Quadrocopter Control Design and Flight Operation

Katherine Karwoski[1]
Massachusetts Institute of Technology, Cambridge, MA, 02139

A limiting factor in control system design and analysis for spacecraft is the inability to physically test new algorithms quickly and cheaply. Test flights of space vehicles are costly and take much preparation. As such, EV41 recently acquired a small research quadrocopter that has the ability to be a test bed for new control systems. This project focused on learning how to operate, fly, and maintain the quadrocopter, as well as developing and testing protocols for its use. In parallel to this effort, developing a model in Simulink facilitated the design and analysis of simple control systems for the quadrocopter. Software provided by the manufacturer enabled testing of the Simulink control system on the vehicle.

Nomenclature

x	=	state vector (n x 1 vector)
v	=	input vector (m x 1 vector)
f(x)	=	nonlinear function of the system states (n x 1 vector)
G(x)	=	nonlinear function of the system states (m x n matrix)
ϕ	=	bank (roll) angle (rad)
θ	=	pitch angle (rad)
ψ	=	azimuth (yaw) angle (rad)
Kp	=	proportional gain
Kd	=	derivative gain
Ki	=	integral gain

I. Introduction

For spacecraft design, an important and often complex component is the control system, which keeps the vehicle on track and adjusts for unforeseen variations. However, spacecraft are expensive to test and take significant time to set up. With the continuing advances in control theory, it is helpful to be able to test the new algorithms, and while computer software has good simulation capabilities, the lessons learned from real-world testing can be invaluable.

Recently, small air vehicles have gained great capabilities and have become suitable for research purposes. One example is the line of quadrocopters from Ascending Technologies (AscTec), which provide a small, lightweight, and relatively inexpensive platform to test control programs on. A quadrocopter is an air vehicle that is lifted and controlled by four rotors. With the onboard microprocessors, they are powerful machines and can be used to quickly and easily perform tests.

The control systems design and analysis branch (EV41) at Marshall Space Flight Center (MSFC) recently purchased an AscTec Hummingbird quadrocopter. The main purpose of this project was to set up the system once it arrived. This included establishing protocol and areas for its use and learning how the hardware and software operated. In conjunction with this task, this project also designed a model and simple controller in Simulink.

II. Quadrocopter Setup

A. AscTec Hummingbird

The quadrocopter is an AscTec Hummingbird. Table 1 lists many of the technical details of the quadrocopter[1]. The frame is made out of balsa wood and carbon fiber[2], making it strong and lightweight. For sensors, the copter has a pressure sensor, an acceleration sensor, and three gyroscopes (one for each axis). It also has a three-axis compass and a GPS unit. All the sensors besides the GPS compose the Inertial Measurement Unit (IMU). The AutoPilot circuit board has two microcontrollers—a low level processor (LLP) and a high level

[1] Summer Intern, EV41- Control Systems Design and Analysis, Marshall Space Flight Center, MIT

processor (HLP). The LLP compiles the IMU data, sends commands to the motor controllers, and also has the basic attitude controller that comes with the quadrocopter. The HLP controls the GPS, but is mostly free as a space for the user-defined programs.

Table 1. Technical Details of the Quadrocopter.[1]

Model	AscTec Hummingbird with AutoPilot
Manufacturer	Ascending Technologies GmbH
Battery	2100 mAh LiPo
Takeoff weight	480 g
Distance between motors	34 cm
Propeller	8” flexible standard propellers
Motors	AscTec X-BL 52s with X-BLDC controllers
Radio controller	Futaba FAAST 2.4 GHz
Telemetry system	Xbee 2.4 GHz

The quadrocopter is controlled solely by motor speeds. As shown in Fig. 1, the front and rear propellers spin clockwise and the left and right motors spin counterclockwise. In this way, all three axes can be controlled through changing the rotation speeds. Roll is controlled by manipulating the speeds of the right and left motors; pitch is similarly controlled by the front and rear motors. Yaw is controlled by a combination of all four motors by speeding up the motors spinning in one direction and slowing the ones spinning the other way. This turns the quadrocopter by causing a change in angular momentum, but it does not affect the pitch and roll axes.

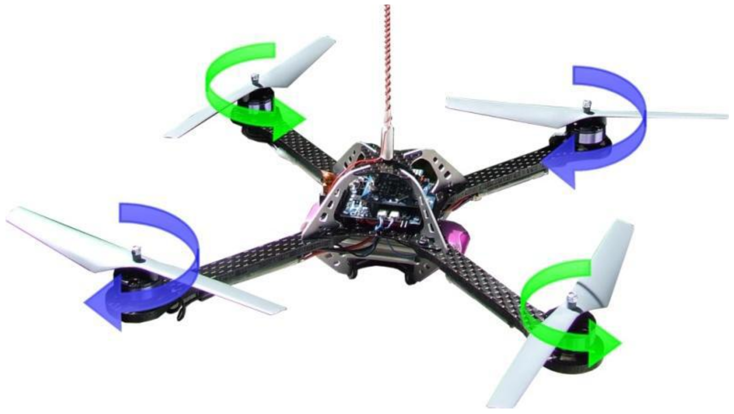

Figure 1. AscTec Hummingbird Quadrocopter.[1] *This image is of a standard Hummingbird Quadrocopter. The arrows show the rotation direction of the propellers, with the front and rear spinning clockwise and the left and right spinning counter clockwise.*

For safety, the quadrocopter was ordered with propeller protection, which connects to the arms of the quadrocopter and consists of lightweight corner pieces connected by carbon tubes.

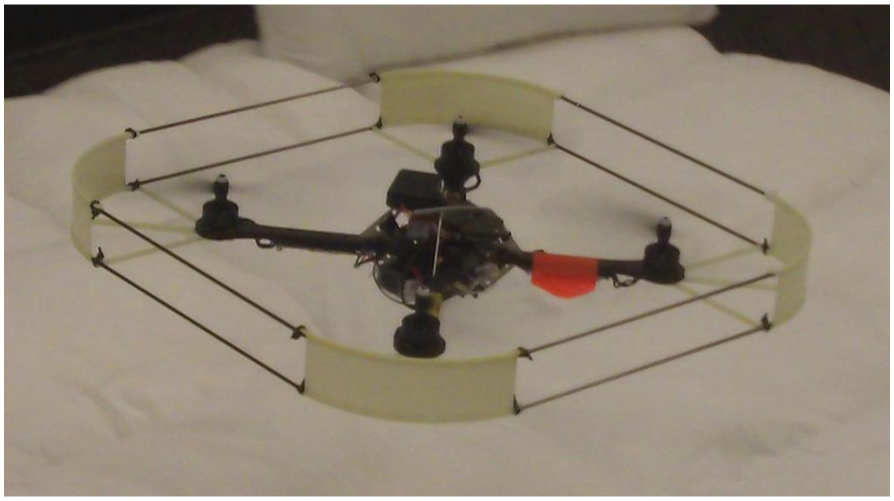

Figure 2. Quadrocopter in Flight. *This image is of the vehicle with the propeller protection on, during indoor flight.*

B. Indoor Flight Environment

The quadrocopter will be mainly used in an indoor environment. To protect people, property, and the quadrocopter, it was decided that there should be a caged area that is designated as controlled flying space. Initially, there were two planned spaces, one in the cubicle and one in a spare file room. Both were built of PVC pipe, with netting and padding.

The smaller space took up a corner of the cubicle and was built to be partially on the desk with the rest of the area extending the full distance to the floor. The idea of this space was to be for quick testing of sensor data readings. The first flights of the quadrocopter were in this space, but it was quickly determined that the quadrocopter was noisy enough to be a bother in the office and this space was dismantled.

The larger space is a 10’x10’x8.5’ cube in a spare file room. This area is made of 1.25” PVC pipe, with wood supports on the corners and foam padding on the structure. Baseball netting covers the whole structure, stretched taught to be able to catch and contain the quadrocopter; the floor is covered with pillows, which were the cheapest and softest material to act as padding. A great feature of the quadrocopter is the soft propellers, as they do not cut through the netting or pillows when they make contact. When impacting the netting, the propellers become caught, stopping the quadrocopter. The cage was designed such that it can be dismantled and moved if necessary.

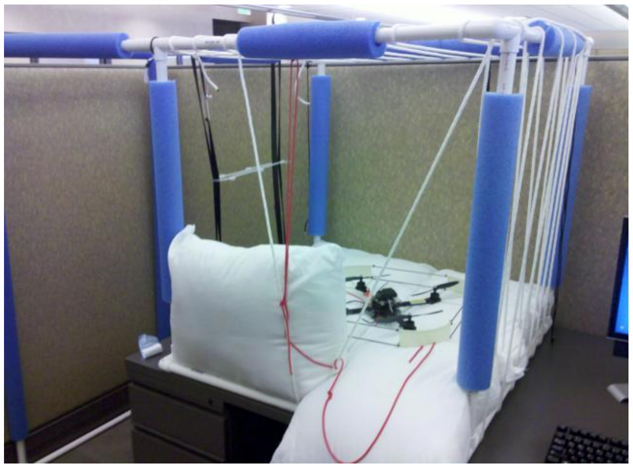
Figure 3. Small Flying Area. *This image shows the quadrocopter inside the small flying area, before it was dismantled.*

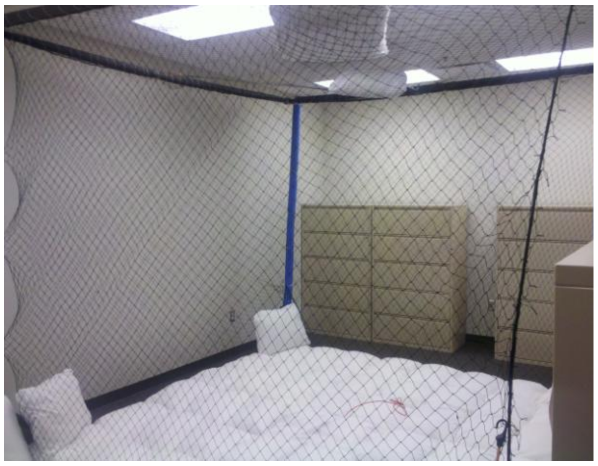
Figure 4. Quadrocopter Cage. *This image shows the completed large flying area, with the pillows and netting.*

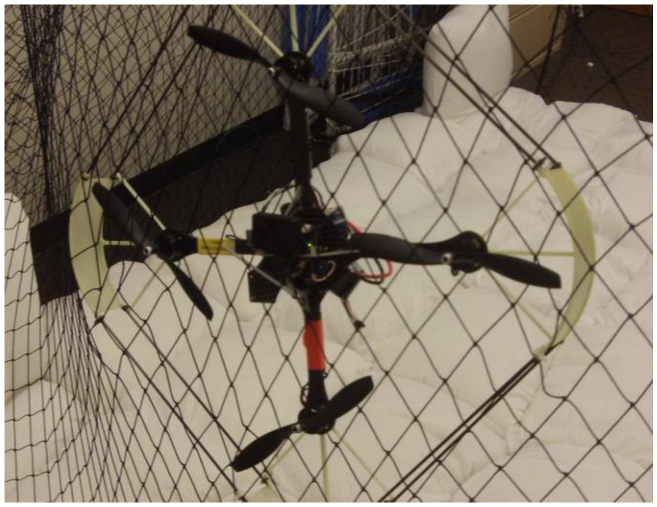
Figure 5. The Effectiveness of the Cage. *After hitting the net, the quadrocopter is stuck, effectively stopping it.*

C. Mode Testing

The quadrocopter comes with three different modes of operation: manual, height control, and GPS. Height control and GPS are intended for outdoor use, so the industrial safety department cleared the quadrocopter to fly in an outdoor location, on the softball field at MSFC. The grass there was short and soft, making it a good location. While outdoors, all modes were tested, as well as maneuvering capability, full speed, and range.

1. *Manual Control*
   The majority of time was spent in this mode, in which the pilot controls all aspects of the quadrocopter through the controller and it flies much like a normal RC aircraft. For safety reasons, a pilot must learn to fly in manually before using any of the other modes.

2. *Height Control*
   AscTec recommends that this mode is only used outside. In this mode, the throttle commands an ascend/descend rate instead of thrust, and when the stick is centered, the quadrocopter will stay at a constant height.

3. *GPS Mode*
   This mode can only be used outside, as the GPS unit needs a clear view of the sky to work. Height control is enabled, the system uses the GPS to hold position, and roll/pitch/yaw maneuvers are speed controlled to 2 m/s. The user can send waypoint commands from the computer to the quadrocopter. By sending a list of points to visit (waypoints), a path is created for the quadrocopter to follow.

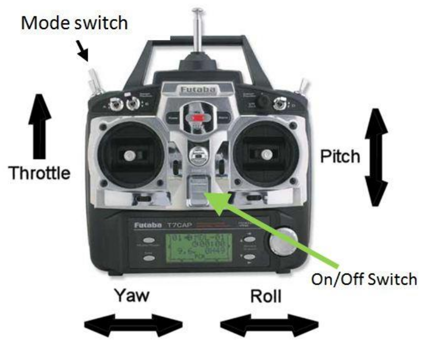

Figure 6. Radio Controller[2]. *Using the Futaba 7-channel transmitter, the quadrocopter can be controlled in all three axes, with throttle and mode control as well.*

III. Control Design

A parallel part of this project was to do control work with a computer model of the quadrocopter. Control design seeks to make a system respond as precisely as possible to a given input, regardless of dynamics or disturbances. A control system uses sensors to measure the state of the plant (the process being controlled), determines the difference between the measured and desired values, and adjusts the plant accordingly with actuators[3]. In this case, the plant is the quadrocopter and the actuators are the motors and propellers.

A. Quadrocopter Model

Before any control design can be done in simulation, the plant must be modeled. A common form of modeling a plant is through a state-space model, which takes the differential equations defining the system and puts them in a matrix format that can be used to better manipulate the system[4]. State-space models are traditionally represented as $\dot{x} = Ax + Bu$ and $y = Cx + Du$ (Ref. 4), but this representation is only applicable to linear systems. When a plant is non-linear the representation changes to $\dot{x} = f(x) + G(x)\cdot\nu$ (Ref. 1). Below is the non-linear state-space representation of the quadrocopter plant[1]. In this model, $x = (\phi \;\; \dot{\phi} \;\; \theta \;\; \dot{\theta} \;\; \psi \;\; \dot{\psi})^T$. A more thorough explanation and derivation of this model can be found in Ref 1. The control systems were all formed and tested around a Simulink[5] version of this model.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_2 x_4 \tan x_3 + \dfrac{x_4 x_6}{\cos x_3} \\ x_4 \\ -x_2 x_6 \cos x_3 \\ x_6 \\ x_2 x_4 \dfrac{1}{\cos x_3} + x_4 x_6 \tan x_3 \end{pmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & \sin x_1 \tan x_3 & \cos x_1 \tan x_3 \\ 0 & 0 & 0 \\ 0 & \cos x_1 & -\sin x_1 \\ 0 & 0 & 0 \\ 0 & \dfrac{\sin x_1}{\cos x_3} & \dfrac{\cos x_1}{\cos x_3} \end{bmatrix} \cdot \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} \Phi \\ \Theta \\ \Psi \end{pmatrix} \tag{1}$$

The state-space model represented by Eqs. (1) were formulated into the Simulink model in Fig. 7. In the following models, R2011a edition of Matlab/Simulink was used. Figure 8 is the upper level of the Simulink program in which the control programs were run. The model in Fig. 7 is the contents of the Quad Plant block; the control design was done in the Attitude Control block, and the contents of the block for each control system are found in the following section. A constant command was fed into the system and the data was output into the Matlab workspace for analysis.

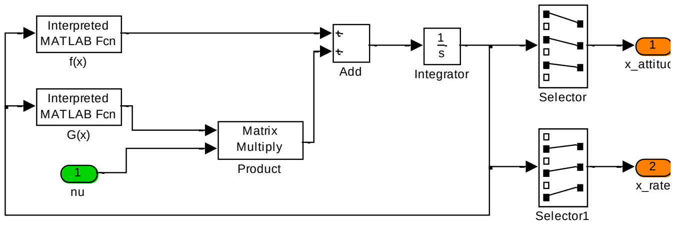

Figure 7. Quadrocopter plant model. *Using Eqs. (1), this Simulink diagram was created as a plant model for control design.*

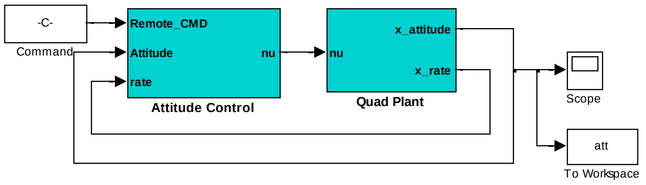

Figure 8. Quadrocopter Control System. *The control design is done in the attitude control block, based on the plant's response to a given command.*

B. PID Control

A common form of control system uses proportional (P), integral (I), and derivative (D) components. The proportional component consists of the difference between the commanded and measured values (the error). The integral component is the integral of the error, and the derivative component is the rate. There are four variations of this, proportional only (P), proportional-derivative (PD), proportional-integral (PI), and proportional-integral-derivative (PID). Each component has a gain associated with it (*Kp, Ki, Kd*). By adjusting the gains of each component, the system response can be tuned.

Controllers are often tested by inputting a step function and analyzing the results. A step function is 0 for time less than 0 and some value for time greater than 0. In the following graphs, a step input of 10 degrees was used. There are four major characteristics of closed loop step response that are used as specifications to tune a system, all related to the steady-state, which is the value the system settles to. Rise time is the time it takes for the output to reach 90% of the desired value; overshoot is how much the peak is above the steady-state; settling time is the time it takes to converge to steady-state; and the steady-state error is the difference between steady-state and the desired value[6]. For this controller, the goals were to have a rise time of less than 0.5 s, an overshoot of less than 10%, a settling time of less than 1 second, and a steady-state error of less than 1 degree. These specifications were chosen because they are reasonable expectations for the performance of the quadrocopter. Each gain has a specific effect on each of these characteristics, summarized in Table 2, so tuning a system is about balancing the effects. When tuning any kind of PID loop, the first gain to start with is the *Kp* gain, using it to decrease the rise time to within specifications. The next step is to use *Kd* to reduce the overshoot and settling time, but this will increase rise time, so it is a process of increasing each gain in increments to see the effects. The last step is to tune *Ki* to eliminate -state error.

Table 2. Effects of PID gains on Response Characteristics[6]

	Rise time	Overshoot	Settling time	Steady-state error
Kp	Decrease	Increase	Small Change	Decrease
Kd	Small Change	Decrease	Decrease	Small Change
Ki	Decrease	Increase	Increase	Eliminate

Figure 9 shows the attitude control block of a proportional controller. There is no rate input as seen in Fig. 8, and the thrust command (the 4th value of the given command vector) is not used in this model. The block Flip is from the Simulink software that was purchased with the quadrocopter[7] and accounts for the full-circle nature of the yaw axis.

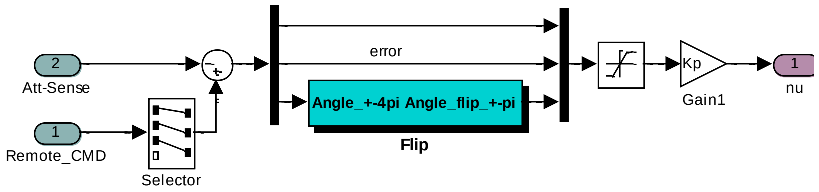

Figure 9. Proportional Control Loop. *This is the attitude block for a simple proportional controller.*

Figure 10 is the output of simulating the model with the control loop in Fig. 9, using varying values of *Kp*. Only the phi (roll) axis was used for the purpose of testing the controllers. All the following PID testing graphs use the same settings unless otherwise noted. The black marks on the graph indicate the command input of 10 degrees and the tuning goals. The mark at (0.5,9) indicates the rise time goal, the middle line indicates the commanded angle, and the lines above and below mark the overshoot and steady-state parameters, respectively. When *Kp* equals 10, the model meets the rise time goal.

The next step in control design is to add a derivative, or rate, component to the control system. Figure 11 shows the attitude control with PD design. The attitude loop block is nearly identical to Fig. 3, and the rate feedback is multiplied by *Kd* and subtracted from the result of the proportional component.

Kp=10 produced the desired rise time, so the PD model ran with *Kp*=10 and varying *Kd* values, as seen in Fig. 12. *Kd*=4 brings the model within the desired overshoot limit, but increases the rise time, so *Kp* must be increased to compensate. Figure 13 shows the model with various values of *Kp* and *Kd*, with *Kp*=25 and *Kd*=6 bringing the model into accordance with the desired performance for both rise time and overshoot.

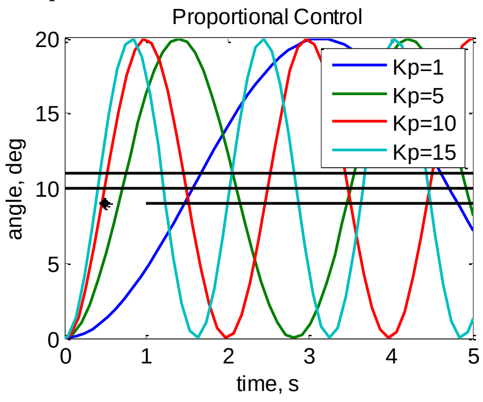

Figure 10. Proportional Control. *A simulation of the model with just a proportional controller, with varying values of Kp.*

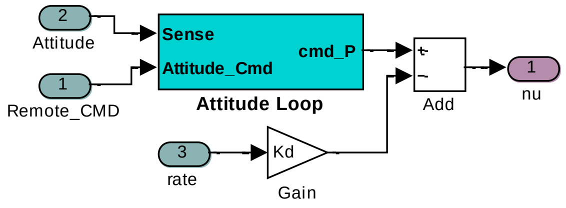

Figure 11. Proportional-Derivative Control Loop. *This is the attitude control block for a PD controller, with a P controller in the attitude loop block and a rate component outside of it.*

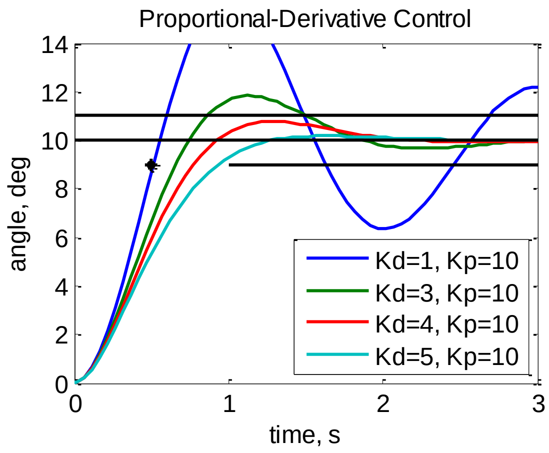

Figure 12. Proportional-Derivative Control. *A simulation of the model with a PD controller, with Kp=10 as found above.*

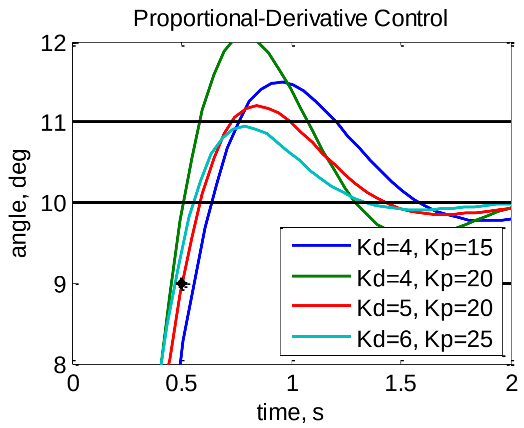

Figure 13. Refined PD control. *Simulation of the model with varying Kp and Kd values, with Kd=6 and Kp=25 fitting the overshoot and rise time specifications.*

Proportional Integral control is also a common control system design; Fig. 14 shows a PI attitude control design. There is no rate component; instead, the error is integrated and multiplied by *Ki* and then subtracted from the proportional control component.

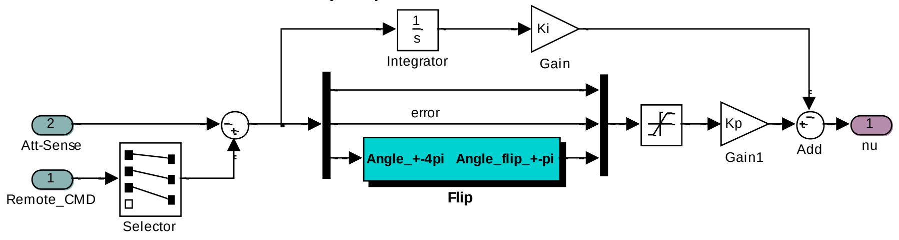

Figure 14. Proportional-Integral Control Loop. *This is the contents of the attitude control block for a PI controller.*

As seen in Fig. 15, simulating the model with a PI controller was an ineffective control solution. The integral component did not solve the oscillations in the system. Low *Ki* values decreased the oscillations, but increasing the value led to integrator windup, resulting in a drop to negative infinity. Changing the *Kp* value with the same *Ki* value also did not have a desirable effect.

Despite the results of the PI controller, for the sake of completeness, a PID controller was also tested on the system.

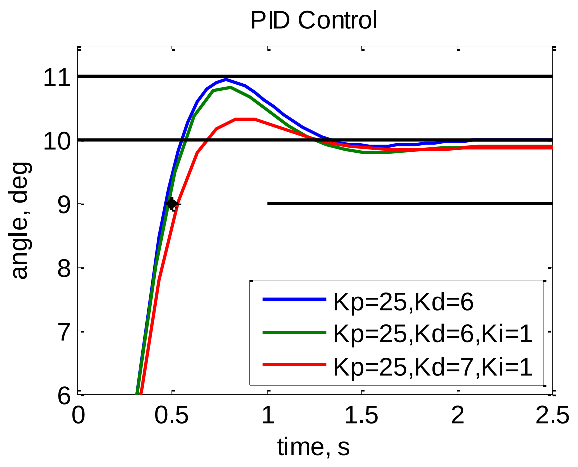

Figure 16. PID Control. *Simulation of the model with proportional, integral, and derivative control components.*

Figure 16 compares PD controller with *Kd*=6 and *Kp*=25 with similar PID controllers. The PID controllers reduce the overshoot slightly, but they also induce a steady-state error, which is not a problem with the PD controller, so a PID controller is not necessary. However, the *Kd*=6 and *Kp*=25 controller meets only three of the design specifications, as the settling time is greater than 1. It is also possible to improve the performance on the other specifications, so new specifications were selected.

As using higher gains for *Kp* and *Kd* leads to better a better system response, new parameters were selected and gains tested to fit the new specifications, which were rise time less than 0.25 s, overshoot less than 1%, settling time less than 1 s and no steady-state error. Figure 17 shows a few comparisons for these parameters. *Kp*=225 and *Kd*=25 fulfilled these specifications. It is worth noting that these gains happen to be close to the gains used in part of the first control system in Ref. 1, but are not exact due to the complexity of the system used in that work.

Next, even higher gains were tested. The results were that system performance could be brought to less than 0.02 s rise time, no overshoot, no steady-state error and settling time less than 0.04 s. However, while the gains tested in these simulations produced good performance, such large gains compromise robustness and may be limited by the fixed-point nature of the quadrocopter microprocessors. Fixed-point numbers have less precision and a limited range of values as compared to some of the other common number types, such as single and double.

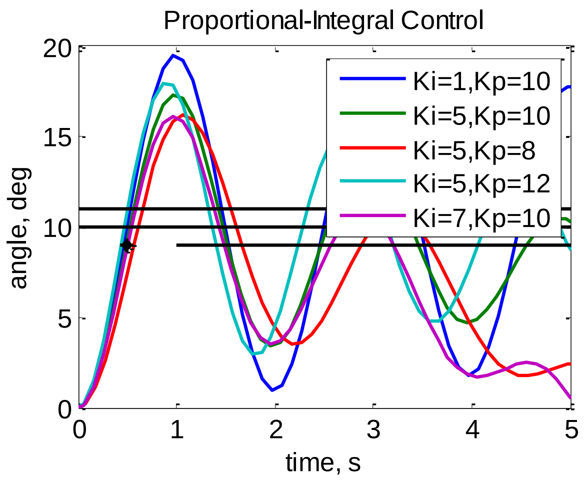

Figure 15. Proportional-Integral Control. *Simulation of the model with varying Kp and Ki values.*

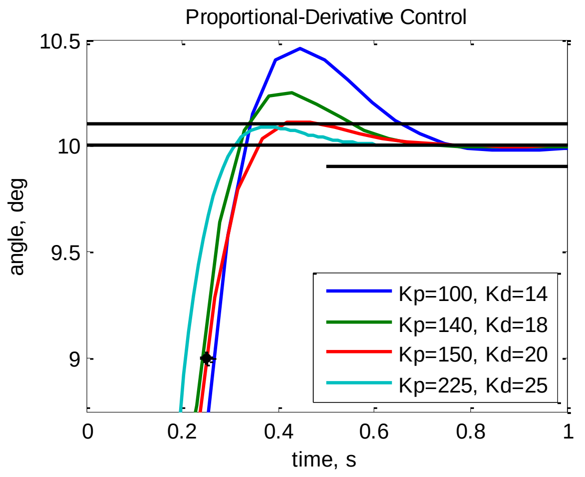

Figure 17. Further PD Testing. *This figure shows using higher gains to further refine the system response.*

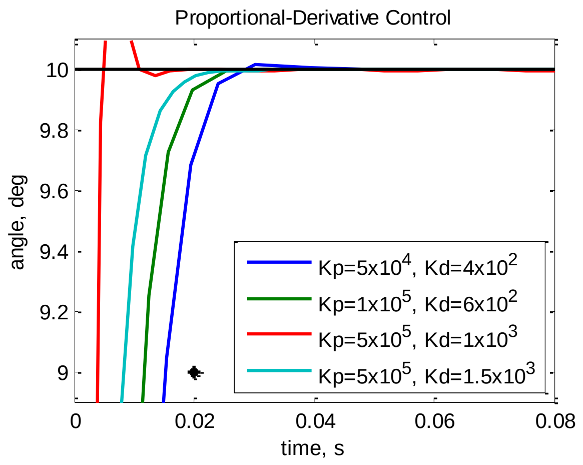

Figure 18. High Gain Testing. *These simulations were run to test very large gains.*

IV. Flight Operation

The remaining part of this project was to combine the hardware and software. One of the powerful features of the AscTec Hummingbird is the ability to have user-loaded code on the HLP. In the long run, the main purpose of the quadrocopter is as a test bed for custom code, so it was important to set up that part of the system. This involved installing and learning software and model from the manufacturer, and receiving data back from the quadrocopter.

A. AscTec Model

The quadrocopter was purchased with a Simulink interface software package. This AscTec SDK[7] (Software Development Kit) included a Simulink model that, when converted to C code, is loaded onto the microprocessor using Eclipse software. The SDK also included a model to receive data back from the quadrocopter, as well as many supporting files.

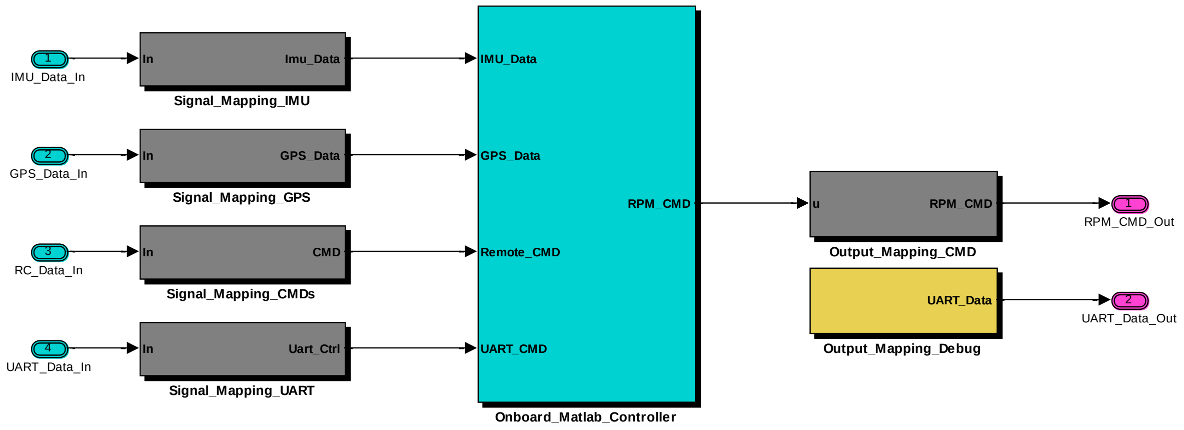

Figure 19. Simulink Quadrocopter Framework[7]. *This Simulink block diagram is the top level structure of the Simulink Quadrocopter Framework that was purchased with the quadrocopter.*

The model shown in Fig. 19 gives an overview of the code framework that runs on the quadrocopter. This model was designed for the R2010b edition of Matlab and was mainly run in that version, but work has been initiated to convert it to R2011b. Data from the IMU, GPS, RC controller, and UART (Universal Asynchronous Receiver Transmitter, a communication interface[1]) are input and can be used as needed. Coding is done in the block Onboard_Matlab_Control, shown in Fig. 20. This block can be modified as needed (or replaced completely) by the user's program. The block Attitude Control contains a similar control system to the ones discussed in the PID control section. The framework model can incorporate parameters that can be sent to the quadrocopter and has many debugging channels and customizable options. Using Real Time Workshop (now called Simulink Coder), this model can be translated into C code. Using the Eclipse software and settings that came with the AscTec SDK, as well as the JTAG adapter (Joint Test Action Group, a programming interface hardware[1]), the code can be sent to the quadrocopter and debugged.

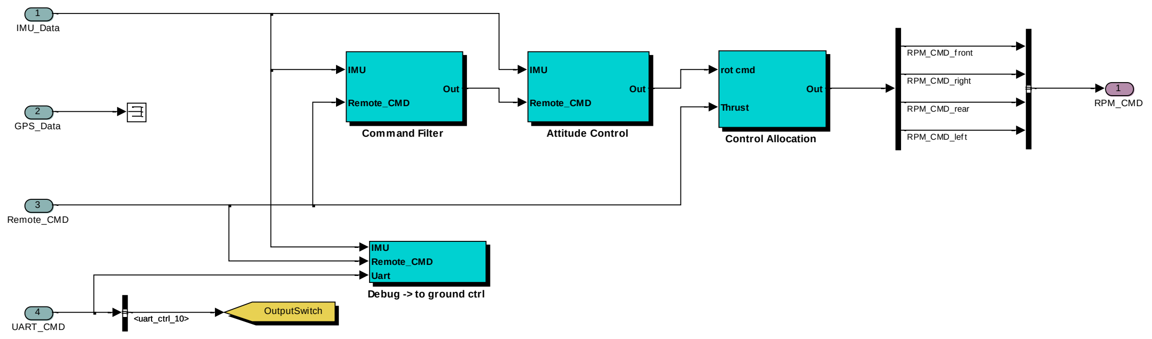

Figure 20. Onboard_Matlab_Controller[7]. *This model shows the contents of the subsystem Onboard_Matlab_Controller from Fig. 19. This is where the user control program would go.*

After initialization and transmission of parameters, the quadrocopter is ready to fly. The quadrocopter uses two wireless communication devices called Xbees. It can be used with one or two, but this system is set up with two, one for transmitting and one for receiving, enabling faster data transfer. The model in Fig. 21 shows the Simulink interface model that can send control channels and receive data from the quadrocopter.

The blocks Quad_Receive and Quad_Send contain C code s-functions to interface with the Xbee hardware. The constant blocks connected to the Quad_Send block are the commands sent to the quadrocopter. Their function can be customized in the framework model. The data coming back from the quadrocopter can also be partially customized in the framework. There are 60 debug channels to choose from, but only 20 channels are sent back to the computer at a time, with a rate of 50 Hz. The first block of ten is transmitted every cycle, including the IMU attitude and rotation, as well as the commanded values from the attitude control loop. The second block of ten is chosen from the remaining five blocks; it can be the same channels every time or it can loop through the blocks of channels as defined by the user. A signal was sent via a control channel and successfully received by a debug channel.

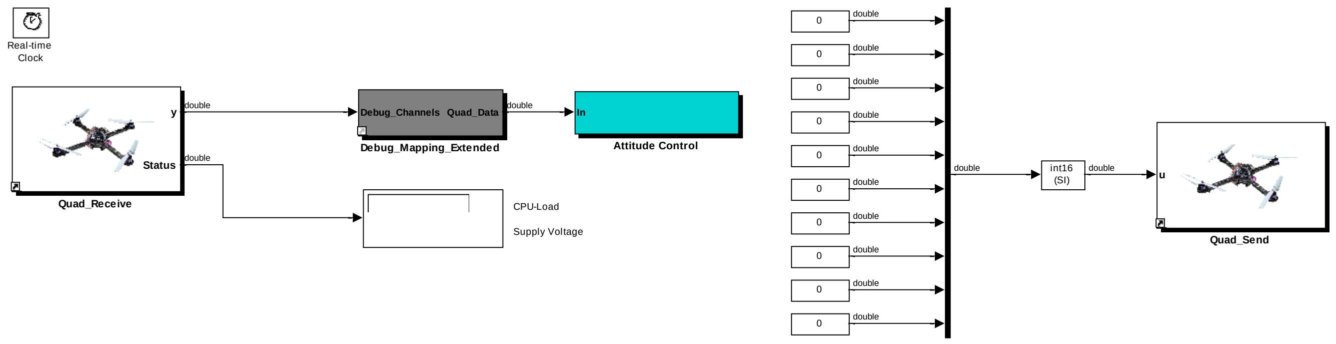

Figure 21. Xbee-Quadrocopter Interface[7]. *This model uses the Xbees to interface with the quadrocopter, sending commands and displaying the data.*

The setup in Fig. 21 displays the CPU load and battery voltage in the model. In the Attitude Control block, there are also scopes displaying the IMU data for attitude and rate, the commanded rates from the attitude control algorithm on the copter, the stick commands, and the resulting motor commands. This section was modified to output the data into Matlab for further analysis.

B. Quadrocopter Data

When flying the quadrocopter, the computer was set up to collect data, which was then analyzed in Matlab. In Fig. 22, the stick input and the response from the attitude loop are compared. Because the stick input is from -1 to 1 and has no real units, it was scaled to be displayed with the attitude loop data. The quadrocopter was purposefully flown in such a manner as to produce the regular oscillations shown. Figure 23, from a different dataset, displays the reaction of the quadrocopter to the given input. The data is from normal flight, so there is no pattern to it. The quadrocopter rotation readings follow the attitude loop input with minor lag, but there is a lot of noise in the signal, whether it is from the sensor or the actual movement of the quadrocopter.

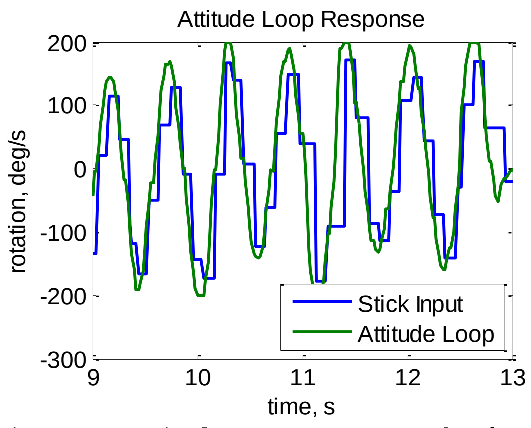

Figure 22. Attitude Loop Response. *This figure compares the scaled input of the stick on the remote control to the commanded rate from the attitude loop.*

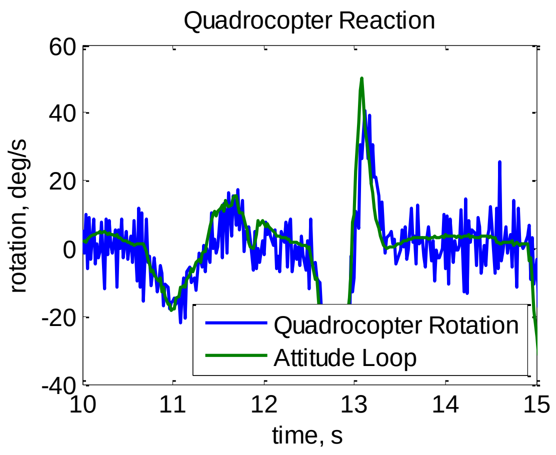

Figure 23. Quadrocopter Reaction. *This figure compares the commanded rotation from the attitude loop with the IMU rotation data.*

The data in Fig. 24 compares the commanded thrust value with the response of the motors. The thrust was taken up to maximum and then brought back down, during which the motors follow the signal with a varying degree of accuracy. When the thrust signal increased, the motor commands deviated, whereas at low thrust values, the motors followed the signal closely. As the thrust is scaled from -1 to 1 coming from the remote control and has no physical unit associated with it, it was scaled to the range of the motor commands. However, the motor commands are also of an arbitrary unit system, as that command is subsequently fed into the motor controllers on the quadrocopter, which take the input and convert it to the appropriate RPM value for the motor.

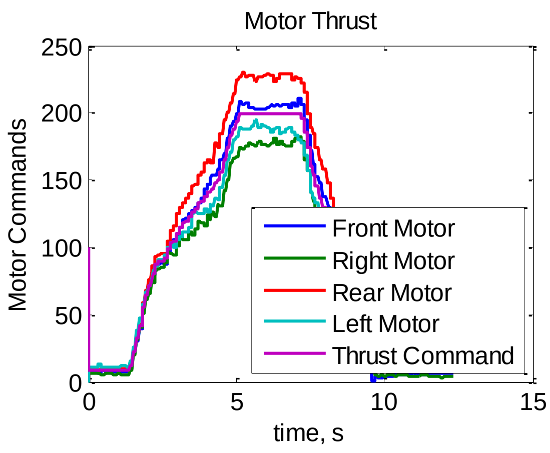

Figure 24. Motor Thrust. *This figure compares the thrust command from the remote control with the resulting motor commands.*

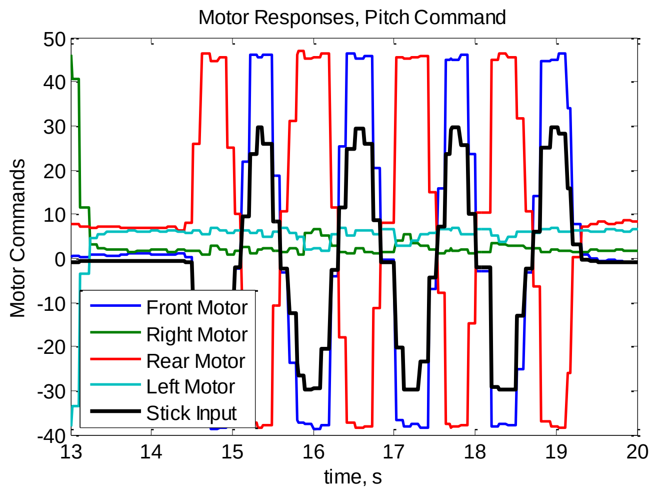

Figure 25. Motor Response to Pitch Command. *By oscillating the pitch command, the correct motor responses were confirmed.*

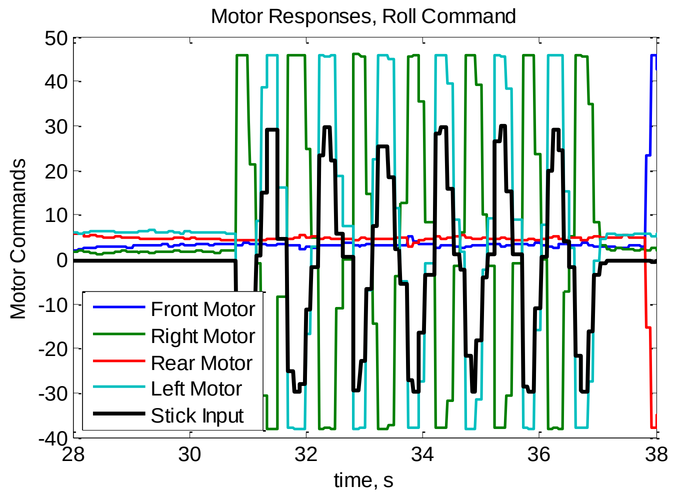

Figure 26. Motor Response to Roll Command. *Confirmation of the correct motor responses to a roll command.*

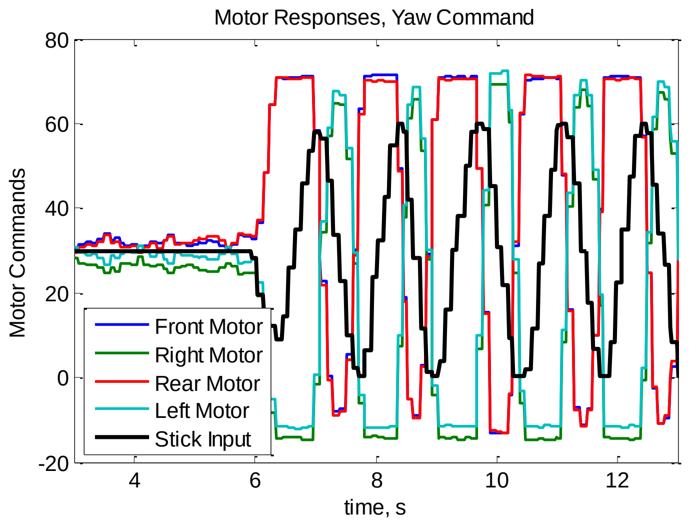

Figure 27. Motor Response to Yaw Command. *As with the pitch and roll figures, the yaw command was oscillated, but as yaw has a different dynamic, all four motors engage as expected.*

In Figs. 25, 26, and 27, the command signal for one axis was oscillated using the RC controller. The resulting motor responses were checked to confirm the response. For the pitch command (Fig. 25), only the front and rear motors are engaged, with the front having the opposite value of the rear. For the roll command (Fig. 26), the right and left motors are engaged. In Fig 27, all four motors are engaged to respond to the yaw command, as explained earlier. The clockwise spinning motors (front and rear) are paired, as are the counter-clockwise motors (left and right), and these pairs oscillate in an opposite manner. This axis has a noticeable lag, as opposed to the other two axes. As pitch and roll both only use two motors, the dynamics of these axes are similar. As is brought up in Ref 1, because the yaw axis is controlled by all four motors, it has different dynamics, causing the different response.

C. Control Variations

The model in Fig. 19 is fully customizable, so steps were taken to make modify it in small increments. The first variation was to put a switch system in place so that one of the control channels in the model in Fig. 21 could be used to send a thrust command to the quadcopter instead of the RC controller. The second variation tried was to have the model output the height data from the IMU back to the computer so that it could be analyzed to estimate how high the quadrocopter could fly when outside. Another variation was to adapt the PD controller model from the earlier control section back into the model in Fig. 19. This was the most successful variation, as the code was run on the quadrocopter and parameters were sent to the quadrocopter real-time. However, all of these variations, and other possibilities, are deserving of more time than could be afforded to them in this project. All of the variations mentioned were coded into C and run on the quadrocopter, but none of the model variations performed quite as expected, and much care had to be taken not to disturb any of the important processes in the model when making changes.

D. Camera Use

Some research and was done into mounting a camera on the quadrocopter and learning to fly it through the first-person view. A small Bluetooth camera was attached on top of the quadrocopter and a live video feed was received while flying. A Simulink diagram that overlays a heads-up display (HUD) onto the video feed from the quadrocopter was also started, but figuring out how to plot data onto such a display proved to be a rather difficult task to complete in Simulink. Contacting Mathworks about this application produced some ideas, but there was no time to explore this further.

V. Conclusion and Future Work

Through this project, the quadrocopter system was successfully set up for future use, fulfilling the primary goal. By establishing a space to fly in and protocol for using the vehicle, others will be able to operate the quadrocopter easily and safely. Documentation of the processes involved in flying, maintaining, and programming the quadrocopter will make it easier for people to become involved in the project and keep the project running after this summer. Through learning to design a simple controller for the quadrocopter, a model has been created for computer simulation of the quadrocopter, and a PD controller was tested. Perhaps the biggest accomplishment of this project was the combination of hardware and software—using the AscTec SDK to program and edit code, and then load and test it on the quadrocopter. Through receiving data back, the system was validated and debugged.

As the purpose of this project was to set up the quadrocopter system for future use, this project has many future possibilities. A further step for testing GPS control would be to use the waypoint command feature. As for control design, a more optimal PD controller should be considered, making the best controller within the limitations of the gains. The controllers should also be verified for all axes and for values close to saturation. More advanced control algorithms and new control designs should be tested on this vehicle in the future, as that is the purpose for this quadrocopter. Besides the control design, much more work can be done to customize the framework model and Xbee model for data input/output from the quadrocopter. The onboard camera and HUD system also bears further development. One other promising option for using this quadrocopter is navigation. There has been an idea about using a camera-equipped room to track the quadrocopter for indoor navigation, and/or using an onboard camera for tracking and navigation. The testing of navigation sensors and algorithms may prove to be the most valuable use of the quadrocopter. The goal is to get more people involved, and as that happens, more ideas will be brought to the quadrocopter project. The project is open to exploration of ideas and improvements in control and navigation logic and is a cost-effective means of testing alternative control programs.

Acknowledgments

I would like to thank the following people:
Mike Hannan for his vision with the quadrocopter, and for his help, mentorship, and guidance.
John Rakoczy, for his mentorship and insight this summer.
Matthew Carter, for his enthusiasm for the project and his help with debugging.
Tannen VanZwieten, for her knowledge, patience, and encouragement when helping me with control theory.
Everyone else at EV41 who helped out and got involved with the project.
Michael Achtelik (Ascending Technologies), for his help in getting this project started, and for providing his thesis and an initial Simulink model for the quadrocopter.
The NASA USRP program for allowing me to come discover and learn about the opportunities at NASA.

References

[1]Achtelik, M., “Nonlinear and Adaptive Control of a Quadcopter,” Dipl.-Ing. Dissertation, Lehrstuhl für Flugsystemdynamik, Technische Universität München, Garching, Germany, 2010.

[2.]“AscTec Hummingbird with AutoPilot User’s Manual.” 2011. Ascending Technologies GmbH. 6 July 2011 <http://www.asctec.de/assets/Downloads/Manuals/AscTec-Autopilot-Manual-v10.pdf>

[3]Avallone, E. A., Baumeister III, T., Sadegh, A. M. (ed.), *Mark’s Standard Handbook for Mechanical Engineers,* 11th ed., McGraw Hill, New York, 2007, Section 16, pp 22, 24.

[4]Leigh, J. R., *Control Theory,* The Institution of Electrical Engineers, London, 2004, Chap. 10.

[5]Matlab & Simulink, Software Packages, Ver. R2010b & R2011a, MathWorks, Natick, MA, 2011.

[6]PID Tutorial.” 1996. The University of Michigan. 14 June 2011 <http://www.engin.umich.edu/group/ctm/PID/PID.html>.

[7]AscTec SDK & Simulink Quadrocopter Framework, Software Package, Ver. 2011, Ascending Technologies, Germany, 2011.