new electronic interface PCB was designed for the Arduino Due since the data acquisition system was no longer needed. The PCB was designed to attach directly on top of the Arduino Due and reroute its pins so that the electronics can be attached easily. The schematic and PCB layout can be found in Figure 5-8 and Figure 5-9.

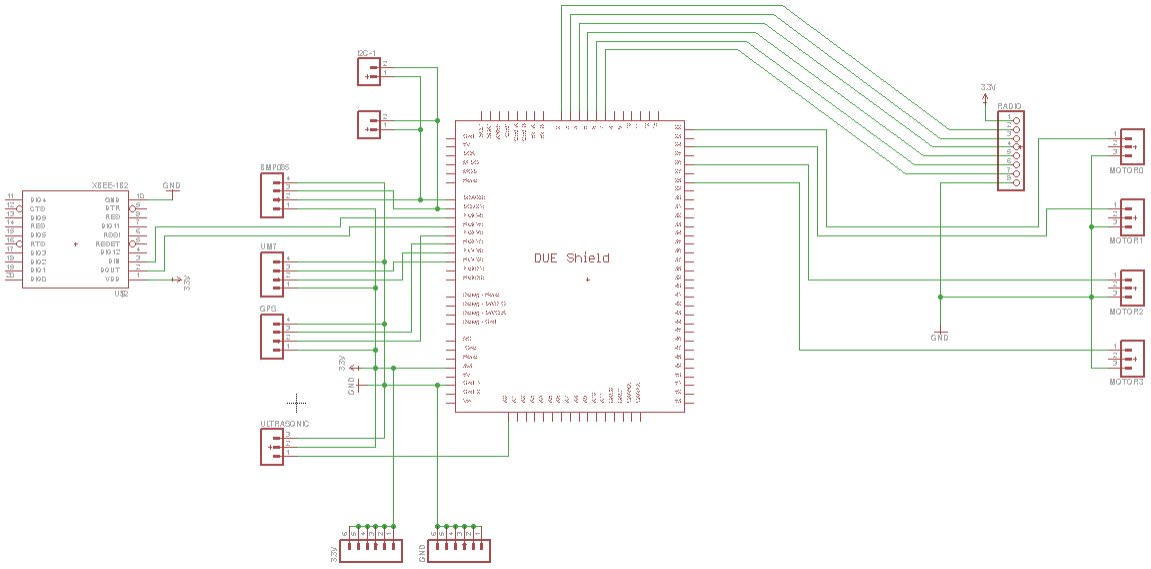


FIGURE 5-8: QUADX FLIGHTOS SHIELD SCHEMATIC

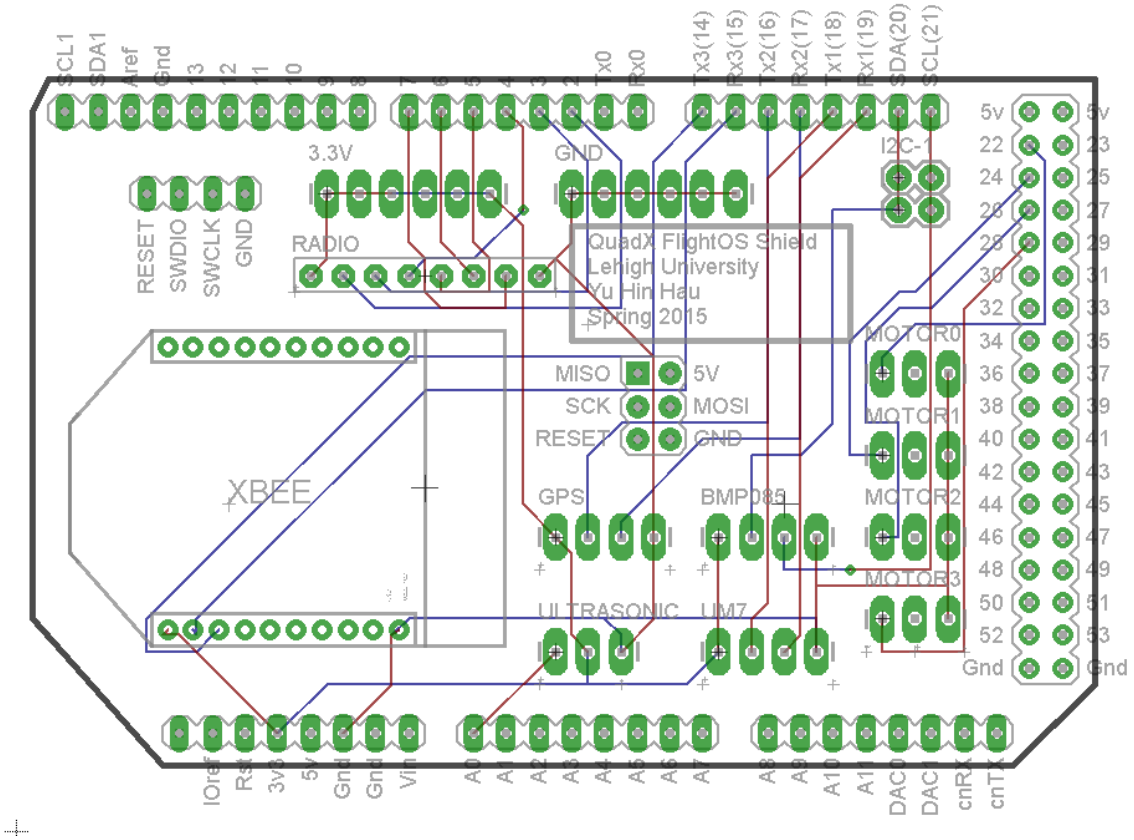


FIGURE 5-9: QUADX FLIGHTOS SHIELD

INERTIAL MEASUREMENT UNIT (IMU)

The inertial measurement unit or IMU is an array of sensors that is designed to estimation orientation. A typical 6 degree-of-freedom IMU is consisted of a 3-axis gyroscope and a 3-axis accelerometer. Though more advanced IMU often in addition include a 3-axis magnetometer for increased accuracy. In empty space, usually only the gyroscope is used due to the lack of a significant magnetic field and gravity field.

The gyroscope is a sensor that is used to measure angular velocity. It can accurately give readings of roll rate, pitch rate and yaw rate when the sensor is rotated. Ideally, the reading from a gyroscope can be integrated to give angular data. Equation for pitching angle is as follow, but this equation works for all roll, pitch, and yaw.

$$\theta = \int \dot{\theta}\, dt$$

$$\boxed{\theta_{gyro_{new}} = \dot{\theta}_{gyro} * dt + \theta_{gyro_{old}}} \tag{5.1}$$

However, the readings from a gyroscope usually have a bias, after integration, this cause the sensor calculated angle to drift away from the actual angle in the long run. Because of this, the angle calculated is usually more accurate in the short term, hence a high pass filter is necessary.

The accelerometer is a sensor that is used to measure acceleration in the sensor's respective axes. In a stationary environment, the gravity vector can be calculated to determine orientation.

Recall that a unit vector is equal to the vector divided by its magnitude. The gravity vector given by the accelerometer is in the body frame, and the unit vector is calculated as follow:

$$\hat{G}_c = \frac{\vec{G}_c}{|\vec{G}_c|} = \frac{1}{\sqrt{G_{c_x}^2 + G_{c_y}^2 + G_{c_z}^2}} \begin{bmatrix} G_{c_x} \\ G_{c_y} \\ G_{c_z} \end{bmatrix}_c$$

Recall the rotation matrix ${}^{c}C^{n}$ derived in Chapter 2. This rotation matrix can be used to convert a gravity unit vector, $[0\ 0\ 1]^{T}$, from inertial frame to body frame. Assuming that that sensor is oriented the same as the body frame is defined in Chapter 2, the relationship is as follow:

$$\hat{G}_c = \quad {}^{c}C^{n} * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_n$$

$$\frac{1}{\sqrt{G_{c_x}^2 + G_{c_y}^{\ 2} + G_{c_z}^{\ 2}}} \begin{bmatrix} G_{c_x} \\ G_{c_y} \\ G_{c_z} \end{bmatrix}_c = \begin{bmatrix} -sin\theta \\ cos\theta\ sin\phi \\ cos\theta\ cos\phi \end{bmatrix}_n$$

Using the relationship above, solve for roll:

$$\frac{cos\theta\ sin\phi}{cos\theta\ cos\phi} = \frac{G_{c_y}}{G_{c_z}}$$

$$\tan\phi = \frac{\mathrm{G_{c_y}}}{G_{c_z}}$$

$$\boxed{\phi_{acc} = atan2\left(\frac{G_{c_y}}{G_{c_z}}\right)} \tag{5.2}$$

Then, the same relationship can be used to solve for pitch:

$$tan\theta = \frac{sin\theta}{cos\theta} = -\frac{G_{c_x}}{cos\theta}\left(\frac{1}{\sqrt{G_{c_x}^2 + G_{c_y}^{\ 2} + G_{c_z}^{\ 2}}}\right)$$

To find $cos\theta$, recall the trigonometric identity $sin\phi^2 + cos\phi^2 = 1$,

$$cos\theta = cos\theta(sin\phi^2 + cos\phi^2) = cos\theta sin\phi * sin\phi + cos\theta cos\phi * cos\phi$$

$$= G_{c_y} sin\phi + G_{c_z} \cos\phi \left(\frac{1}{\sqrt{G_{c_x}^2 + G_{c_y}^2 + G_{c_z}^2}} \right)$$

Finally, plugging the above equation back into the original equation,

$$tan\theta = -\frac{G_{c_x}}{G_{c_y} sin\phi + G_{c_z} cos\phi} = -\frac{G_{c_x}}{\sqrt{G_{c_y}^2 + G_{c_z}^2}}$$

$$\boxed{\theta_{acc} = atan2\left(-\frac{G_{c_x}}{\sqrt{G_{c_y}^2 + G_{c_z}^2}} \right)} \quad (5.3)$$

Equation 5.2 and 5.3 were derived with the assumption that the sensor was stationary and that the only force acting of it was the gravitational pull. While this assumption might be true in most cases, this is not true when an aircraft is doing a sudden maneuver. Hence, low past filter is required on this sensor [12].

IMU Complementary Filter

To get the best out of the gyroscope and accelerometer, usually a sensor fusion algorithm is used to calculate the optimal angle output. A simple implementation is the complementary filter, as shown in Equation 5.4:

$$\boxed{\theta = (1 - \alpha)\theta_{gyro} + \alpha\, \theta_{acc}} \quad (5.4)$$

Again, even though this equation is written for pitch, this algorithm is applicable to both roll and pitch. Unfortunately, yaw angle cannot be calculated from the accelerometer, hence the angle

can only rely on the gyroscope. In more advanced sensor fusion implementation, the magnetometer can assist in calculating the yaw angle.

The complementary filter takes a portion of the estimated angle from the gyroscope and a portion of that from the accelerometer and combines them together. The term α is used to control how much the filter relies on the gyroscope and how much it relay on the accelerometer. This value is also related to the filter's time constant and determines how fast the filter responds to real life changes. Generally, α is chosen at around 0.02, where 98% of the gyro angle is used and only 2% of the acc angle is used. This combination applies a high pass action on the gyro angle and a low pass action on the acc angle. As always, the α should be tuned for optimal performance.

IMU KALMAN FILTER

A more advance method of data fusion is the Kalman filter. In the case for an IMU, the gyroscope data is often more reliable than the accelerometer since it is independent of external forces. Therefore the gyro angular velocity is used as the input. The accelerometer angle is used as the observed angle and the Kalman filter will calculate the differences between the two to output optimal angle estimation [13].

The state space model for roll and pitch estimation is as follow:

Prediction

$$x_k = Ax_{k-1} + Bu + w_k$$

$$\begin{bmatrix} \phi \\ \dot{\phi}_b \\ \theta \\ \dot{\theta}_b \end{bmatrix}_k = \begin{bmatrix} 1 & -dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -dt \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi}_b \\ \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1} + \begin{bmatrix} dt & 0 \\ 0 & 0 \\ 0 & dt \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi}_{gyro} \\ \dot{\theta}_{gyro} \end{bmatrix}_k + w_k$$

The state vector of the system includes roll, pitch and the biases in the gyroscope's measurement. In the prediction phase, the future angle is predicted as the sum of previous angle, the angle calculated from the gyroscope data, and the process noise. In order to make the prediction more accurate, it is necessary to subtract the angle drifted from the gyro bias. The gyro biases are assumed to remain the same. Since the initial state is unknown, the state vector is initialized as a zero vector.

$$w_k \sim N(0, Q_k)$$

$$Q_k = \begin{bmatrix} Q_\phi & 0 & 0 & 0 \\ 0 & Q_{\dot{\phi}} & 0 & 0 \\ 0 & 0 & Q_\theta & 0 \\ 0 & 0 & 0 & Q_{\dot{\theta}_b} \end{bmatrix} dt$$

The process noise is assumed distributed normally with zero mean with covariance Q_k. To simplify, non-diagonal terms are assumed to be zero. The diagonal terms are tuned to achieve optimal performance. In general, the values represent the amount the state deviate from the model in reality. This includes noise in general as well as intentional input changes.

Observation

$$z_k = Cx_{k-1} + v_k$$

$$\begin{bmatrix} \phi_{acc} \\ \theta_{acc} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \end{bmatrix}_k + v_k$$

The two terms being observed are the roll and pitch angle calculated from accelerometer measurements.

$$v_k \sim N(0, R_k)$$

$$R = \begin{bmatrix} R_\phi & 0 \\ 0 & R_\theta \end{bmatrix}$$

The measurement noise v_k, again, is assumed to be distributed normally with a mean of zero with a covariance of R_k. Similarly, to simplify, the non-diagonal terms of the measurement covariance matrix are assumed to be zero. The diagonal terms, however, greatly affect the performance of the Kalman Filter. In general, the greater the value the more the filter favors the estimation predicted by the physics model. On the other, the lesser the value, the more the filter favor the measurements. In order to achieve optimal reading, the measurement covariance must be tuned carefully.

Initialization

$$P_k = \begin{bmatrix} L & 0 & 0 & 0 \\ 0 & L & 0 & 0 \\ 0 & 0 & L & 0 \\ 0 & 0 & 0 & L \end{bmatrix}$$

$$G_k = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Thus far, all variables of the Kalman filter are covered other than the error covariance matrix P_k and the Kalman gain G_k. For the error covariance, again, to simplify, the non-diagonal terms are assumed to be zero. Since the Kalman filter will alter the error covariance over time, if the initial state is unknown, then the P_k matrix should just be initialized with large values to represent large errors in the estimated state. The Kalman gain will be calculate and replaced entirely during the update phase. Therefore, it can simplify be initialized as a zero matrix.

Implementation

To implement the Kalman filter, the prediction and update process equation from Chapter 4 should be used. A Simulink model was constructed to gauge the effectiveness of using a Kalman filter in an IMU and also to verify the equations derived earlier in this section. To begin, a Kalman angle for a single axis is modeled to study the effect of each parameter.

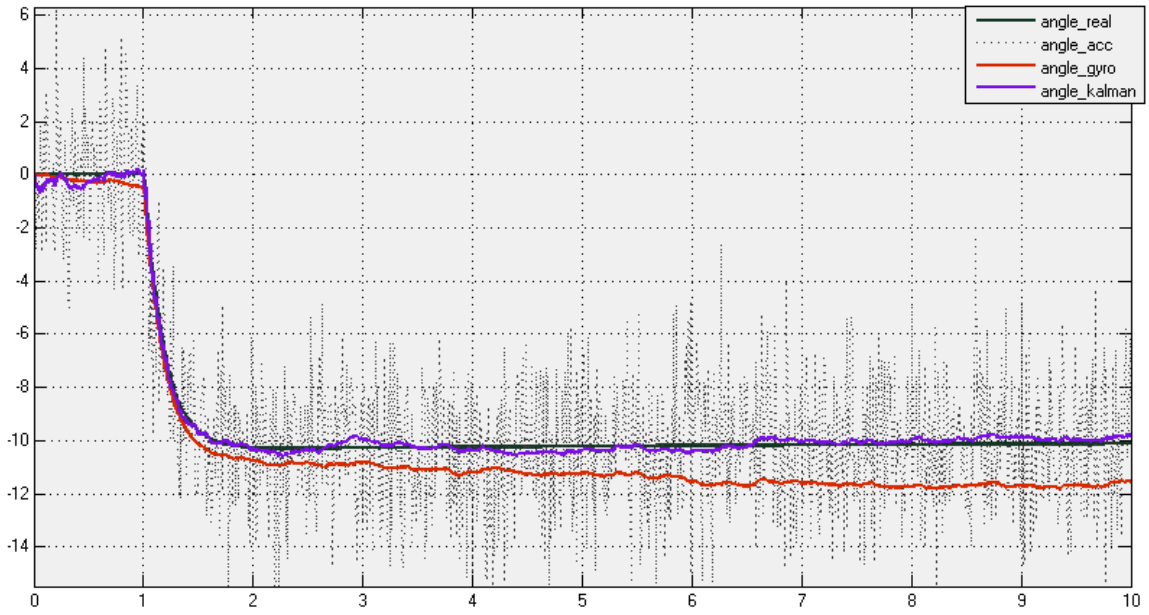


FIGURE 5-10: ATTITUDE ESTIMATION WITH ACCELEROMETER, GYROSCOPE AND KALMAN FILTER (R=5000)

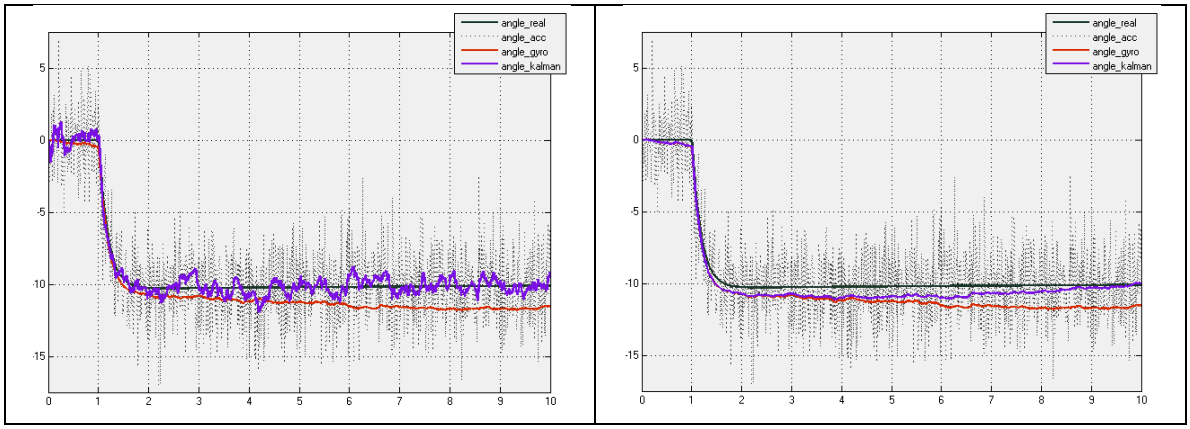


FIGURE 5-11: KALMAN FILTER MEASUREMENT NOISE TUNING (R = 1 VS. R=1000000)

In the simulation, a step command of $-10°$ is issued to a PID controller one second after the start of the simulation. To focus on sensor behavior, the only noises added are to the gyroscope and accelerometer. In addition, it is assumed that the signals going into the PID controller are not corrupted and are connected directly from the physics dynamics simulator.

Comparison between the angles estimated from pure accelerometer, gyroscope and the Kalman filter are illustrated in Figure 5-11. As seen in the figure, the accelerometer suffers from high frequency vibration noise. The signal variance is around $\pm 5°$. On the other hand, the drifting effect or the gyroscope is obvious due to the noise in the measured angular velocity. This cause the estimated angle to drift almost $2°$ at the end of the simulation. The Kalman filter estimated angle is much closer to the actual angle. It exhibit smooth behavior like the gyro angle, yet it overcome the drifting issue by referencing values measured by the accelerometer.

The Kalman filter's behavior is greatly affected by the tuning of the measurement covariance. Figure 5-11 show estimated angle when the measurement covariance are tuned too low and too high. When the measurement covariance is tuned low, it is apparent that the filter favors the data from the measurement. Hence, the estimated angle is a lot more oscillatory. On the other hand, when the measurement covariance is tuned too high, the filter favors the gyro calculated value too much. Hence, it partly suffers from the drifting angle estimation. However, the filter eventually converges back to the actual angle before the end of the simulation.

The Kalman filter is compared with the simpler complementary filter in Figure 5-12. As shown, both estimation are quite similar. One could argue that the Kalman filter is a bit better, but the differences are neglectable. Either way, the 2-axis Kalman filter was modeled with the roll step to $10°$ and pitch step to $-10°$, the result is shown in Figure 5-13.

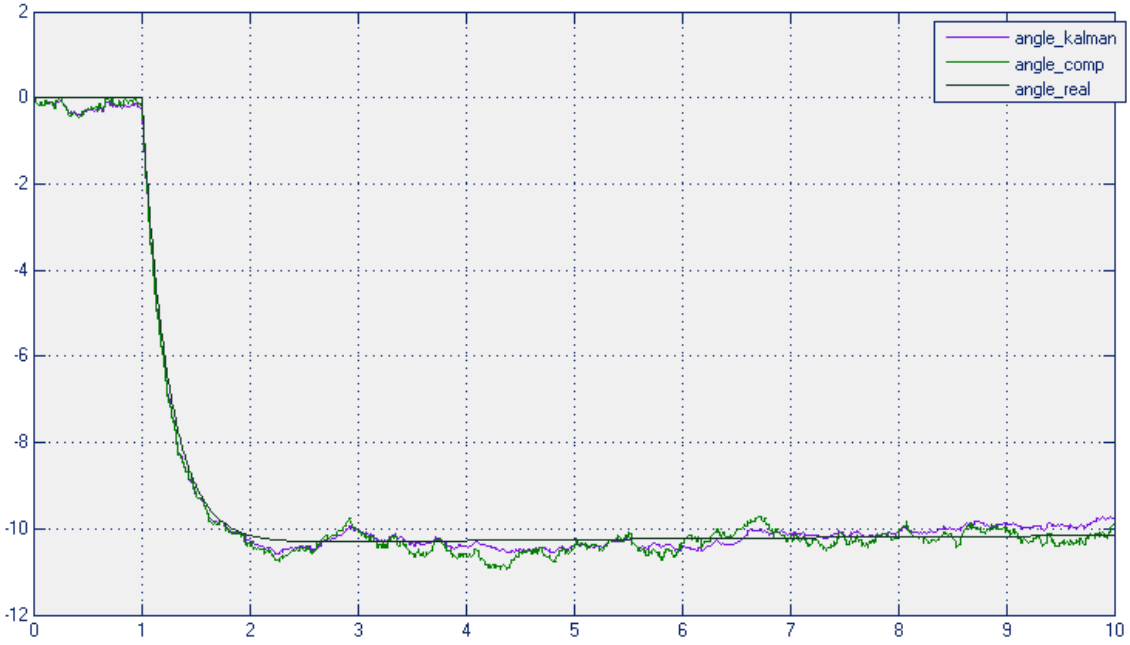


FIGURE 5-12: COMPARISON OF KALMAN FILTER AND COMPLEMENTARY FILTER ESTIMATED ATTITUDE

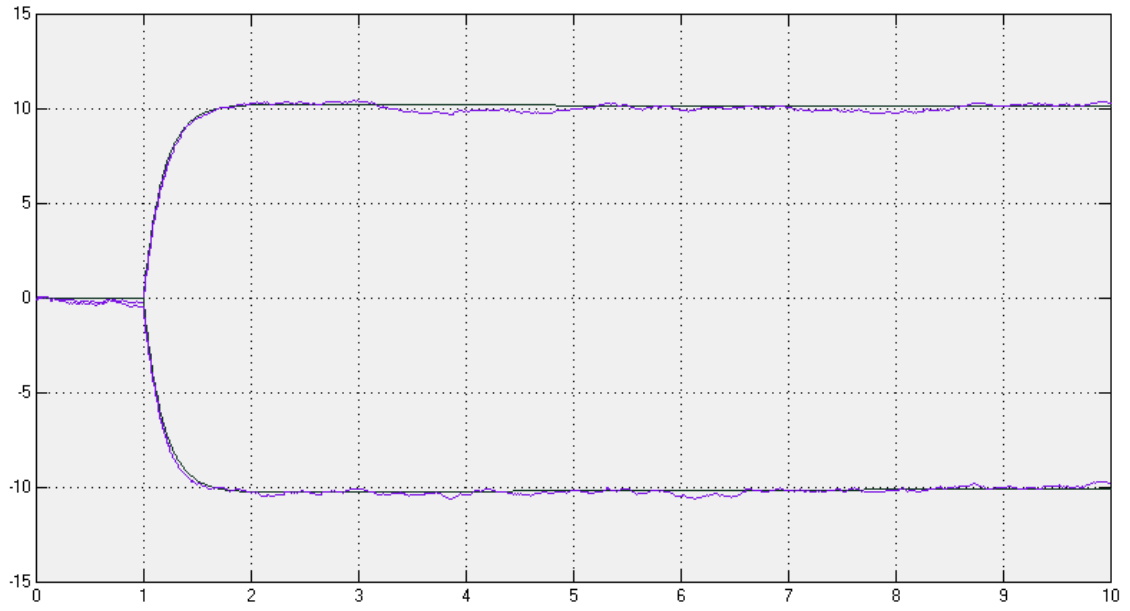


FIGURE 5-13: KALMAN FILTER ESTIMATED ROLL AND PITCH

FIGURE 5-14: CHROBOTICS UM7-LT ORIENTATION SENSOR [14]

The IMU discussed thus far are 6 DOF IMU with only an accelerometer and a gyroscope. Since there is no way for the accelerometer to calculate the yaw angle, it is depended entirely on the gyroscope. This also indicates that the yaw reading will suffer from gyro drift. To overcome this, a 9 DOF IMU is necessary with the addition of a magnetometer.

With the addition of the magnetometer, more advanced sensor fusion algorithm with the Kalman filter is required. Since the physics model for such integration is not linear, and the traditional Kalman filter only works on linear system, the extended Kalman filter (EKF) is required.

Because of this, the UM7 orientation sensor from CHRobotics was chosen. The UM7 is a 9 DOF IMU that has a built-in processor to retrieve measurements from the individual sensors and process them in the EKF. The processor provides an advantage as it off load part of the work that the flight computer must process. The EKF estimation rate is 500 Hz with an accuracy of around $\pm\ 3°$ for roll and pitch while the yaw has an accuracy of around $\pm 5°$. The device is natively 3.3V and can communicate with the flight computer via 3.3V TTL UART and SPI.

BAROMETRIC ALTIMETER

An atmospheric pressure gauge is usually used on aircraft to measure altitude. Similar, the QuadX multirotor is designed to use the BMP085 pressure sensor to determine its height. The BMP085 can be interfaced via I2C, but since it lacks an onboard processor, the flight computer must retrieve the measurement and calculate the altitude.

FIGURE 5-15: BAROMETRIC ALTIMETER

To convert measured pressure to altitude, the follow equation can be used:

$$Altitude = 44330 * \left(1 - \left(\frac{P}{P_0}\right)^{\frac{1}{5.255}}\right)$$

In the equation above, P is the measured pressure while P_0 is the referenced sea level pressure. The average sea level pressure is 101325 Pa. However, to obtain a more accurate reading, the referenced sea level pressure should be calibrated before flight with information from a weather station. It is important to notice that the altitude is expressed as meters above sea level and therefore does not provide a distance measurement above ground. In addition, since the altimeter will be affected by the turbulence created by the QuadX's propellers, the altitude reading might not be very accurate. Hence, an ultrasonic range sensor is required to obtain better altitude estimation.

ULTRASONIC RANGE SENSOR

FIGURE 5-16: MAXBOTIX MB1240 XL-MAXSONAR-EZ4 ULTRASONIC RANGEFINDER [15]

As discussed in Chapter 4, ultrasonic range sensor function by sending emitting an ultrasound wave and use the timing in the echo to determine object range. The MB1240 work similarly. But instead of having the microcontroller does the timing, the sensor take care of that internally and encode the range as a form of analog signal. The MB1240 having the following range to analog conversion relation:

$$\frac{V_{cc}/1024}{cm}$$

In another word every centimeter is represented by the power source voltage, V_{cc}, divided by 1024. To put this is equation form, where V_{analog} represent the voltage read from the analog pin from the microcontroller:

$$Range = \frac{1024}{V_{cc}} * V_{analog}$$

Like most ultrasonic sensor, the MB1240 have a minimal distance of 20 cm where object range within 20 cm will be reported as 20 cm. To accommodate for high acoustic noise, the MB1240 has low sensitive. It operates on a narrow beam pattern and is optimized for reporting large target with a max distance of 750 cm. Hence it is ideal for reporting distance above ground. When working together with the barometric altimeter, the ultrasonic sensor can cover more precise takeoff and landing maneuver while the former can be used to maintain flight altitude.

GLOBAL POSITION SYSTEM (GPS) RECEIVER

The QuadX use a GPS receiver to determine it location. The GPS receiver use the signals received from the GPS satellite constellation to calculate to receiver's longitude, latitude and altitude. 3 satellites are required to get a position fix. However, since the clock on the receivers is often not precise, a 4th satellite is usually required to get more accurate location estimation. For civilian application, the GPS signal has a worse case accuracy of 7.8 meters in space. Many high quality receivers are able to provide a horizontal accuracy better than 3.5 meters. However, this is only the signal in space, not taking into account of atmospheric effect, obstructions, signal bounce, etc. Often the altitude estimated by a GPS receiver is not accurate enough for aircraft altitude control. GPS can often combine with other systems such as the wide area augmentation system (WAAS) to provide better accuracy.

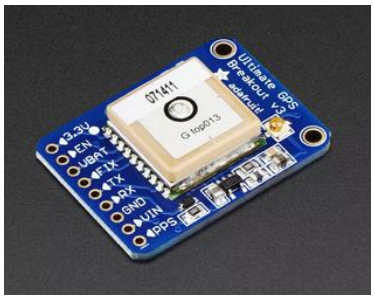


FIGURE 5-17: ADAFRUIT ULTIMATE GPS BREAKOUT - 66 CHANNELS W/ 10 HZ UPDATE [16]

The space hopper simulator uses an Adafruit Ultimate GPS receiver to estimate its location. The receiver is capable of searching for 66 satellites and tracking 22 at once. The position accuracy is less than 3 meters and has a velocity accuracy of around 0.1 meter/s in ideal situation. The interface is 3.3V TTL UART with a theoretical maximum update rate of 10 Hz. However, in reality for implementation, the maximum update rate is only 5 Hz. The receiver is set to output information in NMEA format, therefore a parser is required to extract the information.

RC TRANSMITTER AND RECEIVER

The 3rd generation hopper simulator is designed with a redundant wireless communication system to maintain control of the aircraft even if one of them fails. The primary control for manual flight is a standard RC transmitter and receiver. Older transmitter and receiver are based on the crystal technology. As long as the crystal oscillation frequency match on both the transmitter and receiver, than signal can be linked. More recently, the transmitter and receiver operate on 2.4 GHz wireless band. The transmitter and receiver will need to go through a simple binding process to ensure the radios are linked.

FIGURE 5-18: SPEKTRUM DX7 RC TRANSMITTER [17]

There are many shared transmitter and receiver in the Lehigh Aerospace System Lab. Therefore, the aircraft is control with an available one like the crystal Futaba T6EXA or the Spektrum DX7 in Figure 5-18. The channels in a RC system refer to the number of controls the transmitter can have. The standard rolls, pitch, yaw and throttle stick control already take up 4 channels. There are additional toggle and rotary switches that can be used to program different flight modes.

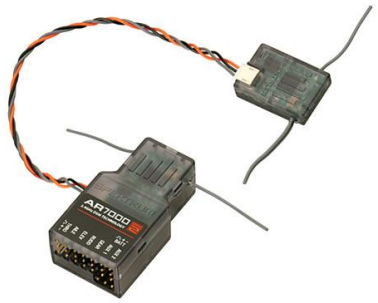


FIGURE 5-19: SPEKTRUM RC RECEIVER [17]

The signals output by a RC receiver take are digital PWM signals. In traditional RC airplane, this PWM signal would directly drive the ESC and servos to control the motor and control surfaces on the aircraft. In multiple, the signal is read by the flight computer to interpret as an input command for the flight controller. The flight computer than compute and generate the appropriate PWM signal to each of the motors to maintain balanced and controlled flight.

Even though RC technology is relatively reliable, there are cases where duty to electronics failure the wireless link was disconnected. This happened during an outdoor flight test where the QuadX was executing a leveled slow ascending maneuver. The link between the controller and transmitter was disconnected and the aircraft maintain the previous command and kept ascending. Eventually the propeller struck a tree branch and the ESC shut off the motor as a safety protocol. This causes the QuadX to free fall onto a brick road around 3 stories high. The end results are broken propellers, bended motor shafts and several fractured carbon fiber struts.

It was fortunate that no one was injured and that a tree caught the multirotor since it would have keep ascending until the battery run out of energy. To ensure that this would never happen again, an originally planned second communication link was soon implemented.

2.4 GHz Radio Modules

The XBee radio module, shown in Figure 5-20, act as a second communication link between the aircraft and a ground station laptop. Implementation of the radio was relatively simple, it interface with the flight computer via 3.3V TLL UART. The interface PCB has a slot to directly plug the XBee into place. On the ground station, a USB adapter was used to interface with the XBee radio module.

FIGURE 5-20: XBEE PRO 60 MW 2.4 GHZ RADIO MODULES

The XBee Pro is a 2.4 GHz radio that has an output power of 60 mW. Before deployment, the units are configured through serial commands with the personal area network (PAN) ID and the device ID so that the radios can achieve point to point or point to multipoint communication. The maximum data transfer rate is 250 kbps while a range of a mile. However, in reality, data would be corrupted at this transfer rate and range. Therefore a slower transfer rate is used for longer range. For the QuadX, a baud rate of 57600 is used to avoid data corruption. In addition, a number of checks are implemented to ensure data packet integrity.

The XBee modules primarily serve as the telemetry system to downlink flight data to the ground station so that the operators can monitor the flight conditions. The uplink was designed to tune the various controller gains in real-time and to issue commands for autonomous flying when

guidance system is implemented. Due to the minor incident that occurred, an control override mode was add so that the ground station operator can override the RC operator control at any time should the communication link failed. In normal override mode, the operator has control over the aircraft's roll, pitch, yaw, throttle and height. The emergency mode automatically reset and command the aircraft for leveled descend. The telemetry and command are implemented in the Ground Station GUI's Flight Command (FlightCOM) tab as shown in Figure 5-21.

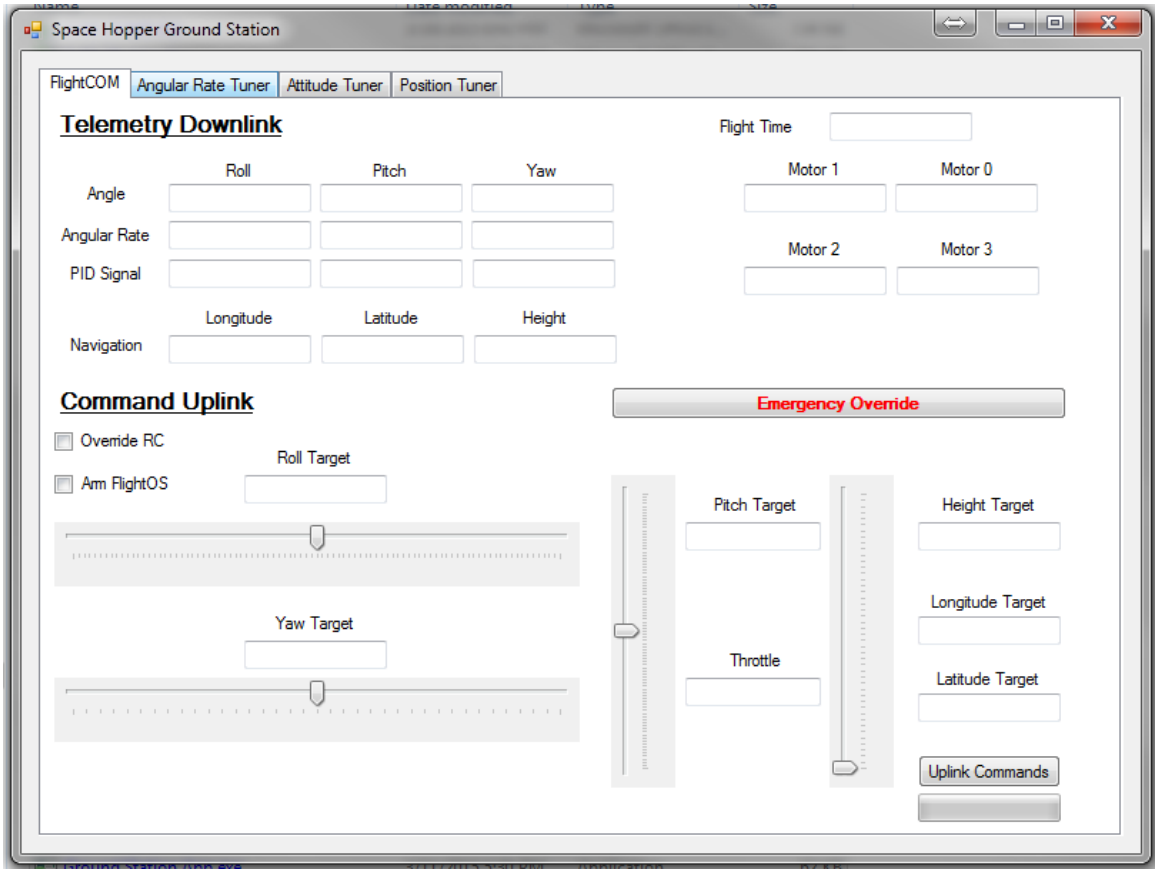
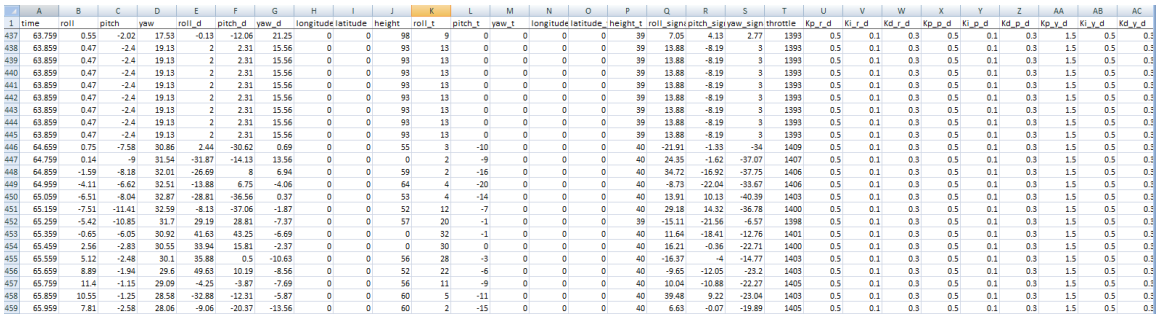


FIGURE 5-21: GROUND STATION FLIGHTCOM TAB

In order to reduce the flight computer's work load, the originally planned onboard Flight Logger is implemented as part of the ground station telemetry system. As soon as telemetry data are received, all the flight variables are dumped into a flight log. This include a large range of parameters such as controller gains, input commands, output PWM signals in additional to the

traditional time, attitude and position. A portion of the flight data is displayed in Figure 5-22 as imported into excel.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	time	roll	pitch	yaw	roll_d	pitch_d	yaw_d	longitude	latitude	height	roll_t	pitch_t	yaw_t	longitude	latitude_	height_t	roll_signi	pitch_sig	yaw_sign	throttle	Kp_r_d	Ki_r_d	Kd_r_d	Kp_p_d	Ki_p_d	Kd_p_d	Kp_y_d	Ki_y_d	Kd_y_d
437	63.759	0.55	-2.02	17.53	-0.13	-12.06	21.25	0	0	98	9	0	0	0	0	39	7.05	4.13	2.77	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
438	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
439	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
440	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
441	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
442	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
443	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
444	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
445	63.859	0.47	-2.4	19.13	2	2.31	15.56	0	0	93	13	0	0	0	0	39	13.88	-8.19	3	1393	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
446	64.659	0.75	-7.58	30.86	2.44	-30.62	0.69	0	0	55	3	-10	0	0	0	40	-21.91	-1.33	-34	1409	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
447	64.759	0.14	-9	31.54	-31.87	-14.13	13.56	0	0	0	2	-9	0	0	0	40	24.35	-1.62	-37.07	1407	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
448	64.859	-1.59	-8.18	32.01	-26.69	8	6.94	0	0	59	2	-16	0	0	0	40	34.72	-16.92	-37.75	1406	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
449	64.959	-4.11	-6.62	32.51	-13.88	6.75	-4.06	0	0	64	4	-20	0	0	0	40	-8.73	-22.04	-33.67	1406	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
450	65.059	-6.51	-8.04	32.87	-28.81	-36.56	0.37	0	0	53	4	-14	0	0	0	40	13.91	10.13	-40.39	1403	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
451	65.159	-7.51	-11.41	32.59	-8.13	-37.06	-1.87	0	0	52	12	-7	0	0	0	40	29.18	14.32	-36.78	1400	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
452	65.259	-5.42	-10.85	31.7	29.19	28.81	-7.37	0	0	57	20	-1	0	0	0	39	-15.11	-21.56	-6.57	1398	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
453	65.359	-0.65	-6.05	30.92	41.63	43.25	-6.69	0	0	0	32	-1	0	0	0	40	11.64	-18.41	-12.76	1401	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
454	65.459	2.56	-2.83	30.55	33.94	15.81	-2.37	0	0	0	30	0	0	0	0	40	16.21	-0.36	-22.71	1400	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
455	65.559	5.12	-2.48	30.1	35.88	0.5	-10.63	0	0	56	28	-3	0	0	0	40	-16.37	-4	-14.77	1403	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
456	65.659	8.89	-1.94	29.6	49.63	10.19	-8.56	0	0	52	22	-6	0	0	0	40	-9.65	-12.05	-23.2	1403	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
457	65.759	11.4	-1.15	29.09	-4.25	-3.87	-7.69	0	0	56	11	-9	0	0	0	40	10.04	-10.88	-22.27	1405	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
458	65.859	10.55	-1.25	28.58	-32.88	-12.31	-5.87	0	0	60	5	-11	0	0	0	40	39.48	9.22	-23.04	1403	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3
459	65.959	7.81	-2.58	28.06	-9.06	-20.37	-13.56	0	0	60	2	-15	0	0	0	40	6.63	-0.07	-19.89	1405	0.5	0.1	0.3	0.5	0.1	0.3	1.5	0.5	0.3

FIGURE 5-22: FLIGHT LOG IN EXCEL

For flight data visualization, a separate application, the Flight Log Player, was developed in vPython. The player's primary interface is an interactive 3D window that allows the user to zoom and rotate the viewing angle. At the same time, the virtual aircraft translate and rotate referencing data recorded by the Flight Logger. The secondary interface is a 2D window that contains a dynamic graph that plots the aircraft's roll, pitch and yaw synchronously with the animation in the 3D window. A screenshot of the Flight Log Player can be found in Figure 5-23.

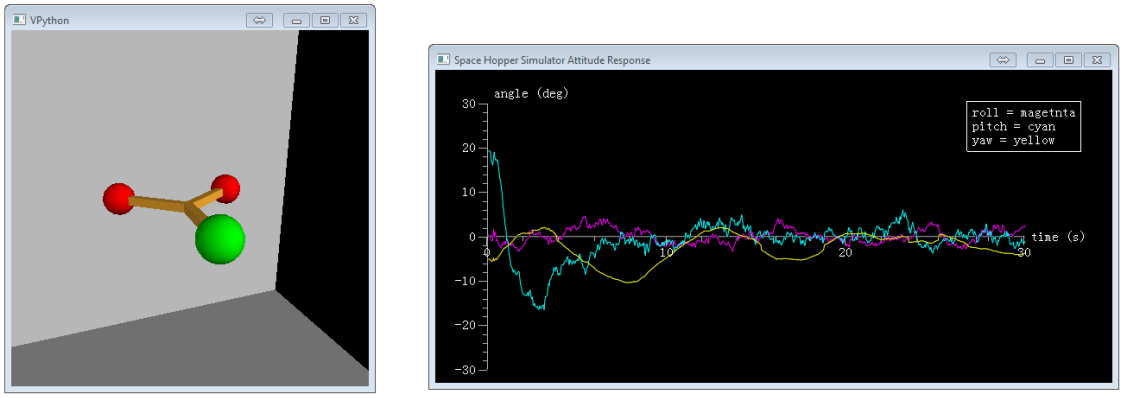


FIGURE 5-23: FLIGHT LOG PLAYER

The Ground Station's secondary function is to support live PID tuning. There are 3 similar tuner tabs for the attitude rate, attitude and position control. Each tab has 3 set of tuners dedicated to the K_p, K_i and K_d parameters of the roll, pitch and yaw axis. In terms of position, this would be longitude, latitude and altitude. The GUI has the ability to save and load previous configured parameter so that it can easily be brought up for the next tuning session.

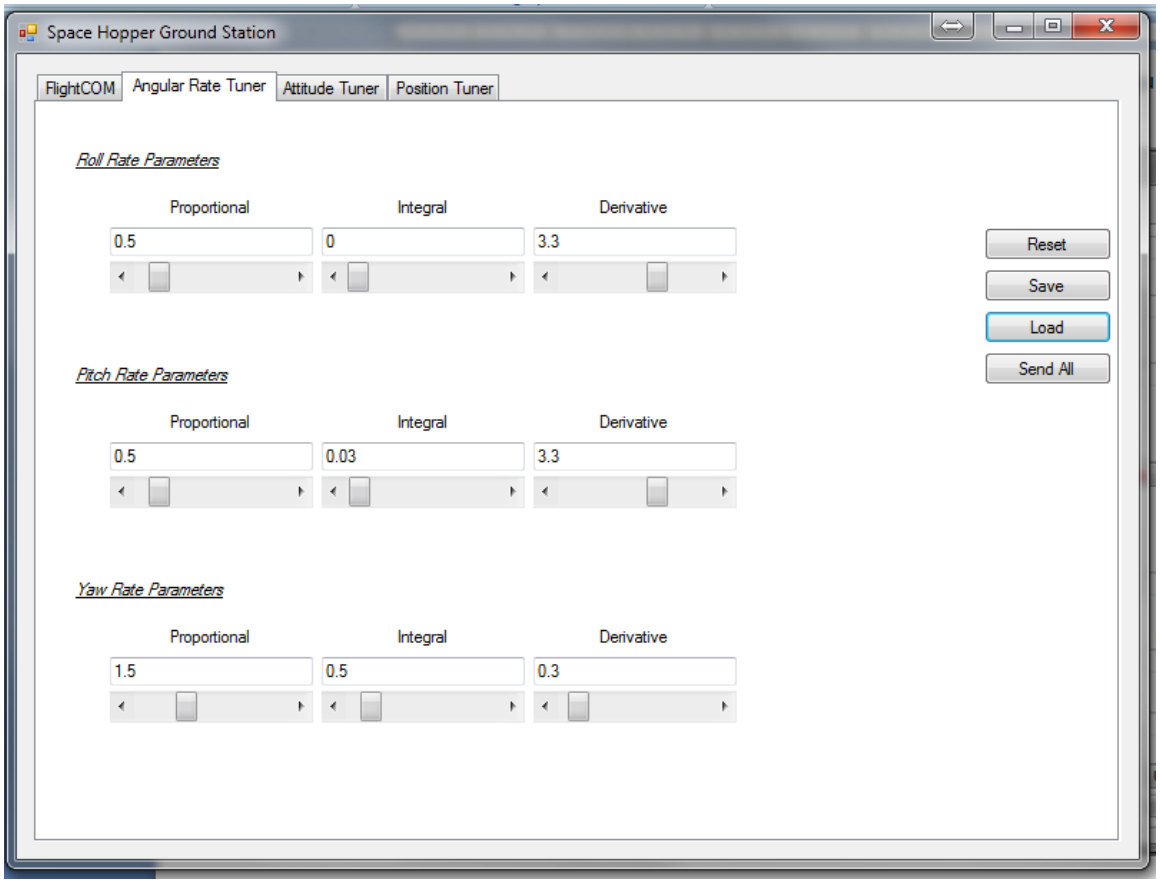


FIGURE 5-24: GROUND STATION PID TUNER TAB

Currently, a new ground station software is being developed by an undergraduate student on the team, as shown in Figure 5-25. As oppose to the previous ground station software which was aimed for aircraft development, the new software is aimed for deployment use. In addition to the basic tuners, the main display is consisted of various attitude indicators and also a map for to indicate aircraft position. The map can also be used to assign way points for the aircraft to travel to.

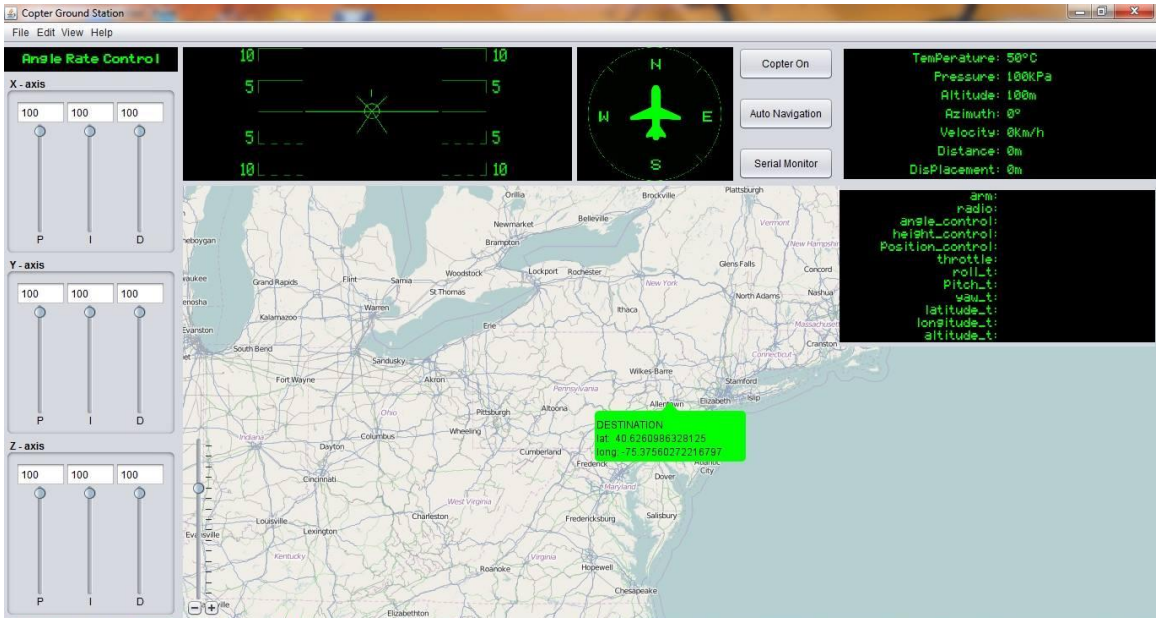


FIGURE 5-25: AUTONOMOUS FLIGHT GROUND STATION SOFTWARE

In complement with the new ground station software, an Android mobile controller is also being developed so that a phone can be used to control the aircraft. The app function by sensing the tilting motion of the phone, and from that commands are either being uplink directly to the aircraft via a Bluetooth radio or to the ground station, then relay to the aircraft via XBee. The software contains attitude indicator display to represent aircraft telemetry data. In addition, it also has a map for selecting way points similar to the ground station. A figure of the App can be found in Figure 5-26.

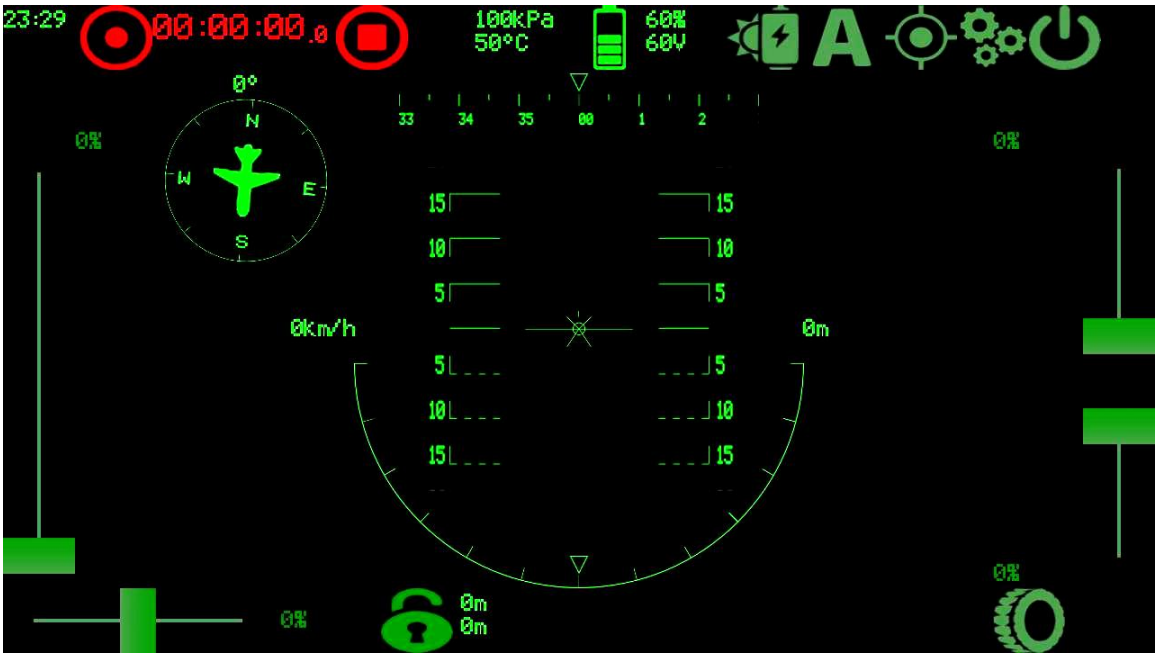


FIGURE 5-26: ANDROID MOBILE DEVICE CONTROLLER

Chapter 6: SOFTWARE

FLIGHT SYSTEM MODULES

The Space Hopper Simulator flight operating system is consisted of a number of software modules as shown in Figure 6-1. The modules contain the source code to interface with various avionics components as well as implementation of digital filters and GNC system. To simplify software development, most modules are implemented as object classes and all have access to a global data object. The data object is updated by the data acquisition modules and is used by the GNC modules to compute the appropriate PWM output to achieve controlled flight.

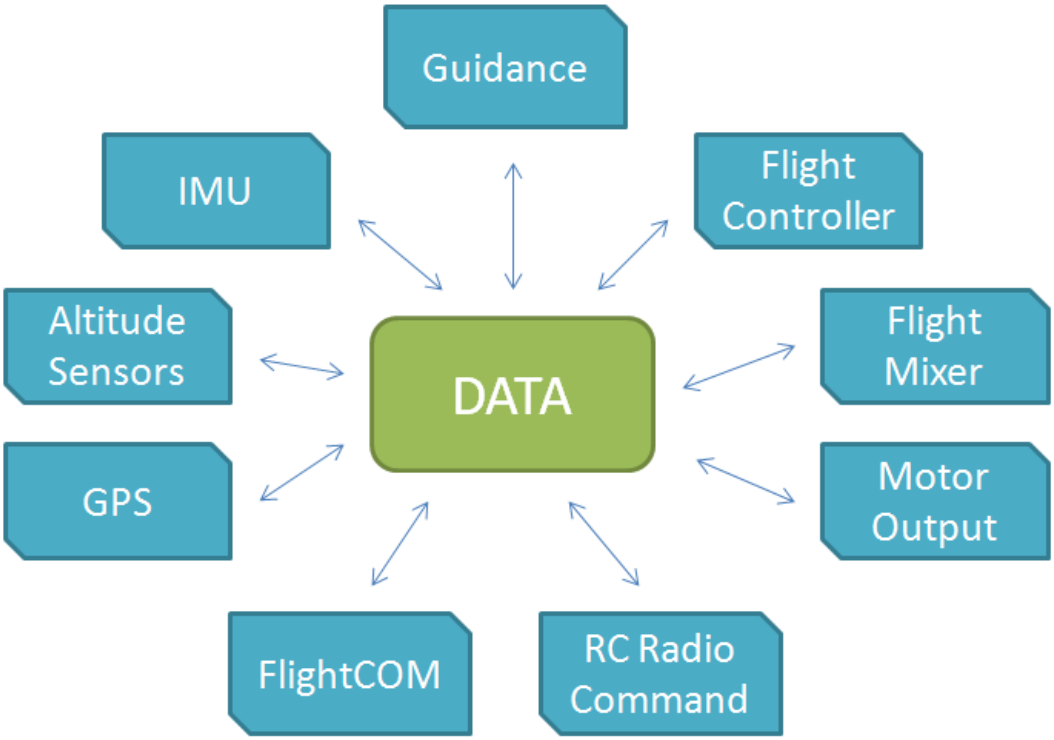


FIGURE 6-1: FLIGHT SOFTWARE MODULES

FLIGHT COMMUNICATION MODULES

The flight communication system is made up of the FlightCOM and the RC Radio Command module. The two modules interface the XBee and the RC receiver respectively.

The FlightCOM as previously discussed interface with the XBee through a serial UART connection. The primary functions of the FlightCOM are to respond to ground station uplink commands and downlink telemetry data. Essentially, when data packets are detected in the XBee serial buffer, the flight computer will read and store the data into a string variable. The string then get pass to a parser to decode whether the ground station try to request a telemetry downlink, tune flight controller parameters or update the target angle, height, position, etc. If the ground station request telemetry, the flight computer respond by downlink the flight data stored in the data object. On the other hand, if the ground station is trying to tune the flight controller or change the target, the respective value in the data object will be updated.

The RC Radio Command module is design to measure the PWM signal output by the RC receiver. Typically, a specialized kind of PWM signal is used by RC electronics such as the receiver, ESC and servos. A sample of the RC PWM signal is illustrated in Figure 6-2.

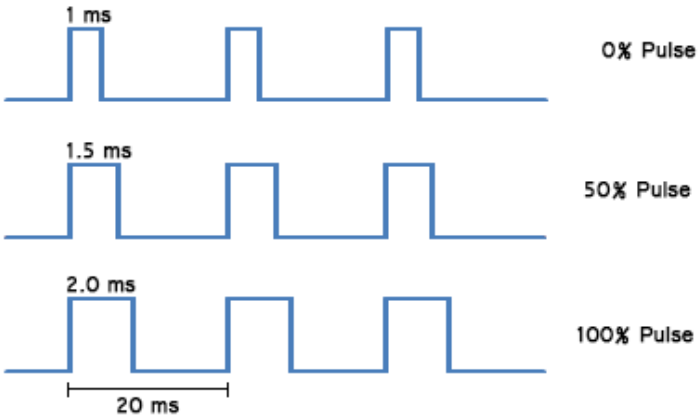


FIGURE 6-2: RC PULSE WIDTH MODULATION SIGNAL

RC PWM signal differ than regular PWM signal in that it don't use duty cycle exactly the way mentioned in Chapter 4. The minimum pulse generally is a signal that has a high pulse width of 1 ms. On the other hand, the medium pulse width is 1.5 ms and the maximum pulse width is 2.0 ms. In an RC servo, the pulse width would represent an angle of $0°$, $90°$ and $180°$. In an ESC for a brushless motor, this would represent 0%, 50% and 100% throttle. The period of the signal is around 20 ms, though exact timing doesn't matter.

In order to measure the pulse width of the PWM signal, each of the 6 channels from the receiver is connected to a digital pin. Arduino has a built-in function to measure the pulse width in which the PWM signal is high. This utilize a timer that begin timing when the signal goes from low to high and stop when the signal goes from high to low. This is however, highly inefficient, since the function block the rest of the flight system from running until the measurement is over. Each measurement takes roughly 10 ms and for all 6 channels, this causes a delay of 60 ms.

It is unacceptable that the radio module run at 16.7 Hz and blocks the rest of the flight system from running. This will cause the rest of the system to run at an even lower frequency. To overcome this, the module is implemented using a series of hardware interrupts, which the Arduino DUE supports in all of its pins. Essentially, when an interrupt is triggered by having the signal changing state, the microcontroller pause its current process loop and jump directly to a section of codes define at startup. Once that code is ran, the microcontroller resume the normal process loop where it was pause. By using interrupt, a timer could be started when the signal goes from low to high, then the microcontroller continue to run the flight loop. When the signal goes from high to low, the microcontroller stop the timer, save the pulse time and resume the flight loop. This implementation scheme allows the module to run much faster with virtually no delay added to the flight control loop.

DATA ACQUISITION MODULES

The data acquisition modules include the GPS, Altitude Sensors and the IMU. The combination of the flight data acquired from the sensor along with various filters and sensor fusion algorithms make up the navigation system.

The GPS interfaced with the flight computer through an UART serial connection and are encoded in the NEMA ASCII sentences. The position is reported as geo-coordinates in degrees and decimated minutes. To make this useful in terms of control, the geo-coordinates are localized by the module so that the positions are reported as meters away from a reference point. The GPS receiver has an update rate of around 5 Hz. In addition, the estimated position bounce quite a bit. Therefore a digital filter is implemented to get a better estimation.

The Altitude Sensors module is made up of the MB1240 ultrasonic range finder and the BMP085 barometric altimeter. The two interface with the Arduino via analog and I2C signal. Since the ultrasonic sensor is better at measuring precise distance at a low height while the altimeter is better estimating high altitude, a digital filter should be used to merge the two.

The UM7 IMU from CHRobotic interface with the microcontroller via UART serial at a baud rate of 115200 to optimize data transfer speed. The communication data packet to and from the IMU are encoded in binary as specify in the data sheet. This involves assembling a binary string to request data. The binary string is consisted of 3 ASCII letter to indicate beginning of a transmission. It is than followed by a byte specifying packet type, a byte for register address, several bytes for data, and finally ended with 2 bytes for checksum to assure serial packet integrity. The data packets are encoded in binary 2's complement and then scaled by some constants. To retrieve actual attitude data, the decoding process must be applied.

GUIDANCE

The Guidance module take the height and position data retrieved from the data acquisition modules to compute a target attitude and throttle for the flight controller. Essentially, this is a position and height control module. The input to the guidance system is either set by the ground station as way points or predefined as the parabolic hopping trajectory.

FLIGHT CONTROLLER

The Flight Controller module implements the control laws researched and simulated in the project. As shown in Figure 6-3, this module could easily be swapped between the PID Controller and the Fuzzy Logic Controller. This will allow the remainder of the flight operating system to remain relatively similar between the Quad+ and QuadX multirotor.

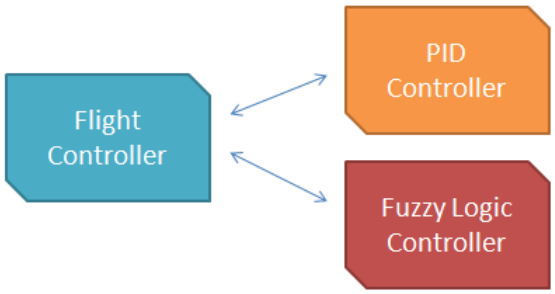


FIGURE 6-3: FLIGHT CONTROLLER MODULE (PID AND FUZZY LOGIC CONTROLLER)

FLIGHT MIXER

The Flight Mixer module takes the roll, pitch and yaw signal generated from the Flight Controller and calculate the PWM signal combination for the multirotor's motors to achieve the desired torque output. To accommodate for different flight configurations, the Flight Mixer predefined 3 mixing set with the flight dynamics discussed in Chapter 2.

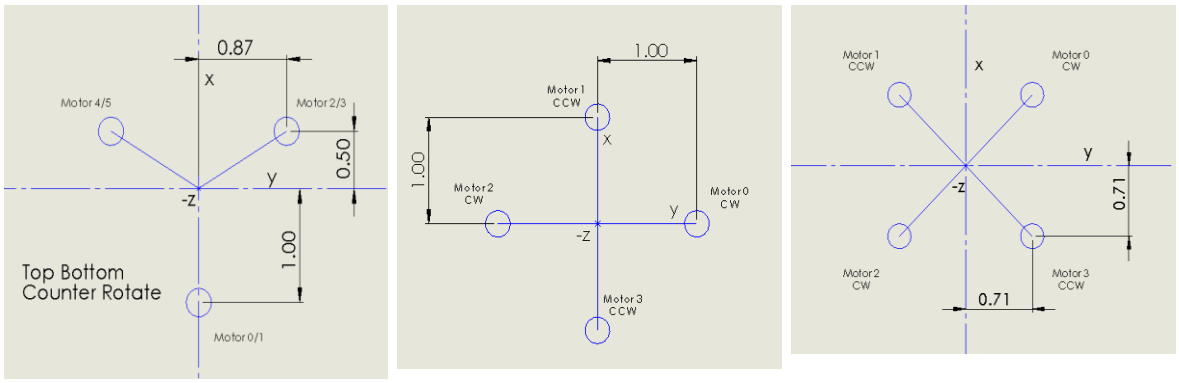


FIGURE 6-4: MOMENT ARM FOR Y6, QUAD+ AND QUADX FLIGHT CONFIGURATION

The Y6, Quad+ and QuadX flight configuration is drawn in a CAD software from a top down view. The moment arm of each flight configuration is measured to determine the motor's contribution to the rolling and pitching torque. In addition, the motor spinning direction is used to determine the yawing torque contribution. The resulting motor mixing table is illustrated in Figure 6-5.

Y6 Configuration Mixer Table

	Roll	Pitch	Yaw
Motor 0	0%	-100%	+
Motor 1	0%	-100%	-
Motor 2	-87%	50%	+
Motor 3	-87%	50%	-
Motor 4	87%	50%	+
Motor 5	87%	50%	-

Quad+ Configuration Mixer Table

	Roll	Pitch	Yaw
Motor 0	-100%	0%	+
Motor 1	0%	100%	-
Motor 2	100%	0%	+
Motor 3	0%	-100%	-

QuadX Configuration Mixer Table

	Roll	Pitch	Yaw
Motor 0	-100%	100%	+
Motor 1	100%	100%	-
Motor 2	100%	-100%	+
Motor 3	-100%	-100%	-

FIGURE 6-5: MOTOR MIXING TABLE FOR Y6, QUAD+ AND QUADX FLIGHT CONFIGURATION

With the motor mixing values in Figure 6-5 and the signal from the Flight Controller, the following equation is used to calculate output PWM signal pulse time for each of the motors. In the equation, the variable x, y and z represent the roll, pitch and yaw contribution for each motor respectively. The pulse time is usually expressed in μs as opposed to ms so that there is enough resolution to generate small differences in thrusts.

$$pwm = throttle + signal_{roll} * x + signal_{pitch} * y + signal_{yaw} * z$$

MOTOR OUTPUT

The Motor Output module simplify take the PWM pulse time calculated from the Flight Mixer and generate a PWM signal for the ESC to the motors. In addition, the Motor Output module is programmed to arm the ESCs in the beginning of flight. It is necessary to arm an ESC before it will spin a motor. This is a safety protocol implemented to prevent injuries. To arm a motor the ESC will first need to be driven with a PWM signal of $1000\ \mu s$ or less for several seconds. The ESC will then beep several times, indicating that it is armed and that the motor will respond to all future PWM signal commands.

FLIGHT OPERATING SYSTEM SOFTWARE ARCHITECTURE

The flight operating system (FlightOS) is constructed with the software modules discussed in the previous section. Since each sub-system is verified to functioning properly in the module level, the development for the FlightOS simply involves linking the modules together. There are several implementation schemes for the FlightOS throughout the development of the Space Hopper Simulator. The first implementation scheme is that of a linear software architecture.

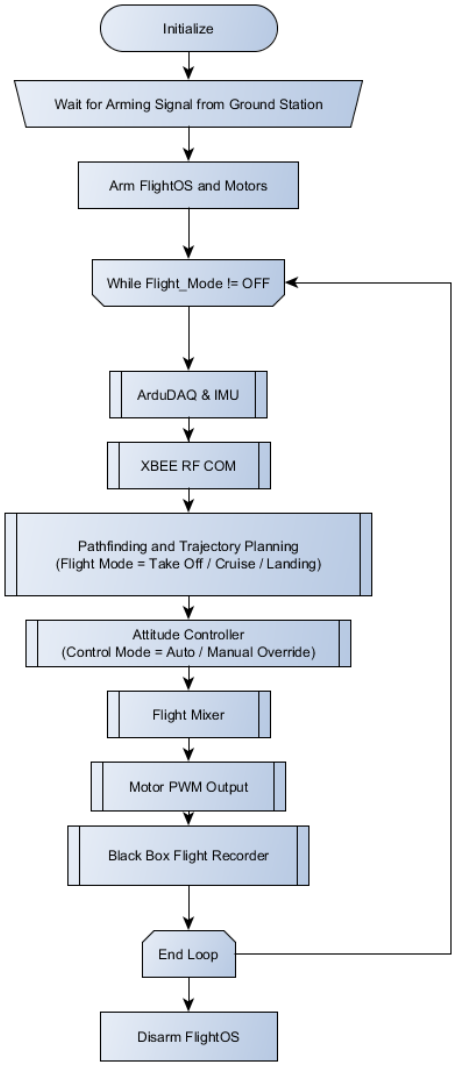


FIGURE 6-6: FLIGHTOS LINEAR SOFTWARE ARCHITECTURE

The linear software architecture is modeled after the intuitive logic that make a system fly. The software first initializes all subsystem and wait for an arming signal from the ground station. When the FlightOS is armed, the system run in a control loop that only end when the FlightOS is disarmed. In the control loop, the flight computer request flight data and downlink that to the ground station. Using the flight data acquired from the sensors, the guidance module calculates the target attitude so that a desired trajectory can be achieved. To meet the target attitude, the attitude controllers compute the appropriate roll, pitch and yaw signal. The flight mixer than take the signal and distribute PWM commands to every actuator. After that is completed, the Flight Logger records the flight data for later analysis. The linear software system flowchart is illustrated in Figure 6-6.

Generally, linear software system has the disadvantage that a function block must complete running before the next is executed. Through benchmark tests, the data acquisition modules appear to be the bottleneck of the system. This is most likely due to the extensive overhead processes that the processor must go through in order to retrieve data from external devices. The ArduDAQ in particular is problematic since to retrieve radio pulse time alone causes a delay of 0.163 sec. This block the IMU and the attitude controller from running fast enough to establish stable flight. Rig testing has shown that a control loop frequency lower or equal to 10 Hz will cause the multirotor to become extremely unstable and uncontrollable.

In an attempt to overcome this process blocking problem, the multithreading software architecture is implemented in the BealgeBone Black. The software flowchart is shown in Figure 6-7. The idea behind the multithreading scheme is that several processes will run in parallel so that none of them will be blocked from each other. The three distinct loops are the guidance / navigation loop, the control loop and the wireless communication loop.

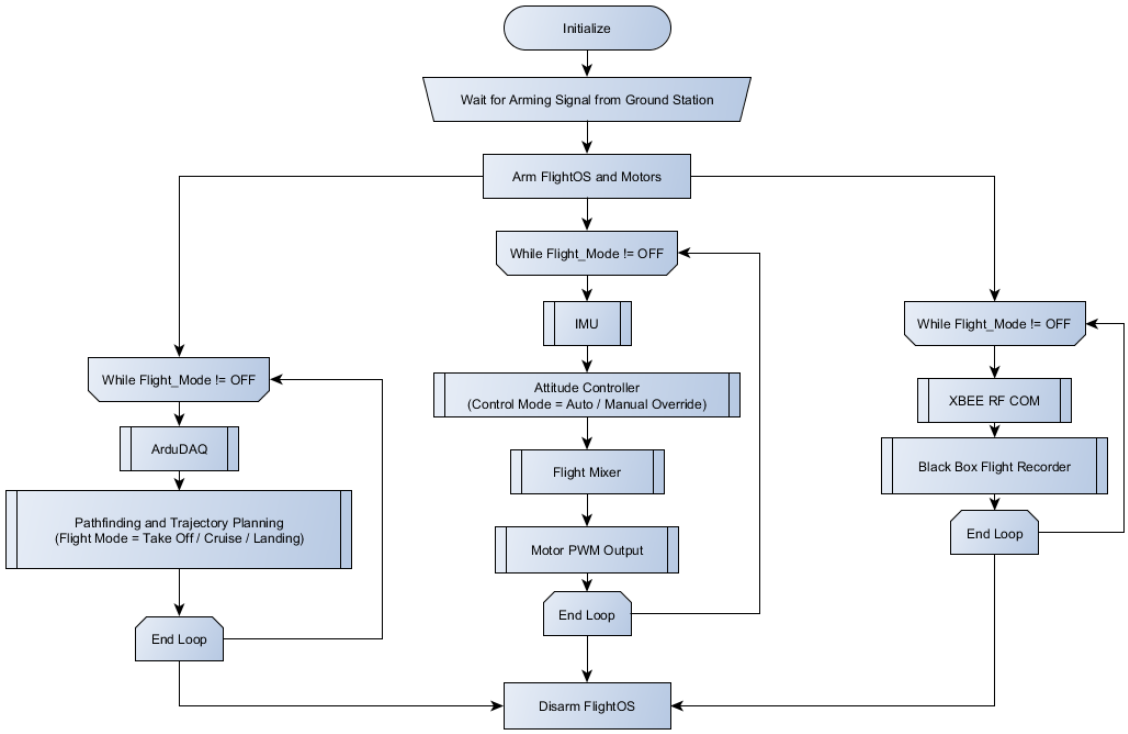


FIGURE 6-7: FLIGHTOS PARALLEL SOFTWARE ARCHITECTURE

Ideally, the parallel processing scheme would greatly improve the speed of the FlightOS. While one loop is stuck waiting for incoming data, the others will keep running. However, this proves not to be the case as the BeagleBone Black only has a single-cored CPU. Therefore, the background Linux services divide processor so that it is shared by all 3 loops. Each loop are only given a portion of the processing time. In addition, the background Linux operating system is also sharing the CPU to run the necessary services to keep the computer running. Therefore, even though the FlightOS did run a bit faster, the performance is still not acceptable.

It is apparent that it is not effective to construct an operating system on top of another operating system. Even though the Linux operating system provides lots of features and support many pre-written libraries that greatly reduce effort in software development, the performance is just not acceptable. Therefore, it was decided that the flight computer should focus on having

minimal overhead so that the processor's full processing capability can be focused on the FlightOS. The Arduino DUE became one of the best candidates.

The Arduino DUE, unlike the previous flight computer, is a microcontroller. Therefore, it does not include a base operating system and must be programmed to perform a specific task, to run the FlightOS. The FlightOS must be developed from scratch in C++ and compile into machine code that the microcontroller can execute. With the lack of a base operating system and that the microcontroller only contains a single-cored CPU, it is impossible to use the parallel software architecture above. The software can only be executed linearly.

Fortunately, as discussed earlier, the Arduino DUE support hardware interrupts in all of its pins. This allow the FlightOS be developed in such a way that parallel processing can be faked by controlling exactly when a function block get processed. The main processing loop can be found in Figure 6-8 while the interrupt function blocks can be found in Figure 6-9.

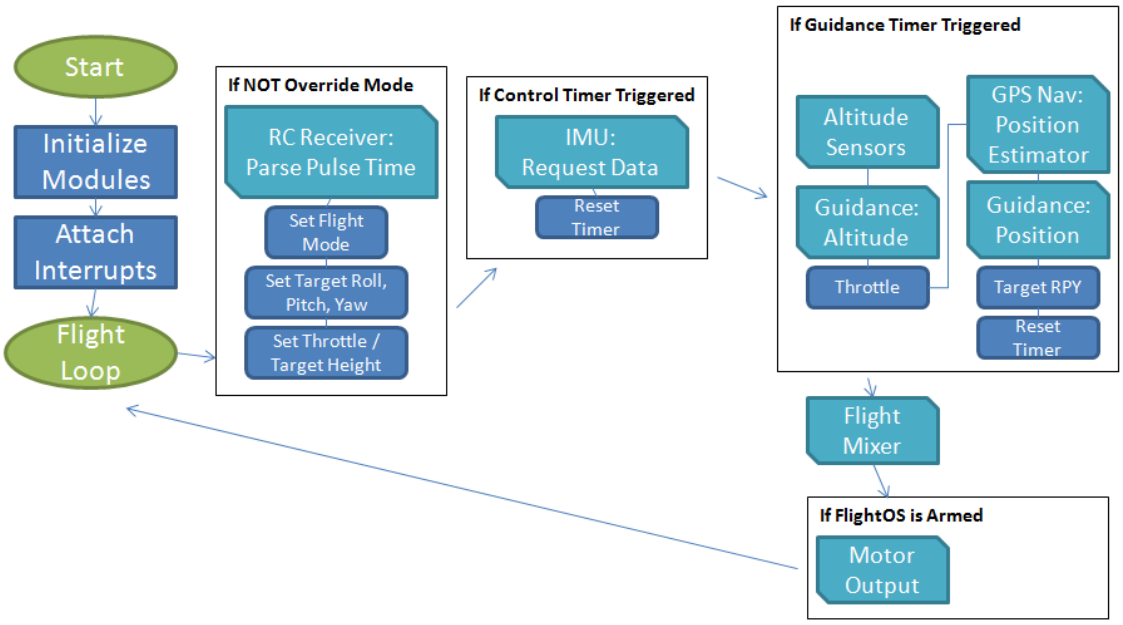


FIGURE 6-8: ARDUINO FLIGHTOS LINEAR PROCESSING LOOP

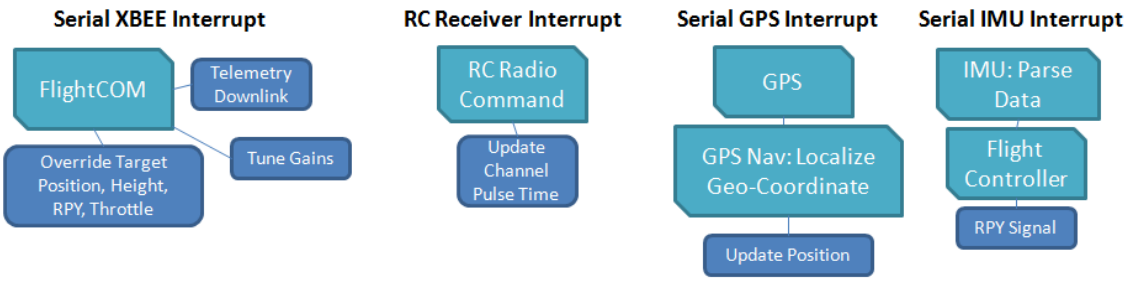


FIGURE 6-9: ARDUINO FLIGHTOS INTERRUPT FUNCTION BLOCKS

The logic of the Arduino based FlightOS is a bit confusing, but it would probably be easier to understand by looking first at the interrupt function blocks. Unlike the main process loop that run repeatedly after setup, the interrupt function blocks only get executed when a hardware interrupt is triggered. When it is triggered, the main process loop is pause until the interrupt function block complete executing. Note that when a second interrupt is triggered while the processor is running the codes for the first triggered interrupt, the codes for the second interrupt will only be run after the first is completed.

The first interrupt function block is that of the XBee communication module. When the flight computer detects activities in the XBee's serial port, the FlightCOM module is activated to retrieve data in the buffer and respond to the ground station appropriately.

The GPS and the IMU interrupt function block are triggered similarly. Though the GPS is set to send serial data at a rate of 5 Hz while the IMU required a request command be sent in order for it to send the most updated data through serial. When the GPS interrupt is trigger, the localization process is ran before the position is updated. On the other hand, when the IMU interrupt is triggered, the attitude controller is also ran to update the proper attitude signal to the flight mixer.

The RC Receiver interrupt function block is triggered when the digital state of the pin is changed as discussed in the previous section. This allows the pulse time be calculated. All data retrieved by the interrupt function blocks are shared with the main process loop through the global data object.

With the functionality of the interrupts in mind, the main process loop should be easier to understand. The FlightOS begin by running the set up functions to initialize all the modules and attach the interrupts to the hardware. Then the process loop is ran repeatedly.

The process loop start by checking whether it is in ground station override mode. If not, then it take the RC receiver pulse time retrieved from the interrupt to calculate target commands and flight modes.

To control the sample rate of specific subsystem, the Control and Guidance blocks are ran by their respective timers. Since the attitude controller is run when the IMU interrupt is triggered, the main process loop only has to send a data request command to the IMU. The aircraft trajectory is controlled by the combination of the altitude and position guidance modules. The altitude guidance module read data from the altitude sensors and determines the required throttle to achieve the desired trajectory. Similarly the position guidance module does the same for target attitude. However, since GPS data are often very noisy, the positions feeding into the guidance module must first run through the GPS Navigation position estimator module.

The flight mixer is ran next to calculate the PWM signal output from the throttle and attitude signal from the various modules. Next, the FlightOS generate the PWM signal to the motor only if the operating system is armed. This is a safety protocol to make sure that the multirotor doesn't accidently take off.

The process loop then repeat indefinitely until power to the microcontroller is lost. Benchmark data show that this hardware and software combination perform the best thus far.

SOFTWARE OPTIMIZATION

Software optimization is a key aspect of the FlightOS development process. The optimization process involved benchmarking and improving performance by implementing more efficient coding logic and configuration avionics such as increasing serial baud rate. The most radical optimization includes changing flight computers and development platforms. Some of the benchmarking and optimization data are reproduced here.

	MATLAB (String)	Python (String)	MATLAB (Binary)	Python (Binary)	Arduino (PulseIn)	Arduino (Interrupt)
Sample Time (s)	0.163	0.090	0.095	0.062	0.060	~0.002
Sample Rate (Hz)	6.1	11.1	10.5	16.1	16.7	500

TABLE 6-1: RC RECEIVER DATA ACQUISITION BENCHMARK PERFORMANCE

The RC Receiver is one of the main bottlenecks of the data acquisition system. For the Intel NUC and the BeagleBone Black, the ArduDAQ must be used to acquire signal and relay the data through serial UART at a max baud rate of 115200. The original system is set up capture the data repeatedly and when requested, the data is transferred to the flight computer via ASCII string. As seen in Table 6-1, this operation takes 0.163 sec in MATLAB and 0.090 in Python. To optimize, the system is modified to transfer data via binary. The sample time improved to 0.095 and 0.062 for MATLAB and Python respectively. Recall that the Pulse In function in Arduino take 0.060 sec, that mean that the communication overhead time is 0.035 for MATLAB and 0.002 for Python. When the flight computer is finally switched to the Arduino DUE, the ArduDAQ is no longer necessary and eliminated the overhead delay. The last optimization effort is to implement

hardware interrupt for all RC receiver channels. Due to the nature of interrupts, there is no effective way to directly measure the sample time. However, using sample time of the analog read operation, the sample time to read all 6 channels is calculated to be 0.002 sec. The implementation of hardware interrupt greatly improves performance for the Arduino DUE. However, the ArduDAQ is run by an Arduino Micro. That particular microcontroller only support 3 hardware interrupts, and hence unable to apply the optimization.

	Python (Desktop)	Java (Desktop)	C# (Desktop)	Python (BBB)	C++ (Arduino)
Sample Time (s)	0.0095	0.0053	0.0048	0.00625	0.0048
Sample Rate (Hz)	105	190	210	160	210

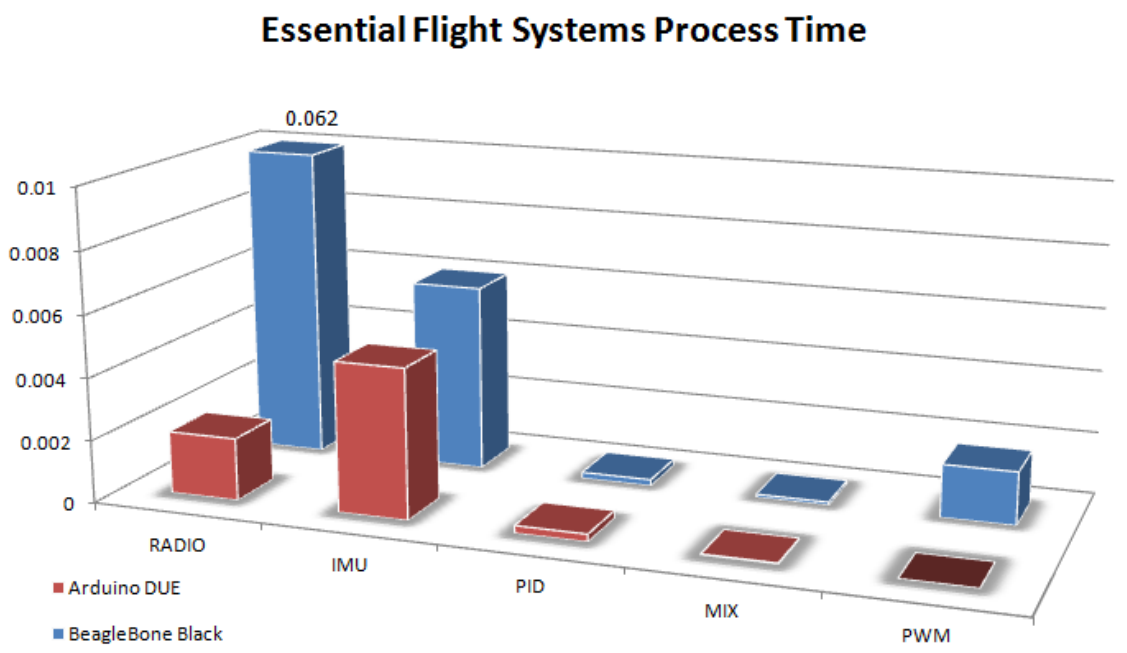
TABLE 6-2: UM7 DATA TRANSFER BENCHMARK PERFORMANCE

The IMU was probably the next bottleneck of the system. Since the IMU was already configured to transfer data in binary at the maximum baud rate, there's not much that can be done to improve the performance other than changing development platforms. Since MATLAB perform poorly in terms of data acquisition, it was eliminated entirely.

In the benchmark, the UM7 was first tested with a desktop computer with various programming languages. The desktop computer has a hex cored AMD Phenom II x6 1090T processor overclocked to 3.40 GHz with 8 GB of DDR3 RAM and run 64-bit Windows 7 on a 120 GB SSD. The benchmark result can be found in Table 6-2. In general, Python perform the worse with a sample rate of 105 Hz and C# performs the best at 210 Hz.

Surprisingly, even though the BeagleBone Black has much less respectable hardware specs when compared with the than the desktop, the same Python script used on the Windows achieve a sample rate of 160 Hz.

The Arduino DUE IMU module is developed in C++ and is able to achieve a sample rate of 210 Hz. Given that the microcontroller has the lowest hardware specs, it is amazing that it achieved the same performance as the C# module in the significant more powerful desktop computer.



	RADIO	IMU	PID	MIX	PWM
Arduino DUE	0.002	0.0048	0.000240	0.000041	0.000004
BeagleBone Black	0.062	0.0060	0.000200	0.000100	0.001700

FIGURE 6-10: BEAGLEBONE BLACK VS. ARDUINO DUE ESSENTIAL FLIGHT SYSTEMS PROCESS TIME

The software modules essential for basic controlled flight are the radio receiver, IMU, flight controller, flight mixer and the PWM motor output. The process time of these modules should be minimized so that the flight control loop can run as fast as possible. The modules' benchmarked processing time for both the BeagleBone Black and the Arduino DUE is illustrated in Figure 6-10. Clearly, the Arduino DUE out perform the BeagleBone Black, especially in the radio receiver module.

However, the Arduino DUE is not without its downfall. The lack of an operating system means that useful features like onboard data logging cannot be done directly. The current flight logging system utilizes the telemetry system. When a downlink request is sent to the QuadX, the flight computer responds by transmitting the flight data via the XBee radio. The flight data is received by the ground station and recorded.

Ideally, the baud rate should be set at 115200 and the process time for that is 0.012 sec. However, this causes the XBEE radio to lock up after several minutes. Therefore, to fix this, the baud rate is reduced to 57600 and the process time is nearly double to 0.023 sec.

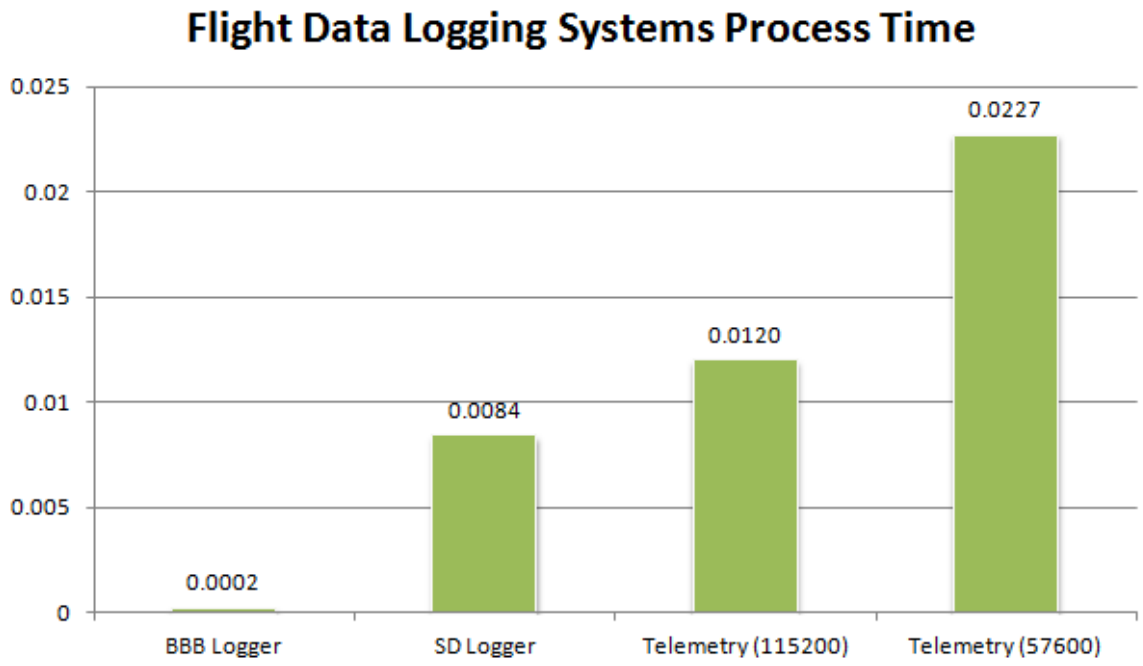


FIGURE 6-11: FLIGHT DATA LOGGING SYSTEMS COMPARISON

A dedicate flight logger module was developed for the Arduino DUE using an external SD card, but this was never implemented into the FlightOS. The theory was that telemetry is already a part of the flight system. The exclusion of onboard data logging would reduce processing load. The benchmark processing time for the SD card flight logger is 0.008 sec, a great improvement

from the current telemetry system. However, keep in mind that the BeagleBone Black's on board data logging system is only 0.0002 sec, the SD logger is still significantly slower.

Since telemetry is not an essential part of flight, a possible way to optimize this is to send a downlink request once every several seconds so that flight control in not impacted as much. However, this is not quite acceptable for analyzing flight performance since data collected at such slow rate might suffer aliasing issues. This can be improved by also using the SD logger. Though the SD logger only be ran once or twice a second to keep interference with flight control to a minimum. The balance between flight data sample rate and flight performance is a topic of trade study.

Chapter 7: CONTROL SYSTEMS

SINGLE AXIS ATTITUDE PID CONTROLLER

The proportional-integral-derivative (PID) controller is one of the most widely used controllers in engineering. It is powerful, easy to implement and relatively intuitive to understand. Therefore, it is used in many systems such as aircraft, spacecraft, robot and automotive. The tuning process, however, is a bit tricky to achieve optimal performance.

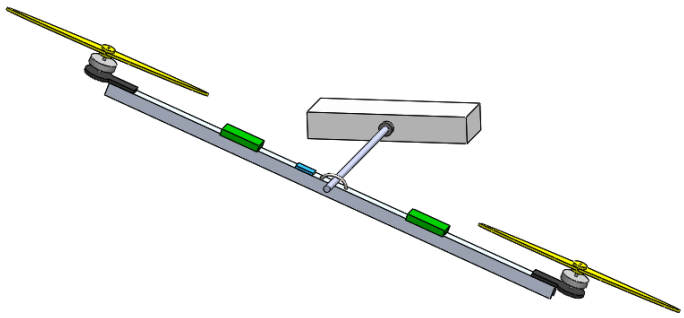


FIGURE 7-1: SINGLE AXIS MOTOR BALANCER

To start, let's consider a single axis motors balancer as shown in Figure 7-1. This is essentially a quadcopter but restricted to a single degree of freedom, allowing the beam to only rotate in one axis. The balancer is consisted of 2 motor, producing thrust in each end of the beam. There's an IMU to report the beam attitude with respective to ground level. The free body diagram for the single axis motor balancer is illustrated in Figure 7-2. The coordinate frame is set such that clockwise motion is positive.

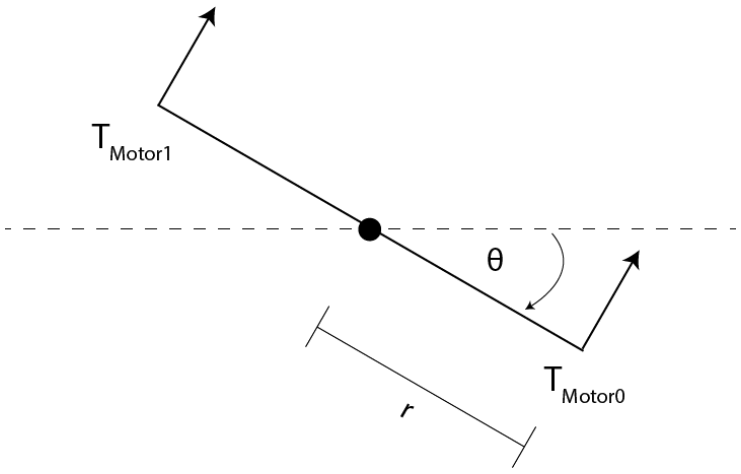


FIGURE 7-2: SINGLE AXIS MOTOR BALANCER FREE BODY DIAGRAM

Let's say that the tilt angle of the beam is measured to be θ and that the referenced input command, or target angle, is r. The error, e, is calculated by subtracting the processor variable y, or in this case, the measured angle θ, from the target.

$$e = r - \theta$$

This error is multiplied by a gain, K_p, to scale the signal proportionally. The result of this operation is the proportional control output signal, u_P.

$$\boxed{u_P = K_p e}$$

This control algorithm by itself is called to proportional controller, or a P Controller. The output signal is feed into the flight mixer to calculate the PWM output signal for the motors. Since Motor 0 contribute torque in the negative direction, the control signal is multiplied by negative one. On the other hand, Motor 1 contribute torque in the positive direction, the control signal remain to be positive. Then the throttle PWM pulse time is added.

$$PWM_{motor0} = -u_p + throttle$$

$$PWM_{motor1} = u_p + throttle$$

If $r = 0°$ and $\theta = 10°$, then the error $e = -10°$. If K_p is tuned to be 1.0, then proportional output signal is -10. Assuming that the throttle pulse time is $1300\ \mu s$, then the PWM signal to Motor 0 and Motor 1 has a pulse time of $1310\ \mu s$ and $1290\ \mu s$.

From the system identification test for the motors, the minimum PWM pulse time is 1150 and maximum pulse time is 1700. This correlate to a thrust of $0\ N$ and around $4.8\ N$. Using linear interpolation, the thrust output by Motor 0 and Motor 1 is estimated to be $1.4\ N$ and $1.2\ N$.

Recall the dynamics equations derived in Chapter 2:

$$\tau_\theta = r\ (T_{motor1} - T_{motor0})$$

$$\ddot{\theta} = \frac{\tau_\theta}{I_{yy}} + \frac{\cancel{I_{zz} - I_{xx}}}{\cancel{I_{yy}}}\dot{\phi}\dot{\psi}$$

Assuming that the radius is 9 *in* or 0.23 m, and that the principal axis of inertia is $0.0021\ kg \cdot m^2$, the angular acceleration is calculated to be -$0.046\ rad/s^2$ or $-2.64°/s^2$. This negative angular acceleration will bring the beam back toward the level position. As the beam is rotating closer to $0°$, the magnitude of the error decreases, hence the magnitude of the control output signal decrease. This, in turn, renders a drop in the motor thrust output difference and causes a decrease in torque magnitude. When the measured angle is negative, however, the control signal and the torque output will be positive to push the beam back to level. A simulation of the P controller with a tuning of $K_p = 1.0$ and $K_p = 0.1$ are illustrated in Figure 7-3 and Figure 7-4.

As seen in the simulations, the tuning of $K_p = 1.0$, even though arrive to the target angle a lot faster, it causes the system to oscillate out of control. Even though the torque is neutral when the measured angle approaches 0, the momentum of the beam causes the system to overshoot.

And the tilt of the beam became too negative. The controller over respond and create torque in the positive direction. This process repeat and the overshoot become worse and worse.

A lower proportional gain yield a lower torque responds as shown in Figure 7-4. Hence, the system remains somewhat under control. However, it is obvious that the system still oscillate by a great amount, a dampening force is necessary to have the system converge to a certain angle.

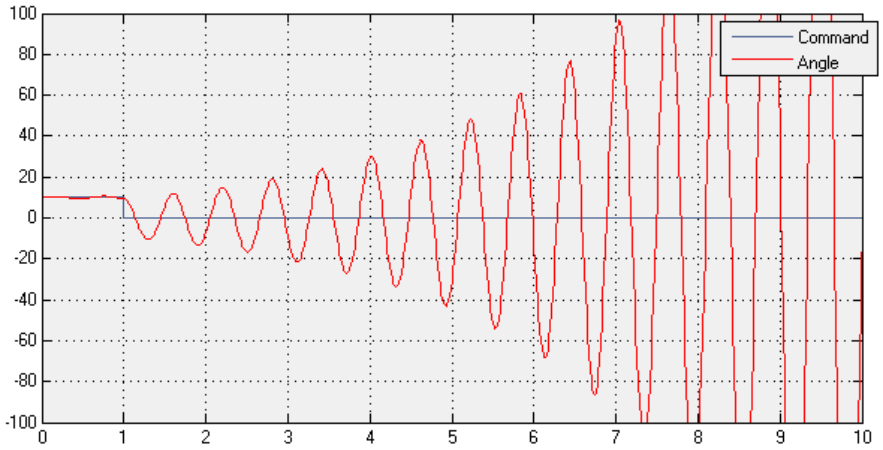


FIGURE 7-3: PROPROTION CONTROLLER ($K_p = 1.0$)

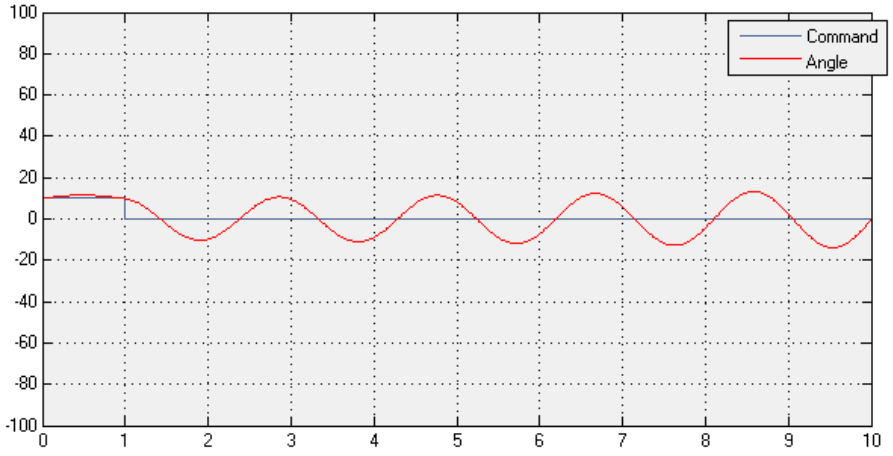


FIGURE 7-4: PROPROTION CONTROLLER ($K_p = 0.1$)

The dampening effect is implemented by adding a derivative action along with the proportional controller. This combined controller is called the proportional-derivative controller, or PD controller. The derivative action is similar to that of the proportional control. Instead of scaling the error by a gain, the derivative action multiple the error rates by the derivative gain K_d.

$$\boxed{u_{PD} = K_p e + K_d \frac{de}{dt}}$$

The derivative action has the characteristic that it will decrease the magnitude of the output signal if the error rate is high. In another word, if the balancer is approaching the target angle too fast, the derivative action will act to reduce the torque.

Let's take the previous example and expand it to the derivative action. Let say that the second reading measured by the IMU is 9° and the measurement time is 0.2 seconds after the initial measurement of $10°$. The error rate would be:

$$\frac{de}{dt} = \frac{e_{new} - e_{old}}{dt} = \frac{-9° - -10°}{0.2} = 5°/s$$

Assume that in this case, $K_p = 1.0$ and $K_d = 1.0$, then:

$$u_{PD} = K_p e + K_d \frac{de}{dt} = (1.0 * -9) + (1.0 * 5) = -4$$

As seen in this example, the proportional action alone would produce a signal of -9. This would yield a rather large torque in the positive direction like before and cause the system to overshoot. The derivative action, however, see that the error rate is high, the system would soon approach the target and reduced the output signal to -4. This cut the signal and torque output by more than half. The derivative action will help dampen the system so that the beam doesn't overshoot as much.

In practice, the measurement rate would be a lot faster than 0.2 sec. Therefore the previous calculations are only to demonstrate the concept. A more realistic simulation of the PD controller with $K_p = 1.0$ and $K_d = 2.0$ is shown in Figure 7-5. Note that in the following simulations, dt was combined into K to simplify implementation. This will be discussed shortly.

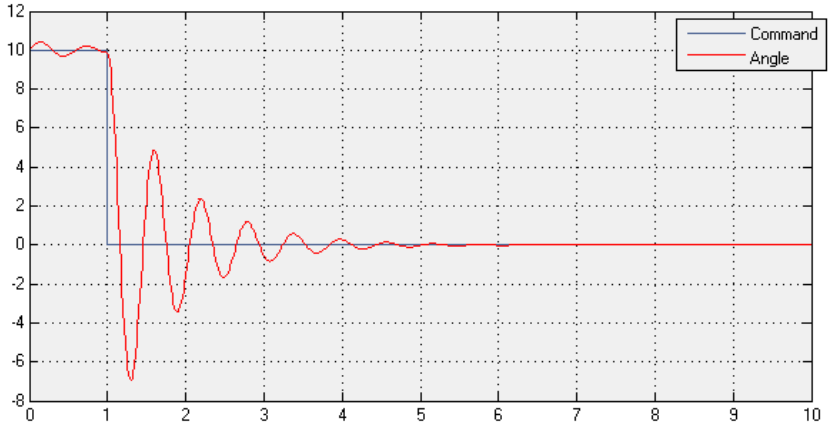


FIGURE 7-5: PD CONTROLLER ($K_p = 1, K_d = 2$)

It can be seen that the system perform a lot better than the P controller alone. Even though not ideal, the beam do settle to the referenced 0° after around 4.5 sec. The system is damped, but under-damped. Hence the first 4.5 sec of the maneuver is still oscillatory. To improve this, the derivative gain can be increased.

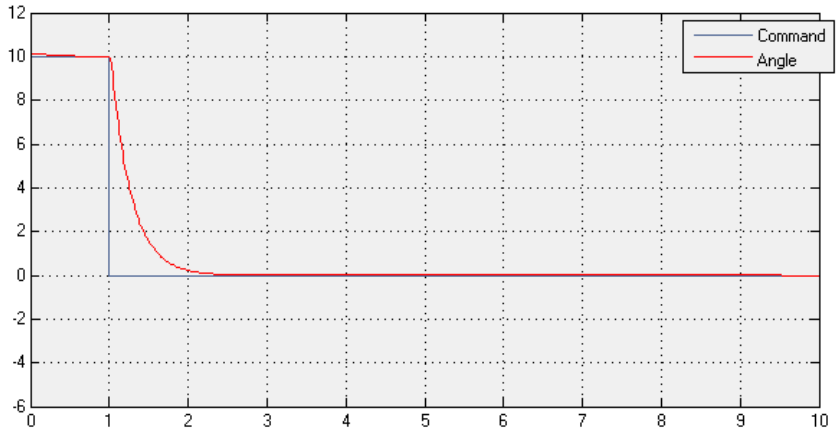


FIGURE 7-6: PD CONTROLLER ($K_p = 1, K_d = 20$)

Figure 7-6 shows the simulation result with $K_d = 20.0$. It can be seen that the oscillation is completely eliminated. However, the system react a lot slower now. Whereas it only take the damped system around 0.2 sec to reach level, the over-damped system take around 1.3 seconds to converge to level. This is an indication that K_d is tuned too high. By decreasing K_d to around 5.0, a critically damped performance can be achieved as shown in Figure 7-7. The system now only takes around 0.5 sec to converge to level.

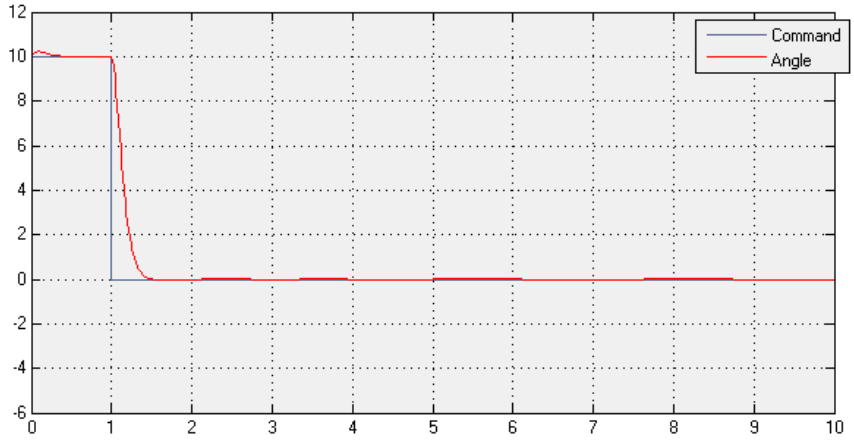


FIGURE 7-7: PD CONTROLLER ($K_p = 1$, $K_d = 11$)

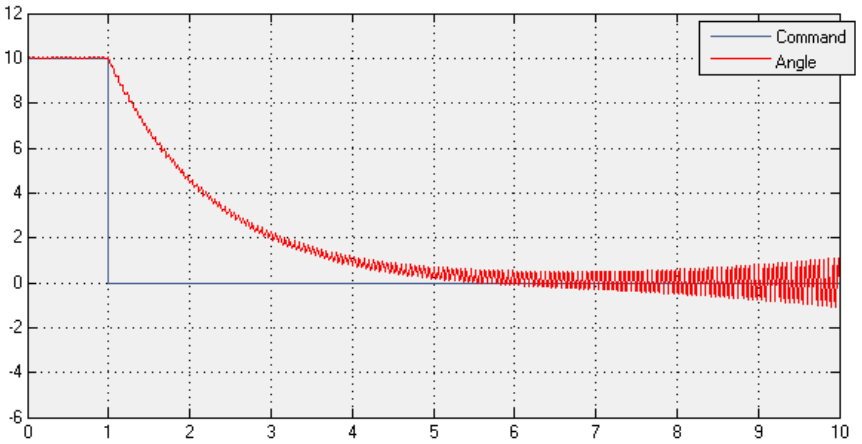


FIGURE 7-8: PD CONTROLLER ($K_p = 1, K_d = 90$)

It should be noted that even though the derivative action provide damping effect to the control system, an overly high derivative gain will add oscillation to the system in additional to the significantly added lag in system response. This effect is demonstrated in the simulation result shown in Figure 7-7.

For most applications, a PD controller is competent for optimal performance. However, there are cases that the response signal will have a steady state error. For a quadcopter, this could be caused by a center of mass not located at the geometric center. In addition, since the hobbyist grade RC motors and props often have poor quality control, they will have different thrust output. Figure 7-9 show a simulation response with the controller tuned to $K_p = 1.0$ and $K_d = 12.0$. Motor 1 is set to output $4.9\ N$ at maximum pulse width while Motor 0 remain at $4.8\ N$. This causes a steady state error of around +1.5° since Motor 0 doesn't have enough thrust to overcome the thrust bias.

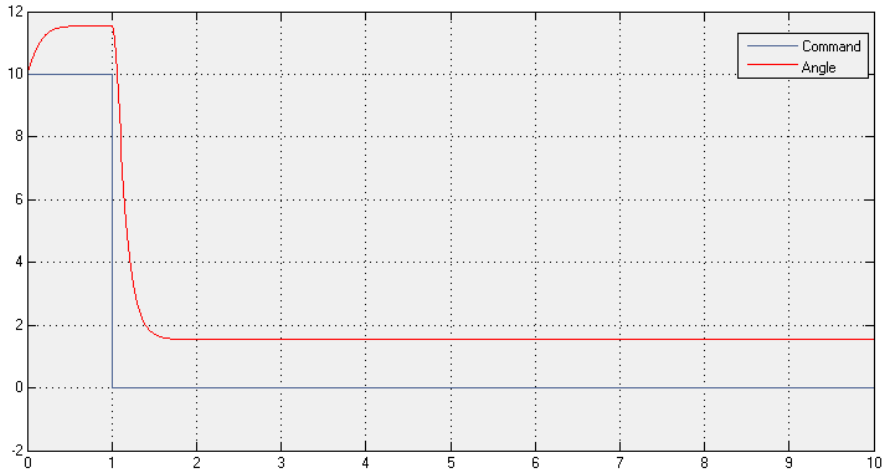


FIGURE 7-9: PD CONTROLLER REPONSES WITH MOTOR BIAS ($K_p = 1, K_d = 12$)

The steady state error can be reduced and eliminated by applying an integral action to the PD controller. All of the components make up the PID controller.

$$\boxed{u_{PID} = K_p e + K_i \int_0^t e\, dt + K_d \frac{de}{dt}}$$

A block diagram of the PID controller is illustrated in Figure 7-10.

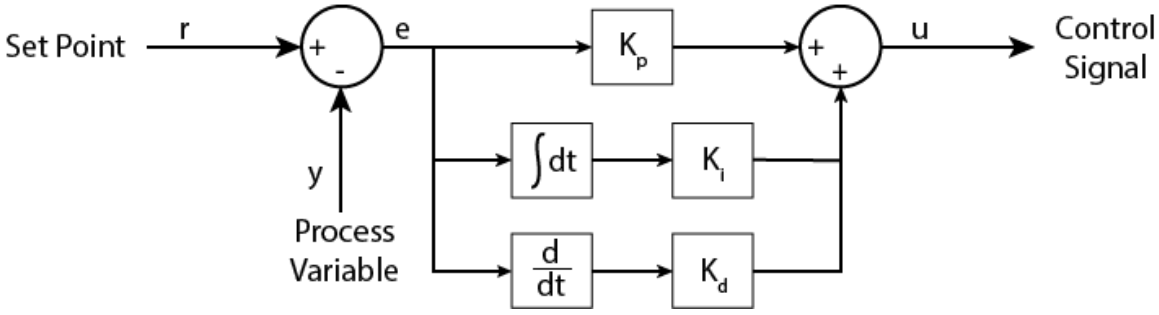


FIGURE 7-10: PID CONTROLLER BLOCK DIAGRAM

The integral action works by accelerating the error over time and then this term is scaled by the integral gain K_i. A simple Riemann sum can be used to implement the integral term digitally.

$$integral_{new} = e \cdot dt + integral_{old}$$

The motor biased simulation is run with the PID controller with the gains $K_p = 1, K_i = 0.01, K_d = 12$. The result is shown in Figure 7-11. As seen in the simulation from time 0 to 1.5 sec, the magnitude of the integral error term is getting larger in each iteration. Hence, it is contributing to a growing negative output signal as long the signal is off from the target. This larger negative signal help converge the signal back to the target.

At time 1.5, it can be seen that the integral term now drive the signal past the target. Since the error signal is then positive, the integral error go from negative toward more positive. Eventually drive the signal back to the target at around 5.8 sec after the step command.

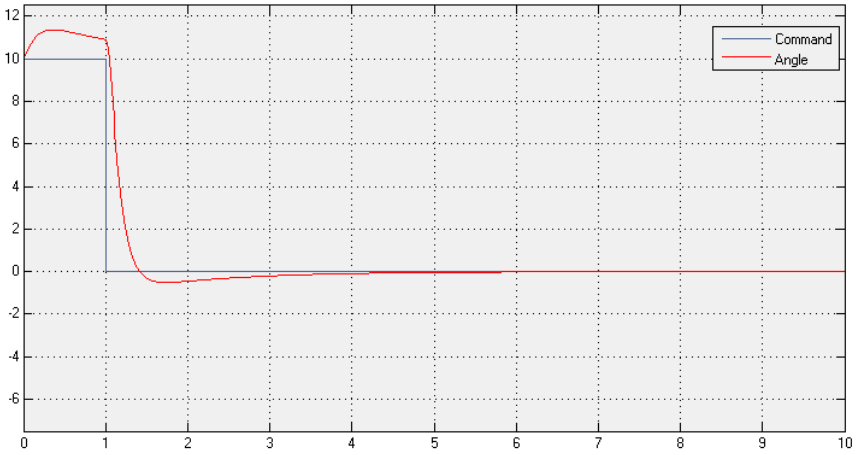


FIGURE 7-11: PID CONTROLLER RESPONSE WITH MOTOR BIAS ($K_p = 1$, $K_i = 0.01, K_d = 12$)

The settling time for the system not quite acceptable. To improve this, the integral gain can be increased. As seen in Figure 7-12, as the integral increases, the overshoot problem worsen. This is called the integral windup.

The Integral windup is cause by an overly large integral term because the system takes too long to converge. Ideally, the integral term should only be large enough so that the bias in the motor could be overcome.

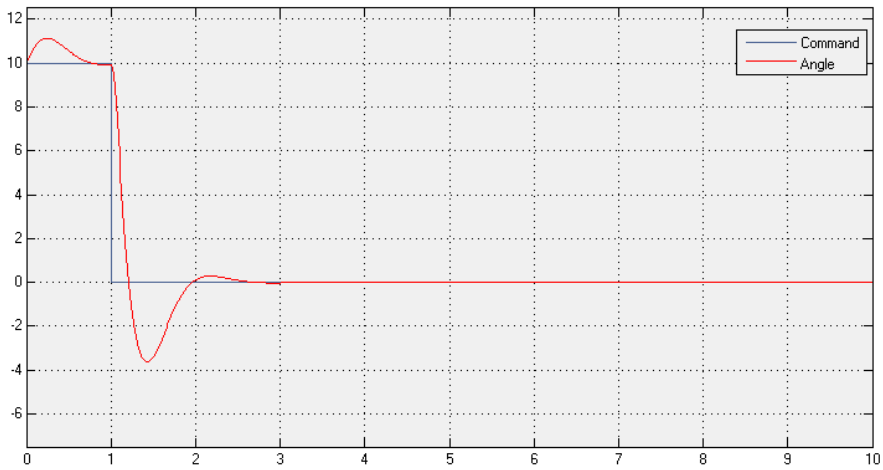


FIGURE 7-12: PID CONTROLLER RESPONSE WITH MOTOR BIAS ($K_p = 1$, $K_i = 0.05, K_d = 12$)

MODIFICATIONS TO THE TRADITIONAL PID CONTROLLER

The PID controller, as shown in the previous section, suffers from the integral windup condition. In addition, the simulations are based on ideal situations where there are no process and measurement noise. When noises are added other problems arise. This section focuses on modifying the PID controller so that it is robust enough for flight control.

The integral windup problem can be addressed by setting a limit on the integral term. By capping both ends of the integral term, the integral action of the controller can be set to just be large enough just to overcome the motor bias and center of mass problems. The accumulating of the error can be modified as follow:

$$if\ (\ abs(integral + error * dt) < integral\_limit)$$

$$integral = error * dt + integral$$

Essentially, the error only accumulates if the absolute value of the operation is less than a certain limit. This will put a cap on the integral error term. The previous example is simulated with an integral limit of 30. The result is illustrated in Figure 7-13.

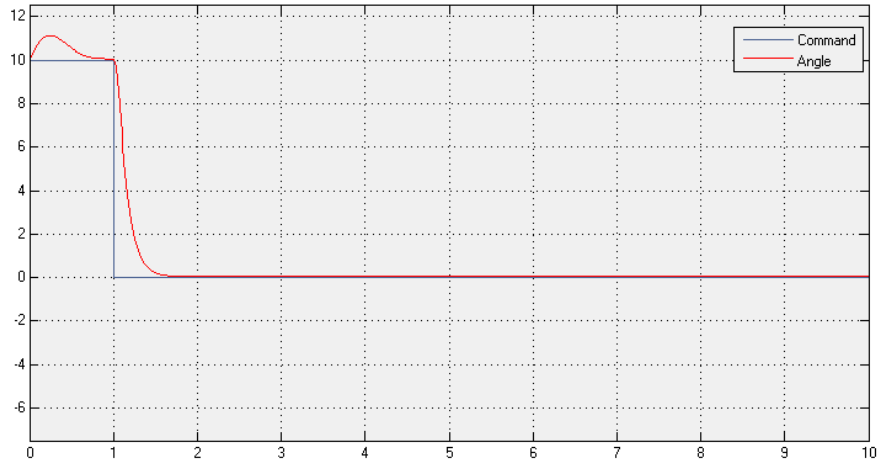


FIGURE 7-13: PID CONTROLLER RESPONSE WITH MOTOR BIAS ($K_p = 1, K_i = 0.05, K_d = 12, Integral\ Limit = 30$)

It is apparent that with the addition of an integral limit, the overshoot problem is solved and that the system converge to the target in 0.5 second. This is significantly better than reducing the K_i since that add a delay in settling time.

Next, the noise issues should be addressed. The derivative term of the PID controller is often problematic due to measurement noise. Recall that the derivative error is calculated by backward differences. With the addition of noise, the equation is as follow:

$$\frac{de}{dt} = \frac{(e_{new} - e_{old}) + noise}{dt}$$

Assuming that the original measurement is $10°$ and that the next measurement is $8°$. The sample time is 0.01 second. In this case, the error rate should be 200. However, let's assume that there's a measurement noise of $\pm 0.5°$, which is rather common, then the derivative error will have an amplified noise of $\pm\ 50°/s$. The noise is quite significant when compared to the original error rate. The noisy error rate could cause more harm than good in terms of dampening the system. This problem could be mitigated by smoothing the signal using a moving average filter as discussed in Chapter 4.

$$\dot{e}_{filtered} = \frac{\dot{e}_t + \dot{e}_{t-1} + \dot{e}_{t-2} + \cdots + \dot{e}_{t-(N-1)}}{N}$$

Figure 7-14 shown the simulation result for a well-tuned PID controller for continuous time like before, but now corrupted with $\pm 0.5°$ of noise in the measurement and now run at a sample rate of 70 Hz. As seen in the simulation, because of the time delay added by running the PID controller as a discrete controller like in real life, the dampening effect is not as good as before. In addition, it can be seen that the noise is amplified in the derivative calculation as shown in Figure 7-15.

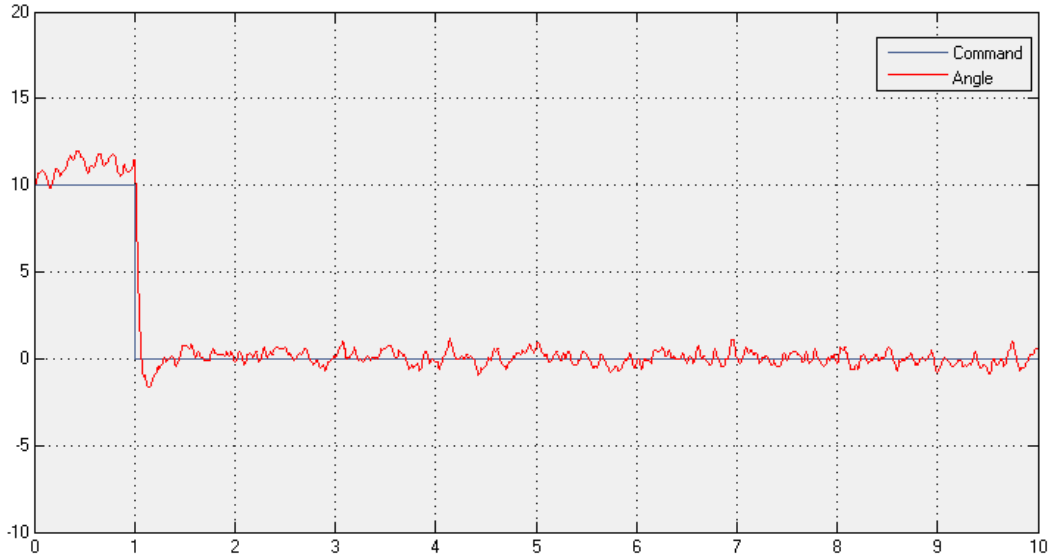


FIGURE 7-14: PID CONTROLLER AT 70 HZ WITH NOISE

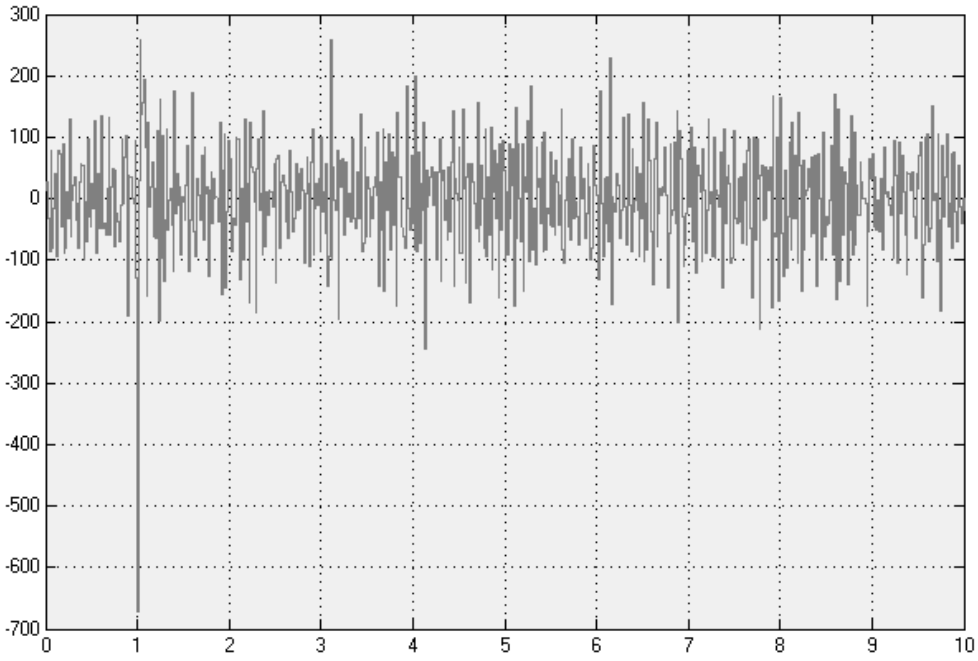


FIGURE 7-15: ERROR DERIVATIVE AT 70 HZ WITH NOISE

Another simulation is run with the implement of a 5-term moving average filter. The results are shown in Figure 7-16 and Figure 7-17. In comparison with the simulation without the filter, it is clear that the high frequency oscillation is gone reduced. The error derivative signal is a lot less noisy and the spike seen in Figure 7-15 is reduced. However, the overall angle signal is still oscillatory at a lower frequency. In addition the overshoot during the step change is increased.

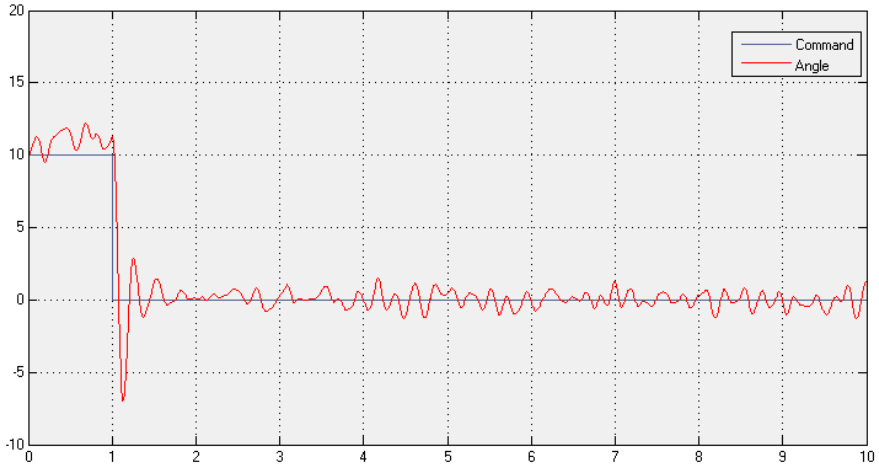


FIGURE 7-16: PID CONTROLLER AT 70 HZ WITH NOISE AND MOVING AVERAGE FILTER

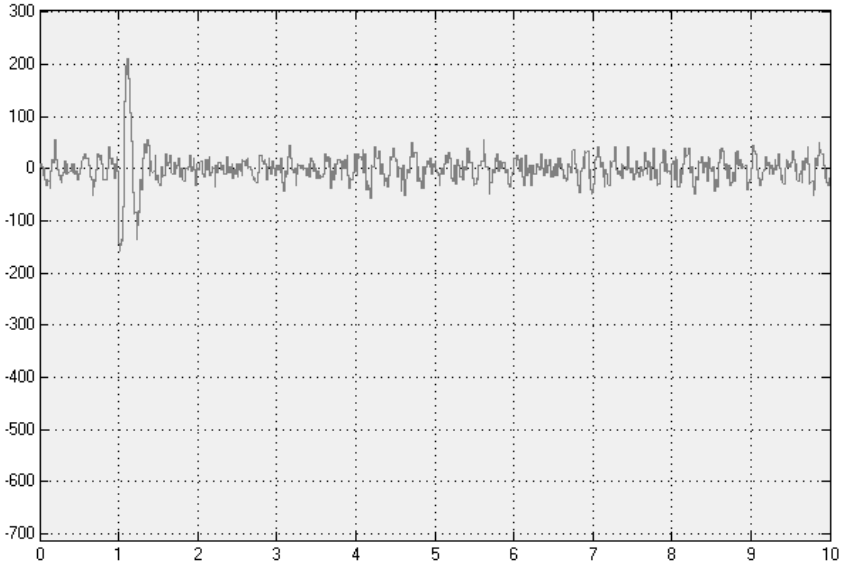


FIGURE 7-17: ERROR DERIVATIVE AT 70 HZ WITH NOISE AND MOVING AVERAGE FILTER

The moving average filter is more effective at higher sample rate. In the simulations shown in Figure 7-18 and 7-19, the controller sample rate has been increased to 200 Hz. In addition, a time delay has been added to the motor for more realistic simulation. As seen in the graphs, the error derivative increased significantly from Figure 7-15 due to the division of a small dt. This makes rendered the PWM output have high oscillation amplitude. The attitude performance is also rather poor, with the angle oscillate most at $\pm 5°$.

The implementation of a moving average filter here significantly reduced the error derivative noise amplitude. This help keeping the PWM output under control. Hence, the attitude performance is better, oscillating at roughly $\pm 2.5°$.

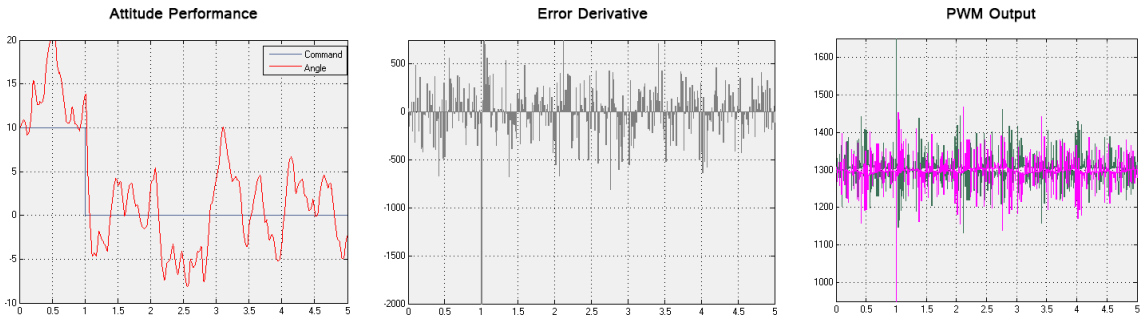


FIGURE 7-18: SIMULATION WITH NOISE AND MOTOR TIME DELAY

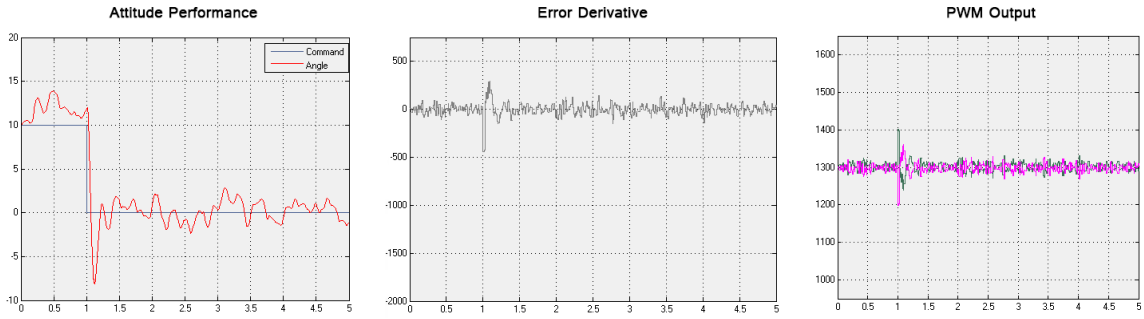


FIGURE 7-19: SIMULATION WITH NOISE, MOTOR TIME DELAY AND MOVING AVERAGE FILTER

The sample time of the PID controller play a major role in determine system performance. The original PID controller is set up so that it is run whenever the process gets to it. Therefore, the time step is varying. This could cause inconsistent performance as shown in Figure 7-20 and make it hard to tune correctly. A way to solve this is to run the PID controller at a fixed time step using a timer. The controller is run only when the timer is triggered. Consider that now the time step is a constant, the PID control law can be simplified as follow:

$$u_{PID} = K_p e + K_i \sum_{0}^{t} e + K_d (e_t - e_{t-1})$$

Essentially, the dt is merged with the K_i and K_d, therefore allowing the integral to be calculated as the sum of all previous error and the error rate as the difference between the current and the last error. An advantage of this implementation is that the controller doesn't need to calculate the time that had passed since the last iteration. In addition overflow problem for the derivative term can be avoided since the calculation no longer divide the error difference by a small number. Unless otherwise specified, this formula simulated continuously in all previous simulations in this section with the exception of the moving average filter demonstrations.

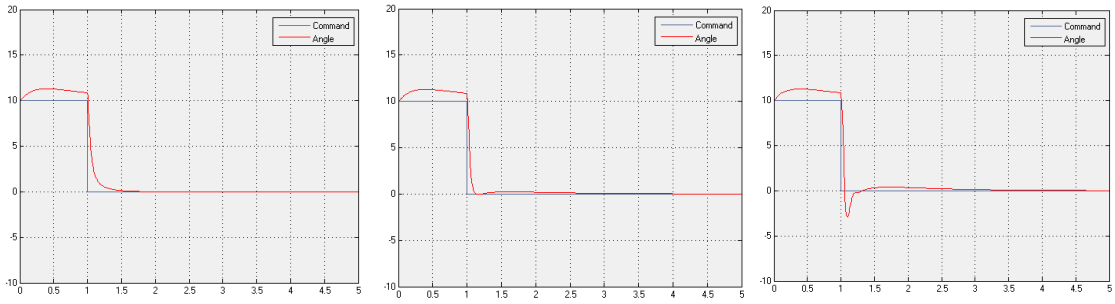


FIGURE 7-20: PID CONTROLLER RESPONSE AT 200 HZ, 100 HZ AND 50 HZ

There were several other modifications that were experimented with but was never fully implemented in the final version of the Flight Controller due to time restriction.

One of the modifications was to replace the derivative action with the derivative of the process variable. Consider a normal PID controller, when a step command is executed, the sudden increase in error will yield a large error rate. Hence, the result is a large output signal and often in the wrong direction as shown in Figure 7-21. This is call a derivative kick.

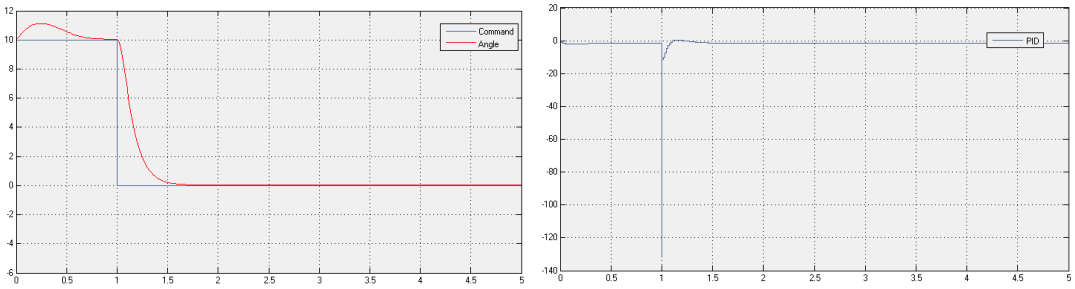


FIGURE 7-21: PID CONTROLLER RESPONSE AND OUTPUT SIGNAL

The derivative kick, although not shown in simulation, might have some undesirable effect on the performance since it will surely saturate the actuators. This condition can be improved by replacing the error derivative by the negative derivative of the process variable. Mathematically, they are equal assuming that the reference signal is constant.

$$\frac{de}{dt} = \frac{d(r-\theta)}{dt} = \frac{dr}{dt} - \frac{d\theta}{dt} = 0 - \frac{d\theta}{dt} = -\frac{d\theta}{dt}$$

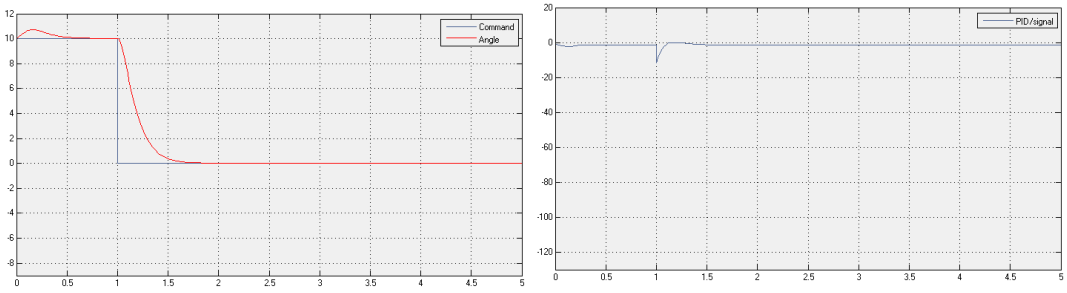


FIGURE 7-22: PID CONTROLLER RESPONSE AND OUTPUT SIGNAL WITH MITIGATED DERIVATIVE KICK

The other modification is also based on the derivative action of the PID controller. Instead of damping the system using the error derivative, this uses the derivative of the process variable like before. An advantage of this type of control scheme is that the error derivative calculation noise can be avoided. The control law is as follow:

$$u_{PID} = K_p e + K_i \int_0^t e\, dt + K_d (K_r e - \frac{d\theta}{dt})$$

The error signal is scaled by an angular rate gain K_r to calculate the target angular rate. The target angular rate is then subtracted by the measured angular rate and result in the angular rate error. This is multiplied by the derivative gain and joins the other parts of the controller.

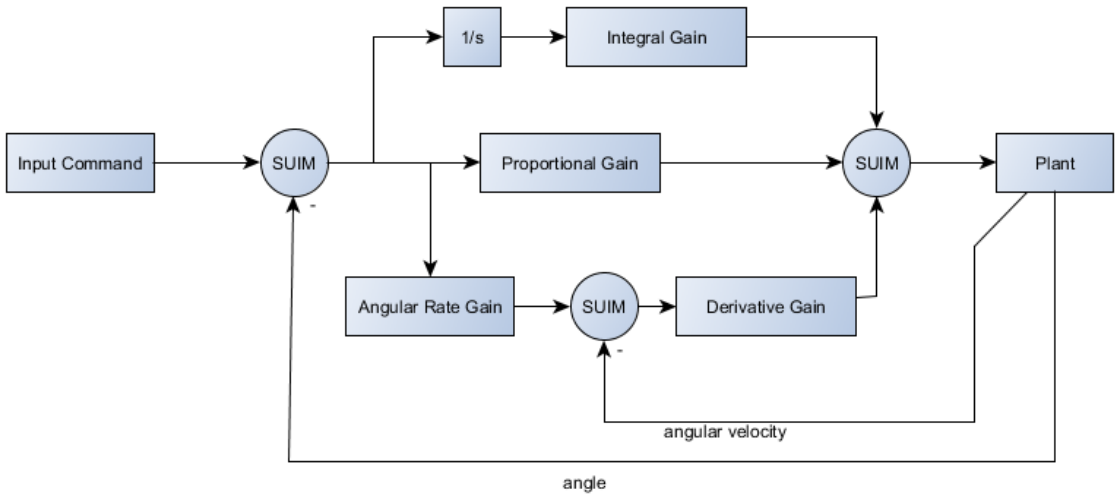


FIGURE 7-23: MODIFIED PID CONTROLLER BLOCK DIAGRAM

This modified controller was used in the early development stage of the control system and went through several rig tests on the prototype hexacopter and the QuadX. Flight data collected from the test were promising. However, the PI-PD cascade controller was eventually implemented in the FlightOS due to the superior damping behavior shown in the Single Axis Motor Balancer test rig.

CASCADE PID CONTROLLER

One of the reasons that makes PID controller powerful is its ability to be cascaded onto each other. The idea of a cascade controller is simple, one controller is used to control the set point of another. For multirotor aircrafts, the primary controller will use the error in angle to calculate a target angular rate. The secondary controller will in turn use the error in angular rate to calculate an output signal for the flight mixer.

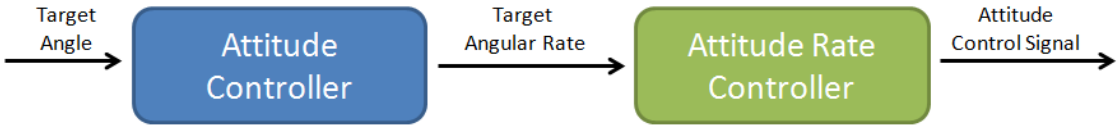


FIGURE 7-24: CASCADE FLIGHT CONTROLLER

The implementation of the cascade controllers starts with the secondary controller and then the primary controller. Since the integration of angular rate is the angle, a PD controller is usually sufficient for the attitude rate controller. The steady error will be handled by the attitude controller. At the same time, a PI controller will usually be sufficient for the attitude controller since the damping will be handled by the attitude rate controller. Together, this forms the PI-PD controller. The experimental results will be discussed in Chapter 10.

QUADX FLIGHT ATTITUDE CONTROLLER

The QuadX flight attitude controller is a 3-axis control system. It is consisted of 2 cascade PID controllers for roll and pitch control. For the manual flight control setup, a PID controller is used for the yaw rate. This allows the user to turn the quadcopter using a RC transmitter or the ground station. Since the yaw angle is with respective to North, it will be inconvenient for the pilot to always have to keep track of the direction. A block diagram of the QuadX flight control system in the manual control setup is illustrated in Figure 7-26.

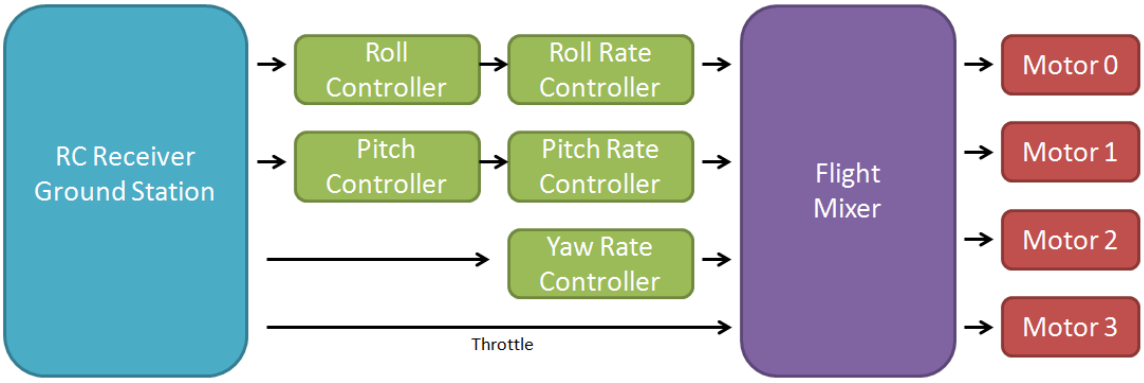


FIGURE 7-25: QUADX FLIGHT ATTITUDE CONTROLLER MANUAL CONTROL SETUP

In autonomous flight, it is necessary for the quadcopter to have complete attitude control. Therefore, a cascade controller is also used for the yaw angle. The guidance module will provide the roll, pitch and yaw commands as oppose to manual inputs by the pilot. A block diagram of the controller in the autonomous flight setup is shown in Figure 7-26.

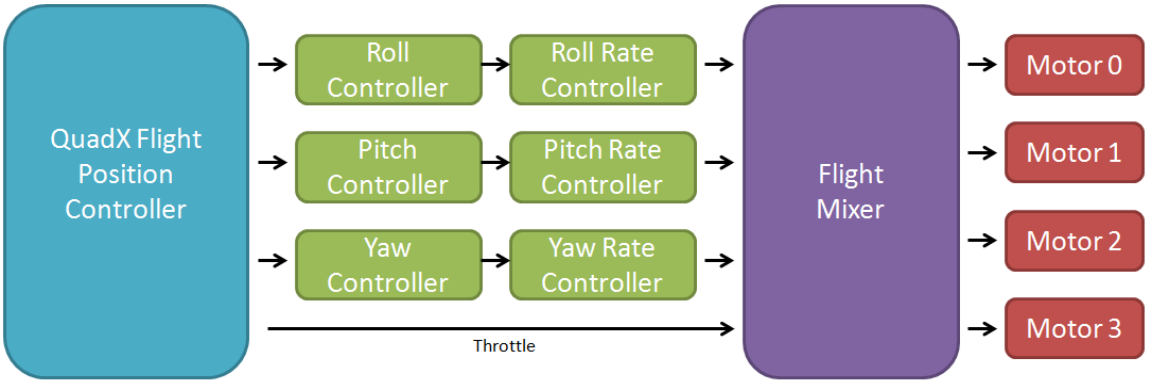


FIGURE 7-26: QUADX FLIGHT ATTITUDE CONTROLLER AUTONOMOUS FLIGHT SETUP

The yaw angle provided by the IMU is commonly reported from $-180°$ to $180°$, where $0°$ is configured to point at the magnetic North. The normal PID controller error calculation algorithm is not the most efficient when the shortest distance between the target and actual angle crosses South. For example, assume that an aircraft is currently at $-135°$ and the target yaw angle is $135°$ as shown in Figure 7-27.

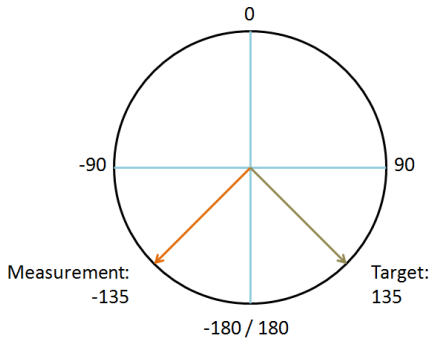


FIGURE 7-27: YAW ANGLE COMPASS

The standard error calculation method will yield a result of $270°$ and signal the aircraft to turn clockwise until the target is reached. However, it is obvious that the closet route to the target is counterclockwise with an error of $-90°$. To mix this, the first step is to remap the angle to the format $0°$ to $360°$ where North is $0°$. Then, the error signal is checked whether it is turning the

most efficient direction. In yaw, the largest magnitude error should only be $180°$. If the error magnitude is larger than $180°$, then it need to be recalculated by manually adjusting the target and measured yaw so that the calculated error will turn the quadcopter in the correct direction. Snippet of the yaw logic code written in Python is illustrated in Figure 7-28.

```
# Yaw Logic Convertor to Calculate the Yaw Error Between Target and Current
def yawLogic(self, yaw, target):

    yaw = self.yawTransform(yaw)
    target = self.yawTransform(target)
    error = target - yaw

    if (abs(error) > 180):
        if (yaw < 180):
            yaw2 = yaw + 360
            error = target - yaw2
        elif (target < 180):
            target2 = target + 360
            error = target2 - yaw

    return error


# Transform IMU Yaw Heading from -180 to 180 to 0 and 360
def yawTransform(self, yawIMU):

    yaw = yawIMU;

    if (yawIMU < 0):
        yaw = (180 + yawIMU) + 180

    return yaw
```

FIGURE 7-28: YAW ERROR CALCULATION LOGIC

QUADX FLIGHT POSITION CONTROLLER

The QuadX's ultimate goal is to achieve autonomous flight to follow a trajectory defined by the guidance system. This requires positional control for localized longitude, latitude and altitude. Recall the hopper simulator's dynamics equation in Chapter 2:

$$\begin{bmatrix} \ddot{N} \\ \ddot{E} \\ \ddot{D} \end{bmatrix}_n = {}^nC^c \begin{bmatrix} \ddot{x}_c \\ \ddot{y}_c \\ \ddot{z}_c \end{bmatrix}_c$$

$$\ddot{N} = -\left(\frac{T_c}{m} - gc\theta c\phi\right) * (s\phi s\psi + c\phi c\psi s\theta) - gc\theta s\phi * (c\phi s\psi - c\psi s\theta s\phi) - gc\theta c\psi s\theta$$

$$\ddot{E} = \left(\frac{T_c}{m} - gc\theta c\phi\right) * (c\psi s\phi - c\phi s\theta s\psi) + gc\theta s\phi * (c\phi c\psi + s\theta s\phi s\psi) - gc\theta s\theta s\psi$$

$$\ddot{D} = gs^2\theta + gc^2\theta s^2\phi - c\theta c\phi * \left(\frac{T_c}{m} - gc\theta c\phi\right)$$

It is clear that all 3 components of the aircraft's position are coupled with the attitude and thrust. The dynamics is highly non-linearly, but an attempt was made to control the position using the linear PID controller. To do this, some assumptions must be made.

Primarily, the position controllers will be controlling positions in the yaw frame $\hat{a}$ as defined in Chapter 2. This way, the yaw frame longitude along the $\hat{a}_y$ direction can be controlled by the aircraft's roll. At the same time, the yaw frame latitude along $\hat{a}_x$ can be controlled using pitch. As for altitude, due to the nature of the yaw rotation matrix, it is the same in body and inertial frame. The altitude is assumed to be controlled by the throttle. A flow chart displaying the control system is illustrated in Figure 7-29.

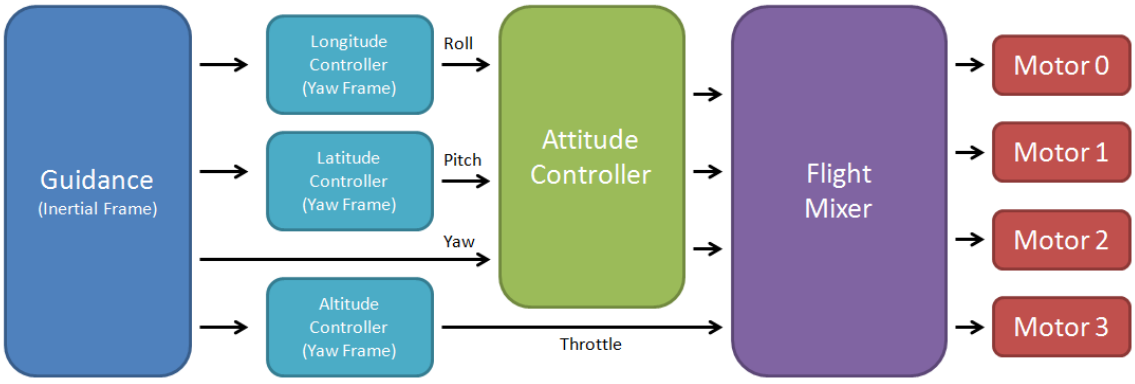


FIGURE 7-29: QUADX FLIGHT POSITION CONTROLLER SETUP

The reference signal provided by the guidance system will be in the inertial frame. In addition, the localized GPS location along with the height provided by the ultrasonic sensor will also be in

the inertial frame. Therefore, it is important transform the measurements and reference signal to the correct frame using the yaw angle and the ${}^{a}C^{n}$ rotation matrix before calculating the error signal for the PID controllers. Since it is more useful to know the aircraft's height above ground for low altitude, the height in the guidance and control system is modified to $U = -D$. This makes it easier to interpret plots and data later on.

$$target_{yaw} = {}^{a}C^{n}\begin{bmatrix} N_t \\ E_t \\ U_t \end{bmatrix}_n$$

$$actual_{yaw} = {}^{a}C^{n}\begin{bmatrix} N_{actual} \\ E_{actual} \\ U_{actual} \end{bmatrix}_n$$

$$error_{yaw} = \begin{bmatrix} e_{y_a} \\ e_{x_a} \\ e_{z_a} \end{bmatrix}_{yaw} = target_{yaw} - actual_{yaw}$$

The output of the yaw frame longitude and latitude controllers are the target angles and are directly fed into the attitude control system. The altitude controller signal output, however, must be combined with the hovering PWM pulse width before it can be used as the throttle pulse width. With this modification, the PID controller can increase or decrease the multirotor's altitude without much effort. The hovering pulse width for the QuadX with 1650 g of mass is around 1350. The simulation and experimental result of this value matches closely.

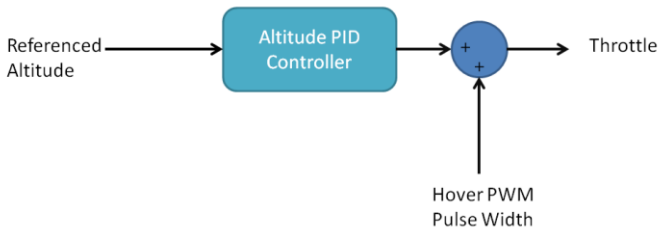


FIGURE 7-30: ALTITUDE CONTROLLER TO THROTTLE

Chapter 8: NAVIGATION SYSTEMS

The guidance and navigation system enable autonomous flight for the space hopper simulator. Though there was not enough time for full implementation into the FlightOS, the two were studied in computer simulations and partially experimented.

GPS LOCALIZATION

The navigational data received from the GPS are in geo-coordinate format. This is not particularly useful in positional control since it is not possible to directly compute the displacement error differences. To do that, the geo-coordinate should first be localized to a reference position. This reference position would most likely be the initial position of the aircraft when first power on.

For example, geo-coordinates collected with the GPS in a test in front of Packard Lab in Lehigh University are plotted in Figure 8-1.

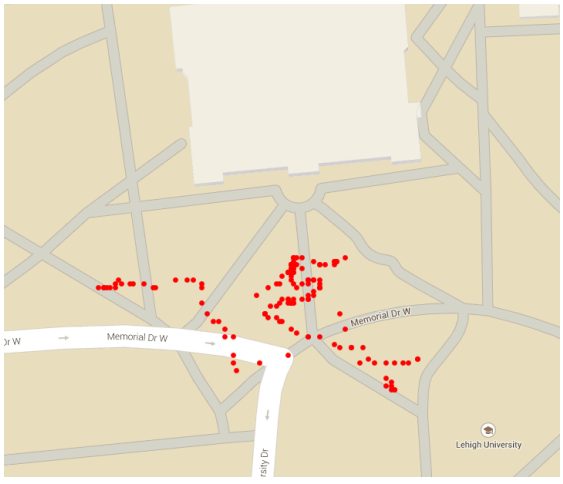


FIGURE 8-1: RAW EXPERIMENTAL GPS POSITION DATA

To the human eye, the plot is very useful in determining the position of the sensor since we can relate the position with respect to the roads and buildings. However, to a machine, this means nothing since it doesn't take into account of those parameters.

GPS Coordinate String to Radian Coordinate Conversion

The geo-coordinate reported by the GPS is usually in a degree format that must be processed before localization. Consider the following coordinates from the Ultimate GPS:

$$4036.5400N, 7522.6313W$$

The string must be first rearranged into degree and minute format.

$$40° \; 36.5400'N, 75° \; 22.6313'W$$

Then, to obtain the decimal degree form, the minute term should be divided by 60. In addition, North and West should be translate to + and – with respective to the inertial frame.

$$40.609 \; , -75.377$$

Finally, the coordinate should be converted into radian by multiplying by $\pi/180°$.

$$0.709 \; , -1.315$$

This string to decimal process should be automated and implemented in the flight navigation system. In addition, the significant digits should be retained along the calculation process for the most accurate position calculation.

FLAT EARTH APPROXIMATION

To localize the geo-coordinates, displacements must be calculated for all points with respect to the initial coordinate. For small distances away from the poles, a simple flat Earth approximation could be used [18]. The displacement calculated are in the standard North, East, Down inertial frame. Note that in the following equations, the geo-coordinates are expressed in radian.

$$D_{North} = R_1 * dLat$$

$$D_{East} = R_2 * \cos(lat0) * dLon$$

$$dLat = lat1 - lat0$$

$$dLon = lon1 - lon0$$

R_1 is the meridional radius of curvature and R_2 is the prime vertical radius of curvature. The radius can be calculated with the World Geodetic System (WGS) parameters equatorial radius $a = 6378.137E3\ m$ and flattening $f = 1/298.257223563$.

$$R_1 = \frac{a(1-e^2)}{(1-e^2\sin(lat0)^2)^{1.5}}$$

$$R_2 = \frac{a}{\sqrt{1-e^2\sin(lat0)^2}}$$

$$e^2 = f(2-f)$$

The localized experimental data from Figure 8-1 is plotted in Figure 8-2. The processed data are expressed in meters displaced from the origin in the North and East direction. This is perfect for comparing with the trajectory generated by the guidance module and can be directly plugged in to position PID controller.

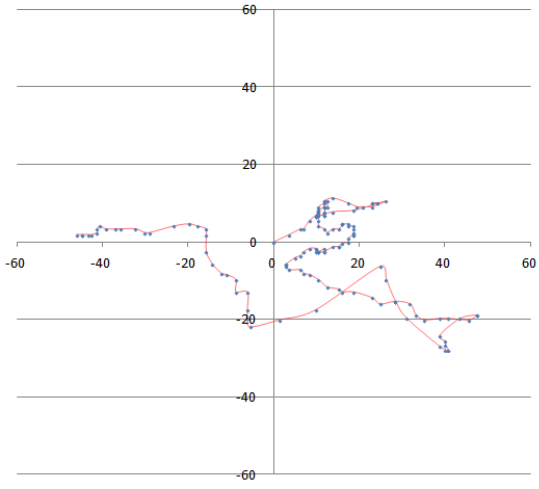


FIGURE 8-2: LOCALIZED EXPERIMENTAL GPS POSITION DATA (M)

GPS SENSOR NOISE

One of the biggest problems with the GPS is that there seems to a lot of noise and delay in the data. The precision and accuracy of the reported position are determined by a number of external factors as the atmospheric condition and signal bounces that are sometime unavoidable. Even though the minimum number of satellite required for a GPS receiver to calculate the position is 3, in reality, the positions calculated are quite poor in quality.

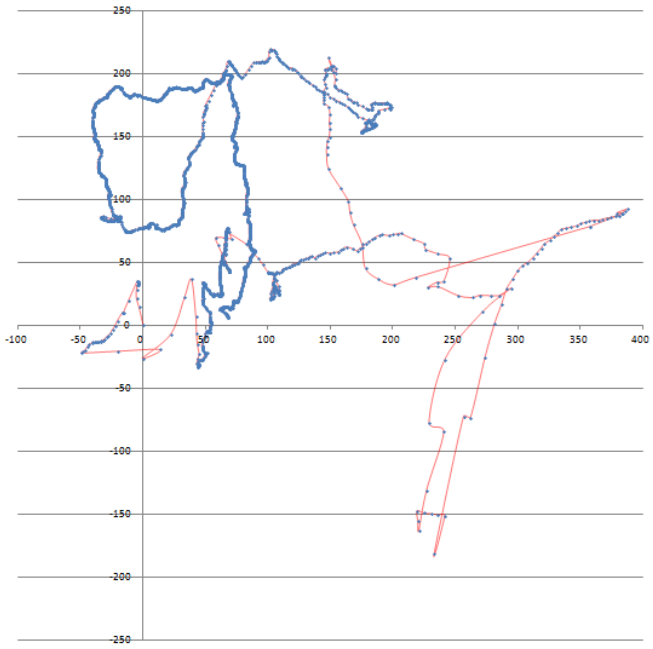


FIGURE 8-3: LOCALIZED EXPERIMENTAL DATA COLLECTED AT LEHIGH UNIVERSITY ASA PACKER CAMPUS

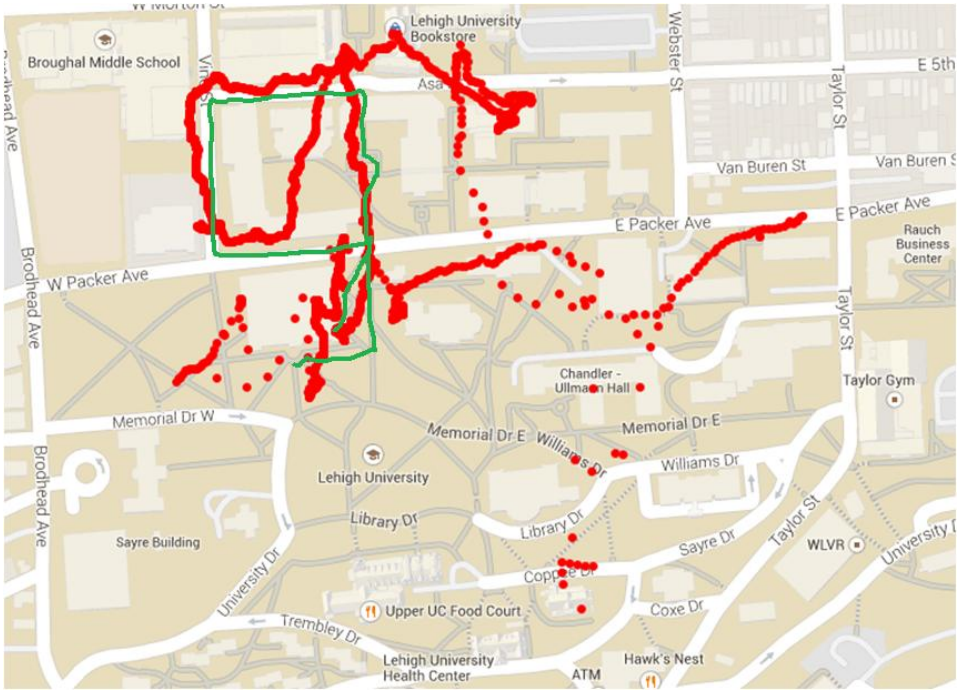


FIGURE 8-4: ACTUAL VS. MEASURED PATH TRAVELLED (GREEN = TRUE PATH / RED = GPS PATH)

Several GPS tests were conducted in the Asa Packer Campus at Lehigh University with the assistance of an undergraduate student. Figure 8-4 illustrated the differences between the actual and measured travel path. On the other hand, Figure 8-3 shows the order in which the data points are collected. As seen in the figure, there is a lot of noise in the measurements.

The actual travel path started at the front of Packard lab and head North until intersecting with Packer Ave. Then, the test rig was brought along the side of STEP toward West Packer Ave and turn North into Vine Street. The group then turned into Asa Drive along the parking garage, going through the bridge at the Fairchild-Martindale Library and head south back to Packard Lab.

By analyzing Figure 8-4 with Figure 8-3, it can be concluded that the beginning measurements were the noisiest. The data shown that after exiting Packard Lab, the position jumped East to Packard Memorial Church and Fritz Lab. Then, the signal jump South through Chandler-Ullmann, Linderman Library and rest at Coppee Hall. The position then jumps Northeast toward Wilbur Power House and stop at the Heating Refrigeration Building across from Rauch Business Center. The signal then shoots Northwest crossing Neville Hall, Sinclair Lab and end up at the bookstore. Then the sensor finally cut through STEPS and converges to the actual travel path and meets up at West Packer Ave.

Looking at the density of the measurement point, it is clear that the earlier data points jump all around the campus. In the actual travel path, the group was walking under a path covered with tall trees overhead. Referencing the logged data from the GPS, the lock during that period consisted only of 3 satellites. The condition improve significantly after meeting with Packer Ave since that area is out of the main campus area and is more open. The satellite lock count increased to 6 and the position slowly converges closer to the actual travel path.

It is apparent that the raw data collected from the GPS is not always reliable and is directly related to the number of satellites the receiver can get a signal lock. In order to get a high number of locks, the receiver must be in an open area with minimal overhead blockage that will induce signal bounce. It is a big limitation that the guidance, navigation and position control system cannot be used in the Asa Packer Campus because of the trees. In this case, the optimal location to develop the navigation and positional control system is Goodman campus. However, it is located very far from the Aerospace Systems Lab thus hinder the development process.

A possible solution to this problem is to add an optical flow sensor for sensor fusion with the GPS. The optical flow sensor is a common electronic used in laser mouse. Basically, it is a camera combined with an ultrasonic range finder that detects velocity by taking picture of the ground and comparing the shift in pixels. The data is similar to that of a gyroscope but instead of attitude rate, this report position rate in the yaw frame. The sensor data can be integrated to find position relative to the starting position, but care must be taken to convert the data back to inertial frame.

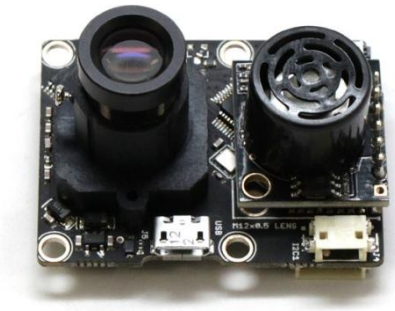


FIGURE 8-5: 3D ROBOTICS PX4FLOW OPTICAL FLOW SENSOR [19]

GPS Position Kalman Filtering

The data noise in Figure 8-4 is somewhat extreme, the measurements are unusable and most likely no simple filter will be able to help that. However, in normal operation with a more reasonable number of satellite locks, a moving average filter will help smooth out the noisy data. Though care should be taken to avoid adding too much delay into the system.

A Kalman Filter could be used to combine the aircraft's dynamics, kinematics, and the noisy GPS measurements to obtain a more accurate position estimation. Consider the following kinematic equations:

$$x_1 = v_0 t + \frac{1}{2} a t^2 + x_0$$

$$v_1 = v_0 + a_t$$

Applying the kinematic equations above for the inertial frame and treating the accelerations as input from the dynamics equations, the following state space model and process covariance matrix is obtained:

Prediction

$$x_k = A x_{k-1} + B u + w_k$$

$$\begin{bmatrix} N \\ \dot{N} \\ E \\ \dot{E} \\ D \\ \dot{D} \end{bmatrix}_k = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} N \\ \dot{N} \\ E \\ \dot{E} \\ D \\ \dot{D} \end{bmatrix}_{k-1} + \begin{bmatrix} 0.5 * dt^2 & 0 & 0 \\ dt & 0 & 0 \\ 0 & 0.5 * dt^2 & 0 \\ 0 & dt & 0 \\ 0 & 0 & 0.5 * dt^2 \\ 0 & 0 & dt \end{bmatrix} \begin{bmatrix} \ddot{N} \\ \ddot{E} \\ \ddot{D} \end{bmatrix} + w_k$$

$$w_k \sim N(0, Q_k)$$

$$Q_k = \begin{bmatrix} Q_N & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_{\dot{N}} & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_E & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_{\dot{E}} & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_D & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_{\dot{D}} \end{bmatrix} dt$$

The inputs of the state space model are an acceleration vector in the North, East and Down direction. Using the roll, pitch and yaw angles obtained from the IMU, along with an estimated thrust from the PWM pulse width output, the accelerations can be calculated. The thrust is estimated through linear interpolation to relate PWM pulse width with thrust. For the Kalman filter simulation:

$$pwm_{min} = 1150\ \mu s$$

$$pwm_{max} = 1700\ \mu s$$

$$Thrust_{min} = 0\ N$$

$$Thrust_{max} = 1.07 \pm 0.03 * 9.81\ N$$

$$Thrust = (pwm - pwm_{\min}) * \frac{Thrust_{max} - Thrust_{min}}{pwm_{max} - pwm_{min}} + Thrust_{min}$$

$$T_c = Thrust_{motor0} + Thrust_{motor1} + Thrust_{motor2} + Thrust_{motor3}$$

With the thrust data available, the dynamics equation from Chapter 2 can be used to calculate acceleration in the Body frame. Then, a ${}^{n}C^{a}$ rotation matrix can be used to transform all the acceleration back to inertial frame.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_c = \begin{bmatrix} -gs\theta \\ gc\theta s\phi \\ gc\theta c\phi - \frac{T_c}{m} \end{bmatrix}$$

$$\begin{bmatrix} \ddot{N} \\ \ddot{E} \\ \ddot{D} \end{bmatrix}_n = {}^{n}C^{c} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}_c$$

OBSERVATION

$$z_k = Cx_{k-1} + v_k$$

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} N \\ \dot{N} \\ E \\ \dot{E} \\ D \\ \dot{D} \end{bmatrix}_k + v_k$$

$$v_k \sim N(0, R_k)$$

$$R = \begin{bmatrix} R_N & 0 & 0 \\ 0 & R_E & 0 \\ 0 & 0 & R_D \end{bmatrix}$$

The data observed by the GPS and the ultrasonic range finder are position in the North, East and Down. Since the measurements are noisy, the R values are usually tuned high to trust the dynamics prediction more than the noisy measurement data collected.

INITIALIZATION

$$P_k = \begin{bmatrix} L & 0 & 0 & 0 & 0 & 0 \\ 0 & L & 0 & 0 & 0 & 0 \\ 0 & 0 & L & 0 & 0 & 0 \\ 0 & 0 & 0 & L & 0 & 0 \\ 0 & 0 & 0 & 0 & L & 0 \\ 0 & 0 & 0 & 0 & 0 & L \end{bmatrix}$$

$$G_k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For initialization, the error covariance and Kalman gain matrix should be initialized to a diagonal matrix with large elements and a zero 6 by 3 matrix respective. The initial states vector can be set to 0 and the Kalman filter will automatically adjust and converge to the actual value later in estimation.

IMPLEMENTATION

The Kalman filter was implemented in Simulink using the processes described in Chapter 4. Noise is added to the attitude and thrust estimation for the simulation to be more realistic. In addition, the GPS and ultrasonic sensor are set to have a Gaussian noise of 10 m and 0.3 m respectively. Initially, the Kalman filter was set to run at 5 Hz. However, it seems that this frequency is too slow for the filter's dynamics model predicts the position with enough accuracy. Therefore, the sample frequency was set to 70 Hz. With the measurement noise matrix Rset to $diag([10000000\ 100000\ 100000])$, the simulation filtering result is as follow.

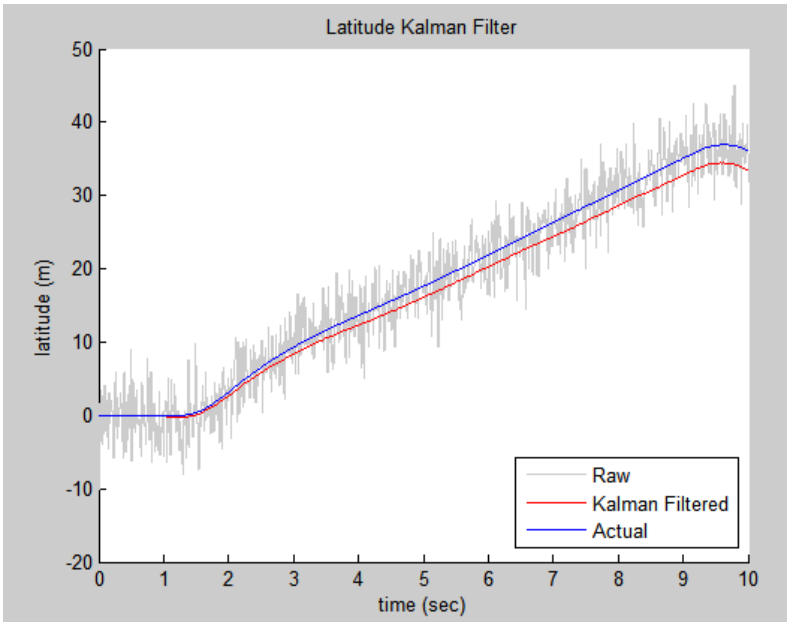


FIGURE 8-6: KALMAN FILTERED LATITUDE DATA WITH $R_N = 10000000$

As seen in Figure 8-6, the Kalman filter is able to effectively filter out the high frequency noise in the system. However, it is apparent that there's a drift issue with the noise and uncertainty in the state space inputs. This indicated that the filter is relying too much on the physics model and R_N should be lowered to rely more on the sensor data.

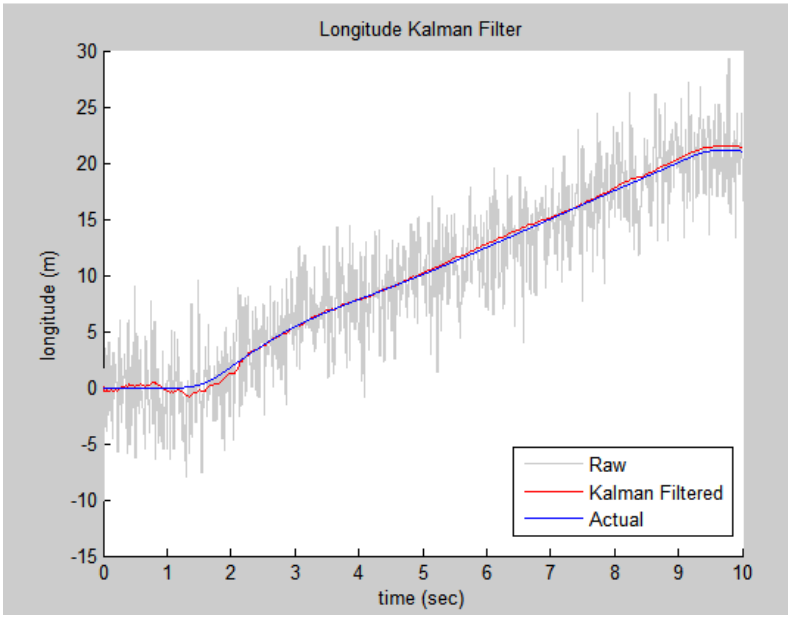


FIGURE 8-7: KALMAN FILTERED LONGITUDE DATA WITH $R_N = 100000$

Figure 8-7 illustrated the longitude filtered result. As seen in the figure, even though the signal is not as smooth as filtered signal from Figure 8-6, the drift problem is mostly solved. At the same time, Figure 8-8 shows a well-tuned signal that has no noticeable drift issues.

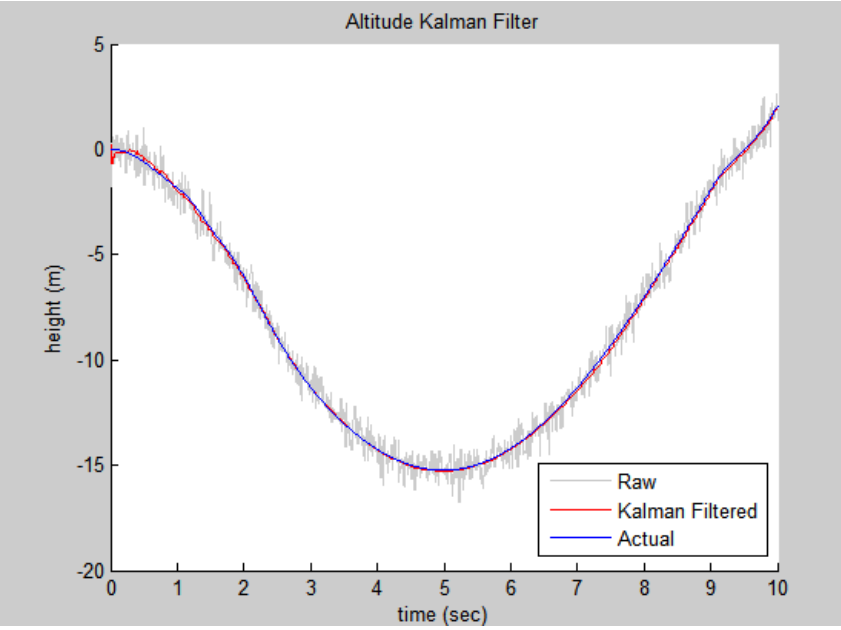


FIGURE 8-8: KALMAN FILTERED ALTITUDE DATA WITH $R_N = 100000$

The downfall of a 70 Hz Kalman filter is that the processing time is very long in Simulink due to the amount of linear algebra involved. Therefore, the sampling frequency is reduced to 20 Hz. In addition, the Gaussian noise generator is set to generate at 5 Hz to reflect the GPS sampling rate in real life. This, however, causes a drop in performance for the Kalman filter. The time step in the physics model is larger, hence, the prediction error is larger and the predicted signal tends to drift over time. Therefore, the R matrix must be tuned lower to rely more on the measurement data. That in turn causes more oscillation.

The simulation result with the measurement noise matrix set as $diag([1500\ 2000\ 60])$ is illustrated in Figure 8-9.

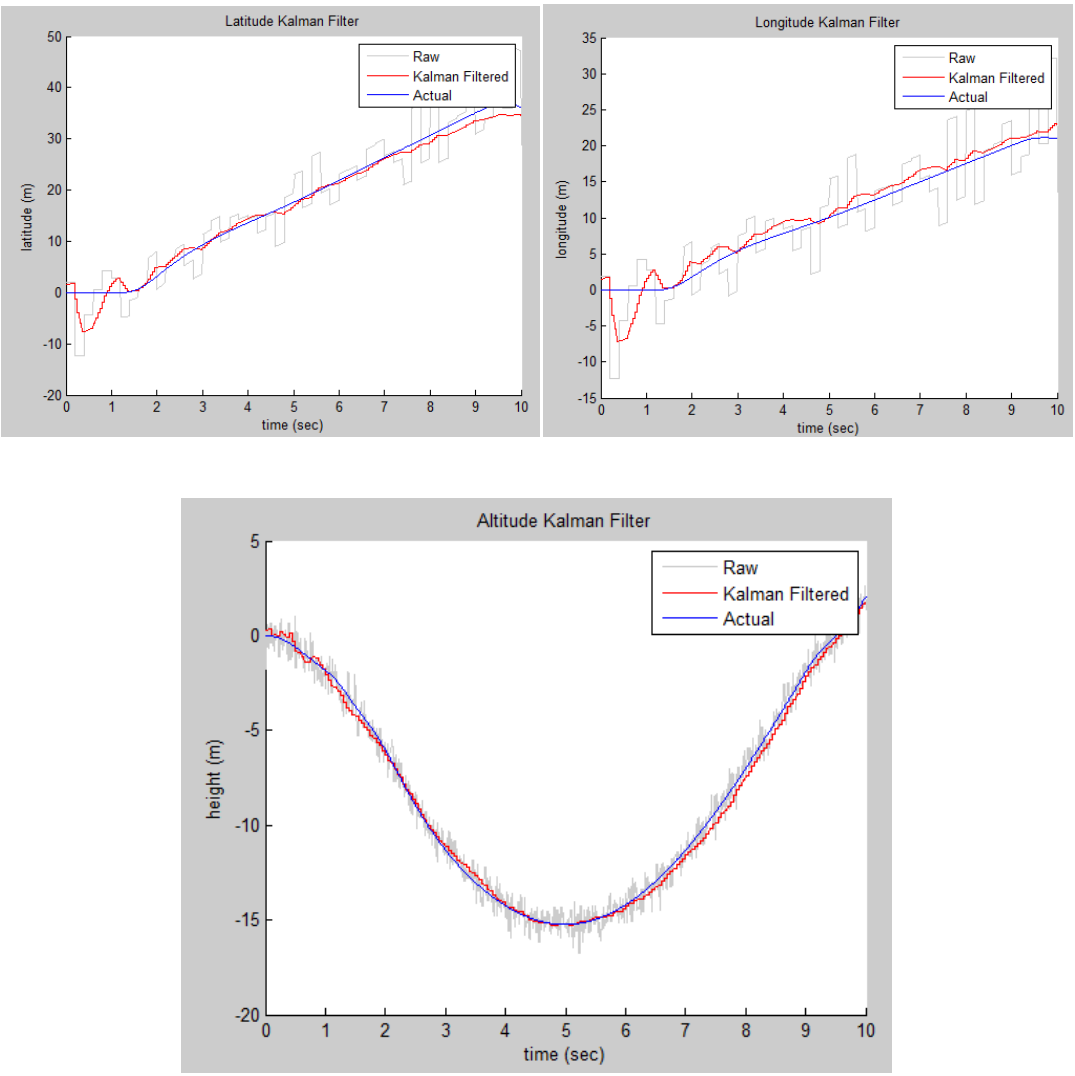


FIGURE 8-9: KALMAN FILTER SIMULATION RESULT AT 20 HZ

One of the advantages of the Kalman filter is its ability to use the dynamics model to predict aircraft position. The Kalman gain is computed by using the differences between the measured and predicted position. In spacecraft navigation, measurements between data points are often slow and might sometime not available due to communication blockage like during an Apollo mission at the far side of the moon. With the Kalman filter, it is possible to continuously estimating the spacecraft's position using the dynamics model. The simplest way to implement this is by disabling the Kalman gain state update phase when there is no measurement:

$$\hat{x}_k = \hat{x}_k + G_k(z_k - C\hat{x}_k)$$

A simulation was conducted with measurement missing from 3 to 5.5 seconds and the results are illustrated in Figure 8-10. In comparison with Figure 8-9, the estimated signal drifted away from the actual signal, but converges back after measurement is available again.

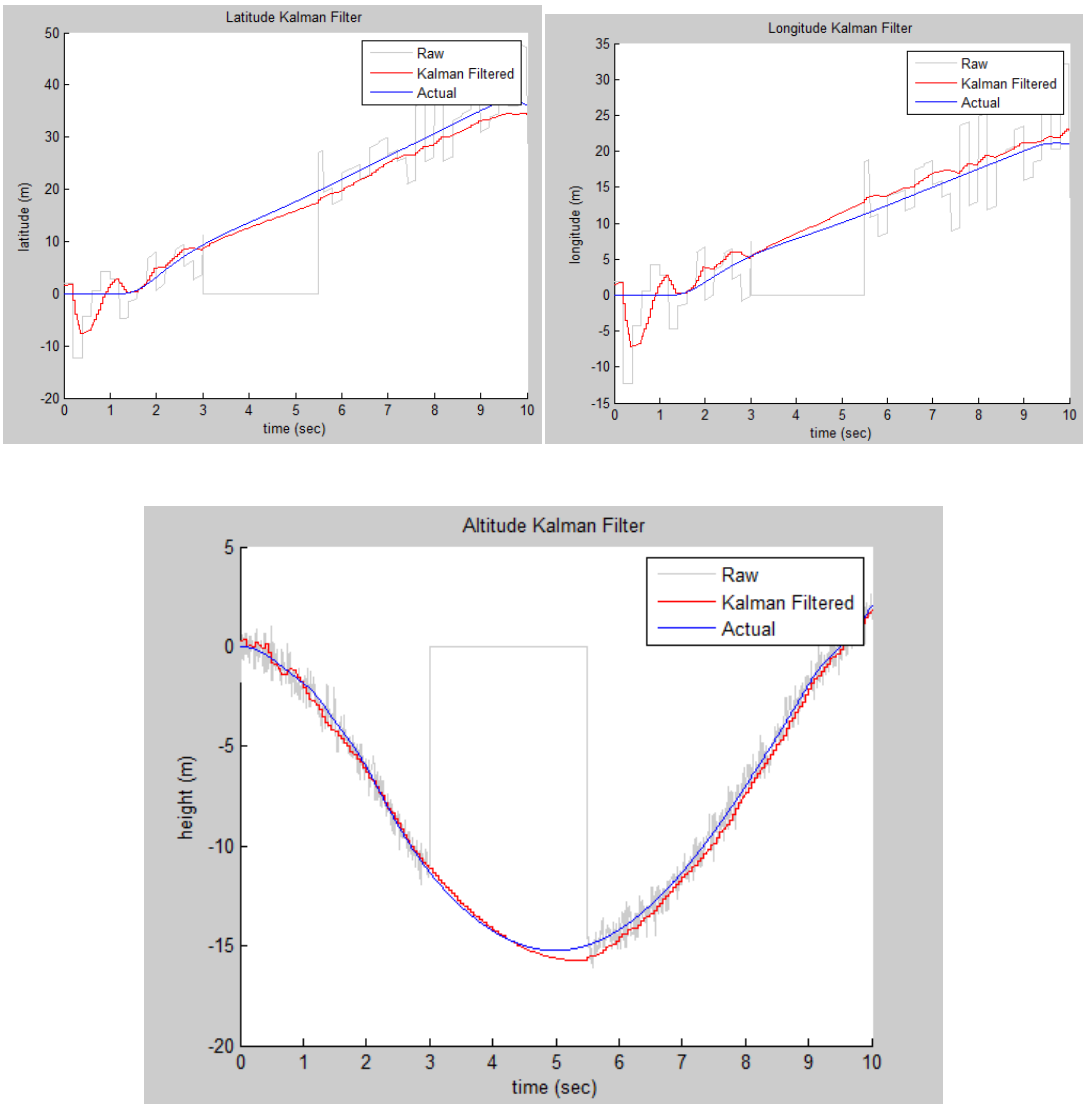


FIGURE 8-10: KALMAN FILTER SIMULATION RESULT AT 20 HZ WITH MISSING MEASUREMENT

Chapter 9: Guidance Systems

The Hopping Maneuver

The guidance system is in charge of generating a reference trajectory for an aircraft to follow. Traditionally, the trajectory would be a straight line connects two way points, since this is the shortest route in flight. For the hopper spacecraft, the robot would first be landed on the lunar surface. It would take a lot of propellant to take off, maintain a constant altitude, maneuver horizontally and land. The most optimal trajectory to travel large distance is through a hopping maneuver. The spacecraft would thrust at an angle to take off and provide a horizontal velocity. Near the end of the parabolic flight, the spacecraft would execute a reverse thrust to cancel out the horizontal velocity and slow down the descending rate.

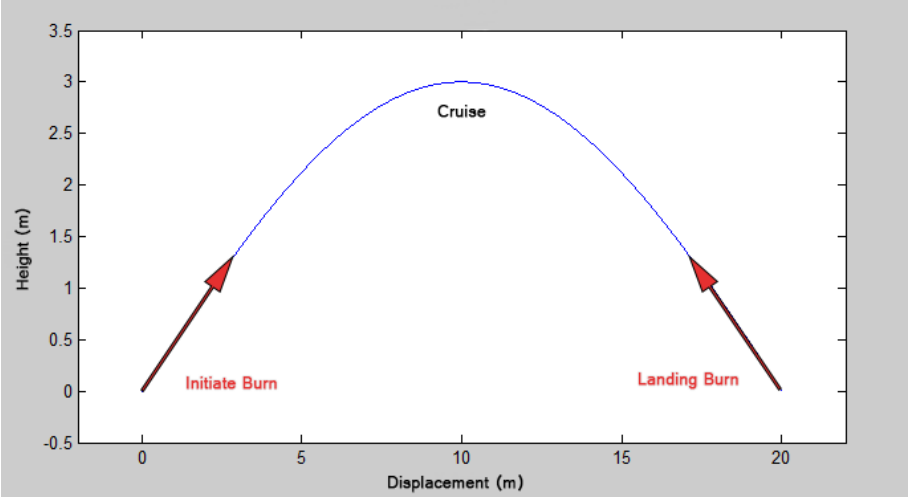


FIGURE 9-1: HOPPER SPACECRAFT HOPPING MANEUVER TRAJECTORY

A major disadvantage of this trajectory is that during the landing burn, any overshoot in the altitude control system would likely damage or destroy the drone. To prevent this, the trajectory is modified so that the aircraft first take off and slowly hover to a pre-defined minimum safety altitude. The aircraft then execute a hop and end targeting the safety altitude. This way, if there

is any overshoot during the reverse thrust, there will be a buffer zone before the aircraft impact with the ground. Finally the aircraft will slowly descend to ground level for landing.

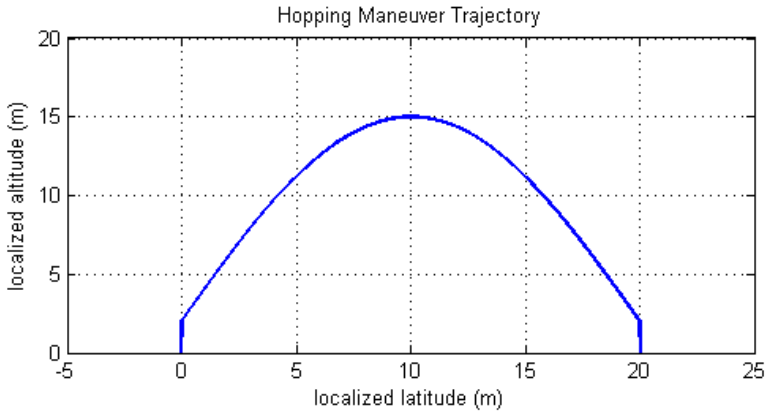


FIGURE 9-2: MODIFIED HOPPING MANEUVER TRAJECTORY

HOPPING TRAJECTORY GENERATOR

The hopping trajectory is defined by several key parameters: the destination coordinate, max height, min safety height, hopping duration and the initial take off/landing duration. These parameters directly affect the characteristic of the flight path and should be adjusted to match mission specs to satisfy geographic restrictions such as the lunar terrain.

The overall hopping maneuver trajectory is consisted of the longitudinal, latitudinal, and altitudinal component. These components are generated separately and feed directly into the flight position controller.

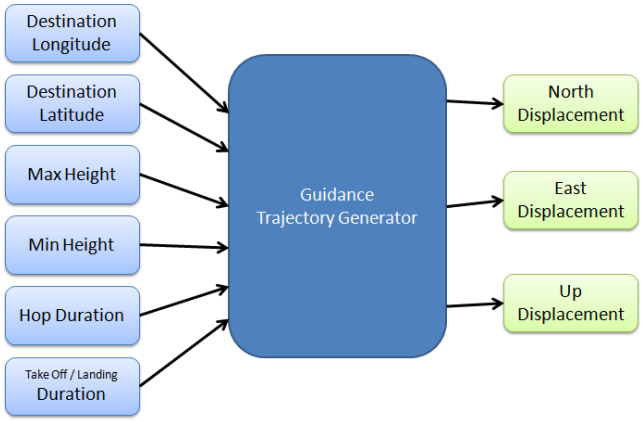


FIGURE 9-3: HOPPING TRAJECTORY GENERATOR

LONGITUDE / LATITUDE TRAJECTORY GENERATOR

The longitude and latitude trajectory are generated using the same method since they both control how the aircraft move horizontally. In the hopping maneuver, the horizontal position remains the constant as the aircraft slowly reaches the safety altitude. During the hopping phase, the aircraft translate at constant speed to its destination and stop when the target is reached. The aircraft then remain at the same horizontal position as it descends. Overall, the horizontal trajectory should look like a ramp command as shown in Figure 9-4.

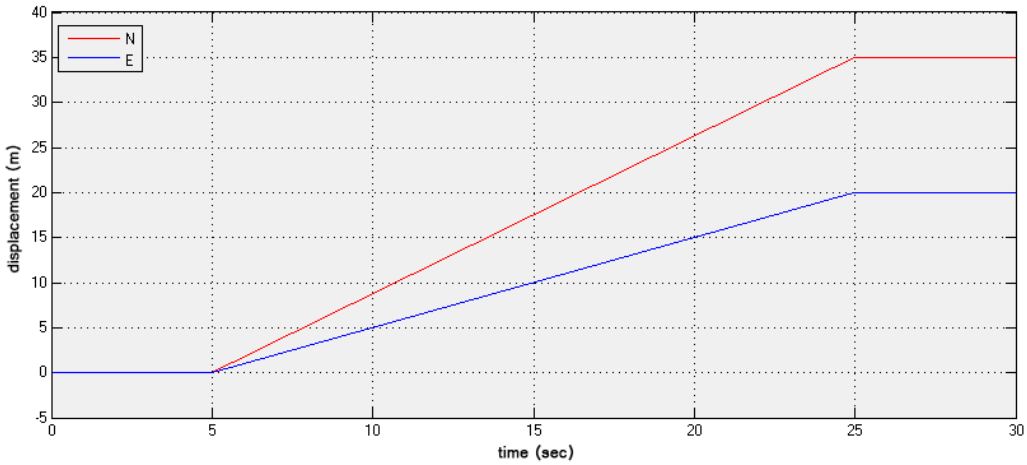


FIGURE 9-4: HORIZONTAL TRAJECTORY FOR LOCALIZED DESTINATION (35 N, 20 E)

As seen in Figure 9-4, the horizontal trajectory is broken into 3 parts. The initial position holds for takeoff, constant velocity cruising during hop and final position hold for landing. Let's assume that t_{hold} represents the position holding duration and t_{cruise} represents the cruising duration defined by the operator. In addition, the variable t represent the time after a hop command is executed. For position, lon_t and lat_t represent the localized destination coordinate while lon_0 and lat_0 represent the coordinate of the origin. The longitudinal trajectory is generated as follow. Again, the latitudinal trajectory is generated using the same method.

Take Off Phase $(\boldsymbol{t \leq t_{hold}})$**:**

$$E = lon0$$

Cruise Phase $\left(\boldsymbol{t > t_{hold}\ and\ t < (t_{hold} + t_{cruise})} \right)$:

$$E = \frac{lon_t - lon_0}{t_{cruise}} * (t - t_{hold})$$

Landing Phase $\left(\boldsymbol{t \geq (t_{hold} + t_{cruise})}\right)$:

$$E = lon_t$$

ALTITUDE TRAJECTORY GENERATOR

The altitude trajectory is also broken into the same 3 phases. However, the overall geometry is a bit more complicated. In the takeoff phase, the QuadX is command to slowly rise to the minimum safety height in t_{hold}. This will be a ramp command. Then in the hopping maneuver, altitude is controlled by a sine wave to achieve a parabolic trajectory. At the end of the hop, the aircraft enter landing phase and ramp down to the ground. An example vertical trajectory is shown in Figure 9-5.

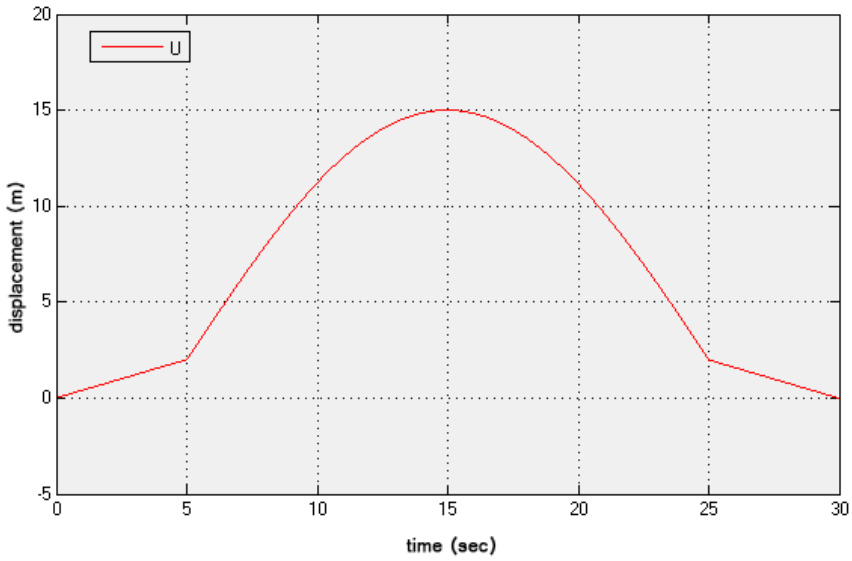


FIGURE 9-5: VERTICAL TRAJECTORY (MAX ALTITUDE: 15M / MIN ALTITUDE: 2M)

To calculate the trajectory, let's assume that $height_{max}$ is that maximum height the trajectory will achieve while $height_{min}$ is the minimum safety height that provide a buffer zone should the aircraft overshoot at the end of the hopping maneuver.

Take Off Phase:

$$U = \frac{height_{min}}{t_{hold}} * t$$

Cruise Phase:

$$A = height_{max} - height_{min}$$

$$U = A * sin\left(\frac{2\pi}{2 * t_{cruise}} * (t - t_{hold})\right) + height_{min}$$

Landing Phase:

$$U = -\frac{height_{min}}{t_{hold}} * \left(t - (t_{hold} + t_{cruise})\right) + height_{min}$$

TARGET YAW DETERMINATION

3 METHODS OF TRANSLATION

There are generally three ways an aircraft could translate to its destination. The first involve the yaw direction consistently being lock toward the waypoint. The aircraft would then pitch to accelerate toward the waypoint.

The second method involves the yaw direction being lock to a certain direction, such as North. In that case, roll would be used to control the longitude and pitch would be used to control the latitude. If the defined direction is of some other angle, then a simple 1D yaw rotation matrix could be used to calculate the displacement components with respect to the aircraft.

The final method involves the aircraft facing a certain target at all time. This would be useful if the mission is to photograph a certain point of interests during flight to a destination. However, this would require a time varying yaw and changing rotation matrix. In the beginning of the flight, the aircraft might be pitching forward, but near the end, it might be pitching backward. Because of this, stable flight control might be difficult to achieve.

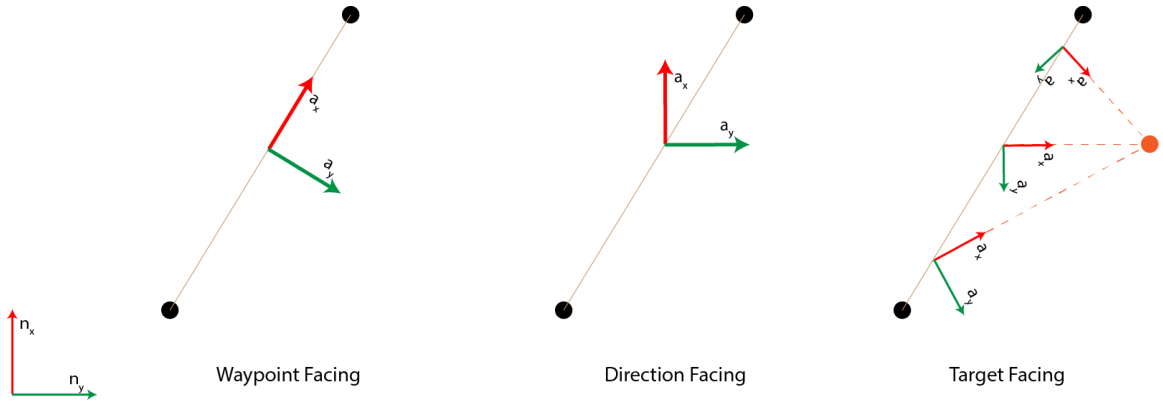


FIGURE 9-6: THREE METHODS OF MULTIROTOR TRANSLATION

Bearing Calculation

Trigonometry can be used to compute the reference yaw signal to the attitude controller. Note that the arctangent function has the following format $atan2(y, x)$ and that mod is the modulus function. The resulting bearing is the degree deviated from North. The following equations work in most cases. However, be must be taken to make sure that $dLon$ and $dLat$ is not overly small, or else there will be significant error in the bearing calculation.

$$dLon = lon_1 - lon_0$$

$$dLat = lon_1 - lon_0$$

$$yaw_t = mod(\ atan2(dLon, dLat), 2\pi\ ) * \frac{180}{\pi}$$

Waypoint Facing Method

With the waypoint facing method, the aircraft is always facing the target destination. This way, only pitch is necessary to complete the flight. Let's assume that lon and lat are the current aircraft position. For this method the differences in longitude and latitude are calculated with the following equations:

$$dLon = lon_t - lon$$
$$dLat = lat_t - lat$$

Direction Facing Method

The direction facing method is the simplest. The target yaw is simply predefined by the operator and doesn't require any special calculations. As with all the methods described here, the 1D yaw rotation matrix in the position controller will help the QuadX to figure out the correct roll and

pitch combinations to achieve the translation to the destination while facing at direction specified.

TARGET FACING METHOD

For the target facing method, the differences in longitude and latitude is calculated as follow. Assume that lon_f and lat_f are the localized position of the facing target.

$$dLon = lon_f - lon$$

$$dLat = lat_f - lat$$

Chapter 10: Simulation Modeling and Results

The QuadX flight system was simulated in Simulink before implementation. To construct the foundation of the simulation, a dynamics block was created using the equations derived in Chapter 2. In addition, The QuadX's GN&C system, flight mixer and motor thrust output were modeled. The simulation architecture can be found in Figure 10-1.

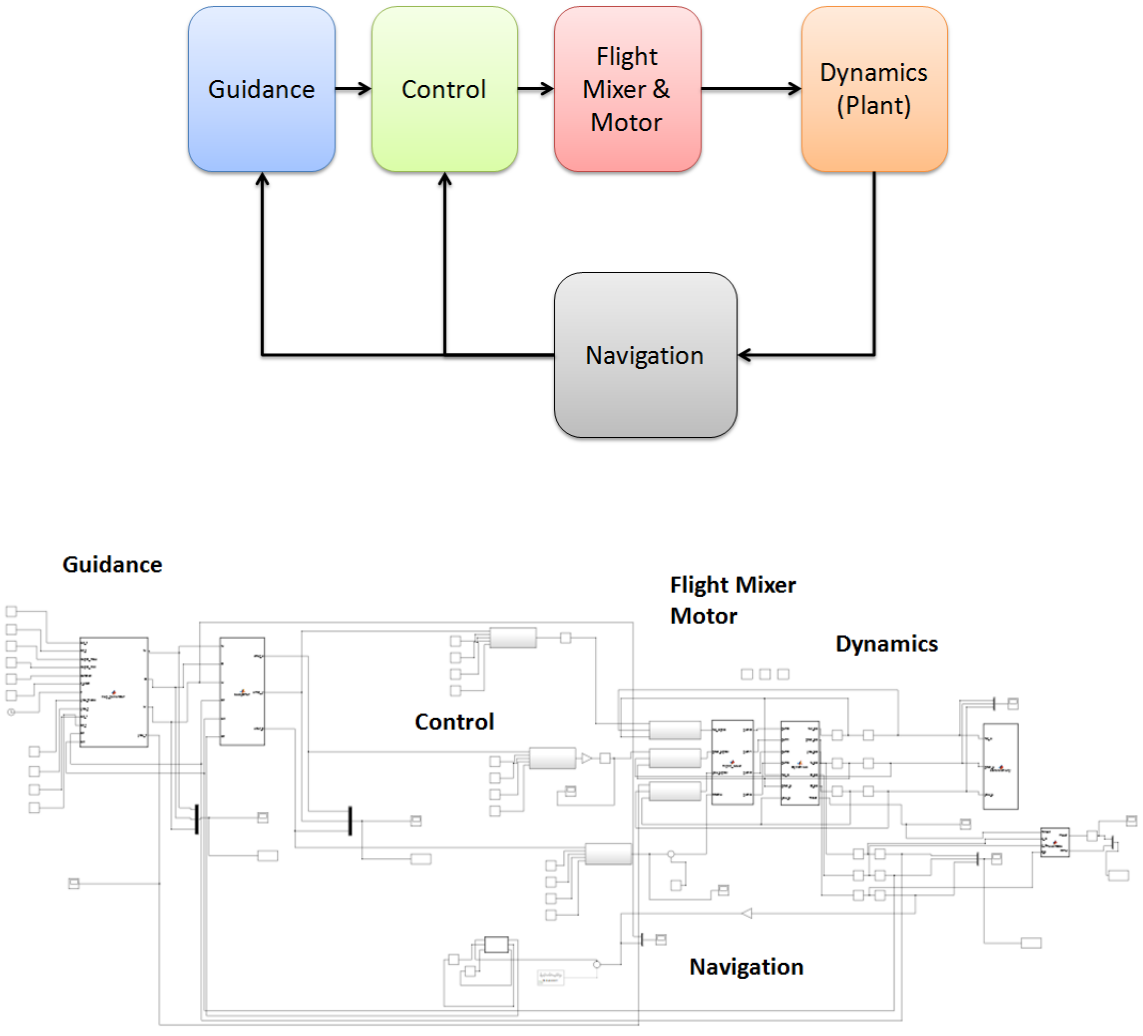


FIGURE 10-1: SPACE HOPPER SIMULATOR SIMULINK SIMULATION ARCHITECTURE

MOTOR SYSTEM IDENTIFICATION

A large portion of the simulation can be ported from codes already developed for the QuadX. This will verify that the software logic is functioning properly. The motor output and dynamics function block, however, required several physical parameters. The aircraft's mass and moment of inertia can be obtained from the 3D CAD model created in SolidWorks. However, the motor thrust and torque output must be modeled after experimental data. Hence, the NTM 28-36 750 KV motor went through the system identification process using a thrust stand in the aerospace system laboratory.

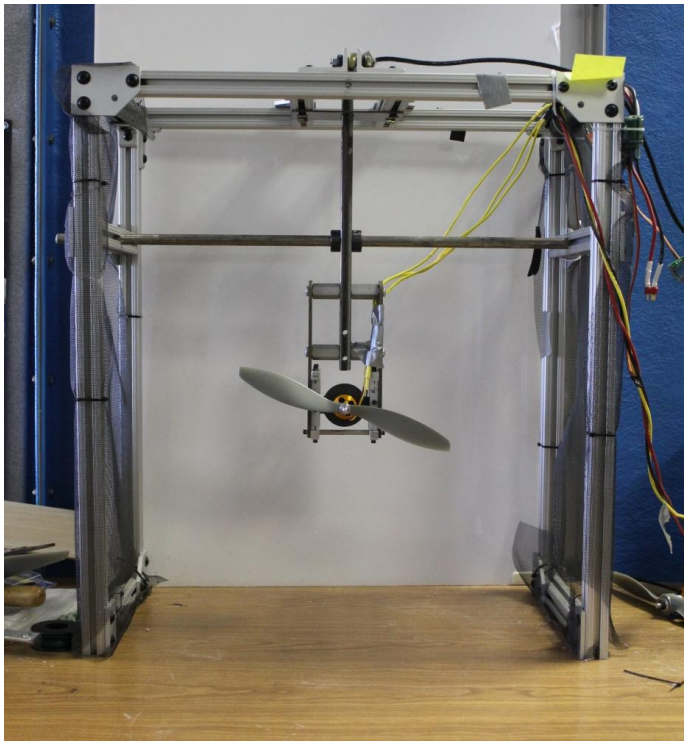


FIGURE 10-2: RC MOTOR THRUST STAND

The thrust stand, as shown in Figure 10-2, is set up using the lever system. A motor is attached to the motor mount at the bottom of the lever. At the other end, a force gauge is connected to a National Instruments data acquisition system to measure the force being applied to it through

the lever. Since the motor and the force gauge's distance away from the pivot point are slightly different, a simple ratio factor is required to calculate the thrust outputted by the motor.

THRUST STAND CALIBRATION

The thrust stand is calibrated by removing the thrust gauge and hanging various calibration weights on it. The data are plotted and a linear equation was obtained to convert the analog signal into thrust measured in gram-force. Measurements in gram-force will allow easier calculation in determining the thrust to weight ratio.

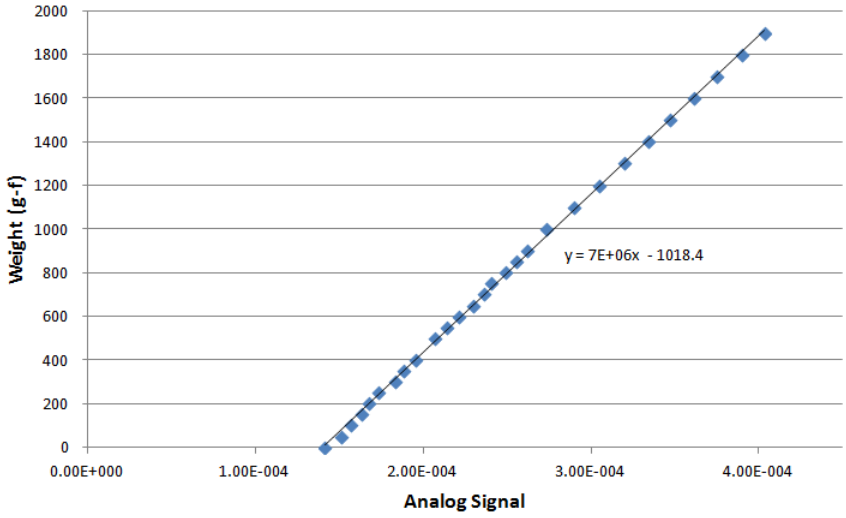


FIGURE 10-3: THRUST STAND CALIBRATION DATA

RAW THRUST DATA

Thrust testing was conducted for multiple motors the propellers combinations. The data for the NTM 28-36 750KV motor and a 12x4.5 propeller will be discussed here since that is the combination being used in the propulsion system.

The experiment was conducted in MATLAB in which the Pololu Micro Maestro 6 Ch USB Controller was used to generate PWM signal for the motor. On the other hand, the NI DAQ was used to collect analog signal generated from the force gauge with a sample time of 0.0006 sec.

The testing sequence was predefined in a MATLAB script. The motor is to go through a ramp command from minimum to maximum PWM pulse width. Then, the motor is commanded to go through a series of random step commands and finally decrease back to minimum throttle through a decreasing ramp command. The raw thrust data is reproduced in Figure 10-4.

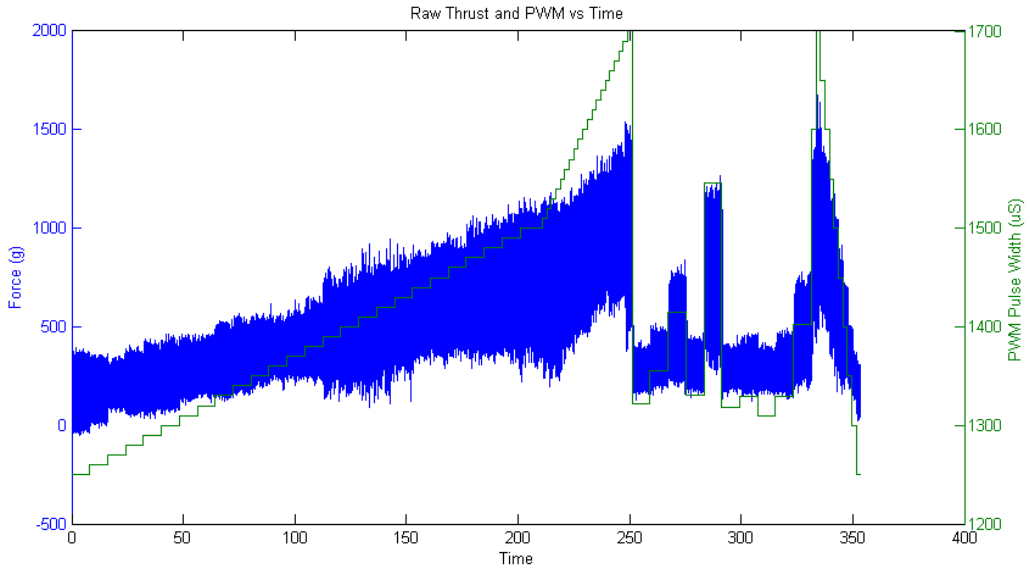


FIGURE 10-4: NTM 28-36 750KV WITH 12X4.7 PROP AT 15.8V

As seen in the chart above, there is a lot of noise in the measurement. This is most likely caused by vibrations from the loose parts in the set up. Several attempts were made to dampen the thrust stand, but it was no use. Therefore, digital filters were used to post process the data.

FILTERED THRUST DATA

In an attempt to eliminate the high frequency noise in the measured data, the raw thrust data were processed through Fast Fourier Transform and plotted in Figure 10-5. It can be seen that most data are resided in the left end of the graph while the right end contain data for the high frequency noise.

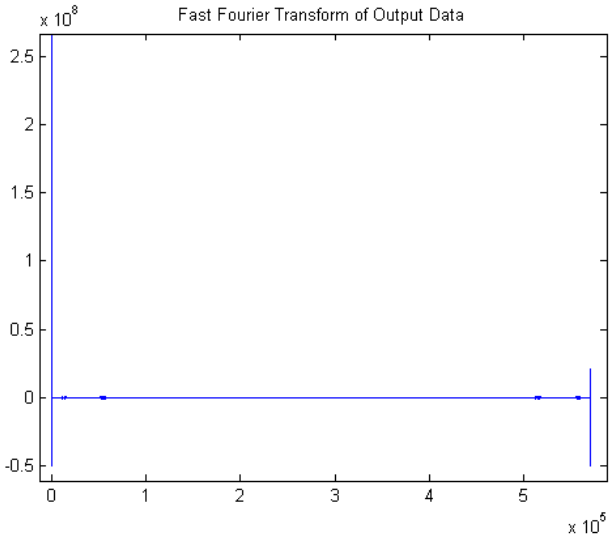


FIGURE 10-5: FAST FOURIER TRANSFORM OF RAW THRUST DATA

To eliminate the unwanted noise, a portion of the data was extracted from the left end of the FFT data and inverse Fourier transformed back in to time domain. The result was plotted in Figure 10-6. It is obvious that the thrust data improved drastically when compared with the raw thrust data in Figure 10-4. However, the max thrust output is now around 800, whereas the raw data show a max thrust of around 1500, which is considerably higher. The reason for this drop in amplitude is because the low pass filter eliminated all the higher frequency signal, including the actual thrust data that make up the amplitude.

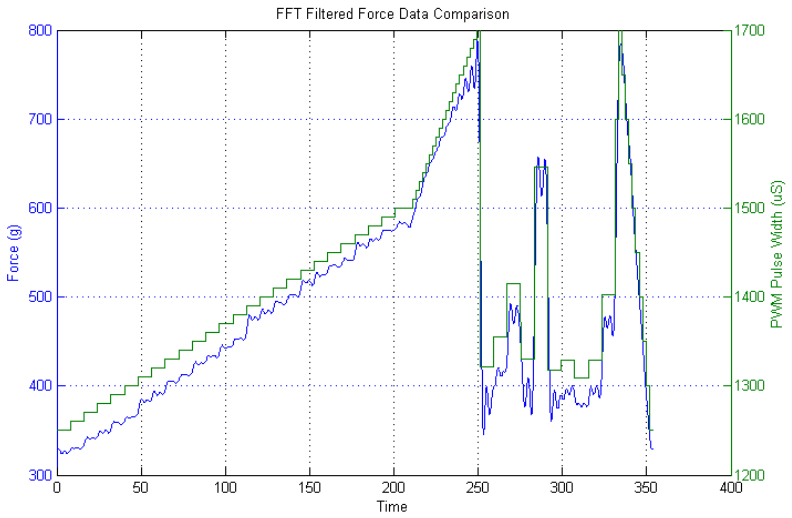


FIGURE 10-6: INVERSE FAST FOURIER TRANSFORM OF LOW FREQUENCY THRUST DATA

After that, a moving average filter of 10, 100, 500, 1000, 5000 terms are used. As seen in Figure 10-7, the higher the average terms, the smoother the signal. However, at the same time, this causes delay and too much smoothing blur the response differences between each PWM pulse commands. It is necessary to shift the data in order to have the timing line up correctly.

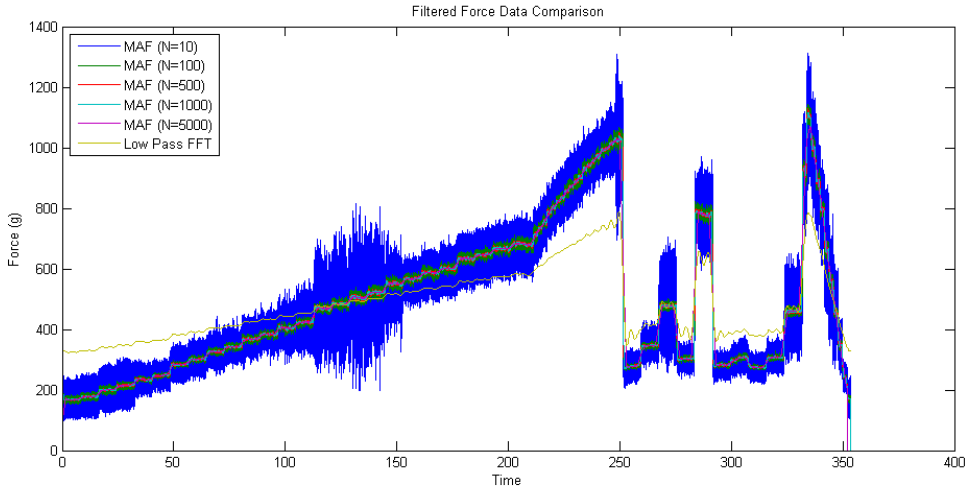


FIGURE 10-7: COMPARISON OF FILTERED THRUST DATA

Eventually one of the filtered results was chosen and is illustrated in Figure 10-8. As seen in the data, the PWM pulse width range from around 1150 to 1700 and corresponding to a thrust of 0 to 1050 g. This data is used to construct a simple PWM to thrust output for the simulation through linear interpolation.

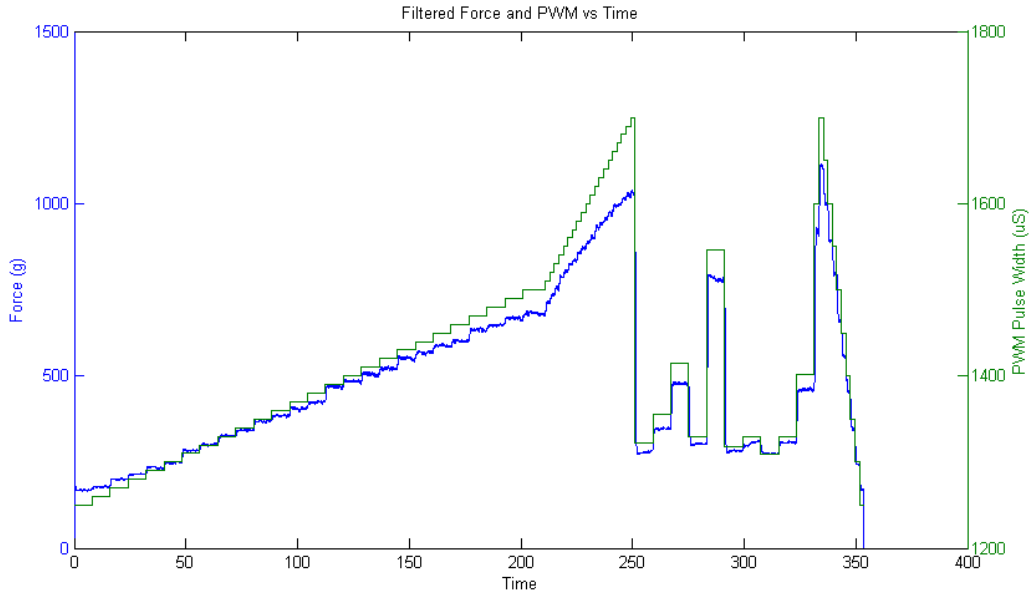


FIGURE 10-8: FILTERED THRUST DATA

After modeling the motor's thrust output, the torque output data is required. The motor is then rotated to its side and remounted onto the test rig. Using the same process, the motor's torque was measured as force from the lever. This "torque force" is required to be scaled before using in the simulation. The result for the torque output is shown in Figure 10-9.

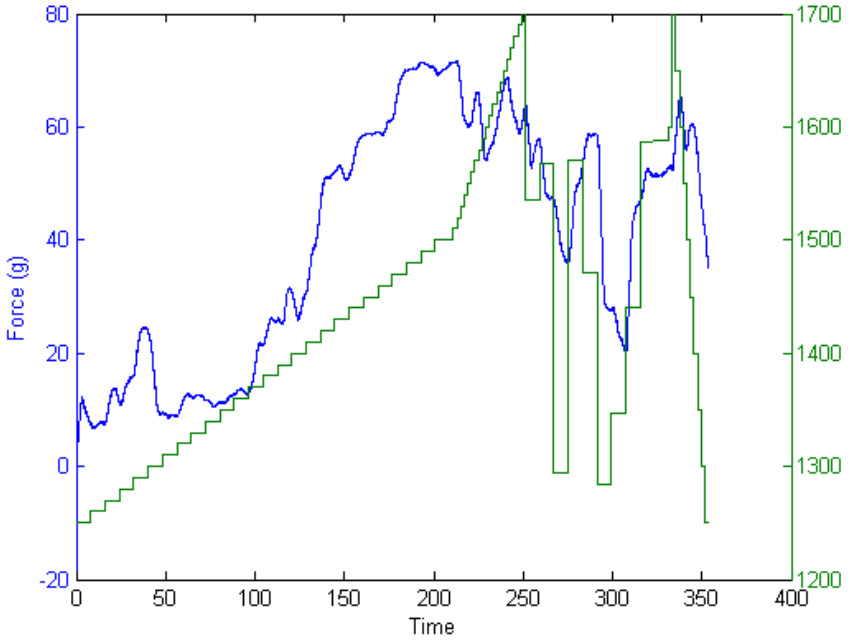


FIGURE 10-9: FILTERED TORQUE DATA MEASURED AS FORCE

As seen in the chart, the torque output is not linear and is generally proportional. The min to max throttle yield a torque output from 0 to around 70 grams. To simplify the simulation, the torque data is treated as linear and the conversion from PWM pulse width to torque is once again based on linear interpolation.

SYSTEM IDENTIFICATION TOOLBOX

The linear interpolation method currently used in the simulation is simple and effective. However, it is not very accurate and does not take into account of physical behavior such as the motor and propeller's delay in thrust response. In an attempt to create a more accurate motor model, the MATLAB system identification toolbox was used to identify a transfer using one set experimental data and verify with another set of experimental data.

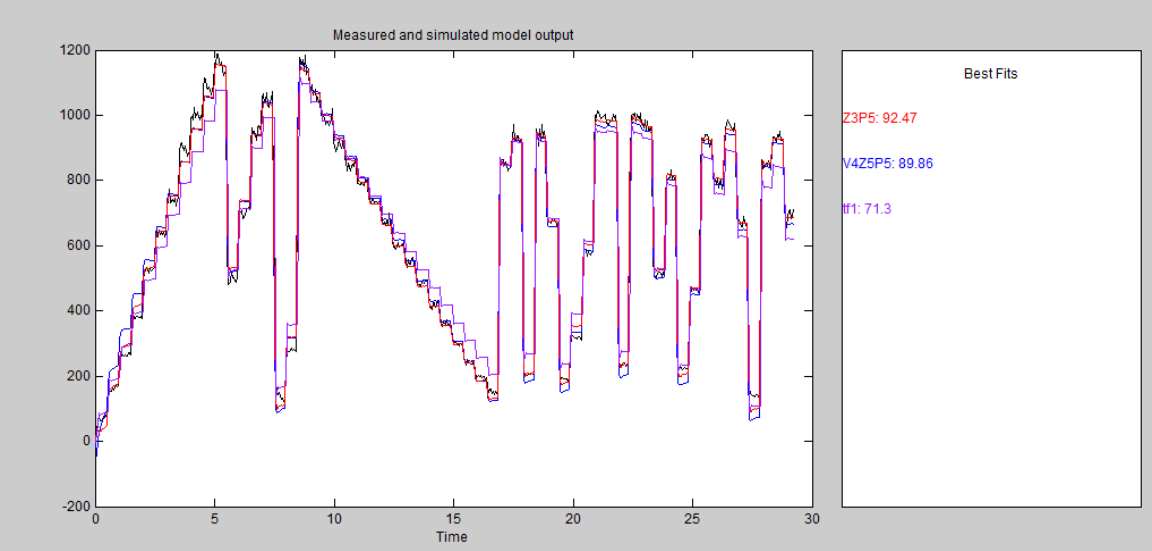


FIGURE 10-10: SYSTEM IDENTIFICATION TOOLBOX VERIFICATION

Figure 10-10 shows a comparison of various models identified by MATLAB. It seems that a 3 zeros 5 poles transfer function best fit the system with a 92.47% fit. Due to time constraint and that a high fidelity model is not absolutely necessary for logic testing, investigation for modeling the motor was halt.

POWER CONSUMPTION ESTIMATION

For effective guidance, the energy consumption of the hopping maneuver should be optimized in the future. The thrust and power equations (3.1 and 3.2) discussed in the propulsion section of Chapter 3 can be used to solve for energy with parameters gathered from the simulation.

$$V_0 = \sqrt{\dot{N}^2 + \dot{E}^2 + \dot{D}^2}$$

$$4.23333E - 4 * pitch * RPM^2 - V_0 * RPM - \frac{Thrust * \sqrt{pitch}}{4.392399E - 8 * d^{3.5}} = 0$$

The RPM of the motor is then solved by using the quadratic equation. With that RPM, the power consumption at that iteration can be solved using the propeller's prop constant and power factor as discussed previously.

$$Power = Prop\ Constant * \frac{RPM^{Power\ Factor}}{1000}$$

To obtain energy over time, the power is integrated.

$$Energy_{new} = Power * dt + Energy_{old}$$

SIMULATION PARAMETERS

The simulation is constructed such that the aircraft modules are simulated as discrete function block. On the other hand, the dynamics of the system is continuous. The sample frequency for each block can be found in Table 10-1. In addition, the gains used in the control system can be found in Table 10-2.

	Guidance	Navigation	Position Control	Attitude Control	Dynamics
X Axis	200 Hz	Inherit	5 Hz	120 Hz	Continuous
Y Axis	200 Hz	Inherit	5 Hz	120 Hz	Continuous
Z Axis	200 Hz	Inherit	20 Hz	120 Hz	Continuous

TABLE 10-1: SIMULATION MODULES SAMPLE FREQUENCY

	Position Control	Attitude Control	Attitude Rate Control
X Axis	10/0.1/50/20	15/0.2/0/10	1/0/0.1/20
Y Axis	10/0.1/50/20	15/0.2/0/10	1/0/0.1/20
Z Axis	150/1/1500/10	5/0.3/0/50	7/0/2/20

TABLE 10-2: SIMULATION PID CONTROLLER GAINS ($K_p/K_i/K_d/\ Integral\ Limit$)

SIMULATION RESULT

WAYPOINT FACING HOPPING MANEUVER

This simulation was conducted using the waypoint facing translation method. The guidance trajectory generator parameters are shown in Table 10-3. The total duration of the simulation is 30 seconds and the total distance traveled will be around 40.3 meters

Target Latitude	Target Longitude	Max Height	Min Height	Hopping Duration	Landing Duration	Yaw Mode
35m	20m	15m	2m	20s	5s	Waypoint

TABLE 10-3: SIMULATION GUIDANCE TRAJECTORY PARAMETERS

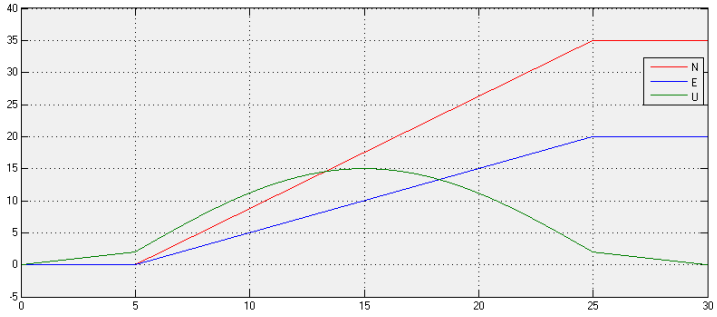


FIGURE 10-11: WAYPOINT FACING HOPPING MANEUVER TRAJECTORY (M)

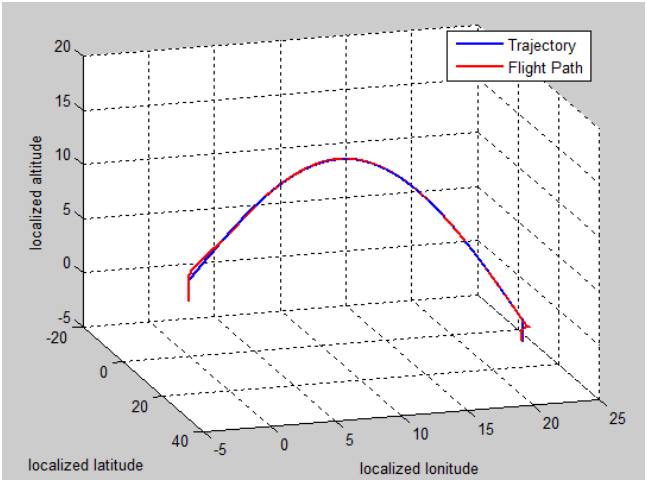


FIGURE 10-12: 3D COMPARISON OF TRAJECTORY AND SIMULATED FLIGHT PATH (M)

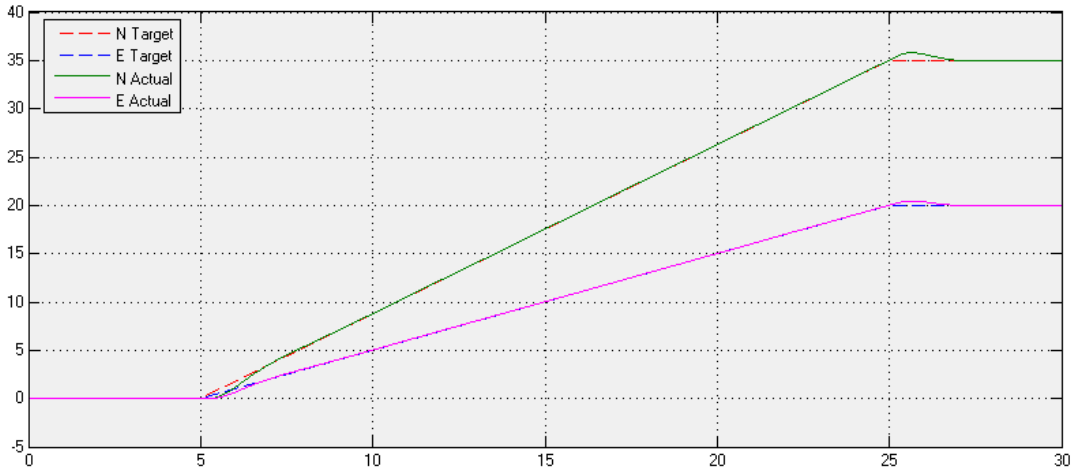


FIGURE 10-13: LATITUDE / LONGITUDE TRAJECTORY AND SIMULATED FLIGHT PATH (M)

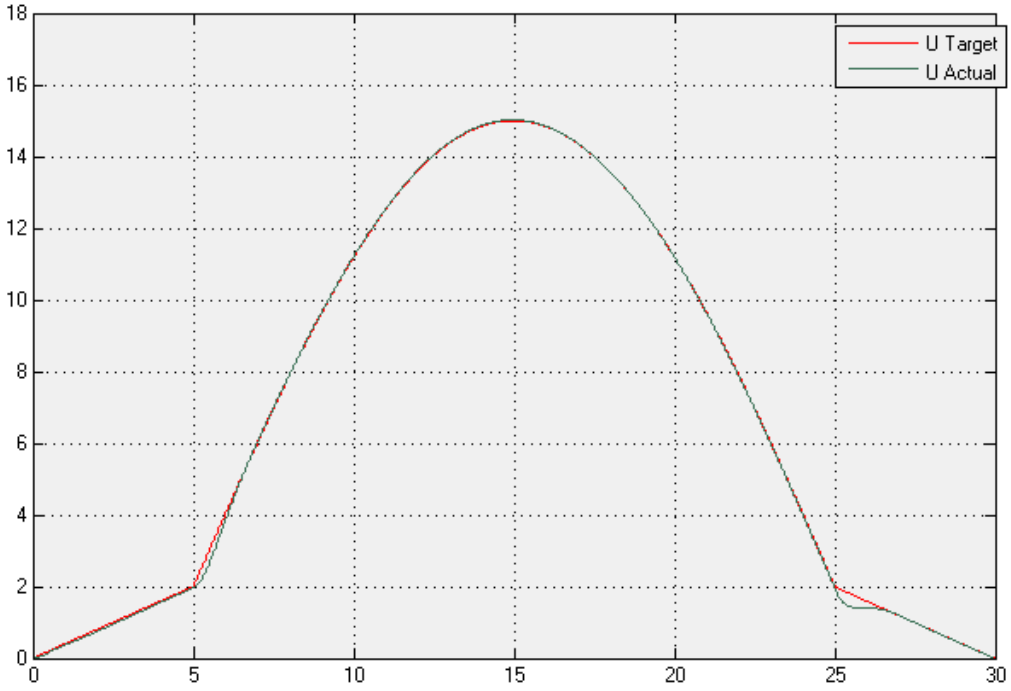


FIGURE 10-14: ALTITUDE TRAJECTORY AND SIMULATED FLIGHT PATH (M)

From the figures above, it can be seen that the position and attitude controller work very well in achieving the target trajectory with about 0.5 m of overshoot in altitude during the end of the hopping maneuver. This is not a problem thanks to the buffer zone allocated by the guidance.

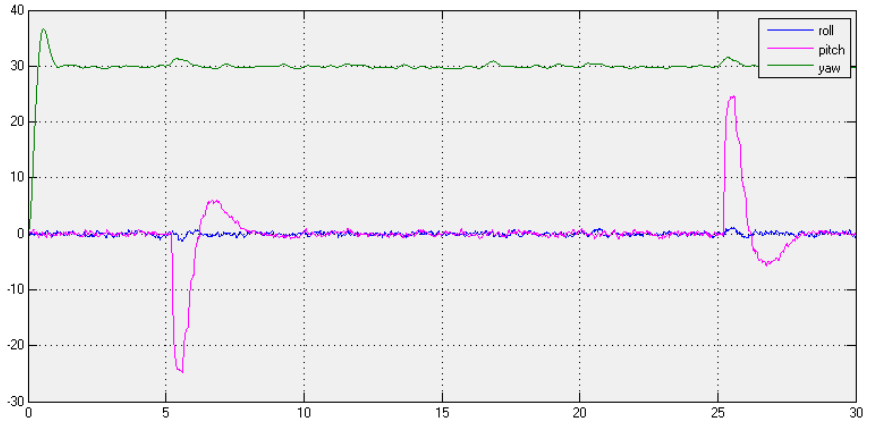


FIGURE 10-15: SIMULATED ATTITUDE (DEGREES)

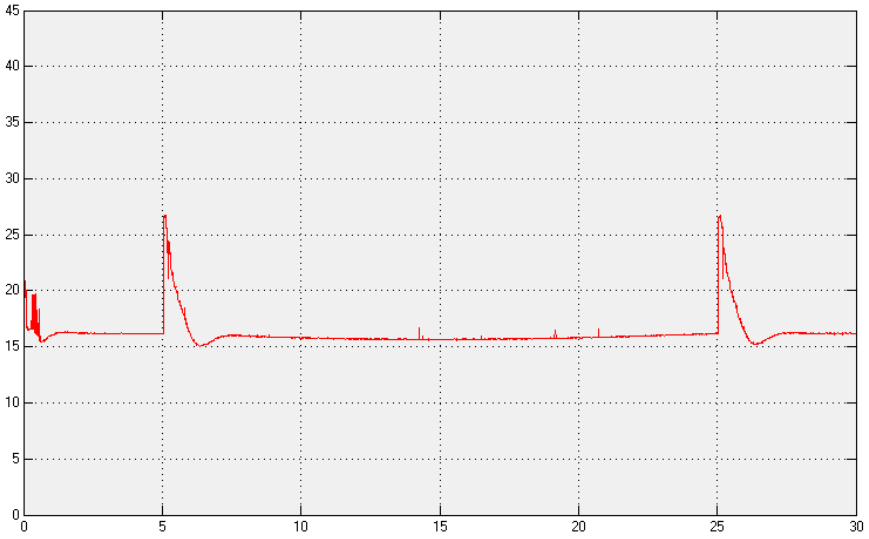


FIGURE 10-16: SIMULATED THRUST OUTPUT (N)

Figure 10-15 and Figure 10-16 reflect the attitude and thrust required to achieve the hopping trajectory. Since the initial hop and landing phase take place at $t = 5$ and $t = 25$, it can be seen that the attitude and the thrust has the most rapid changes there. The translation mode for this simulation is waypoint facing. Therefore, only the pitch is used to achieve a change in horizontal velocity, peaking at around $\pm 25°$. On the other hand, the thrust is in charge of the altitude. To

initiate the parabolic hop and to cancel out the vertical and horizontal velocity at landing, the motors used about 27 N of thrust. That is 63% of the total available thrust. In order to maintain the parabolic flight path at a near hovering status, the motors remain generating 16 N of thrust throughout the flight.

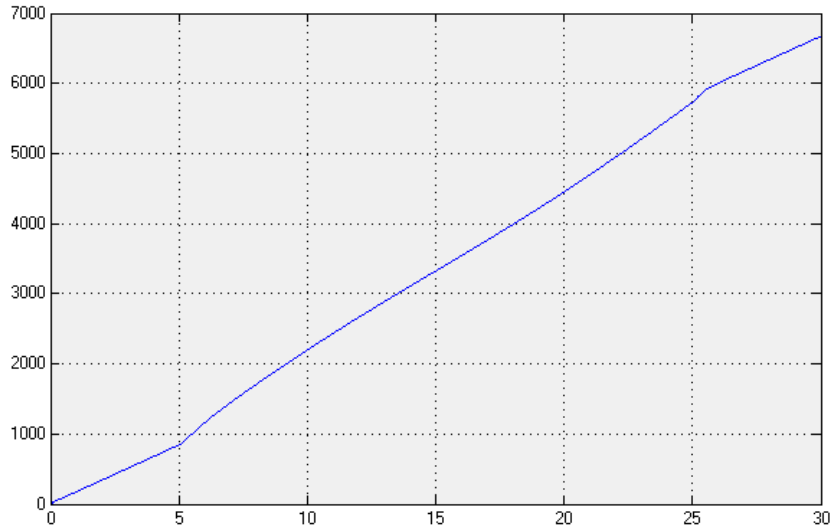


FIGURE 10-17: SIMULATED ENERGY CONSUMPTION (J)

As seen in the energy consumption chart in Figure 10-17, the takeoff and landing phase take the least amount of energy. Then, the energy consumption rate increase as the aircraft enters the hopping maneuver. The total energy consumption is around 6600 Joules.

WAYPOINT FACING HOPPING MANEUVER (AGGRESSIVE TRAJECTORY)

The entire flight duration for the previous simulation is 30 seconds. Lots energy is spent in maintaining the flight altitude. To reduce energy consumption, a more aggressive trajectory can be generated by reducing the hopping and landing duration by half. The new guidance parameters and results are shown below.

Target Latitude	Target Longitude	Max Height	Min Height	Hopping Duration	Landing Duration	Yaw Mode
35m	20m	15m	2m	10s	2.5s	Waypoint

TABLE 10-4: AGGRESSIVE SIMULATION GUIDANCE TRAJECTORY PARAMETERS

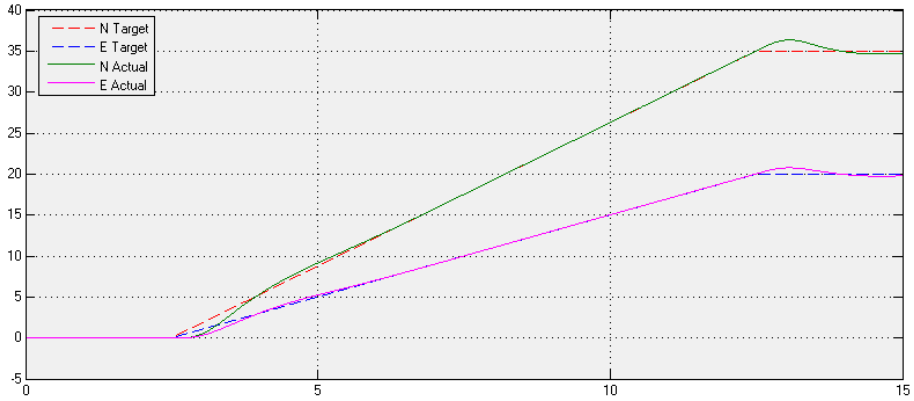


FIGURE 10-18: AGGRESSIVE LATITUDE / LONGITUDE TRAJECTORY AND SIMULATED FLIGHT PATH (M)

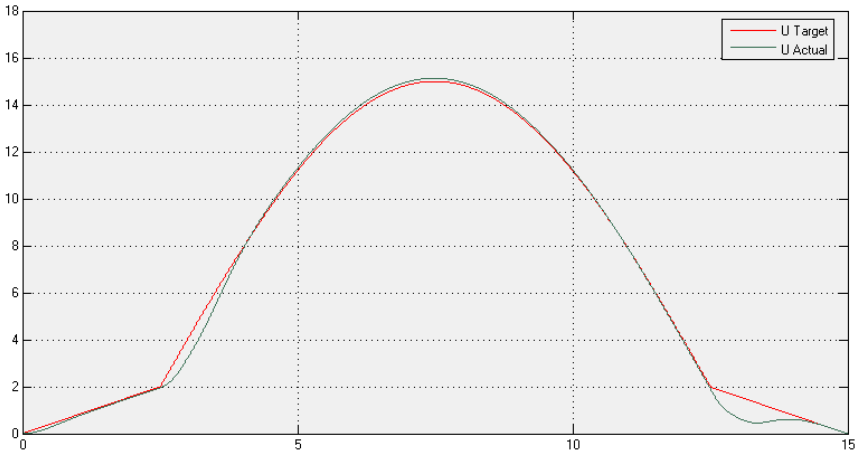


FIGURE 10-19: AGGRESSIVE ALTITUDE TRAJECTORY AND SIMULATED FLIGHT PATH (M)

With the new trajectory, a higher horizontal velocity is necessary. Therefore, as seen in Figure 10-15 and Figure 10-16, a much higher pitching angle and thrust is required to keep up with trajectory. The peak angle is now at around $\pm 45°$ and the peak thrust output is $37\ N$. That is around 86% of the total available thrust. It can be seen that because the aircraft dedicated more of the thrust to the horizontal velocity, there is now an overshoot of around 1.5 m into landing.

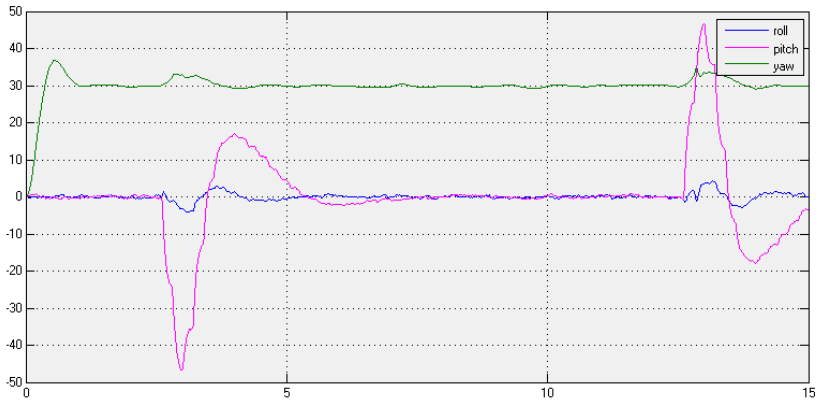


FIGURE 10-20: AGGRESSIVE SIMULATED ATTITUDE (DEGREES)

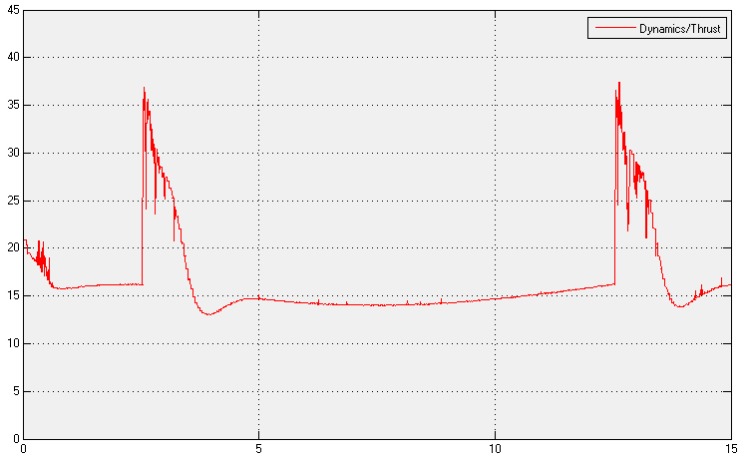


FIGURE 10-21: AGGRESSIVE SIMULATED THRUST OUTPUT (N)

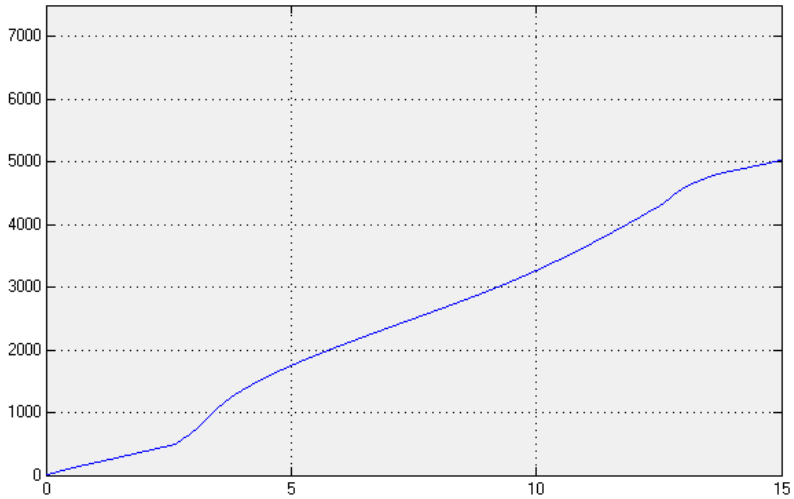


FIGURE 10-22: AGGRESSIVE SIMULATED ENERGY CONSUMPTION (J)

Due to the reduced time in flight, less thrust is necessary to keep the aircraft in the air hovering, as the motion switch from ascending to ascending rather quickly. The thrust to maintain the aircraft in the air is now reduced to around 15N. When analyzing Figure 10-17, it is clear that the energy consumption rate is increased for $t = 2.5$ and $t = 12.5$, but reduced overall everywhere else. The total energy consumption reduced from 6600 Joules to 5000 Joules. This is roughly a 25% drop in energy consumption and it is possible to get to the destination in half the duration. The tradeoff is that the flight overall less stable and that the actuators are being stressed.

TARGET FACING HOPPING MANEUVER

The Google Lunar XPRIZE competition required that the spacecraft move a distance of over 500 meters. Therefore, a hop was simulated from Packard Lab to Zoellner Arts Center while facing the Fairchild-Martindale Library.

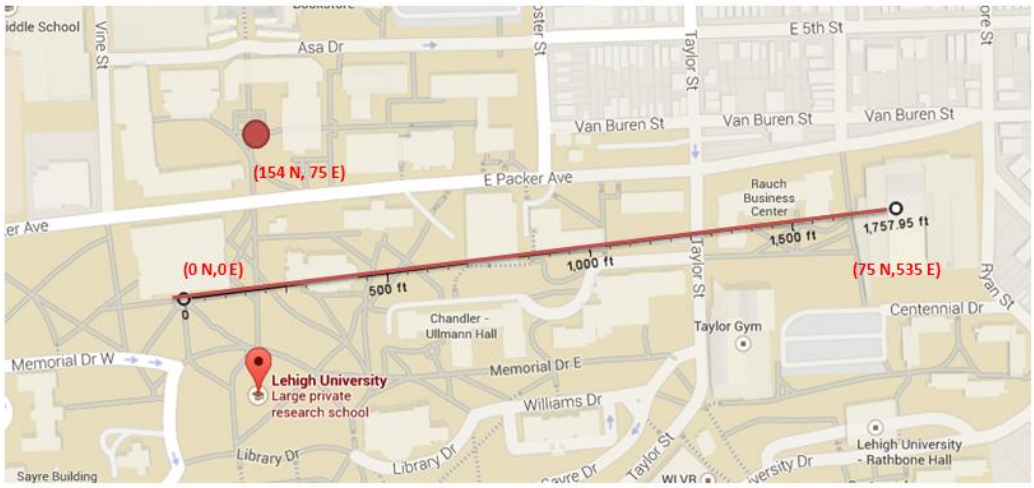


FIGURE 10-23: HOPPING ORIGIN, DESTINATION AND FACING TARGET

Target Latitude	Target Longitude	Max Height	Min Height	Hopping Duration	Landing Duration	Yaw Mode
75m	535m	30m	5m	100s	5s	Target (75,154)

TABLE 10-5: TARGET FACING HOP GUIDANCE TRAJECTORY PARAMETERS

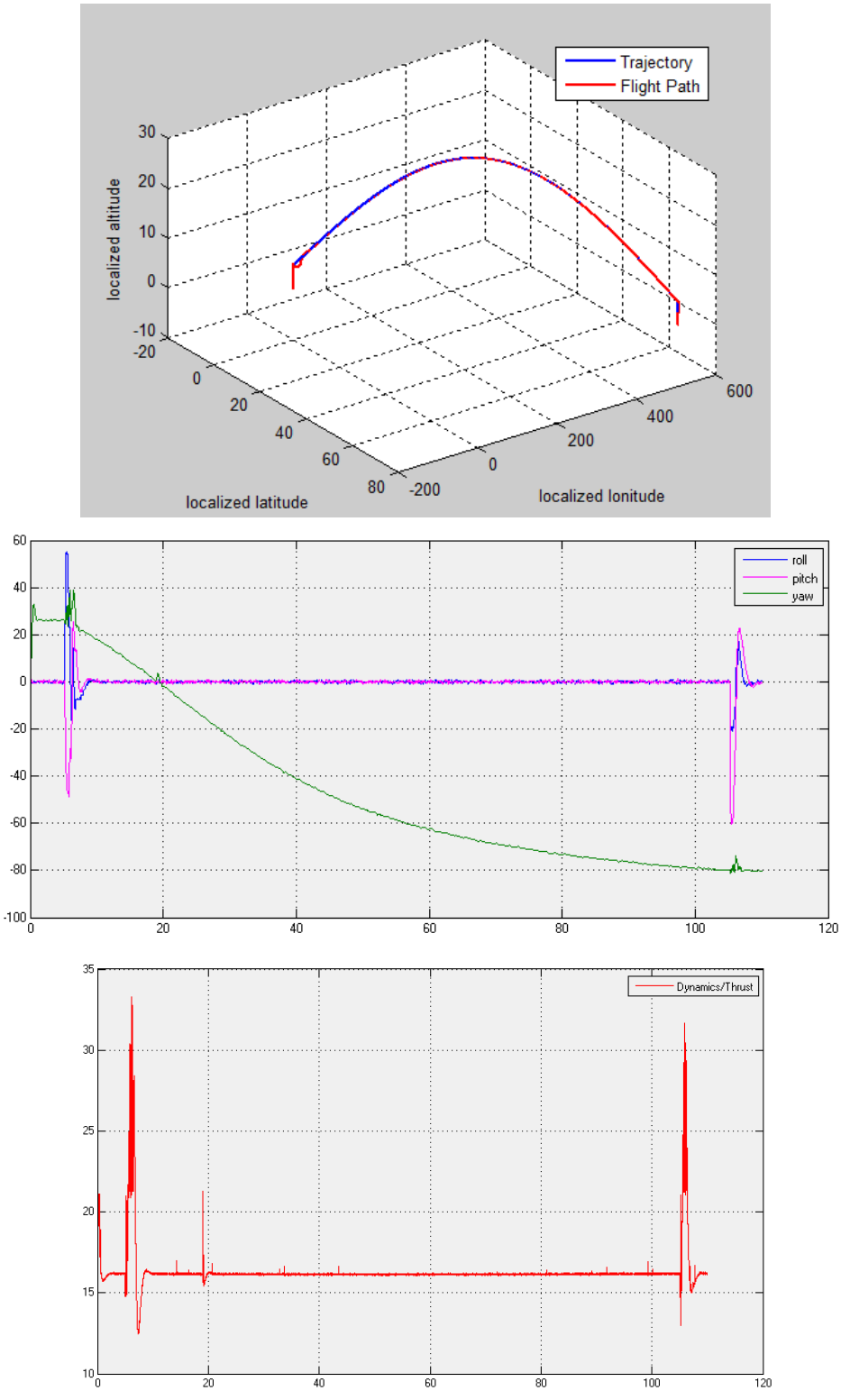


FIGURE 10-24: TARGET FACING HOP FLIGHT PATH, ATTITUDE AND THRUST OUTPUT RESULT

As seen in the simulation results from Figure 10-24, completing this hopping maneuver of around 537 meters in 100 seconds is actually relatively aggressive compare to normal flight. On average, the QuadX have to travel around 5 meters per seconds, or 12 miles per hour. To achieve this, the motors are required to output a maximum of 33 N of thrust.

Since the aircraft is set to target facing mode, the quadcopter is constantly facing the library. Hence, the yaw is constantly changing as shown in the attitude graph. The angle changes most rapidly when the aircraft crosses the target. Because of the changing yaw, the original maneuver to initiate the hop is consisted of a mixture of roll and pitch. During the landing phase, the maneuver is mostly pitch since the target is facing toward the west.

In summary, the cascade PID control system is able to handle both large and short distance maneuver without modifications to the gain as long as the differences is not too great and that the trajectory is not too aggressive. The flight performance is also very good and has some degrees of noise rejection.

The gains used in this simulation are not applicable to real life implementation due to several assumptions to simplify the modeling process. For example, the model does not take into account of atmospheric drag. In addition, the derivative gain for altitude controller will most likely cause problems when tuned that high from noise. However, the simulation serves the purpose of proving that the concept logic will work. In the future, for simulation in the lunar environment, the gravity in the dynamics block can be adjusted to $1.622\ m/s^2$. The gain will need to be retuned, but the rest of the system should remain mostly the same.

Chapter 11: SINGLE AXIS MULTIROTOR BALANCER

HARDWARE AND ELECTRONICS SETUP

In order to choose the correct control law for the space hopper simulator, a single axis multirotor balancer was designed and constructed. As shown in Figure 11-1, the balancer is consisted of an aluminum beam 18" in length. Two brushless motors are mounted at the ends of the beam and a connecting rod is mounted at the center of the beam to provide a pivot point. The connecting rod is connected to bearings inside a block that's mounted to a table in the lab. Collars are used to lock the rod in the axial direction while allowing rotation motion. It is calculated in SolidWorks that the principal inertia for the rotating portion of the balancer is $0.0021\ kg \cdot m^2$.

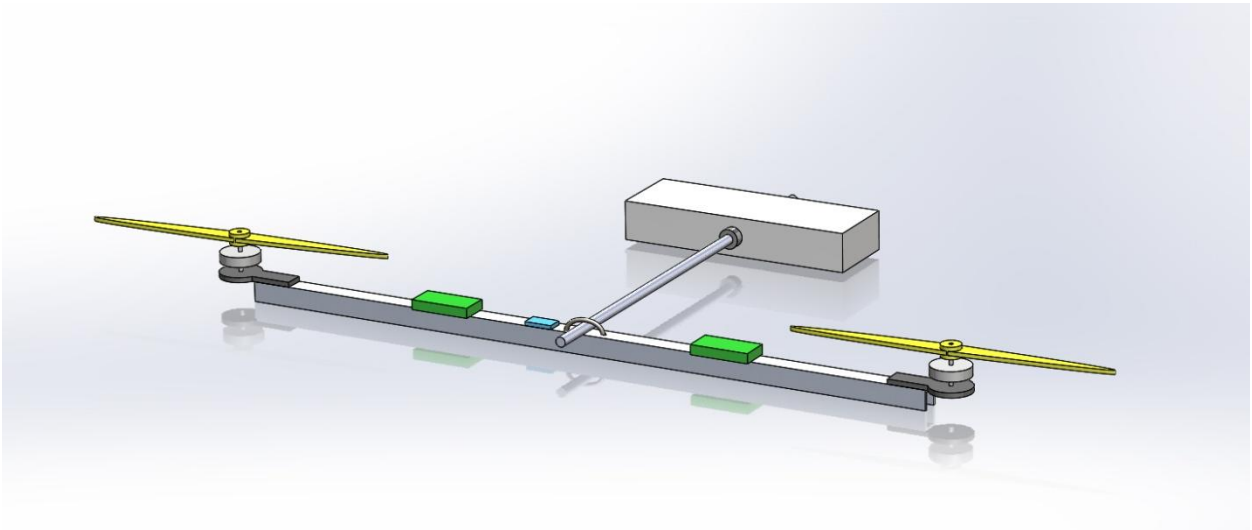


FIGURE 11-1: SINGLE AXIS MULTIROTOR BALANCER

The electronics for the test rig are harvested from an existing quadcopter. Therefore, the hardware is a bit different than that used in the hopper simulator. In the balancer, the LazerToyz Blue Wonder 1000 kV 24g brushless motors are used. Two Turnigy Multistar 15Amp ESC are

used to drive the motors. A MPU-6050 IMU is used to sense attitude data and an Arduino UNO is used to control the test rig. An adjustable DC power supply is used to power the entire system.

Software development for the balancer is relatively easy thanks to the software modules that have already developed for the hopper. Most of the modules can be integrated with a few simple modifications. The only challenge is interfacing the InvenSense MPU-6050 IMU considered the IMU module is designed for the CH Robotics UM7.

The MPU-6050 is a popular low cost IMU that's widely used. As opposed to the UM7's $130 price point, the MPU-6050 only cost around $3. It interfaces with microcontroller via I2C interface and contain a Digital Motion Processor to calculate MotionFusion data. The hardware seems to be an excellent choice considers its price and hardware specification. However, community support in using the Digital Motion Processor is almost non-existence. Hence most non-professionals can only retrieve the raw gyro and accelerometer data. Sensor fusion algorithm will have to be processed in the microcontroller, which add significant processing load. Luckily, the online open source community reverse engineered and released a comprehensive library to access the Digital Motion Processor.

In order to optimize the performance of the Arduino UNO, Serial transmission is set at a speed of 115200 bauds per second. The data logging is set at a sample rate of 10 Hz to allocate the remainder processing bandwidth to the IMU, Controller, Flight Mixer and Motor modules. A heavily simplified version of the Communication module is implemented to compensate for the UNO's lower processing speed and available memory. Logging data at such a low rate might cause aliasing issues. However, high data logging rate affect system performance, hence,

recorded data will not be accurate. Therefore, a balance point must be established. Ultimately, this yield a control loop rate of 70 Hz.

PID Controller Tuning

To start off, a modified attitude PID controller as described in Chapter 7 was implemented at 70 Hz with the integral limit capped at ± 500. In addition, a moving average filter is implemented on the error derivative calculation to filter out amplification of noise.

The PID controller is tuned by adjusting the K_p term until oscillation starts. Then, the K_d term is tuned to dampen the oscillation as much as possible. Then the K_i term is used to reduce steady state error.

While tuning, various tests are conducted to gauge system performance. It seems that increase K_d is not very effective at reducing oscillation in comparison to reducing K_p. In fact, when K_d is increased past a certain point, the oscillation effect often worsen. However, it is observed that before the stability threshold, increasing K_d also increase the amount the motors "fight back" when the balancer is being hit by a rod. This allows the balancer to return to neutral position faster. When reducing K_p, the oscillation effect is reduced. However, rise time to step command suffers. Multiple set of gains were tuned before arriving at a set that optimize stability while taking into consideration of other parameters like overshoot, rise time, settling time and steady state error. The PID gains are as follow:

$$K_p = 0.3 \quad K_i = 0.02 \quad K_d = 13.8$$

COMPARISON BETWEEN SIMULATION AND PRELIMINARY EXPERIMENTAL RESULTS

In an attempt to simplify the tuning process, a Simulink model was designed. Using data acquired from the thrust stand, linearized equations of motion and mass properties calculated in SolidWorks. A step command test of 10° was conducted both computationally using Simulink and experimentally with the balancer. The gains in both sides were set to be the same with no derivative filter. Initially, the results were completely off. However, after adjust parameters such as initial conditions, sample time and time delay, the results became more similar as shown in Figure 11-2 and Figure 11-3.

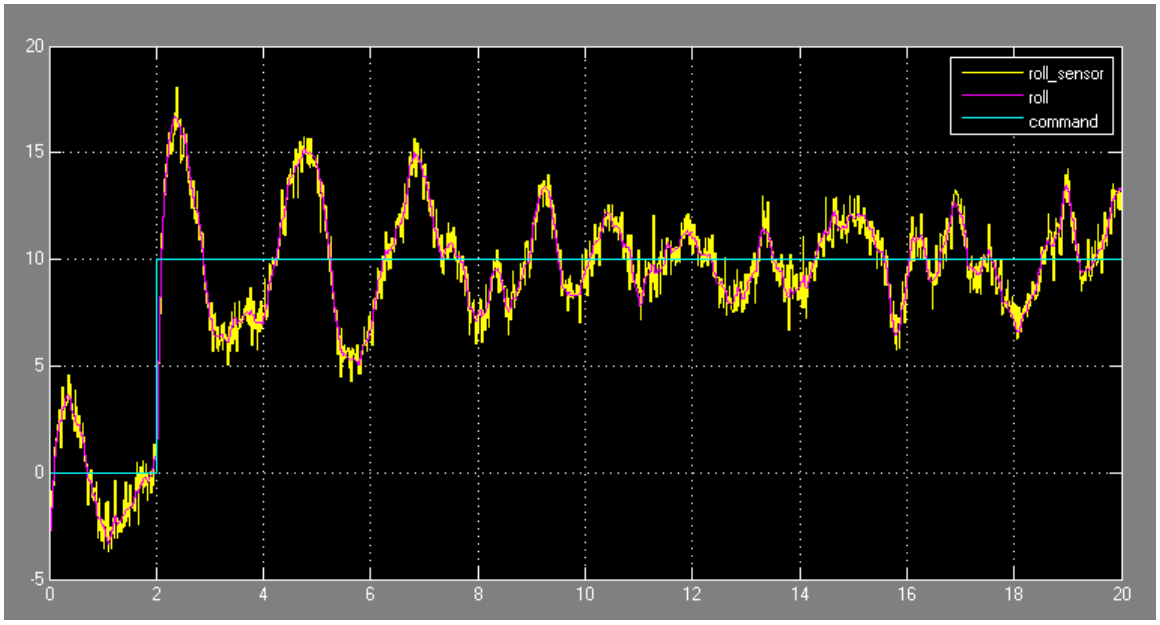


FIGURE 11-2: SIMULATED PID CONTROLLER STEP RESPONSE

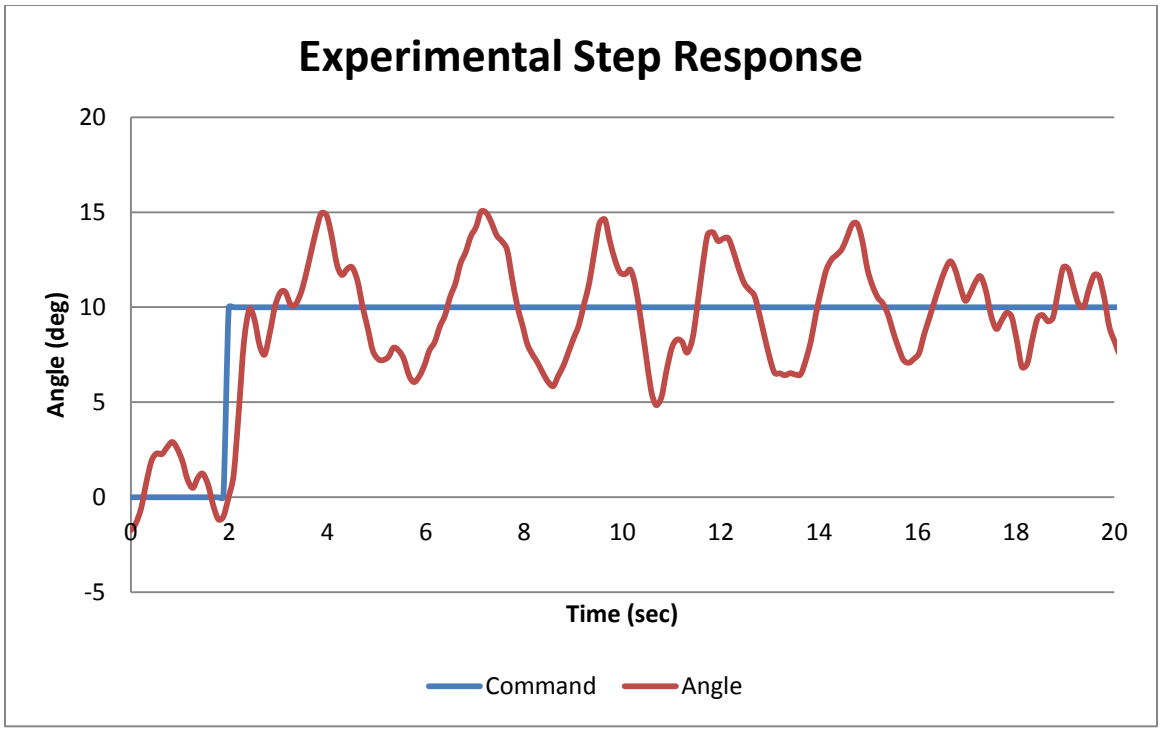


FIGURE 11-3: EXPERIMENTAL PID CONTROLLER STEP RESPONSE

Figure 11-2 illustrates the step response calculated in simulation while Figure 11-3 illustrated data collected experimentally. As seen the both plots, the angle oscillate between $5°$ and $15°$ after the step command to $10°$. The rise time and oscillation frequency of both are similar, however, it that the simulation demonstrates faster performance. The rise time in simulation is almost instantaneous while experimentally, it takes about 0.5 seconds. The oscillation period in simulation is around 2 seconds while experimentally, it is around 3 seconds.

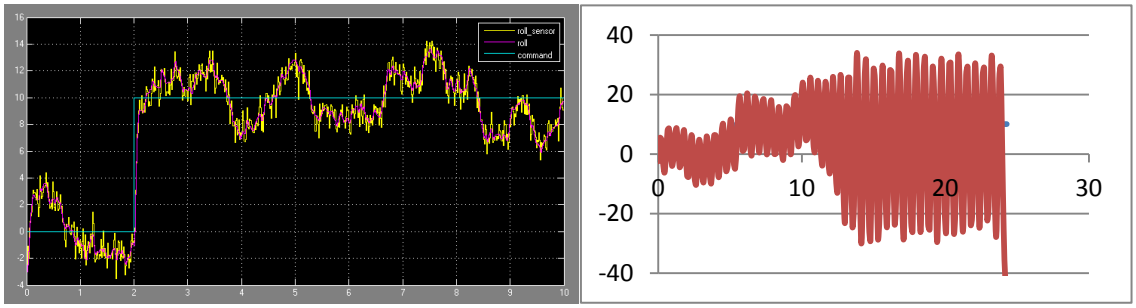


FIGURE 11-4: SIMULATION AND EXPERIMENTAL STEP RESPONSE WITH INCREASED K_D

To further verify that the simulation model is correct, K_d is doubled and step responses are once again collected. The results are illustrated in Figure 11-4. As seen in the plots, Simulink shows that the system should behave better with oscillation amplitude decreased. However, in reality, the derivate term actually introduce even more oscillation into the system. The system ended up becoming unstable and oscillates between $\pm$ 30 °.

This demonstration shows that all models are wrong to a certain degree. In order to have an accurate enough model for offline tuning, careful dynamics, geometric, electronic and system analysis must be conducted in order to take into account of the various observable and unobservable parameters. While offline tuning is almost a must for spacecraft, for a relatively simple and inexpensive small team project such as the hopper simulator, it is probably more feasible to use simulation as a tool for concept proofing while moving into the experimental stage as soon as possible to make real progress.

PID Controller Experimental Results

After tuning the PID controller with a set of gains optimized for stability, the impulse, ramp and step response tests were conducted. In addition, several tests were conducted to compare the effectiveness of the moving average derivative filter with 2 and 3 terms against a controller with no derivative filter.

During the impulse test, one end of the balancer was strike two times with a plastic rod to impulse the system. The first strike was a hard strike aim to bring the balancer to a peak angle of around 40 to 50 degrees. The second strike was a soft strike to bring the balancer at an angle less than the first impulse. Impulse response for the non-filtered controller is plotted below.

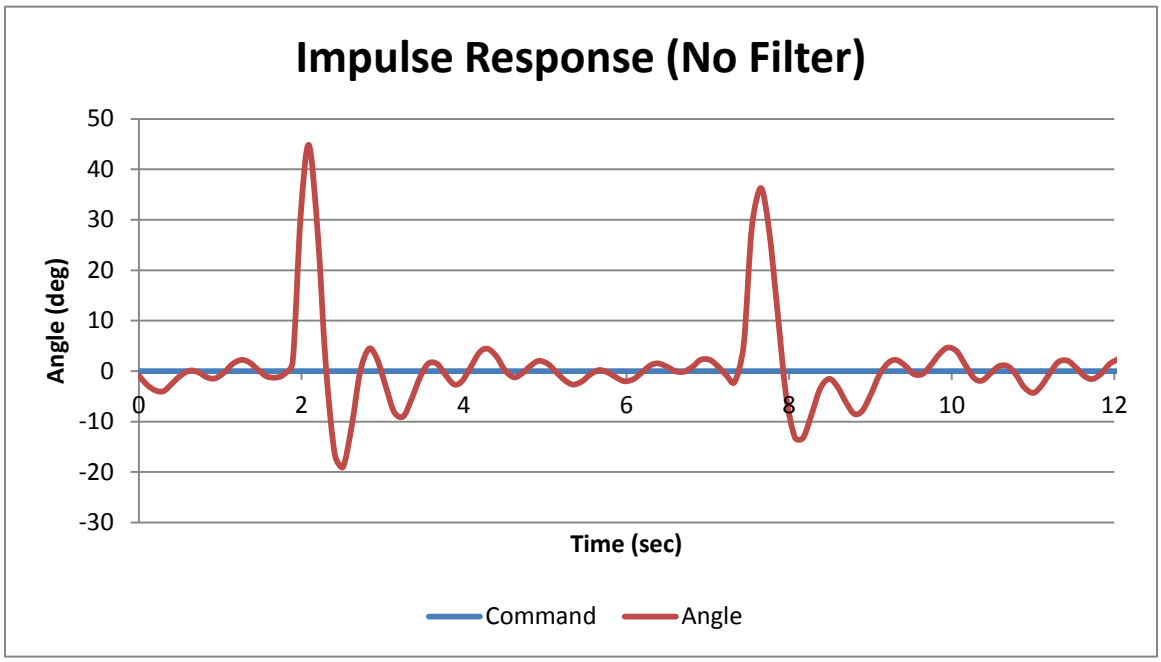


FIGURE 11-5: IMPULSE RESPONSE OF PID CONTROLLER WITH NO DERIVATIVE FILTER

As shown in Figure 11-5, the PID controller was able to hold the balancer angle at around 0 degree relatively stable. Throughout the experiment, the system is a bit oscillatory with an amplitude of less than 5 degrees and a period of around 0.7 second. It can be seen that after the balancer was impulse to an angle of 45 degrees, the controller drive the system back to the nominal position after around 1.5 seconds with an overshoot of -20 degrees. The overshoot in the second impulse response is a bit less at around -12 degrees, but again take roughly 1.5 seconds to settle.

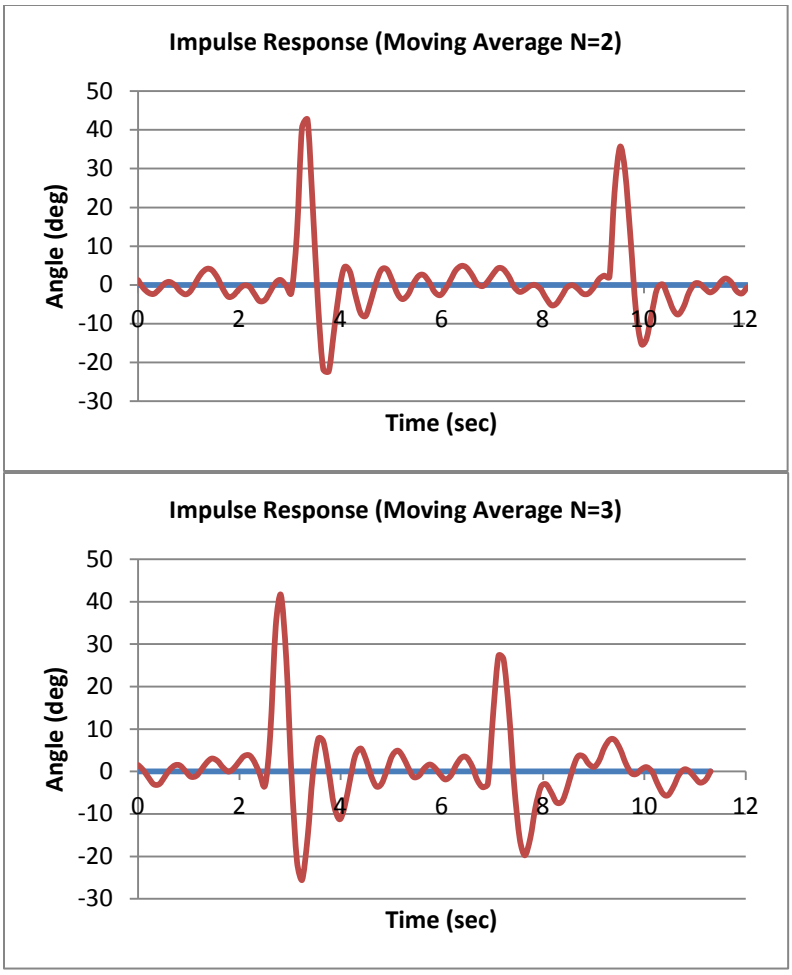


FIGURE 11-6: IMPULSE RESPONSE OF PID CONTROLLER WITH MOVING AVERAGE FILTER OF 2 AND 3 TERMS

Impulse response of PID controller with moving average filter on the derivative action is plotted in Figure 11-6. In comparison with the control subject, the controller without the moving average filter, the amplitudes of oscillation seem to be slightly larger. After the impulse the control with a 2-term filter still settle in around 1.5 to 2 seconds. It just than when compared with the response with no filter, the amplitude is larger. However, the 3-term filter is noticeably different. In its impulse response, not only is the oscillation amplitude larger, the overshoot is also higher at -25 degrees after the first impulse and -20 degrees after the second impulse. The settling time also seem to be longer at around 3 seconds. The performance of the filtered

response is overall worse in an impulse response. However, the reaction time does not seem to be much different.

The ramp tracking tests were then conducted after the impulse tests. The setup involve the balancer tracking an angle command sweeping from -30 to 30 degrees at a speed of $30°/s$ and $10°/s$ for the high speed and low speed test respectively.

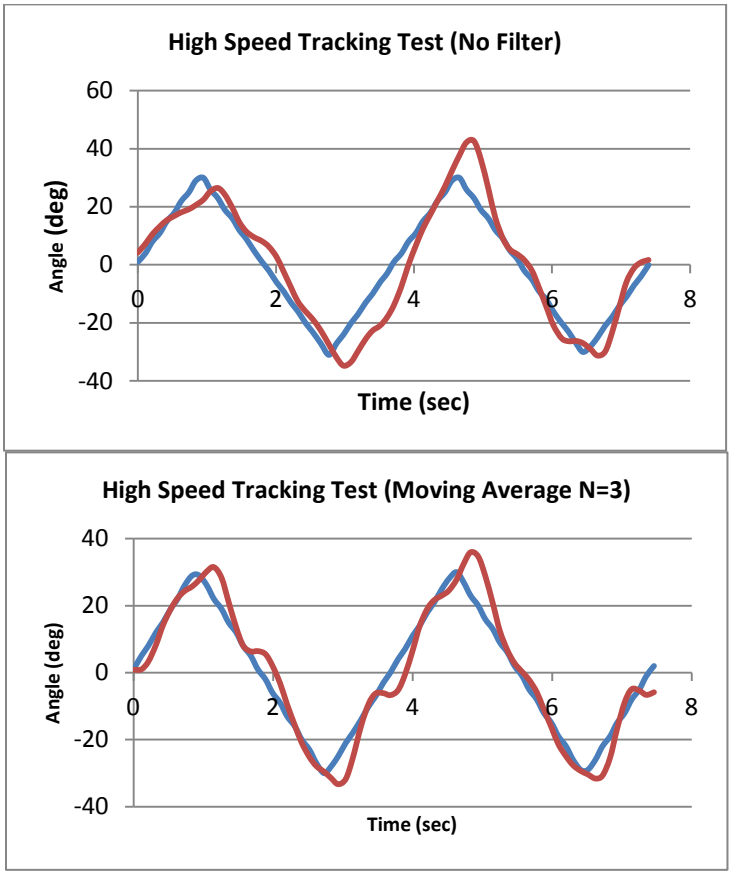


FIGURE 11-7: 30°/S TRACKING RESPONSE COMPARISON BETWEEN FILTERED AND NON-FILTERED CONTROLLER

High speed tracking response for the 3-terms filtered and non-filtered controller is plotted in Figure 11-7. At first glance, it appears that both set of response are pretty similar. They both track the command signal relatively well with some oscillation and an overshoot of around 10 degrees. With more careful analysis, one will notice that the non-filtered controller actually

have a larger phase delay than the filtered term. Considering that filter usually introduce delay in signal response, this is somewhat counterintuitive. However, recall that filtered controller have higher oscillation amplitude. The oscillation pushed the balancer back and forth across the command angle. Depending on application, this could be a good property. The same behavior is shown in the low speed tracking responses in Figure 11-8. In comparison, the low speed tracking performance is much better than that of high speed tracking.

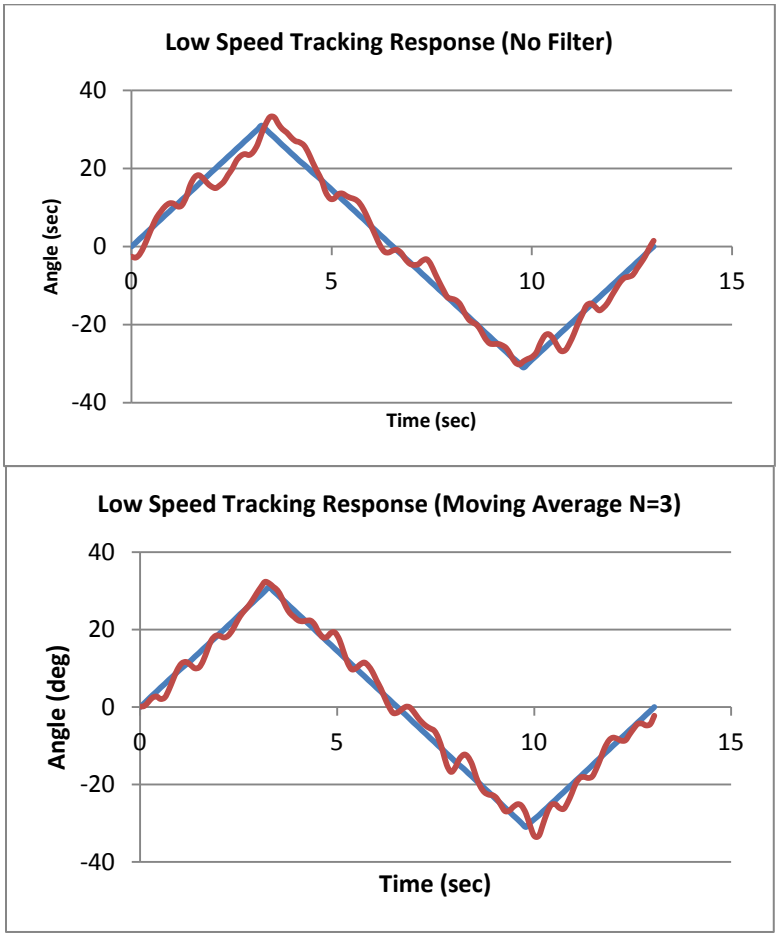


FIGURE 11-8: 10°/S TRACKING RESPONSE COMPARISON BETWEEN FILTERED AND NON-FILTERED CONTROLLER

The final experiments for the PID controller were the step response tests drawn from a preset command array with 2 seconds interval. The logged data were plotted in Figure 11-9. Whereas the overall performance of the non-filtered controller was somewhat satisfactory, the step

response seems to have a lot of overshoot. In the lab balancer, overshoot does not matter much. However, that could pose a problem for the flying platform. It also appear the system don't settle in time before the next command is executed. As expected, the filtered controller had a response that is much worse. Aside from greater overshoot amplitude at times, the increased oscillation is also affecting stability.

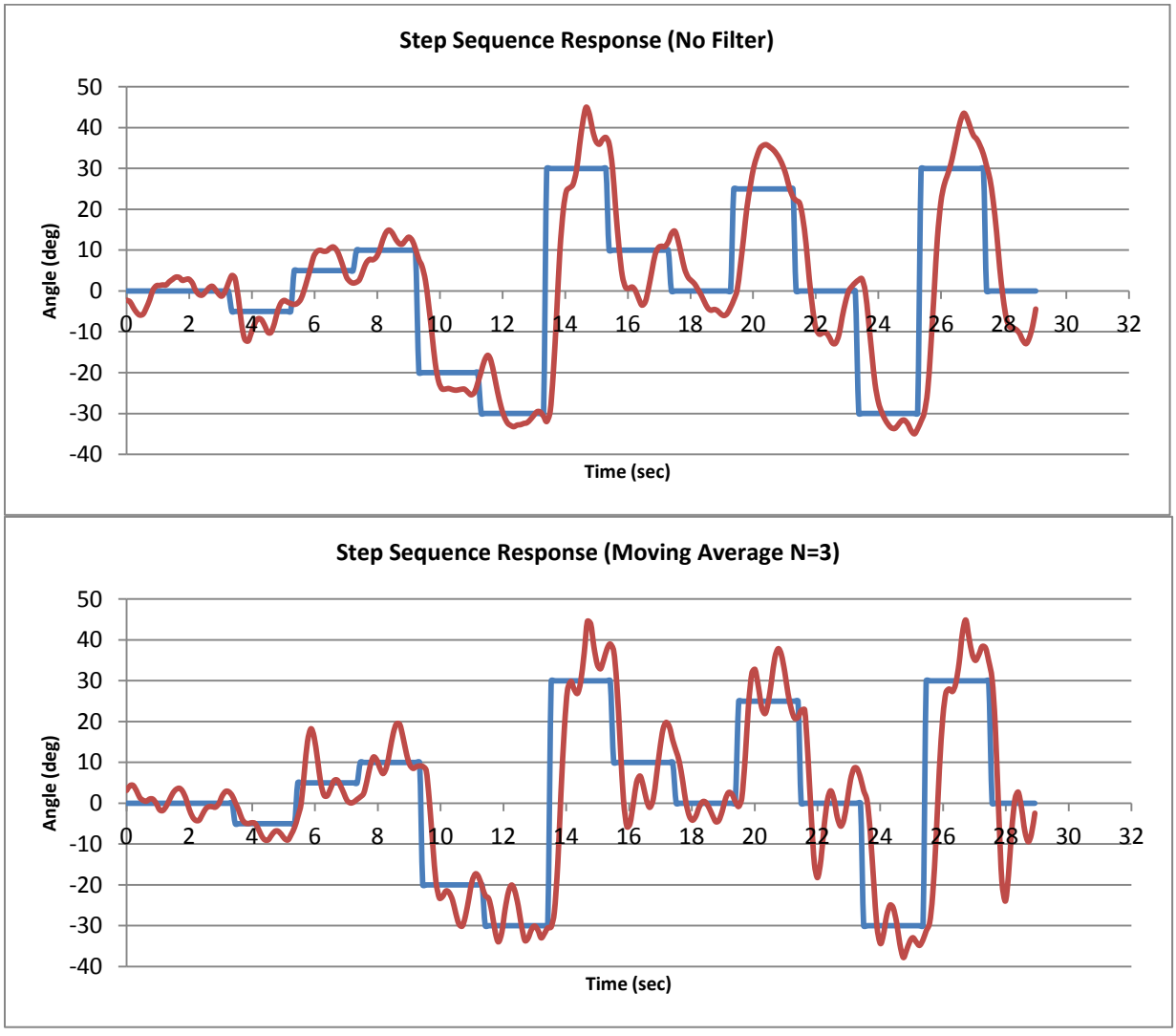


FIGURE 11-9: STEP SEQUENCE RESPONSE OF FILTERED AND NON-FILTERED PID CONTROLLER

In summary, the moving average filter in this case does more harm than good to the system. This could probably be drawn from the fact that the sample time of the control loop is only 70

Hz to start with. While the moving average filter smooth out sensor noises using a simple algorithm, it also causes time delays. The additional time delay on the derivative action might have cancel out a portion of the damping effect. Therefore, the system will need to be retune in order to obtain satisfactory result. Several attempts were made to retune the balancer, however, the oscillation problem remains. Therefore, it is advice that the filter should only be used in a system with faster sample rate.

CASCADE PID CONTROLLER TUNING

The second control law investigated in the single axis multirotor balancer is the cascade PID controller as described in Chapter 7. As oppose to a PID controller where the error in attitude is used to directly compute a signal for the flight mixer, the cascade controller is consisted of 2 PID controller stack onto each other. The first PID controller uses the error in attitude to compute a desire attitude rate or angular rotation rate. The desired angular rate is then compared with sensor angular rate from the gyroscope. The error is then feed into a second PID controller. The output from the attitude rate controller is then sent to the flight mixer to compute PWM output for the motors.

In comparison with ordinary PID controller, the cascade PID controller has twice as much parameter to be tuned. Fortunately, a PI-PD controller is usually sufficient instead of the full PID-PID controller. A PI-PD controller is essentially two PID controller stack on each other with the derivative term removed in the primary attitude controller and the integral term removed in the secondary attitude rate controller. The tuning process for the cascade controller is similar to that of a PID controller with the secondary controller tuned first and the primary controller tuned last. The primary controller should be disconnected when tuning the secondary controller.

When tuning the secondary controller, it is important to remember that the process variable is angular rate as oppose to angular position. Therefore, the proportional gain $K_{p_{rate}}$ should be tuned for the balancer to oppose or foster motion. The derivative gain $K_{d_{rate}}$ should be tuned to remove oscillation. In relation to the PID controller tuned in the previous section, the $K_{p_{rate}}$ is in charge of the "fight back" factor in an impulse scenario. While the proportional term opposes motion, it does not control angular position. Therefore, if the balancer turns due to a force stronger than that provided by the motor, at this point, the controller will not correct for that.

After tuning the attitude rate controller, several impulses, step and ramp tests were conducted to ensure controllability for the primary attitude controller. The attitude controller was then tuned using the same method as an ordinary PID controller. The derivative term is no longer necessary since it's already covered by the secondary controller. The tuning that yields the best stability performance are:

$$K_{p_{rate}} = 0.5 \quad K_{d_{rate}} = 3.3$$

$$K_{p_{attitude}} = 1.7 \quad K_{i_{attitude}} = 0.02$$

CASCADE PID CONTROLLER EXPERIMENTAL RESULTS

The same sets of tests were conducted for the cascade controller. Similar to the PID controller, the moving average derivative filter add oscillation and amplify overshoots in system responses. Since it's more useful to compare to contrast the difference between an ordinary and cascade PID controller, the experimental results for the filtered tests will be omitted.

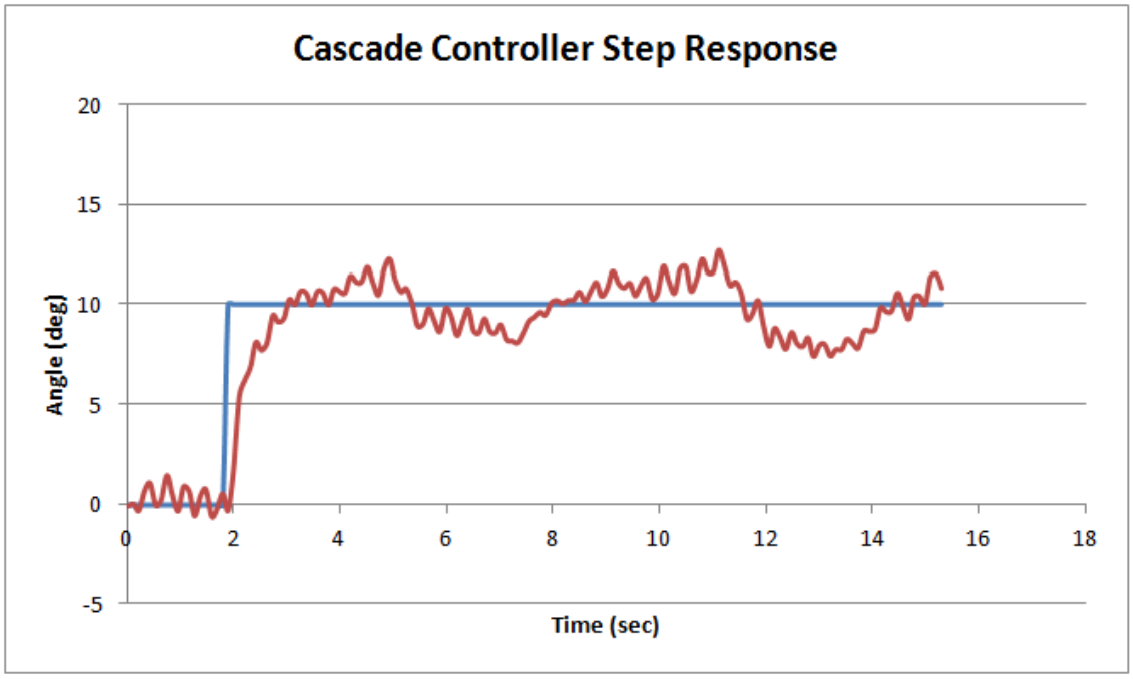


FIGURE 11-10: CASCADE CONTROLLER STEP RESPONSE

Recall the PID controller's step response in Figure 11-3, the cascade controller's step response is plotted in Figure 11-10. When comparing the two, it is apparent that converge better at the target angle of 10 degrees. The oscillation amplitude is only around $\pm 2°$ as oppose to the traditional PID controller's $\pm 5°$. The oscillation period is also larger at 5 seconds instead of 3 seconds. The rise time is a bit slower, but the settling time is roughly the same.

A Simulink model is also constructed for the cascade controller. Similar to the previous simulation model, the simulation result demonstrate behavior that's similar, but more optimal than real life in a way. Using the same gain parameters, the simulation demonstrated a response that's more damped with minimal oscillations. The rise time in the simulation is actually much worse than in the experiment. Consider that parameters like sensor noises, process noises and the sample rate of 70 Hz is included in the model, there must be other disturbances that were overlooked.

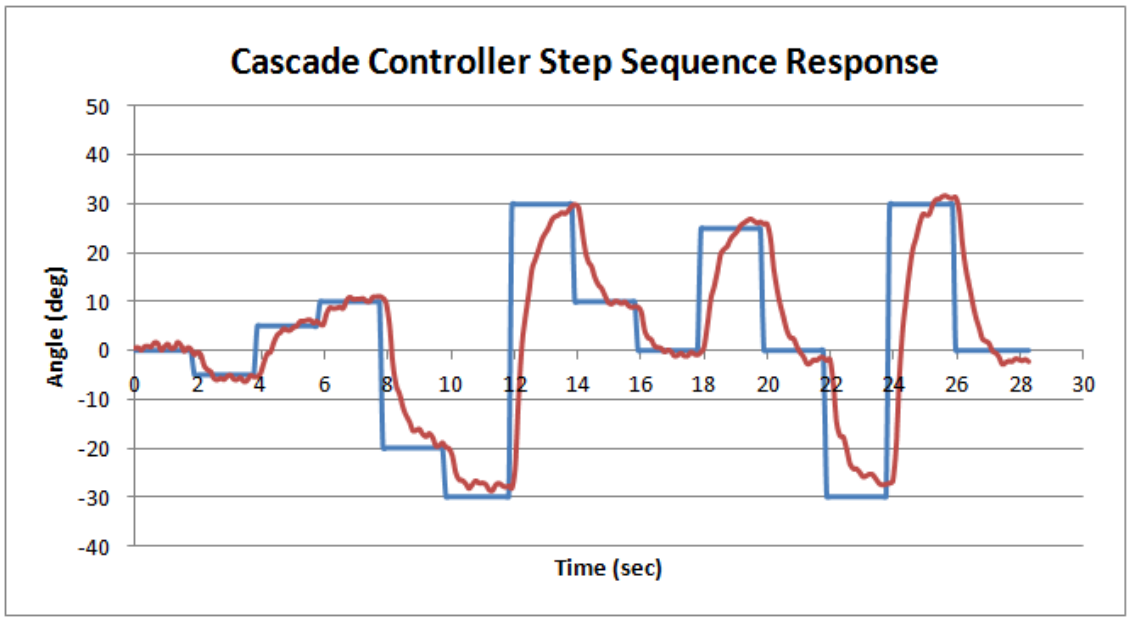


FIGURE 11-11: CASCADE CONTROLLER STEP SEQUENCE RESPONSE

While analyzing step response of the cascade PID controller, data for the step sequence response is reproduced in Figure 11-11. In comparison with the sequence response in Figure 11-9, the cascade control exhibit much better performance. While the rise time suffers a little depending on step size due to the controller's damping behavior, the steps are actually recognizable with barely any overshoot and oscillations. The smooth transition from angle to angle is highly desirable as it will yield smoother flight response in the air. This will in turn make guidance and navigation perform better.

The cascade controller's damping behavior greatly affects its tracking response as presented in Figure 11-12 and Figure 11-13 for high and low Speed tracking respectively. In comparison with the tracking response of the PID controller in Figure 11-7 and Figure 11-8, cascade controller yield smoother responses. However, the phase lag is also more apparent. Whereas the PID controller goes back and forth on the tracking signal, the cascade controller is consistently lay

behind by almost 0.5 seconds. While this performance will not be acceptable for applications like radar target tracking, the smooth response is preferred in multirotor flight dynamics.

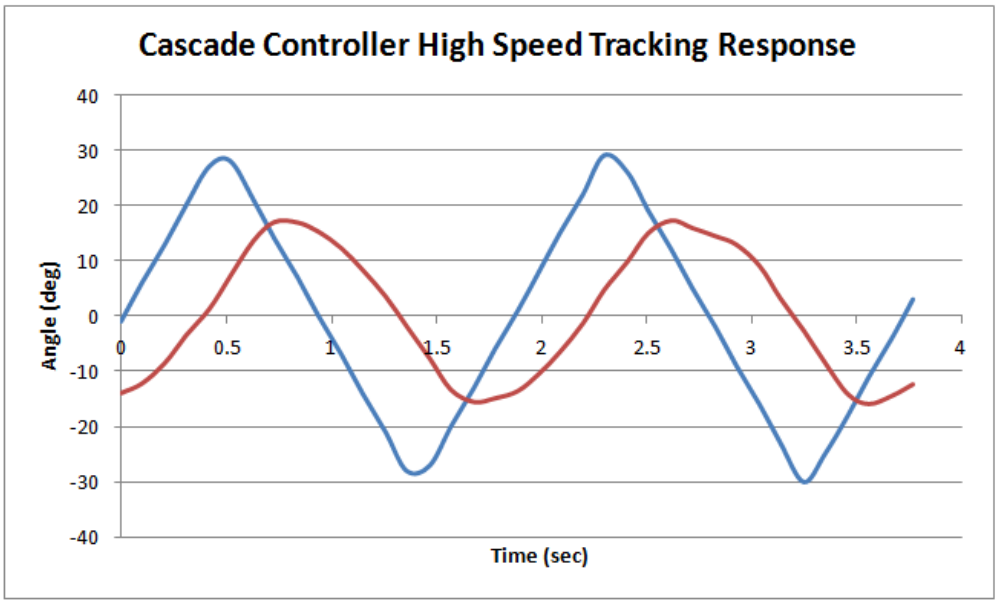


FIGURE 11-12: 30°/S TRACKING RESPONSE FOR CASCADE CONTROLLER

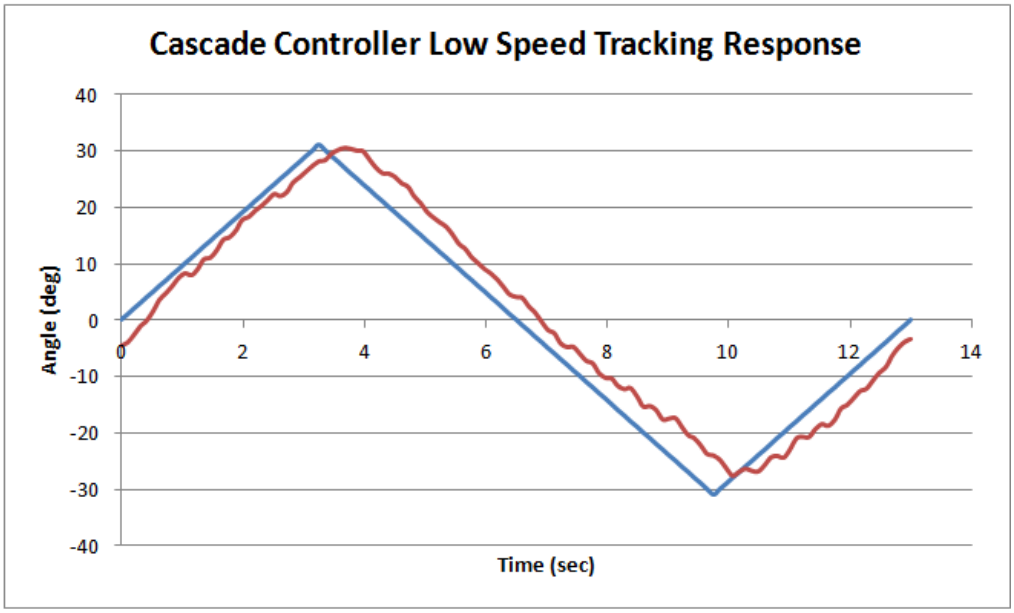


FIGURE 11-13: 10°/S TRACKING RESPONSE FOR CASCADE CONTROLLER

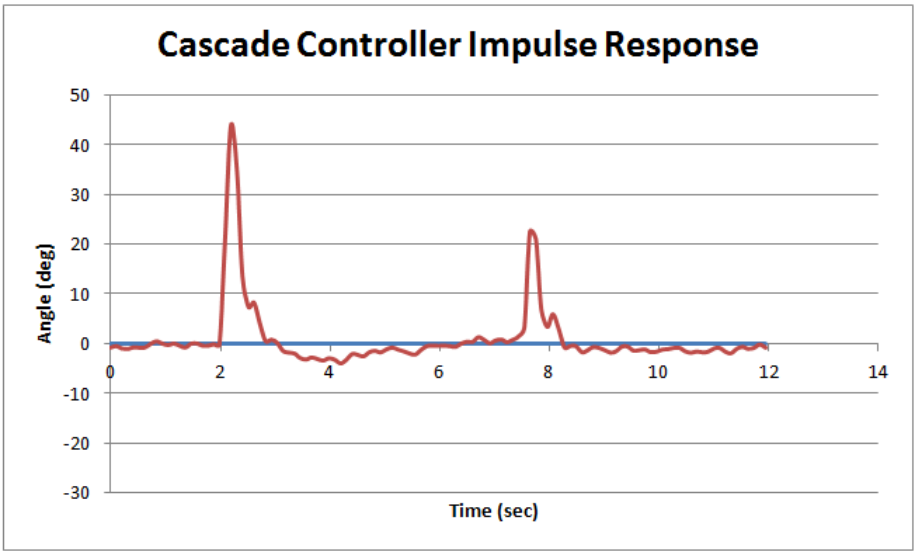


FIGURE 11-14: IMPULSE RESPONSE OF CASCADE CONTROLLER

The cascade controller's forte is probably in its impulse response. As seen in Figure 12-14, the controller is able to maintain the balancer at a level position with neglectable oscillation. The balancer react immediately after an impulse, however, the response is damped so that the balancer slowly settle with minimal overshoot. Whereas the overshoot on the PID controller, as shown in Figure 11-5, is almost -20 degrees after the first impulse, the overshoot in the cascade controller is only around 3 degrees. Since the cascade controller reacts to the impulse more controllably, the system converges to back to acceptable level only after around 1 second depending on the impulse strength.

In conclusion of this section, cascade controller is a better choice to control attitude. Even though it involves more work in tuning, the system response performs much better than that of an ordinary PID controller in terms of system damping. Not only is the controller able to react to commands at an acceptable pace, it is able to do so stably with very little oscillations. If same results can be achieved on the flying platform, then flight performance will be a lot smoother and easier to control.

Chapter 12: QUADX FLIGHT TEST

The QuadX quadcopter was constructed and tuned in the Aerospace Systems Lab. Due to time constraints, the guidance and navigation system is not implemented in the FlightOS. In addition, longitude and latitude position control is also not implemented.

The QuadX went through two phases of testing. The first are indoor rig tests to ensure that the controller is tuned properly and to prevent the quadcopter from losing control. The quadcopter undergo step, ramp and impulse tests to simulate outdoor environmental interference such as wind gusts. The second and final phase of the testing process is outdoor flight test. During the outdoor flight test, the QuadX is primarily controlled by a RC transmitter. A laptop ground station is also nearby to collect telemetry data at 10 Hz and to provide an emergency override should the quadcopter loses control.

FLIGHT TESTING VIDEOS

Several videos were taken to document the QuadX's testing and development process. They can be found in a YouTube playlist at http://goo.gl/vEuD44.

FIGURE 12-1: QUADX SPACE HOPPER SIMULATOR

RIG TEST RESULTS

The QuadX was tuned and calibrated multiple times on the test rig. Obtaining a perfect set of gain seems to be impossible. In comparison with the single-axis balancer, the quadcopter is a lot hard to tune due to the coupling effect. In generally, the quadcopter exhibit oscillatory behaviors that can't be dampen out. Eventually, a set of gains similar to that of the single axis balancer was chosen to procedure with the development.

	$K_{p_{angle}}$	$K_{i_{angle}}$	$K_{d_{angle}}$		$K_{p_{rate}}$	$K_{i_{rate}}$	$K_{d_{rate}}$
Roll	1.7	0.02	0		0.5	0	3.3
Pitch	1.7	0.02	0		0.5	0.03	3.3
Yaw	N/A	N/A	N/A		1.5	0.5	0.3

TABLE 12-1: QUADX CASCADE ATTITUDE CONTROL TUNING PARAMETERS

The 1st and 2nd generation attitude test rig was modified for indoor flight test. A paracord is hung from the ceiling and attach to the test rig by going through a center hole and tying to one of its legs. Both 3rd generation hopper simulators have a center hole so that they can be attached to the test rig through the paracord. The paracord's purpose is to limit the quadcopters' position. That way, it has the freedom to ascend, descent, rotate and limited horizontal travel. This is to make sure that the quadcopters don't crash into people and equipment around the aerospace system lab. The top of the test rig is a foam landing pad. This is to ensure that the QuadX don't crash into the metal rig when landing. Early rig tests are conducted through an umbilical cable that provides power and data transmission.

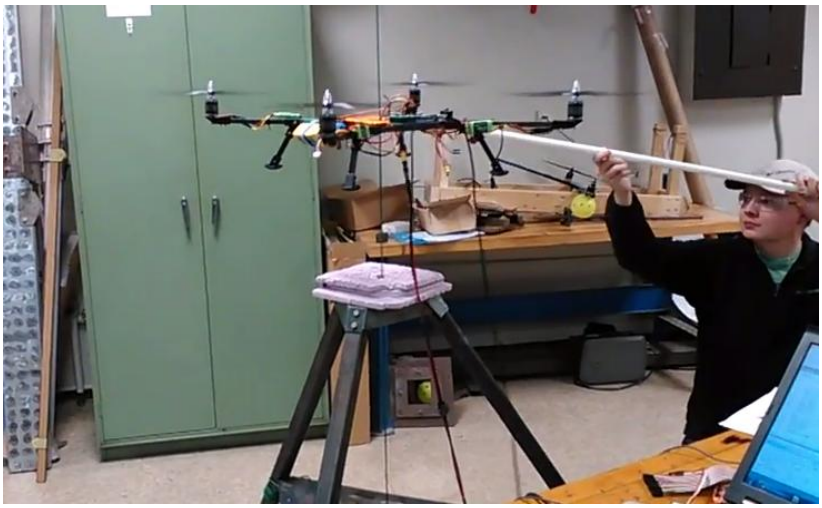


FIGURE 12-2: TETHERED IMPULSE RIG TEST

After the FlightOS became more mature, the rig tests are switched entirely to the internal battery power system and telemetry is handled through the onboard XBee radio. This is to prepare for outdoor flight tests and to remove the cable's damping behavior. In addition, the landing pad was replaced to a larger piece of foam board to test the altitude control system.

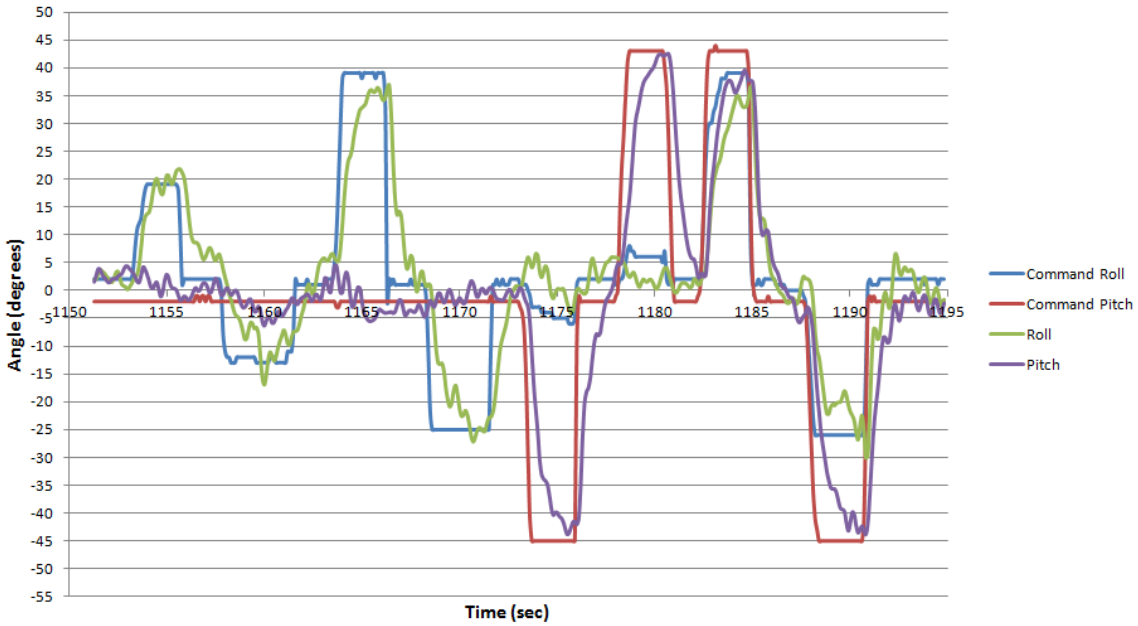


FIGURE 12-3: QUADX RIG STEP COMMAND TEST RESULT

In Figure 12-3, the step command test results are graphed. From $t = 1150$ to $t = 1170$, the pitch signal demonstrates the QuadX's ability to maintain level pitch. As seen in the response, the pitch command remains at around $-2°$, but the pitch signal oscillates between $\pm 5°$. In comparison, the single axis balancer oscillates at only $\pm 2°$. Though not ideal, the multirotor's ability to maintain level pitch is not terrible either. The oscillate might, however, cause drifting issues in the future.

Other than the oscillation the system also have a slow rise time due to the damping behavior of the cascade controller. The graph shows that the system take around 1.5 seconds to converge to the steady state value. In addition, there seems to be a slight steady state error of around $3°$. In theory, the integer term for the PI controller should be increased. However, it seems that doing add oscillation into the system.

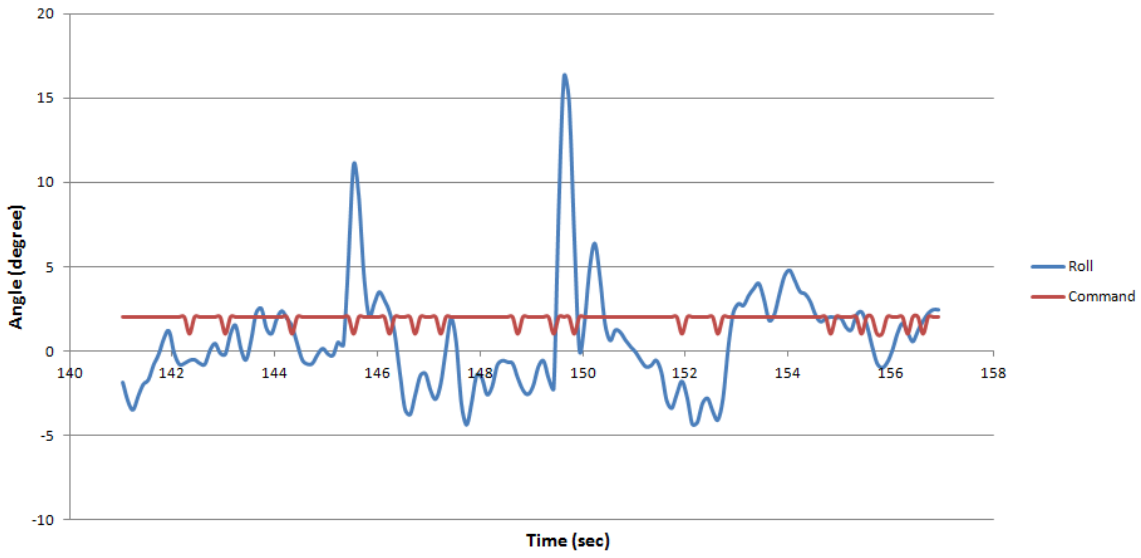


FIGURE 12-4: QUADX RIG IMPULSE TEST RESULT

The QuadX's roll impulse response is illustrated in Figure 12-4. For the impulse test, the multirotor is set to hover in the test rig and then impacted by a rod, similar to the image as shown in Figure 12-2. In the test, the QuadX was stroke by the rod two times. The first was a

soft hit that bring the quadcopter to a roll of around $11°$. The second hit, however, is a strong hit that bring the quadcopter to a roll of around $16°$.

From the test, the cascade controller provided a quick and powerful response to resist the impulse. That's why the QuadX only roll to a maximum of $16°$. Aside from that, the system also responded by driving the system back to the original level angle within 0.5 seconds.

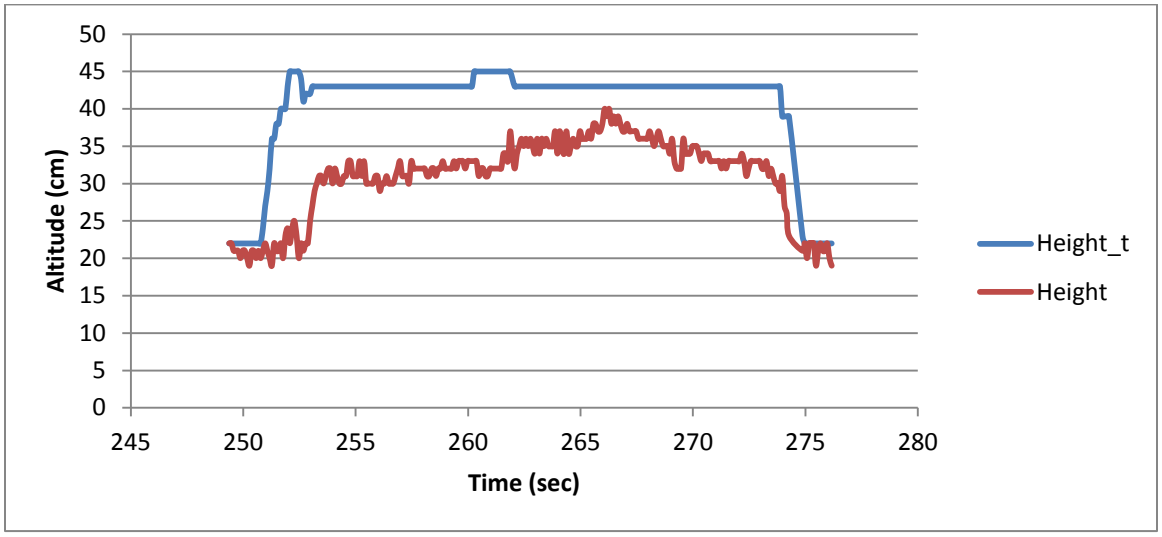


FIGURE 12-5: QUADX ALTITUDE CONTROL TEST RESULT (TAKE OFF AND LANDING)

The position control system was not fully implemented due to the noise in the GPS data. It is necessary to first implement the navigation Kalman filter. The altitude control system only relies on the ultrasonic sensor at the moment. Therefore, a PID controller was implemented and tested with a gain of $K_p = 1, K_i = 0, K_d = 0.05$.

The ultrasonic sensor has a minimum measurement distance of 20 cm. Therefore, the QuadX is actually on the landed pad when the plot reading is 20 cm. From the response in Figure 12-5, it can be seen that the controller does work. However, it's not optimal due to the large steady state error. In addition, the response seems to be slow in taking off from the landing pad, taking

several seconds. A possible cause is that the pulse width value added to the altitude controller output is not high enough to achieve hovering flight at neutral signal output.

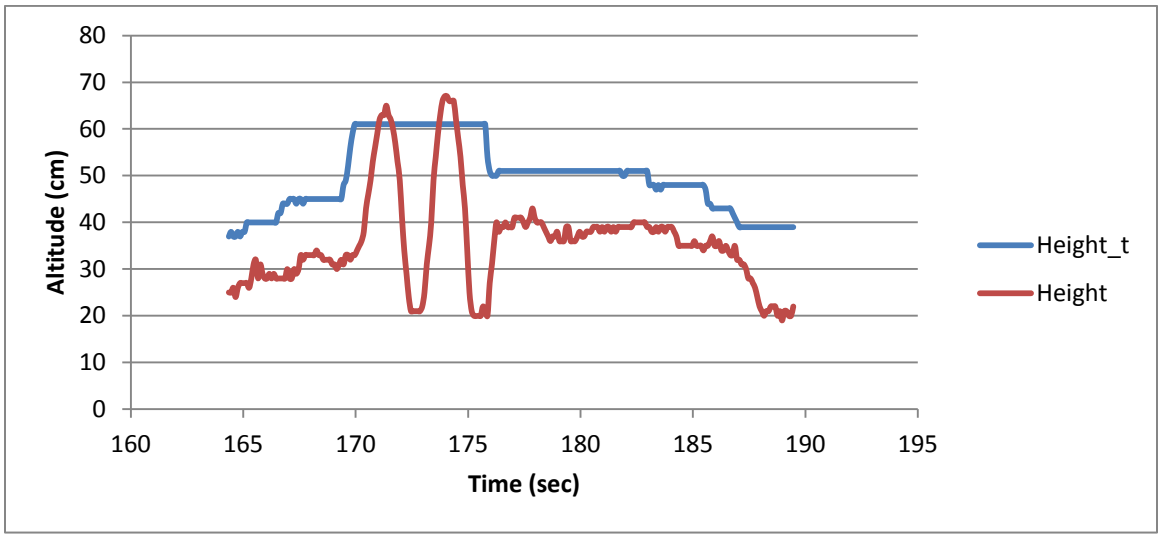


FIGURE 12-6: QUADX ALTITUDE CONTROL TEST RESULT (MID-FLIGHT STEP COMMANDS)

In Figure 12-6, if can be seen that the QuadX response faster to commands while it is already hovering above the landing pad. There is still a steady state error of roughly 10 cm. From $t = 170$ to $t = 176$, the command executes a step from around 45 cm to 60 cm. This causes the QuadX to oscillate out of control, hitting the height limiter at the paracord and the landing pad. When the altitude command is reduce down to 50 cm, the system became stable again.

With the current tuning, the height control is not stable enough for use in outdoor flight due to the oscillations and steady state error. More time will need to be spent with calibrating the hovering PWM pulse width and also filtering out the noise so that the derivative gain can be increased. In alignment in the simulation, the altitude controller seems to be quite difficult to damp. When the QuadX is capable of stable height control, then the integral gain should be added to reduce the steady state error. Several attempts were made to add the integral gain to the controller. However, this simply added more oscillations to the system.

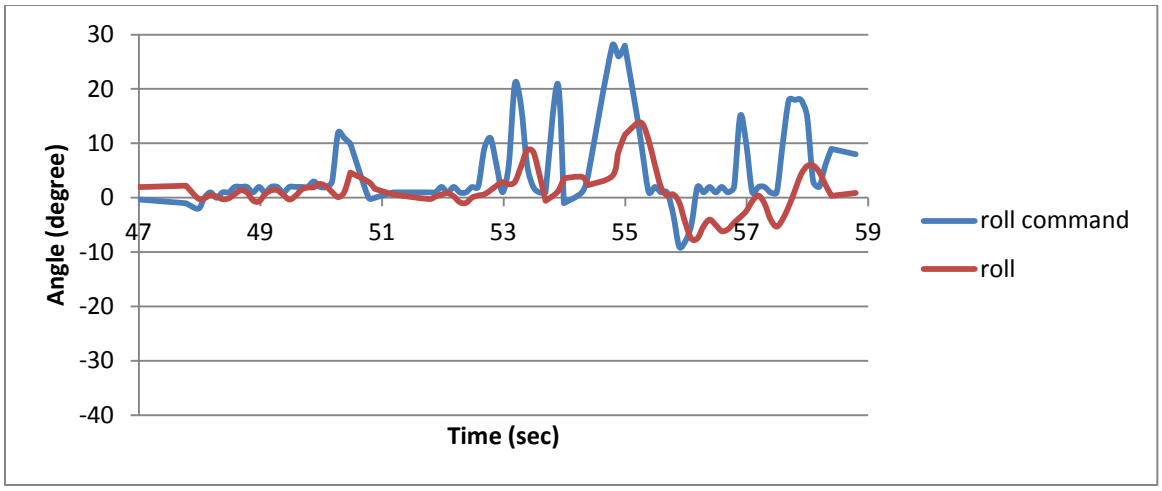
FLIGHT TEST RESULTS

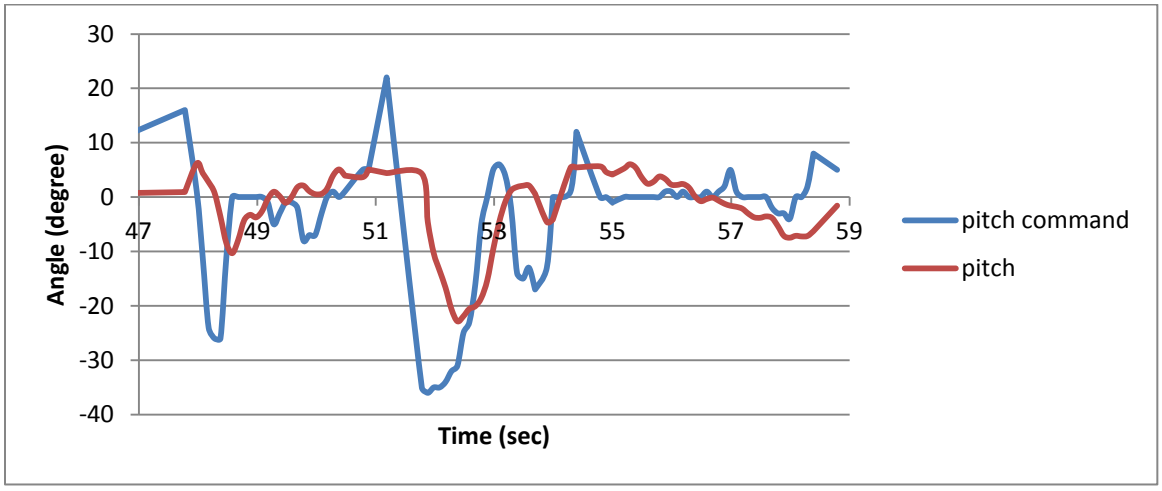
The first outdoor flight test took place on 1/30/2015 and went relatively well initially. However the RC transmitter link was lost several seconds into the flight. Hence the quadcopter kept executing the last command of level ascend. Eventually, the rotors impact with a tree branch and shut down. The QuadX ended up crashing into a brick road, destroying all the propellers and bent several motor shafts.

The 3/19/2015 flight test was the second outdoor test and implemented several design changes. Primarily, a second radio link is set up for telemetry and control override. Along with the ground station, the flight data are logged and analyzed. A third outdoor flight test was conducted on 3/24/2015 with a longer duration to collect more data to determine system response.

The results for this section are generated with telemetry data collected on 3/19/2015 and 3/24/2015. The corresponding can be found at http://goo.gl/2pT7L0 and http://goo.gl/j6KNmD.

FIGURE 12-7: QUADX OUTDOOR FLIGHT TEST AT UNIVERSITY CENTER FRONT LAWN





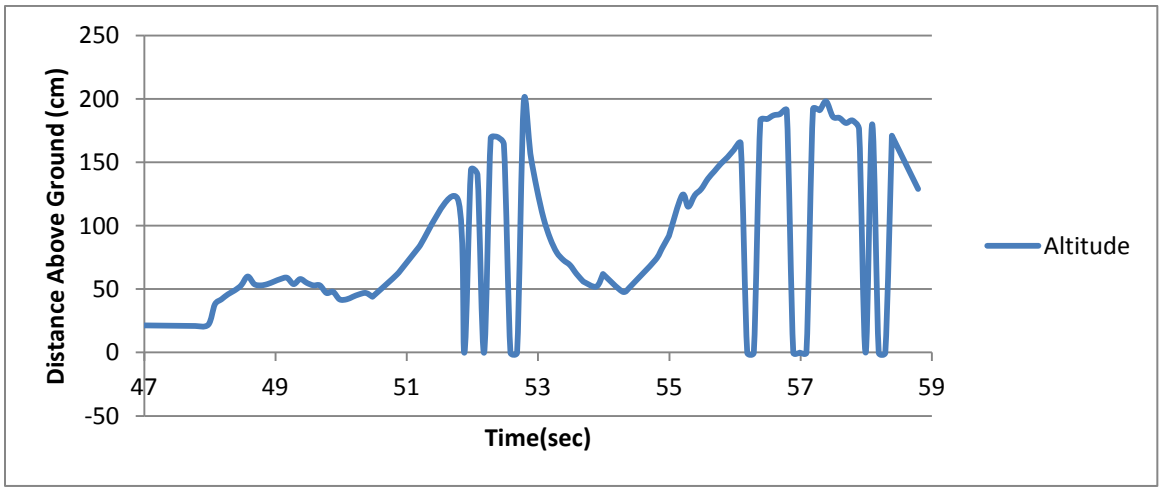


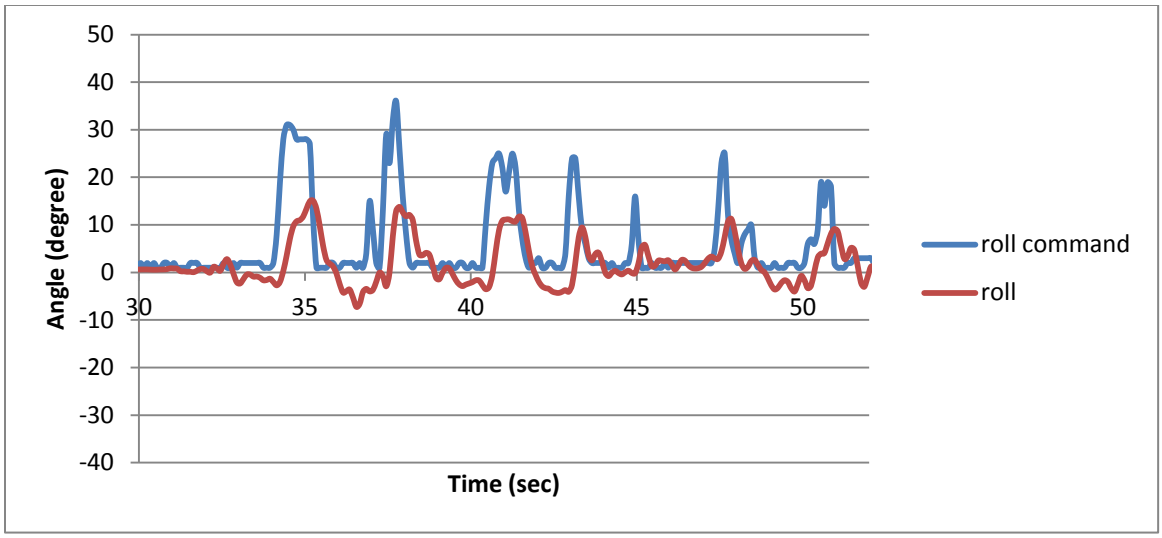
FIGURE 12-8: ROLL, PITCH AND ALTITUDE RESPONSE FROM FLIGHT TEST (3/19)

During the 3/19 flight test, during takeoff, the QuadX is commanded to pitch forward to avoid collision with the concrete platform. It is then commanded to pitch back to reduce forward velocity. Finally, the QuadX is commanded to roll right and land to avoid the student walking by.

As seen in Figure 12-8, the system clearly doesn't have enough time to converge to the target angle. In the rig test, the step commands are given for several seconds so that the system has enough time to converge. The slower rise time is acceptable in the controlled environment. However, in outdoor flight, quick response is required to avoid obstacles. Therefore, each command only lasted around 0.2 seconds

To improve performance, the cascade attitude controller should be tuned more aggressively. However, a balanced gain set must be used so that the aircraft also behave well to wind and overall stability.

The altitude data is also plotted in the graph located at the bottom of 12-8. The reported data are distance above the ground level and they mostly match the height shown in the video. However, it appears that there are several jumps at which the height is reported as 0. In coding the interface for the ultrasonic sensor, a limiter of 500 cm is programmed in to catch corrupted measurements. The value is then manually set to 0. Given that the minimum measurement distance is 20 cm, it is obvious that the measurements at those heights exceeded the limit. Consider that the trend of the flight around $t = 51$ to $t = 59$ is from 50 cm to 200 cm, those jumps are measurements corrupted either by the sensor or interference with the environment.



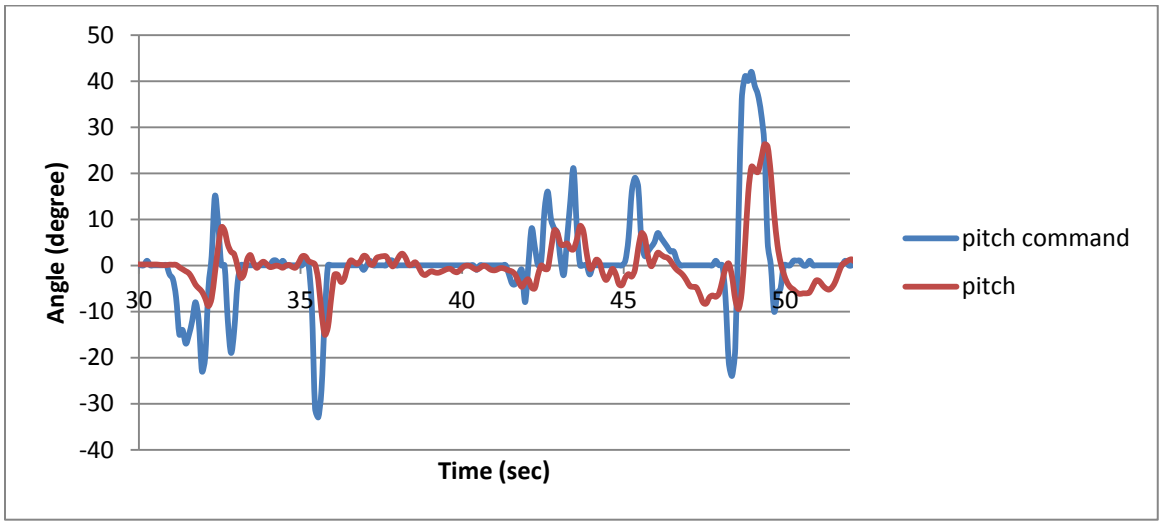


FIGURE 12-9: ROLL, PITCH AND ALTITUDE RESPONSE FROM FLIGHT TEST (3/24)

The 3/24 flight test was conducted mainly to determine the position drifting issues that occurs during the 3/19 test. It appear that the multirotor have a tendency to drift down the hill from University Center front lawn while being command at level as seen in the videos.

Telemetry data in Figure 12-9 show that the QuadX perform relatively well in terms of converging to level flight in both roll and pitch. Therefore, the drifting issue must be caused by external forces such as a wind gradient.

Chapter 13: FUTURE WORK

At its current state, the QuadX Space Hopper Simulator is capable of flying with relatively decent performance. However, the cascade attitude controllers should be tuned to be more aggressive for faster response. In that case, a higher derivative gain will be required to damp the system. The derivative gain continuously to be the problem at gain tuning. The UM7 IMU currently used in the quadcopter might not be properly calibrated, especially the Extended Kalman Filter. This could introduce time delay if not properly set. Therefore, for the next step, it is recommended to develop an attitude estimator using the raw data collected from the IMU as outlined in Chapter 5. In addition, a different IMU could be used. This would ensure that the attitude sensor data are correct and not the source of the oscillation behavior.

Before implementing the guidance, navigation and position control system using the methods simulated in Chapter 7, 8 and 9, the software system should be further optimized to allocate more resources for these additional systems. Currently, as shown in Chapter 6, the bottleneck of the FlightOS is the telemetry due to the slow serial baud rate in order to maintain data packet integrity. The telemetry downlink rate should be reduced from 10 Hz to around 1 Hz. In addition, the SPI microSD flight logger module should be added to FlightOS for high speed data logging.

The guidance system should include the feature to automatically optimize the trajectory for minimal total energy consumption. This could be done by trying to model an equation to describe the relationship between the trajectory parameters and the energy consumption.

Finally, adaptive control should be explored so that the aircraft could adjust its gain automatically to different flying conditions. The tuning process is one of the most time consuming aspects of the project.

BIBLIOGRAPHY

[1] C. Furrball, "RC Wiring Diagrams and such," [Online]. Available: http://www.tjinguytech.com/connectors.

[2] G. Staples, "Propeller Static & Dynamic Thrust Calculation," [Online]. Available: http://www.electricrcaircraftguy.com/2013/09/propeller-static-dynamic-thrust-equation.html.

[3] P. Connolly, "Hyperion Prop Talk," [Online]. Available: http://www.aircraft-japan.com/Datasheet/en/hp/emeter/hp-proptalk.htm.

[4] P. Connolly, "Prop Constants for Efficiency Determination," [Online]. Available: http://www.aircraft-japan.com/Datasheet/en/hp/emeter/hp-propconstants.htm.

[5] "HobbyKing: NTM 750 KV 28-36," [Online]. Available: http://www.hobbyking.com/hobbyking/store/__19611__NTM_Prop_Drive_28_36_750KV_265W.html.

[6] J. Gleick, "A Bug and A Crash," [Online]. Available: http://www.around.com/ariane.html.

[7] T. Hirzel, "Arduino - PWM," [Online]. Available: http://www.arduino.cc/en/Tutorial/PWM.

[8] C. Burnett, "Wikimedia Commons: I2C," [Online]. Available:

http://commons.wikimedia.org/wiki/File:I2C.svg.

[9] SFUptownMaker, "Sparkfun: I2C," [Online]. Available: https://learn.sparkfun.com/tutorials/i2c.

[10] C. Burnett, "Wikimedia Commons: Typical SPI bus: master and three independent slaves," [Online]. Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#mediaviewer/File:SPI_three_slaves.svg.

[11] M. Grusin, "Sparkfun: Serial Peripheral Interface (SPI)," [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi.

[12] M. Pedley, "Freescale Semiconductor: Tilt Sensing Using a Three-Axis," [Online]. Available: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf.

[13] K. Lauszus, "TKJ Electronics: A practical approach to Kalman filter and how to implement it," [Online]. Available: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/.

[14] "Pololu: UM7-LT Orientation Sensor," [Online]. Available: https://www.pololu.com/product/2740.

[15] "MaxBotix: MB1240 XL-MaxSpmar EZ4," [Online]. Available: http://www.maxbotix.com/Ultrasonic_Sensors/MB1240.htm.

[16] "Adafruit: Ultimate GPS," [Online]. Available: https://www.adafruit.com/products/746.

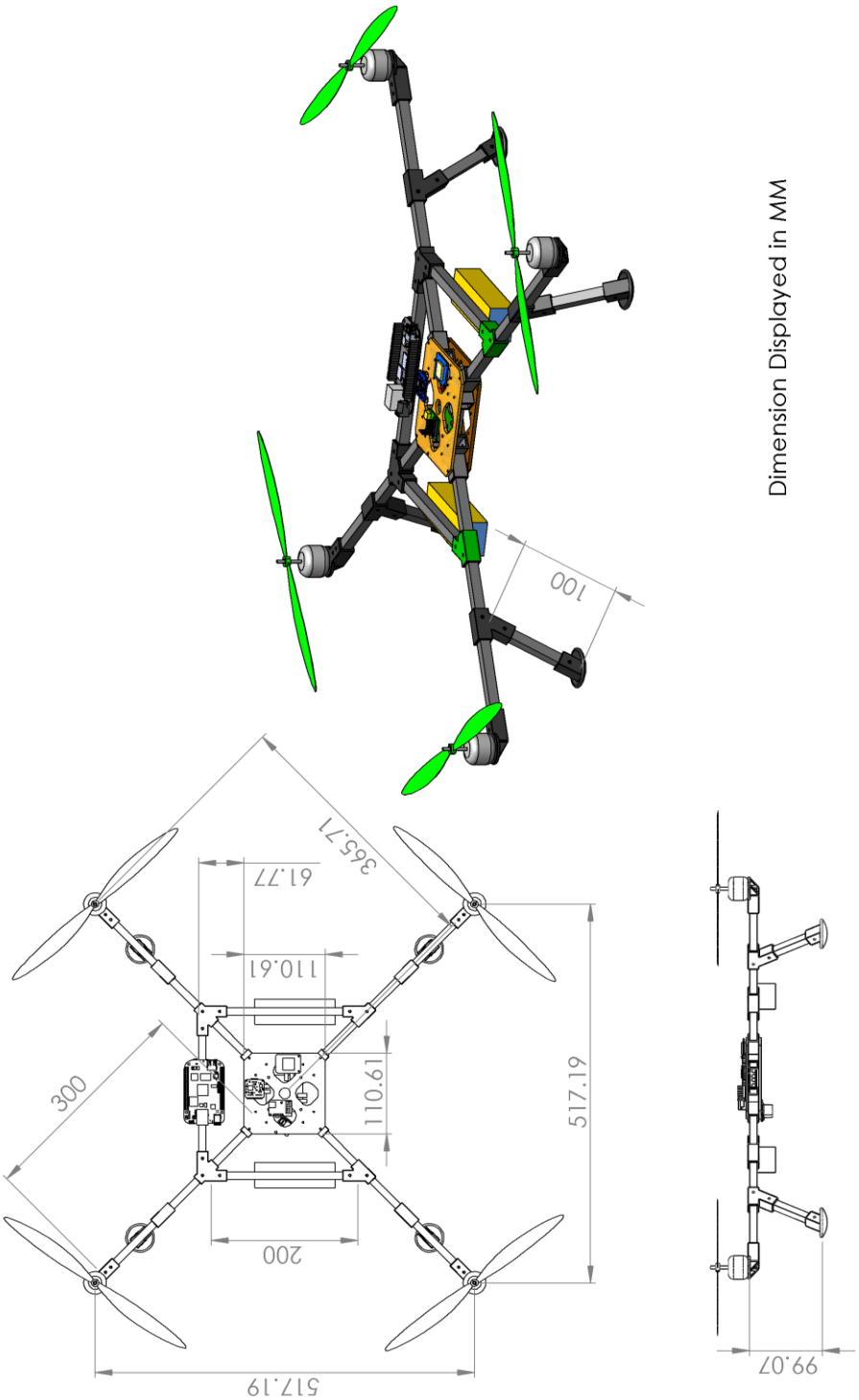
[17] "Spektrum: DX7 7Ch Air w/AR7000 & 4-DS821 MD2," [Online]. Available:

http://www.spektrumrc.com/Products/Default.aspx?ProdId=SPM2710.

[18] E. Williams, "Aviation Formulary V1.46," [Online]. Available: http://williams.best.vwh.net/avform.htm#flat.

[19] "3DRobotics: PX4FLOW KIT," [Online]. Available: https://store.3drobotics.com/products/px4flow.

APPENDIX A – DETAILED CAD DESIGN



APPENDIX B – SOFTWARE SOURCE CODE

The source code for the FlightOS along with the support software modules are open sourced and available freely at GitHub under the MIT license: https://github.com/Billwaa/FlightOS.git

Language	Files	Blank	Comment	Code
C++	7	196	34	673
C/C++ Header	8	81	13	251
Arduino Sketch	1	86	79	323
XML	2	0	0	21
TOTAL	18	363	126	1268

Arduino FlightOS Stats

Language	Files	Blank	Comment	Code
C#	5	309	629	2605
MSBuild script	1	0	0	70
ASP.Net	1	0	0	6
TOTAL	7	309	629	2681

C# Ground Station Stats

VITA

Originally from Hong Kong, Yu Hin Hau (Billy) received his US citizenship and grew up in Holland, Pennsylvania. He received a B.S. in Mechanical Engineering *cum laude* in May 2013 from Lehigh University, with minors in Aerospace Engineering, Computer Science and Asian Studies. Yu Hin was the recipient of the Daniel Horner Memorial Scholarship and was inducted in the Pi Tau Sigma International Mechanical Engineering Honor Society.

During his undergraduate studies, Yu Hin participated in a number of interdisciplinary projects including development of a 10 kW concentrated solar cavity receiver, interplanetary trajectory design for a robotic / human colonization mission to Mars, and also the constructions of an ancient Chinese bridge and the Bethlehem Chinese Harmony Pavilion. With his passion in aerospace, Yu Hin served as the Vice President of the Lehigh Aerospace Club and is currently serving as a graduate student mentor for the club and the newly founded Lehigh AIAA.

In the Fall of 2011 and Summer of 2012, Yu Hin was enrolled in GE Aviation's Early Identification Co-Op Program in the Development and Production Test Operations, working with prominent turbofan and aero-derivative gas turbine engine programs such as the GEnx, GE90 and LM2500. During his rotation at the headquarter, Yu Hin successfully developed software to automate the worker timesheet management process and was given the Above and Beyond Bronze Award for Imagination and Courage.

Yu Hin is enrolled in NASA's Pathway Intern program as an Engineering Student Trainee at Glenn Research Center. He is a member of the Power, Diagnostics and Electromagnetic branch and is assigned to support development of the next-gen space suit, modular spacecraft power system and a rapid deployment communication CubeSat mission. In the Fall of 2014, Yu Hin was presented with the Distinguished Performance Award for his contributions to the center.

Yu Hin Hau is currently pursuing a M.S. in Mechanical Engineering at his alma mater and working with Professor Terry Hart on aerospace GN&C systems. Aside from his research, Yu Hin is a teaching assistant for the Spacecraft Systems Engineering course. Upon graduation, Yu Hin is expected to return to NASA to work as a full time Electrical Engineer.