Quad-Rotor UAV project

Final Report

Prepared for:

Professor Eric Klavins

Charlie Matlack

DSSL Lab

Prepared By:

Justin Palm

Andrew Nelson

Andy Bradford

June 11, 2010

Executive Summary

Quad rotor UAV Project: final report is the final milestone report for EE 449: Controls System Design. A quad rotor test bed has been built that utilizes an infrared vision system to track the position of the vehicle. In order to have a stable system, the quad rotor dynamics are quantified and a controller designed to meet the performance specification of a 10 second step response.

While the system simulation shows results that meet the above requirement, the system implementation has proven unsuccessful. An error analysis of the problem reveals a non-linear mapping between the motor control input and the angular velocity of the rotor blades. This fact combined with an to small of a resolution for pulse-width modulation results in a range of velocities that *cannot* respond to the perturbation of hovering conditions.

Further research is has been conducted on the problem and is linked specifically to the opto-isolator being driven with too high of a voltage giving a fall time that is far too slow. Given additional time and patience, the problem could be fixed by replacing the opto-isolator with a faster device and using a micro controller capable of higher PWM resolution.

Table of Contents

Table of Figures

List of Tables

Project Description

The quad rotor UAV project is was originally selected because of the complexity and *wow* factor. There have been a few other colleges in the world who are dedicating enormous resources to similar projects. Being part of a team that establishes a working test bed for the University of Washington has been a great experience. The project itself is straightforward: using a hobby quad rotor frame, design, build, and test hardware and software control structure that uses feedback to achieve autonomous flight. Actual implementation has proved to be not quite as straightforward. The quad rotor project is a very ambitious task to complete in ten weeks. The trade off in the complexity of the project is enormous intellectual gains.

Customer

Our customer for this project is Professor Mehran Mesbahi who heads up the DSSL lab here at the University of Washington. His lab focuses on numerous areas of controls in engineering such as guidance, navigation and control of both single and multi-platform aerial (and space) systems. Our customer requested that we design a control system able to autonomously fly a quad rotor to a specific location, or waypoint, in 3D space utilizing the overhead Vicon positioning system.

Project Plan

The project plan was divided into five major milestones each spaced approximately two weeks apart.

1) Project Description and Plan of Work
2) System Model
3) Controller Design
4) Controller Implementation / Hardware / Software
5) Project Demonstrations

The sequence that we met these milestones was out of sequence with the required milestones. Experience told us to get the hardware done as soon as possible as this is often requires a lot of debugging time. By doing so, and because of unforeseen difficulties, we fell behind slightly with the System Modeling and Controller design. After working closely with our customer and other professionals we were able to complete the milestones only slightly behind schedule. The final implementation was time stressed toward the end of the quarter and we failed to achieve autonomous hover. In lieu of this, we have done extensive failure analysis and is included later in the report under sections *Experimental Data* and *Further Development.*

Literature review and related work

The quad rotor project required extensive research into similar systems. By reviewing others work, we used this insight to develop our system. To this end, research papers from various quadrotor groups were used as guides in the early development of the dynamics and control theory.

Quad rotor platforms used in research remain somewhat the same, having four electric motors pointed vertically upwards and equally spaced in a square fashion. However, there were some groups whom

designed their own platforms, where as commercial models available to the consumer were the DraganFlyer, the X-UFO and the MD4-200.

One such platform is the Stanford Test-bed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) uses a modified DraganFlyer IV quad rotor (the same frame as we are using). Their system uses LQR techniques as well as Integral Sliding Mode (ISM) control. The STARMAC also incorporated an onboard micro controller/IMU.

The single most valuable resource to us during this project was the work that Brain Hemstra did while working with the DSSL lab last year. Brian's master thesis entitled *Linear Quadratic Methods Applied to Quad rotor Control* provides a working simulation that we used to gain insight and understanding while working through the complicated dynamics of the quad rotor system. Another big advantage of having this resource is that many of the parameters of interest (thrust, inertia, etc) were already well quantified.

System model

In this section, the quad rotor system's multiple inputs and outputs will be identified and the equations of motion governing the dynamics of the system will be derived. The first step is to identify the plant and the inputs and outputs of the plant itself and then to establish the framework to derive the equations of motion using Newton's Laws.

Plant Identification

Figure 1: Plant Identification

Figure 1 shows the inputs and outputs of the plant. The inputs are angular velocities in radians per second and the output is a 12x1 vector which is discussed below. The angular velocity is converted through an airfoil blade. The Dynamics of the airfoil blade is included in the dynamics of the plant. In order to control the angular velocity of the blades, we must control the voltage applied to the motors in the quad rotor. Figure 2 below shows how the motors can be controlled through pulse-width modulation (PWM) the axel of the motor is geared with a five to one reduction gear. The result of varying duty cycles of the motor drive stage results in varying angular velocities.

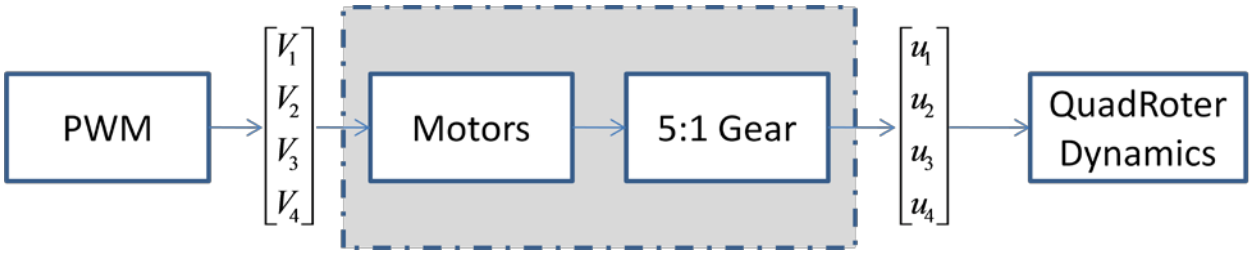


Figure 2: Controlling angular velocity via PWM

Reference Frame

The quad rotor system operates in two coordinate frames: inertial and body. The inertial frame (also referred to as the earth frame) is the coordinate axis where Newton's Laws apply. To complicate matters, the countering forces to achieve hover are applied to the body frame which is fixed to the quad rotor itself and is allowed to rotate and translate. This dual-frame coordinate system is shown below along with a free body diagram of the quad rotor system.

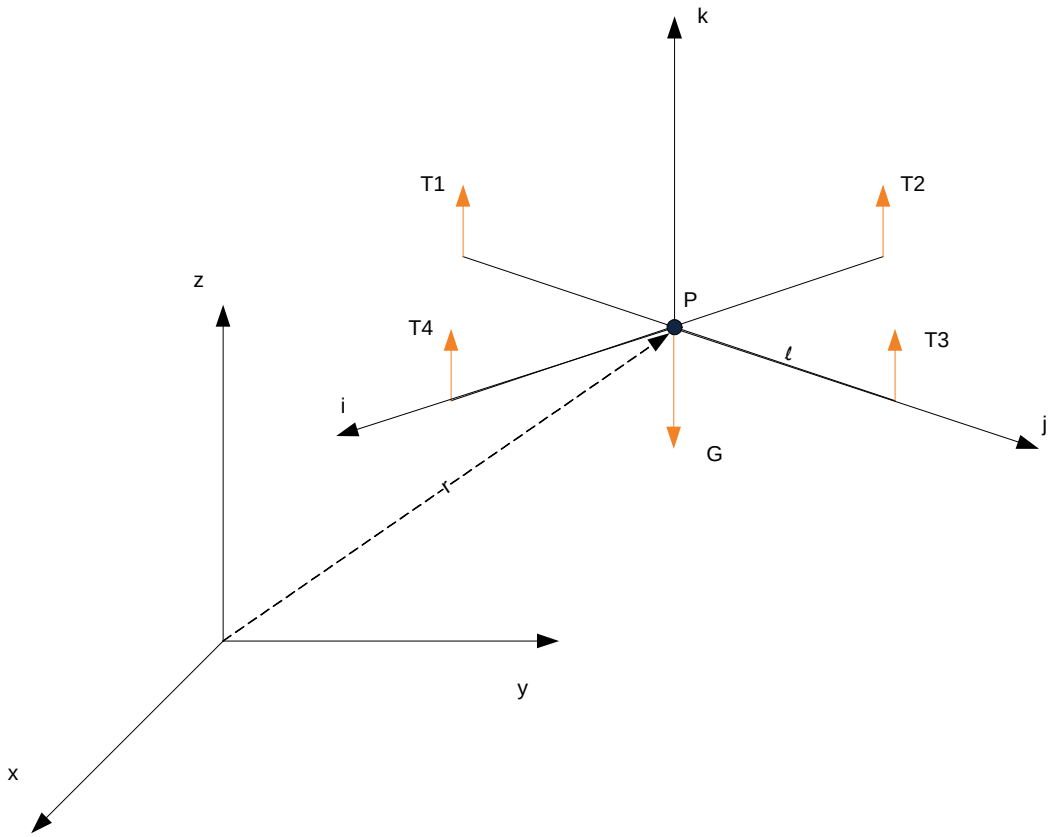


Figure 3: Coordinate Axis and Free-body Diagram

Now that the coordinate reference frame is identified we can begin to represent the system mathematically. The Mechanism through which the quad rotor can be controlled is thrust provided by

airfoil blades attached to four independently controllable motors attached at a fixed distance (ℓ) from the center of the quad rotor (P). By varying the relative magnitudes of the thrusts, we can control the attitude (yaw, pitch, roll) and position (X,Y,Z) of the system in inertial coordinates. As mentioned previously, the thrust forces are applied in the body frame; therefore, transformations must be made.

Model States

In the previous subsection, we showed how the quad rotor system can be described using body coordinates and inertial coordinates. Now we define the states of the system that include a mixture of body and inertial components comprised of translational and rotational positions and velocities. Referring to Figure 1, we define the following vectors:

$$v^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$ translational velocity in the body frame

$$\omega^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$ rotational velocity in the body frame

$$\Phi^i = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$ attitude (yaw,pitch,roll) in the inertial frame

$$r^i = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ position in inertial frame

Combining the four vectors defined above yields the state vector $x = \begin{bmatrix} v^b & \omega^b & \Phi^i & r^i \end{bmatrix}^T$ which is used in the derivation of the quad rotor dynamics shown in the appendix

State-Space Representation

The basic form of the state space equations are as follows:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

Where u is a vector of system inputs and x is the state vector. Because the quad rotor is an unstable system, we must linearize the system about an operating point. To achieve a hovering condition, we effectively want all the states to be zero. Stated differently, if we think of the initial starting condition at some position in space (X,Y,Z) and call that point zero, all the allowed states should also be zero. This is the easiest point to linearize about since many of the elements after the linearization process go to either zero or one.

In order to expedite the linearization process, Matlab is used by creating a model of the system that includes the non-linearized equations of motion. This model is shown below as well as the inner block below. Equations of motion written in code are included in the appendix.

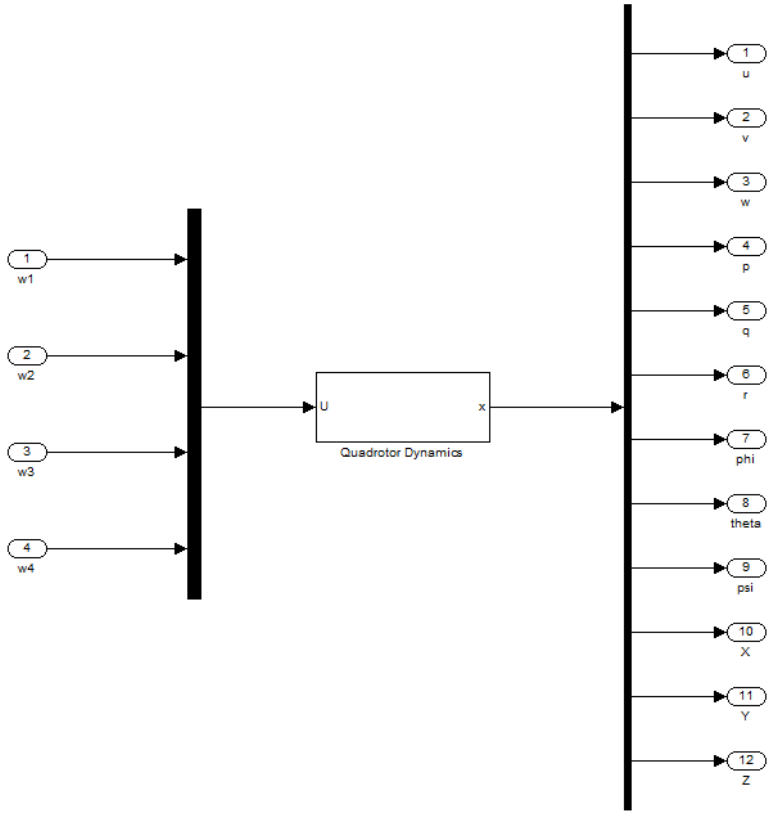


Figure 4: Plant input/output

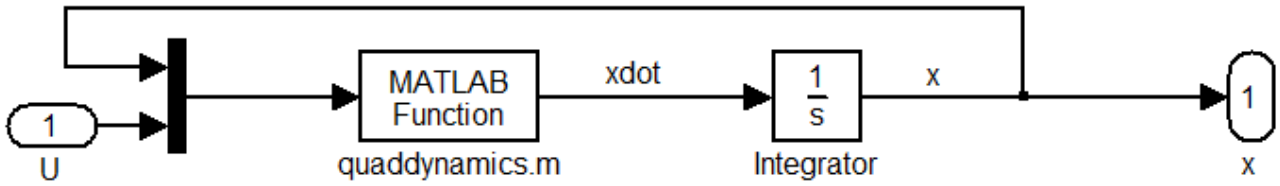


Figure 5: Matlab model used to linearize the quad rotor system via the function linmod()

Figure 2 above represents the plant with angular velocity in radians per second (rad/s) and outputs as the state vector described earlier (mixed units). The Matlab function linmod() requires a set point for the system. In order to achieve hover, the motors must be spinning at a particular speed. To find the set point one simply must set the sum of the forces acting on the system to zero.

$$F_G = F_{Thrust}$$

$$mg = \frac{1}{4} b\omega^2$$

$$\omega = \sqrt{\frac{mg}{16b}} = 82.4981\ rad/s$$

Substituting the calculated value into the linearized input equation $\omega^2 = 2\omega\Delta\omega = 164.98\Delta\omega$ where $\Delta\omega$ is the input perturbation angular velocity. This result shows the motors must spin at 164.98 rad/s for the given system mass in order to achieve hover. To verify that this is correct, the non linearized model is tested with a constant 164.94 (rad/s) applied at the input. The resulting plot is shown below.

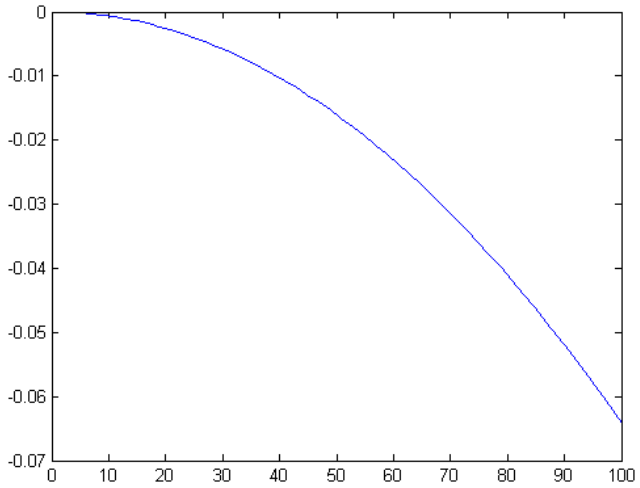


Figure 6: Open loop test for linearization set point

One can see that the value being (164.98 [rad/s]) used is fairly accurate as the altitude only drops by 6cm in 100 seconds without any feedback. Some clarification here is necessary: There is no floor on the simulation executed; that is, the quad rotor is being allowed to travel 'through the ground' just to demonstrate the correct linearized angular velocity was chosen. The plot of the linearized model is omitted here as it is simply zero throughout the 100 second simulation.

Included in are the model parameters identified in the hand derivation included in the appendix. The values shown are taken from Brian Heemstra's work last year using the same quad rotor.

Table 1: Parameters used in model

Parameter	Symbol	Value	Units
Mass	m	.589	Kg
Thrust Parameter	b	4.3248e-5	Kg m
Torque Constant	k	5.96927e-8	N m s^2
Inertial Matrix	I	$\begin{bmatrix} 6.532e-3 & 0 & 0 \\ 0 & 6.6944e-3 & 0 \\ 0 & 0 & 1.2742e-2 \end{bmatrix}$	Kg m^2
Distance to motor 1	r1	$[0.2319\ 0\ \ 0]^T$	M
Distance to motor 2	r2	$[0\ 0.2319\ 0]^T$	M

Distance to motor 3	r3	$[-0.2319\ 0\ 0]^T$	M
Distance to motor 4	r4	$[0\ -\ 0.2319\ 0]^T$	M

Now that we have a means to linearize the system through Matlab, well defined input trim conditions along with a reasonable estimate of the true model parameters, the linear model can be generated. The resulting A,B,C and D matrices below are seen to be Observable and Controllable.

Table 2: Linearized State space representations

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9.81 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -9.81 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ .0297 & -.0297 & .0297 & -.0297 \\ 0 & -.5066 & 0 & .5066 \\ -.2470 & -.2470 & -.2470 & -.2470 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad D = 0$$

Performance Specifications, Controller Design & Simulations

The performance specifications that were designated at the outset of the project versus what has been achieved are quite different. Through iterating through different control designs, we were able to simulate step responses that gave reasonable performance- though still far from our initial specifications. The over-estimated response problem comes from not really knowing what the quad rotor was/is capable of. The current performance specification is 10 second step response with no steady state error. The following sections will describe the Controller design and show the simulation results.

Controller Design

The first iteration through the controller design process used only inertial coordinates and heading (yaw). While altitude and yaw stability were stable, any horizontal command led to oscillatory behavior. This method was quickly thrown out as it does not use full state feedback, the commands could not be decoupled, and tuning the PID controllers was quite a tedious task. For these reasons, a Linear Quadratic Regulator (LQR) is used to find a gain matrix K.

The LQR method works by minimizing a cost function that allows the user to set weights on different degrees of freedom. The effect of this is creating a gain matrix that sets the closed loop eigenvalues further from the imaginary axis to increase the *aggressiveness* of the response. Often the tradeoff for this is overshoot and perhaps some steady-state error.

The Q and R matricies used in the controller that is in place on the quad rotor is shown in the appendix. The effect of the Q and R matrices is to assign a heavier weight to altitude and yaw control efforts. One can see the entries Q(9,9) corresponding to yaw control and Q(12,12) have different weights compared to the rest of the entries. This shoves the poles for those degrees of freedom further to the left. The eigenvalues for the closed loop system are shown below.

Table 3: Eigenvalues of the closed-loop system

-3.3801
-1.6174 + 2.3011i
-1.6174 - 2.3011i
-3.4233
-1.6315 + 2.3073i
-1.6315 - 2.3073i
-0.5667 + 0.5493i
-0.5667 - 0.5493i
-1.0010
-1.0009
-0.2327 + 0.2326i
-0.2327 - 0.2326i

One can see that the simulated system should be stable as all the eigenvalues are in the left-half plane. The gain matrix that gives those eigenvalues is also listed in the appendix. In the next section, the simulation of the system is described and results discussed.

Simulation

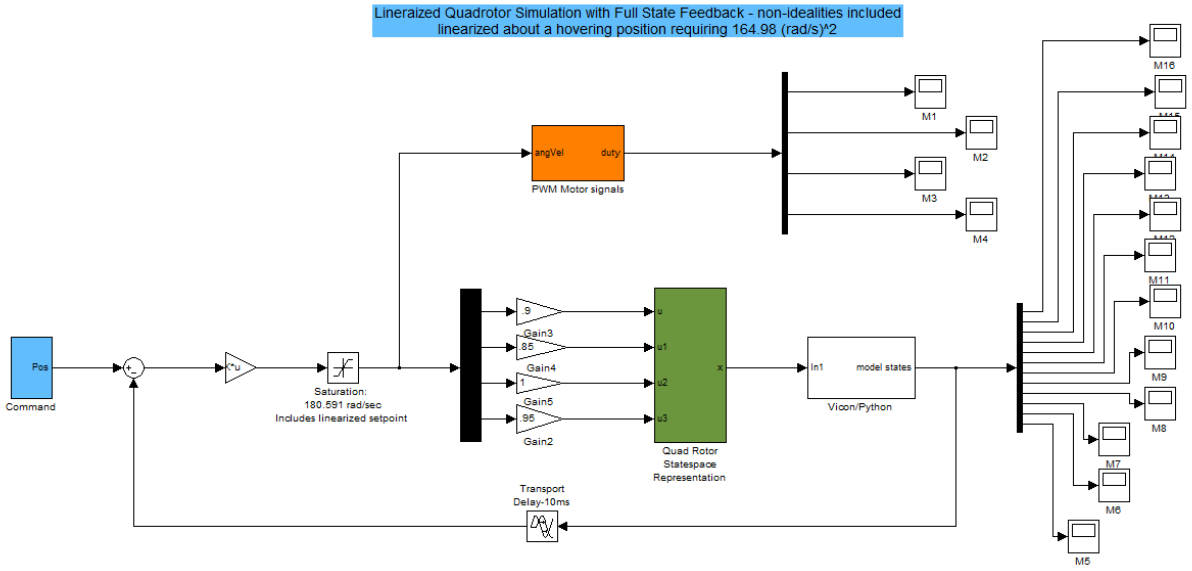


Figure 7: Matlab simulation with non-idealities included

Figure 7 shows a screen-shot of the simulation built to demonstrate the control law design. The Command box on the left includes a set point of a hovering condition and step changes in the X,Y,Z and yaw positions. The error between the setpoint and the actual postion is sent into the gain matrix described in the last section. This gain matrix calculates the necessary angular velocities that are then sent into the motors. The motor saturation and the effect of slightly different responses of the different motors are included in the loop as well to simulate as closely as possible the true nature of the system. The quad rotor dynamics block is shown below.

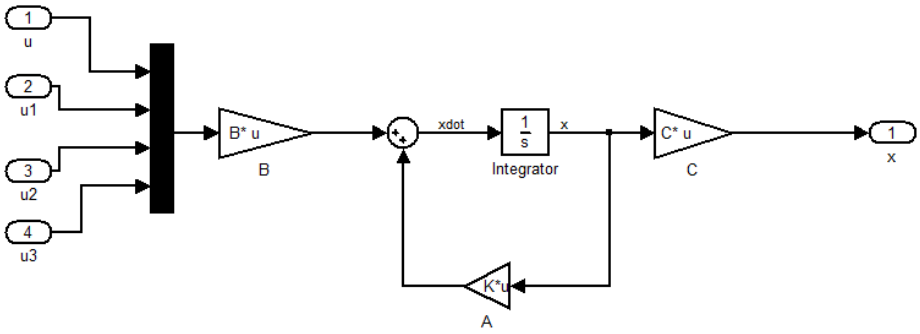


Figure 8: State space representation used in simulation

The simulation includes the A,B,C, and D linearized matrices. The output of the block is the state vector. This vector is then parsed to simulate the effect of vicon data and python script that actually calculates the position, attitude, translational, and angular rates (the latter two in body coordinates). The Vicon/Python block is shown below.

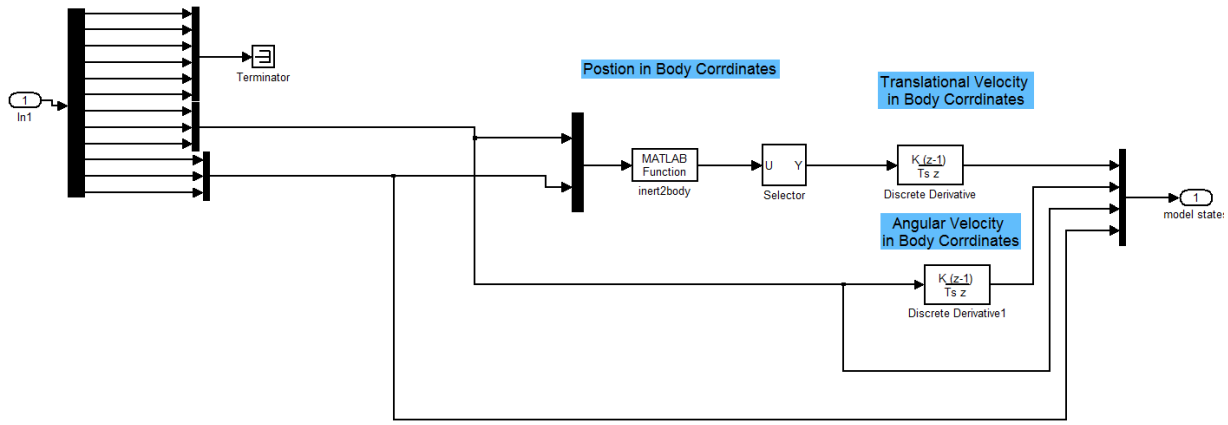


Figure 9: Simulating Vicon data and python script and estimating states

Since the data that is coming in from the Vicon system is in inertial coordinates, the data must be transformed into body coordinates through a direction cosine matrix transformation. The result is differentiated to get a rate and place in the appropriate spot in the state vector. The Euler Angles (yaw,pitch, roll) are also differentiated to give angular rates. The position and Euler angles are passed through the loop and placed in the state vector. Included in the feedback loop is a 10 ms delay to simulate latency of the network communication and the entire simulation is run in discrete steps of 1/120 seconds which corresponds to the maximum frame rate of the Vicon vision system.

Simulation Results

The following four plots show a simultaneous step response in X,Y,Z, and yaw. Each of the translational step changes are 0.5 meters and the yaw step change is 30 degrees. The simulations show a rise time of under five seconds and settles by 13 seconds. These meet the performance specification of a 10 second step response in altitude.

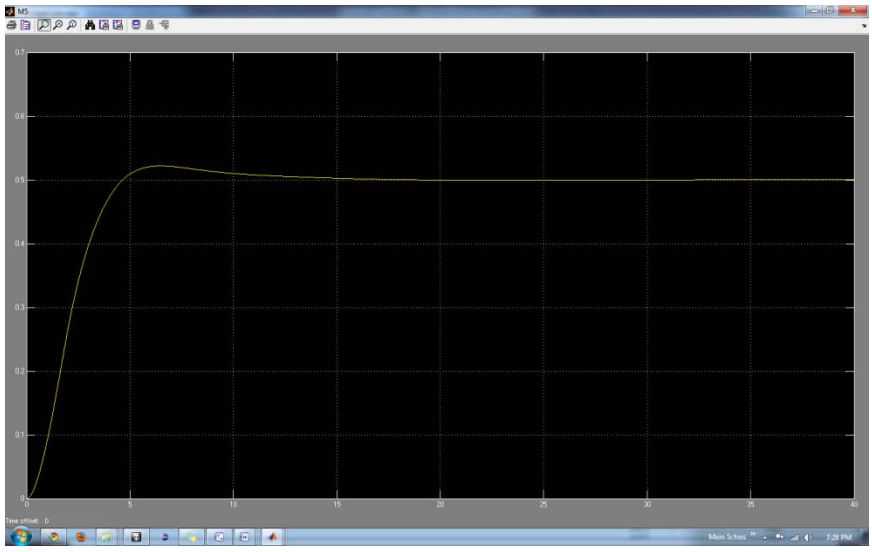


Figure 10: 0.5m step change in Z

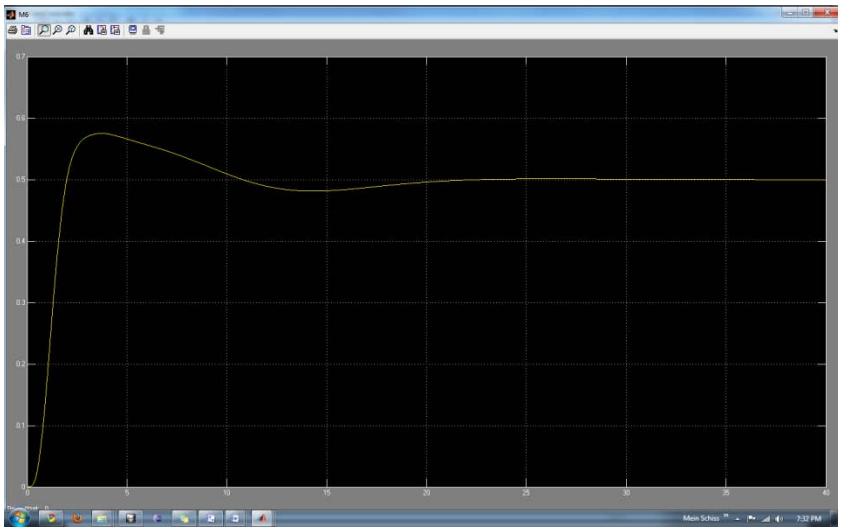


Figure 11: 0.5 m step change in Y

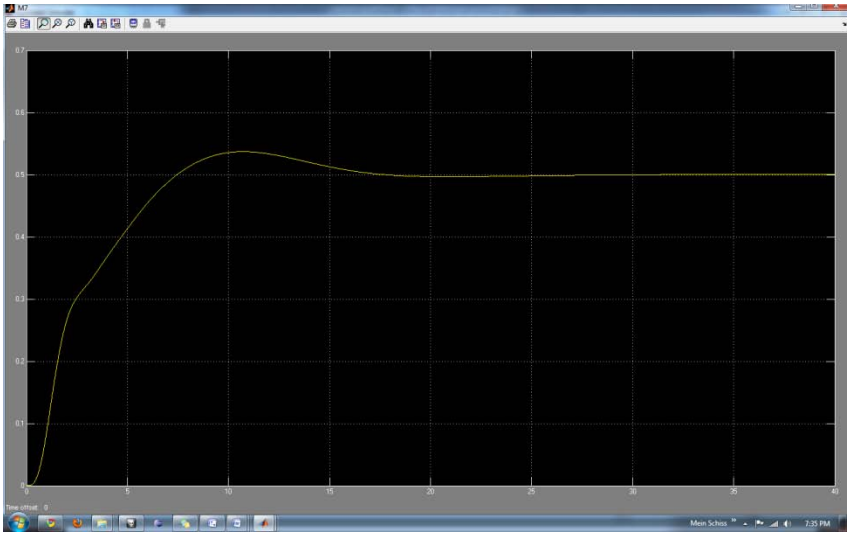


Figure 12: 0.5m step change in X

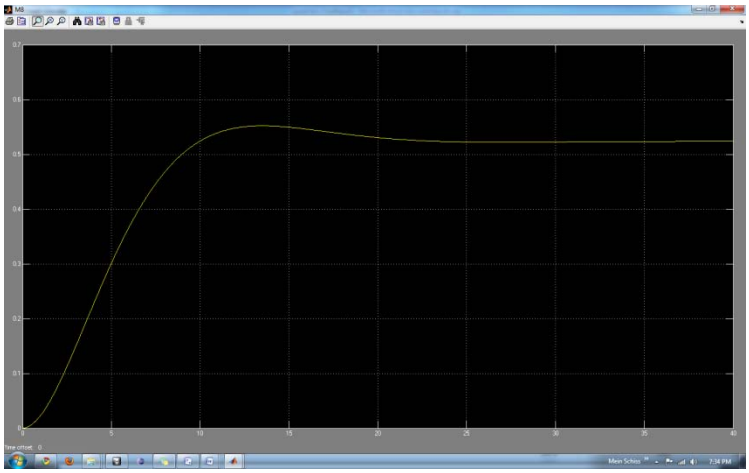


Figure 13: 30 degree step change in yaw

It should be noted here that the simulation does not take into account a necessary mapping from angular velocity to pulse-width modulation counts. The simulation results and how the actual system performs are quite different unfortunately. It is said sometimes that *simulations are doomed to succeed.*

Hardware, Electronics, and Software design

In this section, we will discuss the hardware implementation of the quad rotor system. Figure 8 shows a system level diagram of the hardware software and feedback.

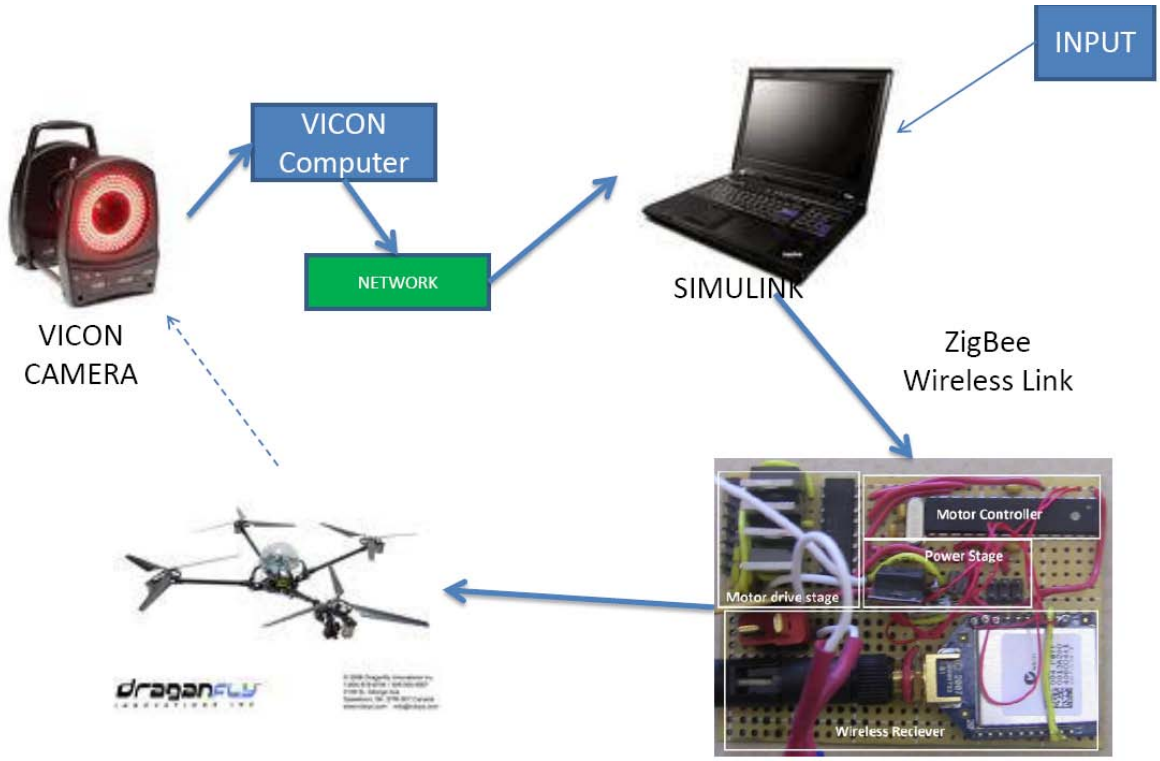


Figure 14: Overview of Hardware level Feedback loop

Hardware

There are three pieces of electronics hardware used in the control setup, the Vicon vision system, a laptop running Simulink with the control loop and the electronics hardware on the quad rotor. Since the laptop is mostly software, discussion of that will be put off until the software section of this report. We will be discussing the other two components here.

Vicon Vision System

The Vicon vision system is comprised of 3 components. The Vicon cameras, the Vicon hardware processor, and the Vicon host computer. There are six Vicon cameras placed on the ceiling in the DSSL lab. Each camera is 2 megapixels and runs at 120fps. The system tracks an area that is about 8'x6'x5'. These cameras are infrared black and white cameras. By using special reflective balls on the subject, the Vicon cameras use IR LEDs to reflect off of the balls producing an image that once filtered is all black except for the balls which reflect the IR light. By figuring out which pixels in each of the six images received back the balls cover, the system is able to accurately place the balls in the area to about .1mm accuracy. This data is processed on the Vicon hardware processor and sent to the Vicon host computer over a dedicated network link. This data is read in real time by the proprietary Vicon Nexus software. This software has a server running that the Simulink laptop can pull the data from the system through and will be discussed further in the software section. We are using this system because it was a requirement from the DSSL lab and gives about the best possible accuracy of any position tracking setup.

Quad rotor Hardware

The electronics hardware on the quad rotor is shown below in Figure 12. There are four sections, the motor drive stage, the motor controller, the power stage which provides 3.3v and 5v power to the board and the wireless receiver. The PCB was assembled using protoboard since putting a breadboard on a flying object didn't seem like a good idea and manufacturing a PCB would have been about $100 for a prototype run. This board attaches directly to the motors and the battery. The only other electronics on the quad rotor are filtering capacitors on the motors. The total cost for the hardware for the quad rotor is $215.51 the price breakdown is shown in the appendix.

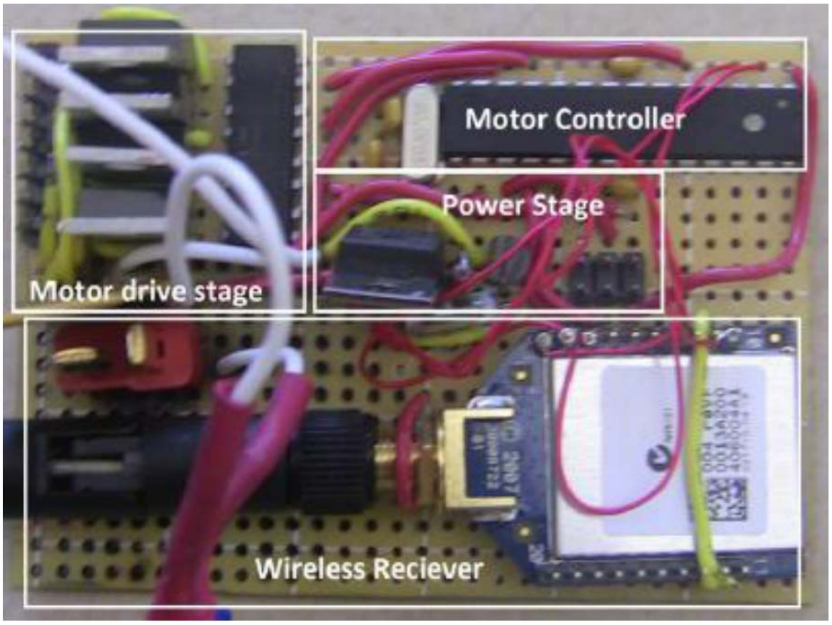


Figure 15: PC Board showing component location

Power Stage

The power stage consists of a pair of linear voltage regulators. A 5 volt TLE7805 regulator is used for the microcontroller and a LM317 for the Xbee transmitter running at 3.3 volts. These parts were chosen due to having them on hand when we went to assemble the PCB.

Motor Drive Stage

In order to control the thrust of the rotors, we must control the speed of the motors. Arguably the most used method to accomplish this is through pulse width modulation. A circuit was designed to take a 0-5V pulsed signal and boost this to a 0-11V signal to use to switch a MOSFET on and off as a DC chopper configuration. The circuit used is shown in Figure 9.

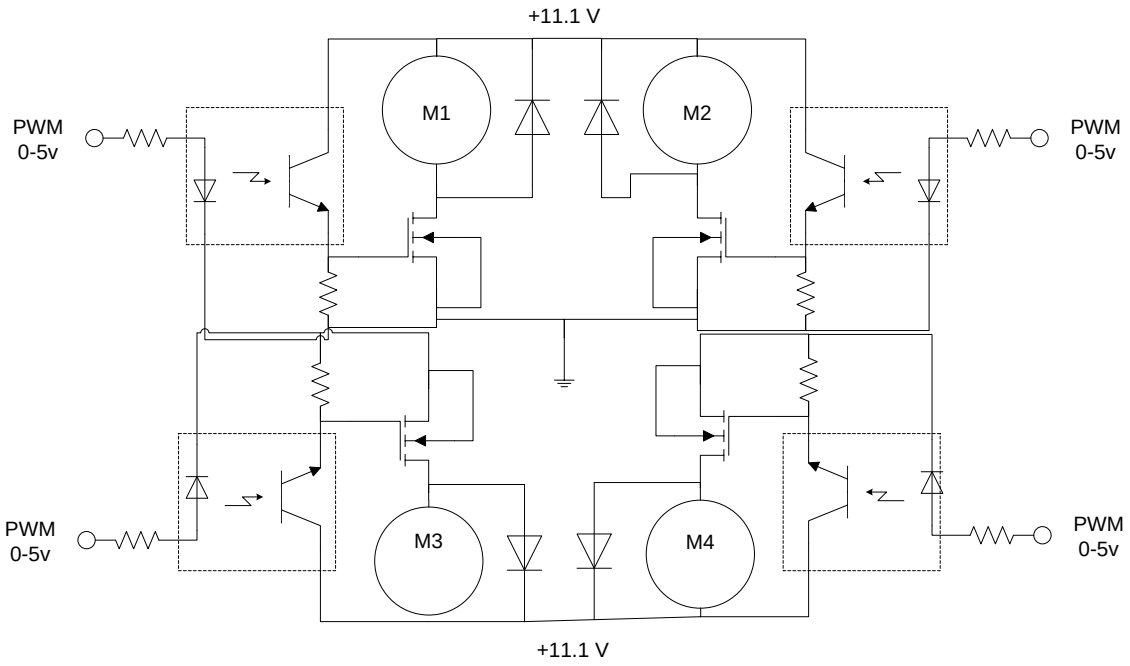


Figure 16: Motor Driver Stage

The components were chosen to handle the current draw requirements of the motors which is around 4 amps max and to provide isolation between the power stage and the micro controller. The MOSFETs (MTP3055) were selected for their performance, reliability, price and due to familiarity having used them on another EE related project. The MTP3055 are rated at 12A max current load, on 0.15 Ω resistance, and are also ideal for switching applications. Next is the Opto-Isolator (LTV-847) which comes as a quad package perfectly suited for our four drive stages, 4µs rise-time which is 25 times our base frequency of 10KHz and were very cheap at $0.75. Switching protection diodes are installed across the Motor to allow for *freewheeling*- current flow due to back EMF as the motor spin during the off cycle. 1 KΩ pull-up resistors are used to bias up the gate of the MOSFET in its linear region. Lastly is the Li-Polymer 3 Cell Battery which is rated at 2100mAhr, giving us a empirically determined flight endurance of approximately 40 minutes. However, we expect a typical flight on the order of 20 minutes. This battery gives us almost twice the capacity as the one that came with the original quad rotor.

Motor Controller

In order to control the motor, we decided on the Atmel ATMega328p microcontroller. This microcontroller was chosen due to having the PWM outputs we needed, the serial link to the Xbee we need, and already having familiarity with the microcontroller and owning the programmer. The motor controller is currently running off an external crystal at 20MHz. This was chosen so we could achieve a reasonable PWM speed of 10KHz and allow a high serial data rate to the XBee of 115200 Kbps.

Wireless Receiver

In order to communicate with the quadrotor we decided to go with a pair of 900MHz XBee radios. These radios offer two way communication with the quadrotor over a standard UART serial link. They provide 156 Kbps data rate and a line of sight range of 6 miles. These were chosen so the quadrotor could eventually be flown outside without having to worry about the signal fading out. The 900MHz radios were chosen due to having compact antennas while providing low link losses and not have issues with interference on the 2.4GHz band. While the current setup only sends 9600 bit/s, the high data rate was chosen due to it's lower latency and having extra bandwith for sending retries creating a more robust wireless link.

Software

Software implementation for the quad rotor project went through numerous iterations during the design process and was paramount to the success that were made. This section discusses the software in detatil.

Previous software attempt

The original plan was to use Realtime Workshop in Simulink to implement our controller. This was chosen since it would be easy to transition from simulation to implementation and would allow for plenty of time to test and troubleshoot the controller. This idea worked well and was implemented up to the point of interfacing the Vicon into Simulink. Since the Vicon isn't supported by default in Simulink, it required writing a custom S-Function to retrieve data. After spending a week working on this, the interface worked in the regular Simulink, but would not compile for Realtime Workshop. The other issues was in trying to get the Simulink model to be timed and trigger when data is received from the Vicon. Since the timing between the model and the Vicon was separate, this could introduce up to an additional 8ms of latency in the system. Due to these issues and not being confident the simulation was calculating time steps properly, we decided to abandon using Realtime Workshop.

Software requirements and implementation

Since using Realtime Workshop in Simulink was not able properly interface with the Vicon, a replacement was required. From the testing of the quadrotor up to this point and looking at what would be required to implement the controller, the following requirements were set:

- Fast and easy to implement and make changes
- Ability to verify implementation
- Realtime diagnostics and data logging capabilities
- Safe way to start and stop the quad rotor

In order to meet these requirements, it was quickly decided that programming the controller to the ATmega was not a good idea since it wouldn't give us any way to verify the implementation or any type of diagnostics on what is happening. Since it was required to use a computer to interface with the Vicon and send the data to the quad rotor, it made sense to run the feedback on the same system. In order to keep the implementation easy, it was decided a language with powerful vector numerical computation capabilities was required. In order to get realtime diagnostics and start and stop the quad rotor, it was decided a GUI was also needed. In order to meet these requirements and use something that was

familiar, we selected to use Python. By itself, Python is a powerful language that is fast and easy to read and write and has a large selection of libraries to use. In order to get the functionally that was required, the following libraries were used:

- Scipy – Scientific computing toolbox for Python. Gives most of the functionality of Matlab to Python.
- Numpy – Provides fast N-dimensional array manipulation to Scipy
- Matplotlib – Plotting library. Used to generate Matlab like plots from Python.
- PyQT4 – Application framework that gives Python bindings to Qt4, which is the framework KDE4 is built off of. Used for it's ability to create GUIs, implement network communication, and support creating and managing multiple threads.
- PySerial – Serial communication for Python. Used to send serial commands to the quad rotor over the Xbee link
- PyGame – Game development framework for Python. Used for it's ability to interface with Joysticks.

Software components

All in total, there was about 1500 lines of python written for our final implementation and about 75 lines of C that ran on the quad rotor. Most of this code ended up going into framework for supporting the controller. In total the controller itself takes up about 200 lines of code. While a lot of code wouldn't have been needed to be written to get the quad rotor working, by creating the framework around the controller and having the ability to save and plot data and see real time output to the motors allowed for easy debugging and troubleshooting of issues when testing. By not having this robust framework, figuring out where the problems were would have been much more difficult and it would have taken more time to test and run than it did to program the extra code.

GUI

Of the 1500 lines of Python were written to control the quad rotor, Over half the code ended up being to dedicated to creating the GUI, which is shown in Figure 17. The GUI is divided into four sections, the Vicon interface, the serial interface, the joystick interface, and the control and plotting interface. The Vicon interface allows the user to connect to the Vicon and shows real time position of the quad rotor both in absolute position and also position relative to a point. This point can be set with the zero button. The serial interface is used for connecting to the serial output that the Xbee is connected to. This section also has four sliders that show the current output to the motors. The sliders change from blue to red when the output is enabled indicating that it is really sending data to the quad rotor. The joystick module is for open loop testing. It allows connecting a joystick and shows four sliders for the axis of the joystick. The control and plotting interface allows the user to control what the program should do. There are three modes of operation, open loop with a joystick, closed loop, and simulation. The user can give a waypoint and a time the quad rotor should start moving to it or load a set of waypoints from a .mat file.

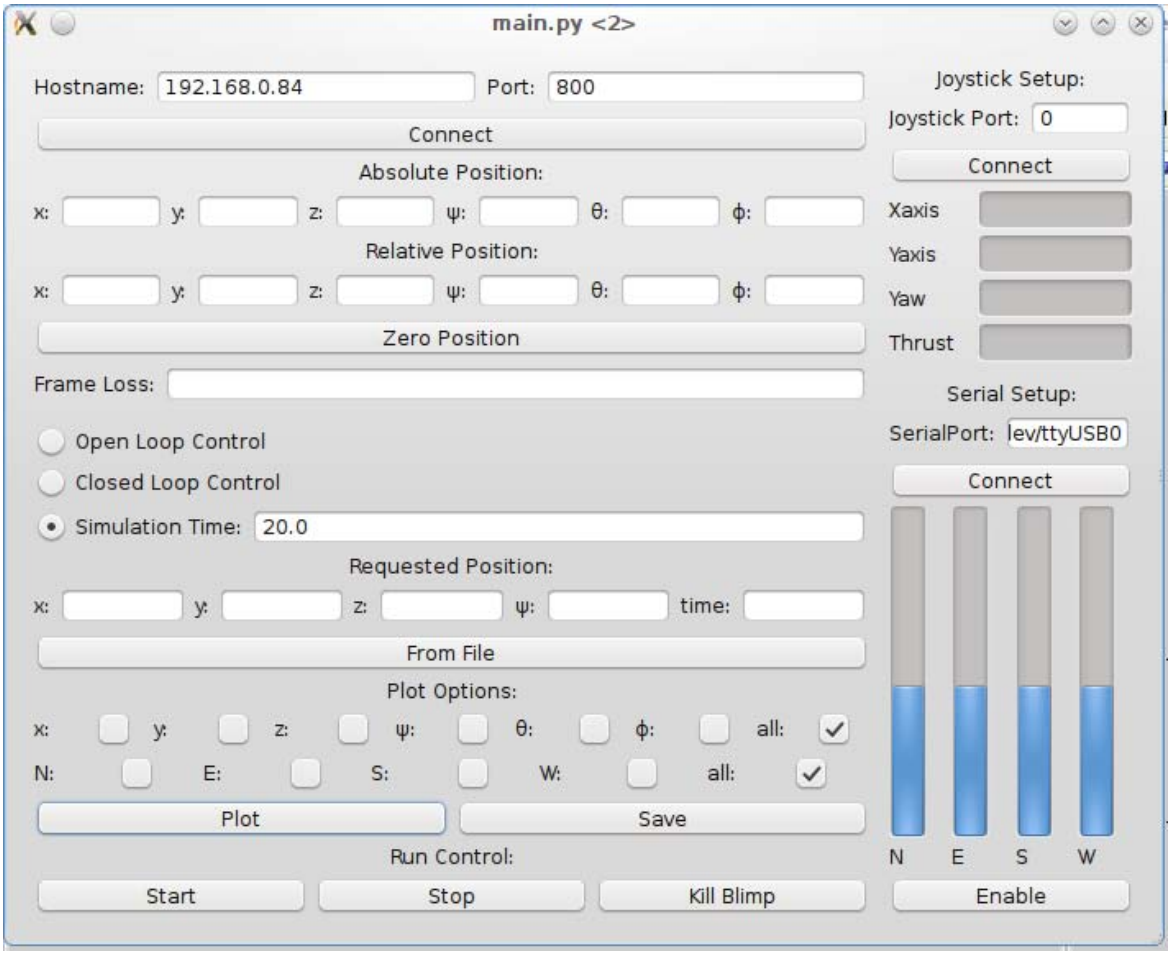


Figure 17: Python GUI used in quad rotor implementation

The output position and demand data can then be plotted or saved into a .mat file. The two buttons, run and stop control when the program executes. The stop button is also setup to stop the motors on the quad rotor when pushed in the event that it flys off to somewhere it shouldn't.

Vicon

The Vicon interface was written in PyQT4's QNetwork module. This module makes writing network servers and clients easy. The module defines TCP on the level of sockets and handles all low level communication leaving the code to just read and write to the socket. By reimplementing the Vicon interface and running under Linux, we were able to drop packet loss to under .5%. Unfortunately running the code under Windows still shows over 50% packet loss. The Vicon was configured to stream the data instead of polling the Vicon for it's data. This also helps reduce latency since the Vicon doesn't have to process the request before sending the data and sends it once it is generated. Since the data being sent is just x,y,z position data of each individual marker, the data needs to be converted into the position and angles of the quad rotor. To do this, there are four balls on the quad rotor, one on each blade. There is another ball to give one rotor a known indicator so the Vicon can tell which rotor is which. By defining two vectors between adjacent pairs of markers, the code finds the center of the quad rotor's position by least mean squares. Once it has the center position, it creates four normalized vectors from this position to the four rotor markers. It then creates a discrete cosine matrix for each set of adjacent rotors and then uses these to find Phi, Theta, and Psi. These angles are then averaged for the four marker pair sets to help improve the accuracy.

Controller

Since Scipy allows for importing data from .mat files, we were able to import the A, B, C, D, and K matrices directly from Matlab's saved files. The Python code then converted these matrices into the discrete time versions by taking their exponential. The actual feedback loop's ran the following steps:

1. Grab data from Vicon
2. Convert coordinates to body coordinates
3. Find velocity and angular velocity by taking the change in position from the last position to the current and dividing by the timestep.
4. Take the difference between the demand and feedback and multiply it by the K matrix.
5. Convert the output from rads/s to PWM counts by our fitted curve.
6. Send the PWM values to the serial object to transmit to the quad rotor

There are a few points to note here. The first is that we blocked on read until the data came in from the Vicon which essentially made the whole loop triggered by new data from the Vicon. This was to reduce as much latency as possible in the system. We decided to go with a pure discrete derivative since the Vicon is more accurate than 1mm, so the quantization noise will be low. By loading Matlab's saved files, it is very easy to change the controller without having to change any of the code. This gives one of the main advantages that Realtime Workshop had. In order to make sure the GUI would still be responsive and not slow down the controller, the entire controller was also implemented into it's own thread. Since the system we ran the program on had a dual core processor, this allows one core to to run the controller without any slowdowns.

Simulator

In order to verify the implementation of the controller, a simulator was built into the program. By comparing the output from Simulink to to the output of the implementation, we were able to guarantee the implementation was bug free. In order to do this, a simulated Vicon was created that ran the model

on the outputs that are sent to the quad rotor. By calling the controller thread with this simulated Vicon, the simulations could be ran without touching a line of code in the controller. A plot of the output from our Simulink model and from the Python simulator for a Z-step change are shown in Figure 18.

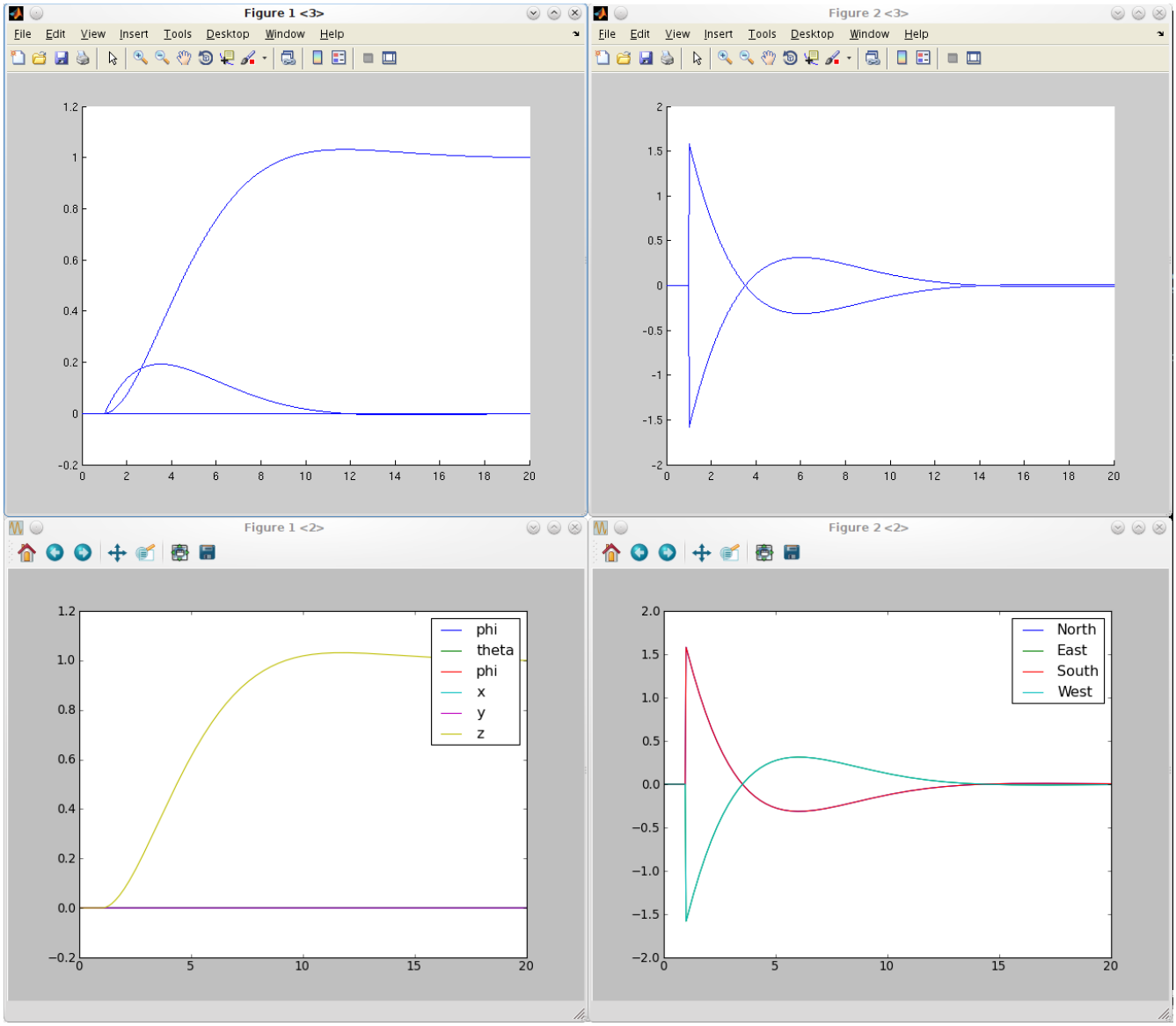


Figure 18: Python simulator step responses

The only difference between the data from the two is the Simulink model also plots velocities where the Python model doesn't since it is not something that is fed back from the Vicon.

Quad rotor software

Since all of the processing and control is done off board the quad rotor, the software that the quad rotor is running is very simple. All it does is wait for data to come in on the Xbee which is attached to it's UART port. When data is received, the Atmega checks for the hex byte 0xAA, which is the start byte. It then receives the motor PWM values twice and a stop byte 0x55. It makes sure the two copies of the PWM values are equal and if they are sends each one to one of the PWM timers. Sending the data twice was

chosen since there were some issues trying to send a checksum from Matlab/Simulink and the 120hz sample rate means the data sent to the quad rotor is only about 10% capacity on the Xbee wireless link.

Experimental Data

There is no data to present regarding the actual flight characteristics compared with the simulation results since the quad rotor did not achieve autonomous flight. This section will describe testing that was done on the quad rotor in order to quantify its parameters.

Motor Validation and PWM to Angular Velocity Mapping

The motors had to be tested to ensure approximately similar characteristics. In order to do this a test apparatus was created to hold the quad rotor stationary as we ramped up the PWM counts. The angular velocity was captured using a home built tachometer and an oscilloscope. The test apparatus is shown below.

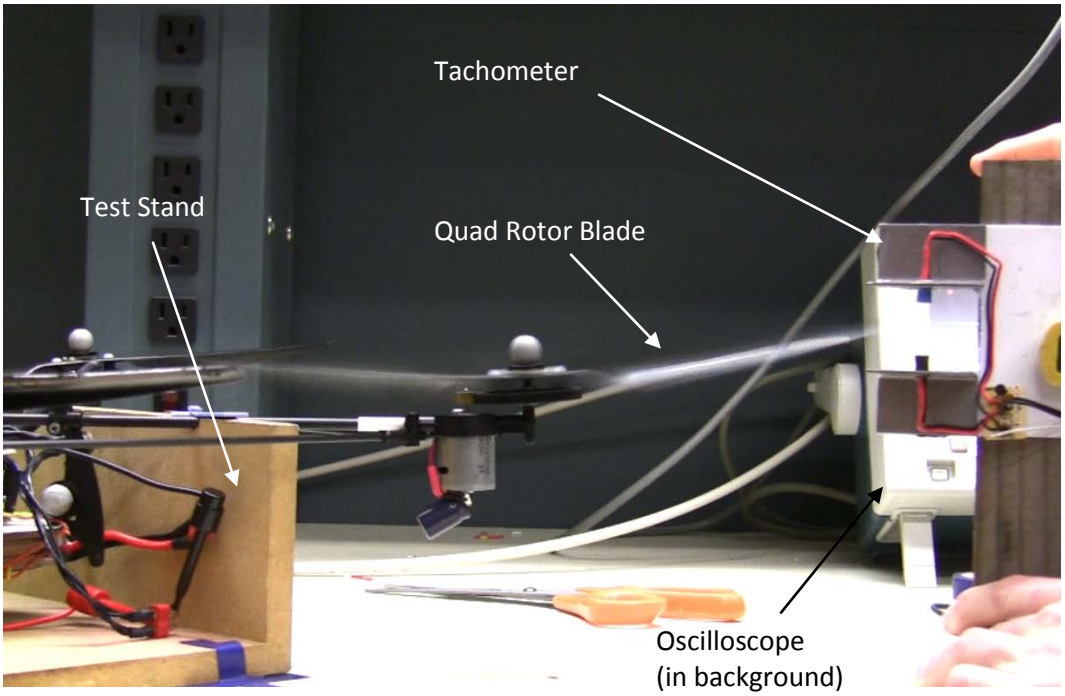


Figure 19: Apparatus for measuring angular velocity

As the blade passes through the tachometer, the tachometer measures two pulses per revolution. In addition to capturing angular velocity, motor voltage, current and PWM counts were recorded and tabulated for each of the four motors. The raw data is included in the appendix and the resulting plot is shown below in Figure 11.

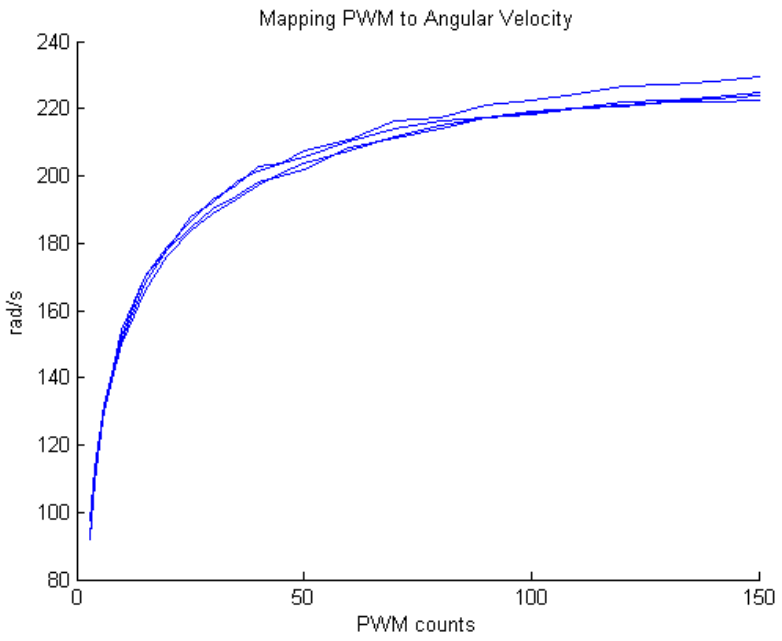


Figure 20: Plot of the 4 different motors speed vs. PWM

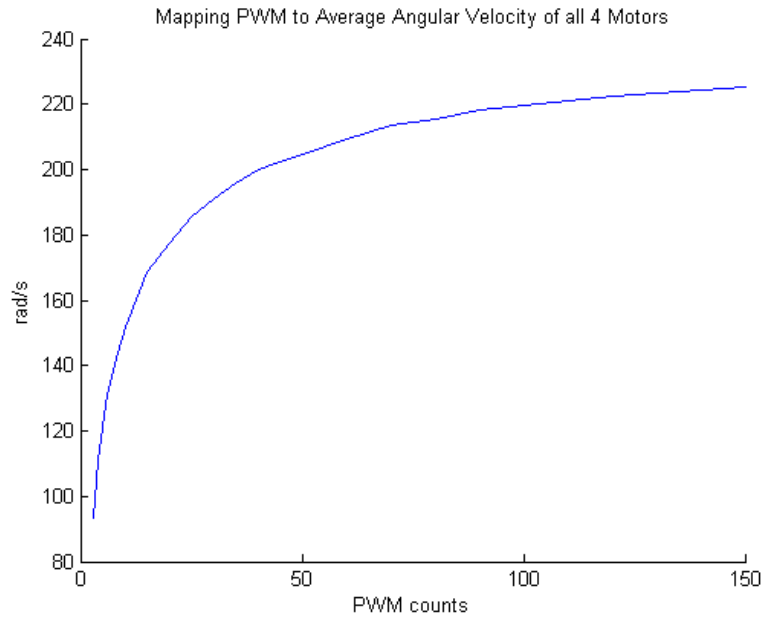


Figure 21: Average Angular Velocity vs. PWM

Figure 12 above shows the average of all the motors' angular velocities versus PWM counts. The average was taken because the plots of the individual motors were all very close. This greatly simplifies the process as we can now use just one function to map the feedback. Inverting and fitting the data give the results below.

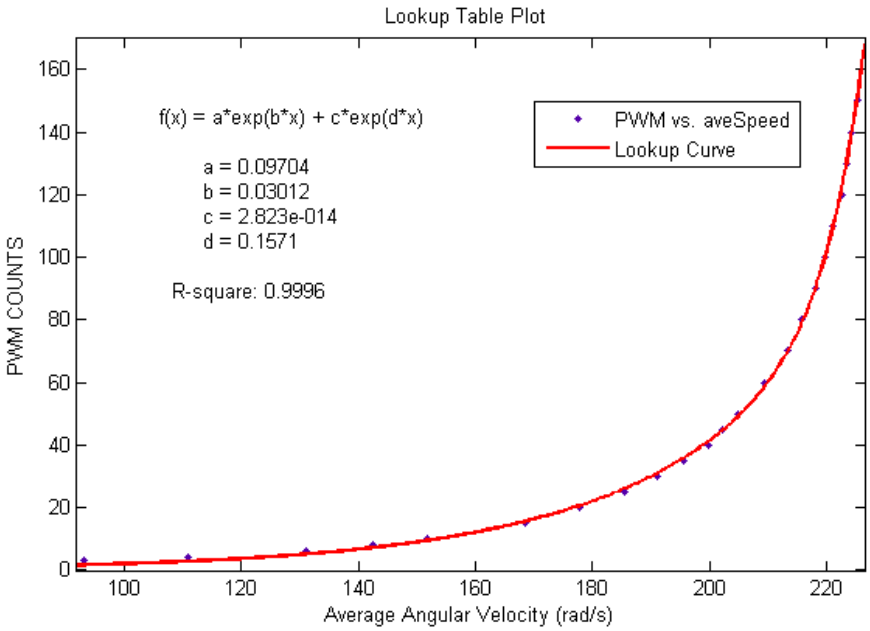


Figure 22: Statistical Fit for the average angular velocity vs. PWM

The results of this test have dire consequences. While it is nice to have a mapping between the angular velocity and PWM counts, the mapping is non-linear and shows the dynamic range that we can actually achieve with the current hardware configuration is a very large limiting factor. The results shown in Figure 13 are being used as a software lookup table but the small dynamic range with only an 8-bit resolution is not adequate.

Conclusions and Further Development

Since the quad rotor is not quite in a state where it can fly, there are several areas in the hardware, software, controller, and model that can be improved in order to get the quad rotor flying. Of all the changes that can and need to be made, it looks like the hardware issues are currently most limiting to achieving flight.

Hardware

There are three main issues with the quadrotor's hardware. The first is the circuit really doesn't have enough bulk capacitance. The second is the PWM counts to voltage needs to be linearized. The third is the resolution of the PWM device needs to be raised.

When the motor's PWM switches, there are huge transients in the circuit which effected the Xbee's ability to communicate. Our current solution was to add capacitance on the motors. This might have not been the best place to put it and these capacitors should have been placed on the power rails. This is because placing the caps on the motors are not well mounted and could easily break off. The second issue is this limits the response of the motor.

The second issue of non-linear PWM to voltage is what really killed our ability to fly the quad rotor. Our testing shows that it is caused by the optio-isolator. In figures 23-26, we show the input and output from the optio-isolator in the circuit for PWM counts 10, 50, 100, and 150. The yellow trace shows the input and the blue trace is the output. For this test, the motors were disconnected to make sure the issue was not caused by them. As can be seen, the blue trace takes a long time to fall on the longer pulses. We then isolated the optio-isolator and tested it straight from a function generator. These plots are shown in figures #### and ####. What this is showing is that there is an issue with the drive voltage on the LED for the optio-isolator. If the voltage is high enough to get the full 11v out, the output transistor takes a long time to fall. If the drive voltage is low, the transistor doesn't turn on hard enough and the output voltage is low. Adding a mosfet to the output of the optio-isolator had negligible effect on it's performance. This means any version 2 will need to use a different optio-isolator.

Table 4: Identifying the problem with the opto isolator

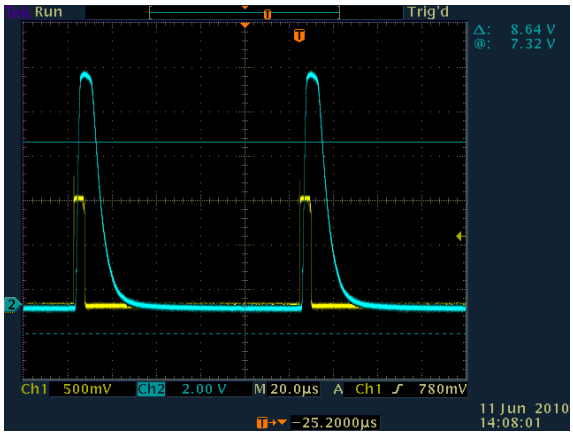


Figure 23: PWM = 10

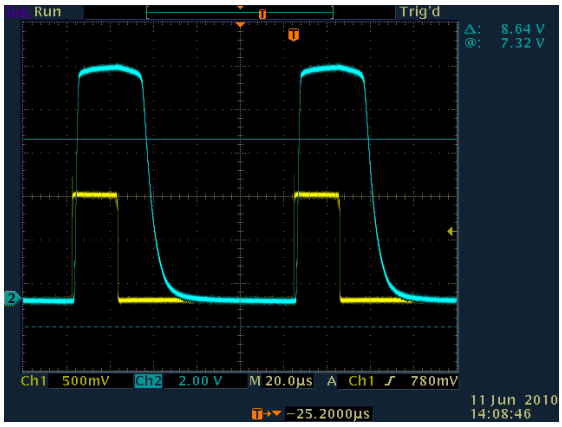


Figure 24: PWM = 50

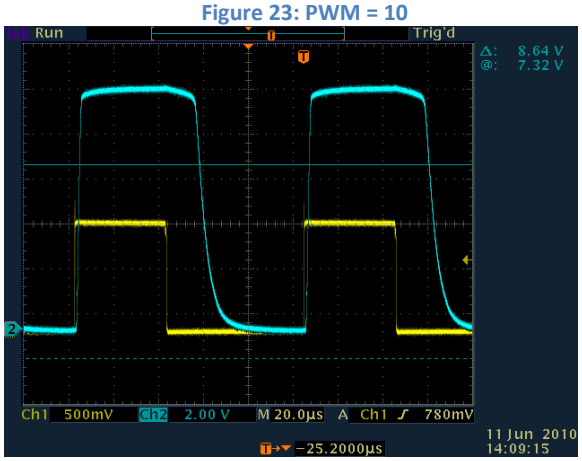


Figure 25: PWM = 100

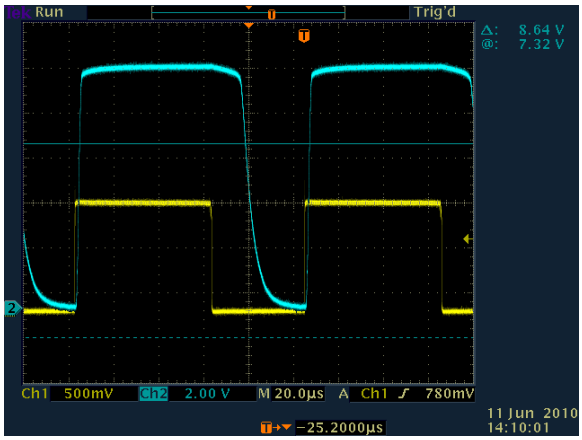


Figure 26: PWM = 150

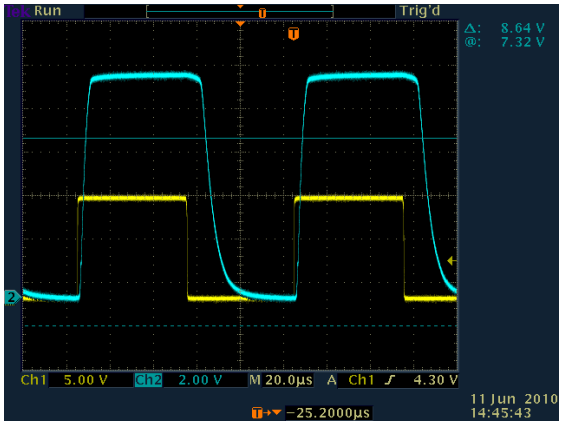


Figure 27

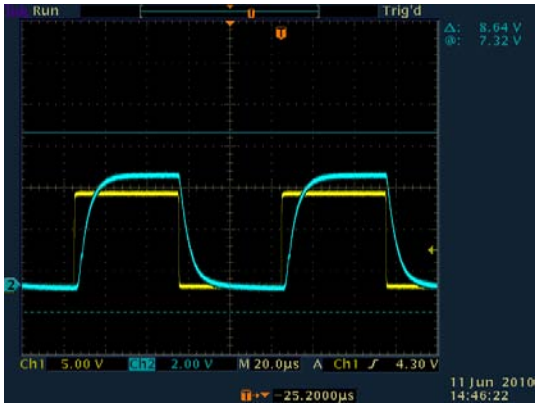


Figure 28

The third and biggest issue is we are currently using 8-bit PWM. Even if our optio-isolator was ideal, our simulations show this would not be enough. With limited output resolution, the quad rotor has to move a long distance from the set point before the controller acts strong enough to change the PWM output and the fine control gets lost in the rounding. By simulating the effect of our non-linear PWM with the quadrotor trying to hover, our results for our aggressive LQR controller show the quad rotor drop over 25 centimeters before it finally stabilizes as shown in figures 29 and 30. By simulating an ideal linear 8-bit PWM, we were able to get the steady state error for this set point down to 18 centimeters as shown in figures 31 and 32. Moving to a 12-bit controller, the performance would be greatly increased with less than 1.5 centimeters of steady state error as shown in figures 33 and 34. The ideal solution is to use 16-bit PWM controllers which would reduce the steady state error to just 1.8 millimeters as shown in figures 35 and36 In order to do this, it would be best to find a chip that just connects up to the existing microcontroller over SPI and outputs 16-bit PWM. Texas Instruments looks to have a line of ICs designed for LED applications that can do this.

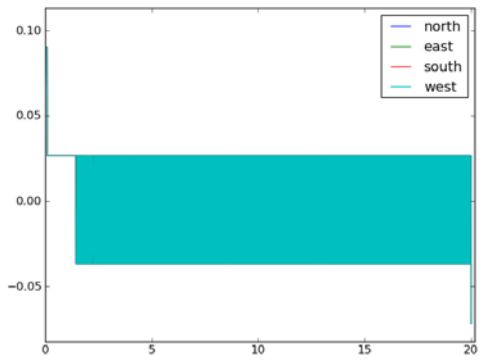


Figure 29

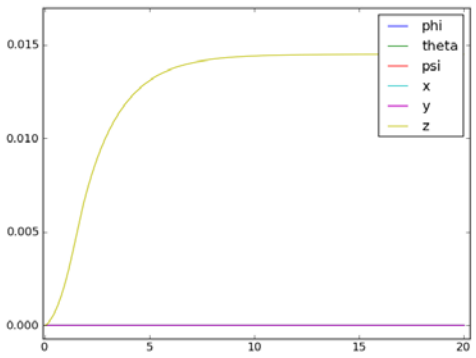


Figure 30

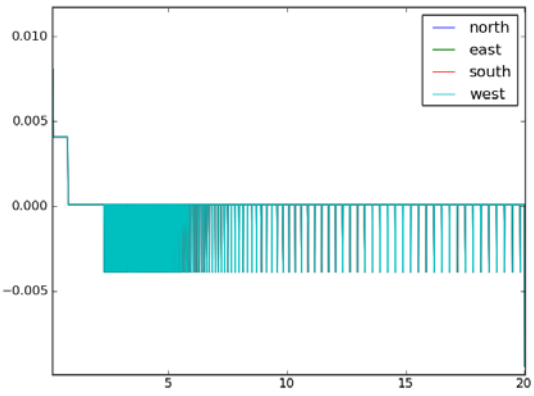


Figure 31

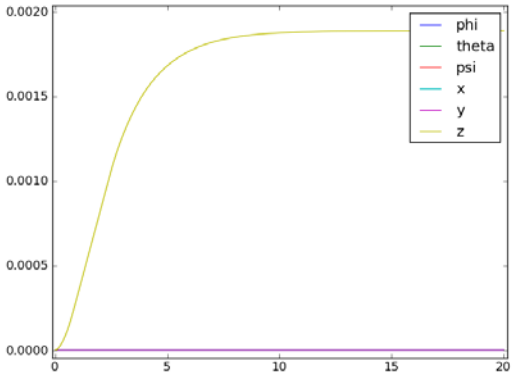


Figure 32

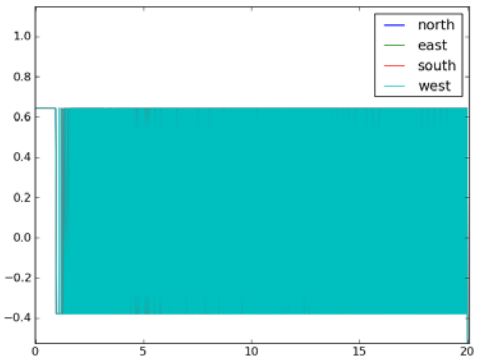


Figure 33

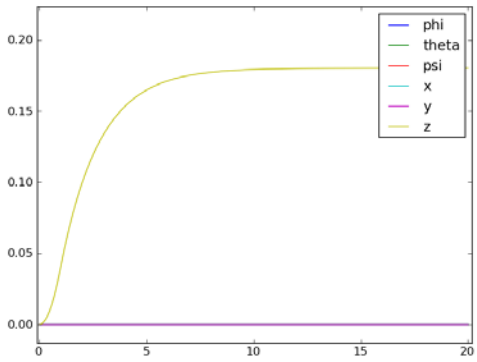


Figure 34

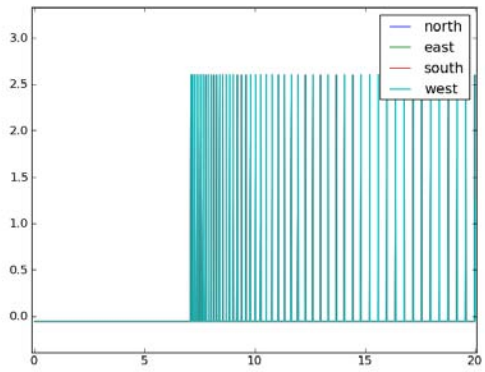


Figure 35

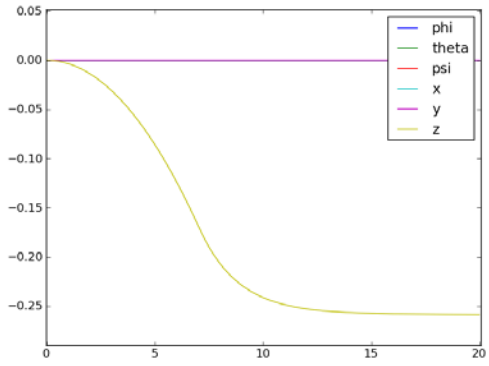


Figure 36

With these three hardware fixes, we are confident that the quad rotor will fly. There are several other hardware options to extend the design even further. These would include adding an IMU to supplement the Vicon data, adding feedback for the battery voltage as it drops over it's charge cycle, and putting this on a PCB and improve mounting. However these are secondary issues that could be left to being addressed until after the quad rotor is flying.

Currently all the software works, but there are a few bugs in various places and parts that can be improved. Probably the biggest is in handling of the translation from Vicon markers to position and angles. Currently, if one of the balls goes out of view, the data we get back is garbage or the position is unsolvable. It may be possible to figure out the data that is needed if one marker disappears. Unfortunately if more than one disappear, there isn't really a way to figure out the position. At this point, the code could be modified to turn off the quad rotor since it may be going completely out of range with the Vicon. The other way to solve this problem would be with an IMU. Another improvement to software would be to add realtime plotting of the data. This would give better feedback besides just the sliders on what is actively going on. This should be doable with the matplotlib. The last major issue that should be fixed in the software is error handling. Right now if something happens, the known errors are just printed to the console and the unknown ones can crash parts of the program. These issues should be caught by an error handler and fed to the user or fixed in the code. The one place were this issue shows up is in the network implementation. It seems that on starting the code gets a packet it doesn't like which requires it to timeout before it starts running. Likewise there is an issue where once the controller is stopped, the program must be restarted because it doesn't like one of the network packets it sees when it is started back up. These bugs should be fixed just to make the software easier.

References

[1] B. Heemstra, “Linear quadratic methods applied to quadrotor control”. M.S. thesis, University of Washington. 2010.

[2] C. Balas, “Modeling and linear control of a quadrotor”. M.S. thesis, Cranfield University. 2007. https://dspace.lib.cranfield.ac.uk/bitstream/1826/2417/1/Modelling%20and%20Linear%20Control%20of%20a%20Quadrotor.pdf

[3] S. Bouabdallah, A. Noth, and R. Siegwart, “PID vs LQ control techniques applied to an indoor micro quadrotor”, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings, vol. 3, pp. 1-6.

[4] P. Castillo, A. Dzul, and R. Lozano, “Real-time stabilization and tracking of a four-rotor mini rotorcraft”, IEEE Transactions on Control Systems Technology, Vol 12, No 4, July, 2004.

[5] P. McKerrow, P., "Modelling the Draganflyer four rotor helicopter", *2004 IEEE International Conference on Robotics and Automation, April 2004, New* Orleans, pp. 3596.

[6] Observability. (2010, March 29). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:26, April 24, 2010, from http://en.wikipedia.org/w/index.php?title=Observability&oldid=352709914

[7] S. Waslander, G. Hoffmann, J. Jang, C. Tonlin, “Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning”, *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[8] A. Kivrak, “Design of control systems for a quadrotor flight vehicle squipped with inertial sensors”, M.S. thesis, Atilim University, 2006.

Appendix

Parts List

Name	Part Number	Distributor	Cost	Qty.	Ext.
2100mAh 11.1v LiPo	TP-2100-3SPL2	rctoys.com	$47.99	1	$47.99
Deans Battery Connector	DE-ULTRA	rctoys.com	$3.55	2	$7.10
900MHz Dipole Antenna	WRL-09143	Sparkfun	$7.95	1	$7.95
Digi XBee Pro 900	WRL-08768	Sparkfun	$44.95	2	$89.90
XBee Explorer USB	WRL-08687	Sparkfun	$24.95	1	$24.95
Atmel ATMega328P	COM-09061	Sparkfun	$4.30	2	$4.30
3W 3.3v DC-DC Regulator	445-2474-ND	Digikey	$11.37	2	$22.74
20.000 MHz Crystal	300-8507-ND	Digikey	$0.63	2	$1.26
22pF Ceramic Cap	BC1005CT-ND	Digikey	$0.08	10	$0.76
6-pin header	609-3218-ND	Digikey	$0.37	1	$0.37
Optioisolator	160-1370-5-ND	Digikey	$0.75	1	$0.75
PTC Fuse	F3189-ND	Digikey	$0.84	1	$0.84
Protection Diodes	1N4007	UW EE Store	$0.20	4	$0.80
Resistors (carbon film 5%)	N/A	UW EE Store	$0.10	8	$0.80
MOSFETs (N-channel)	MTP3055	UW EE Store	$1.00	4	$4.00
5v Regulator	LM317T	UW EE Store	$0.60	1	$0.60
3.3v Regulator	LM78L05	UW EE Store	$0.40	1	$0.40
				Total	$215.51

Controller Gain Matrices

Q =

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

R =

$$\begin{bmatrix} 0.1000 & 0 & 0 & 0 \\ 0 & 0.1000 & 0 & 0 \\ 0 & 0 & 0.1000 & 0 \\ 0 & 0 & 0 & 0.1000 \end{bmatrix}$$

```
    0     0     0     0     1     0     0     0     0     0     0     0
    0     0     0     0     0     1     0     0     0     0     0     0
    0     0     0     0     0     0     1     0     0     0     0     0
    0     0     0     0     0     0     0     1     0     0     0     0
    0     0     0     0     0     0     0     0   100     0     0     0
    0     0     0     0     0     0     0     0     0     1     0     0
    0     0     0     0     0     0     0     0     0     0     1     0
    0     0     0     0     0     0     0     0     0     0     0    10
```

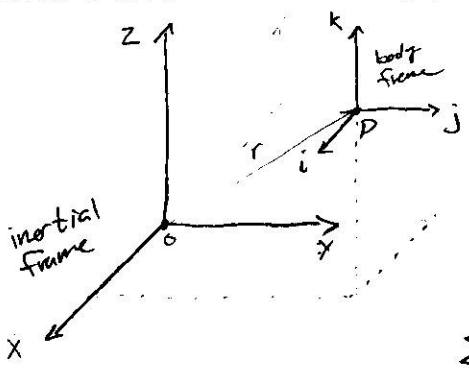
K =

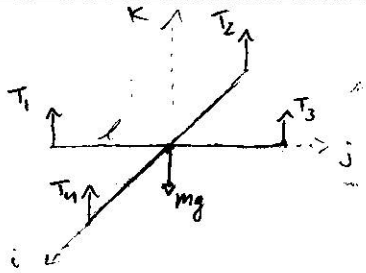
Columns 1 through 7

```
    3.6888    0.0000   -9.0550   -0.0000    6.0434   67.8528   -0.0000
   -0.0000    3.6791   -9.0550   -5.9652    0.0000  -67.8528  -20.0579
   -3.6888    0.0000   -9.0550   -0.0000   -6.0434   67.8528   -0.0000
    0.0000   -3.6791   -9.0550    5.9652    0.0000  -67.8528   20.0579
```

Columns 8 through 12

```
   20.2081   15.7807    2.1651   -0.0000   -4.9764
    0.0000  -15.7807   -0.0000    2.1644   -4.9764
  -20.2081   15.7807   -2.1651    0.0000   -4.9764
    0.0000  -15.7807    0.0000   -2.1644   -4.9764
```





$$\left.\begin{aligned}\Sigma F &= m\ddot{x}\\ \Sigma M &= I\ddot{\theta}\end{aligned}\right\}\ \text{Inertial Frame}$$

translational
u – Velocity in body frame $\hat{i}$
v – " $\hat{j}$
w – " $\hat{k}$
p – rotational velocity in body frame about $\hat{i}$
q – " $\hat{j}$
r – " $\hat{k}$

$$\bar{V}^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \bar{\omega}^b_{b/i} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$ angular velocity (body) W.R.T. inertial frame

$$\underline{\Phi} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$ Euler angles: orientation of the body axis W.R.T inertial frame

$$\bar{r}^i_{b/i} = \begin{bmatrix} P_i \\ P_j \\ P_k \end{bmatrix}$$ Position of body frame W.R.T. inertial frame

state vector

$$X = \begin{bmatrix} \bar{V}^b \\ \bar{\omega}^b_{b/i} \\ \underline{\Phi} \\ r^i_{b/i} \end{bmatrix}_{12\times 1} \qquad \dot{X} = \frac{d}{dt}X$$

$\Sigma F = m \left.\frac{d}{dt}\right|_i \bar{V}$: newton's law is in inertial frame, so

$$\left.\frac{d}{dt}\right|_i \vec{V} = \left.\frac{d}{dt}\right|_b \vec{V} + \omega^b_{b/i} \times V^b$$

Forces are magically xformed into body frame

$$\Sigma F^i = m\left(\left.\frac{d}{dt}\right|_b \vec{V}^b + \omega^b_{b/i} \times V^b\right) \Rightarrow \left.\frac{d}{dt}\right|_b \vec{V}^b = \frac{1}{m}\Sigma F^b - \omega^b_{b/i} \times V^b$$

$$\Sigma F = F_G + F_T \qquad F^i_G = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \Rightarrow F^b_g = mg\begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix}$$

$$F^i_T = \sum_{j=1}^{4} T_j \ ; \ T_j = b\Omega_j^2 \qquad \Omega_j : \text{angular velocity of blade}$$

$$F^b_T = \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix}$$

(1)
$$\frac{d}{dt}\begin{bmatrix} u \\ v \\ w \end{bmatrix}^b = \frac{1}{m}\left(mg\underbrace{\begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix}^b}_{\text{gravity}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix}^b}_{\text{Thrust}}\right) - \begin{bmatrix} p \\ q \\ r \end{bmatrix}^b \times \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b$$

$\Sigma M = I\ddot{\theta} = I\dot{\omega} = \left.\frac{d}{dt}\right|_i I\omega_{b/i}$ only in inertial frame!

$\left.\frac{d}{dt}\right|_i I\omega_{b/i} = \left.\frac{d}{dt}\right|_b I\omega_{b/i} + \omega^b_{b/i} \times I\omega_{b/i}$ same magic as before

$\Sigma M^b = \left.\frac{d}{dt}\right|_b I\omega_{b/i} + \omega_{b/i} \times I\omega_{b/i}$

$\left.\frac{d}{dt}\right|_b I\omega_{b/i} = \Sigma M^b - \omega_{b/i} \times I\omega_{b/i} \Rightarrow \frac{d}{dt}\omega_{b/i} = I^{-1}\Sigma M^b - I^{-1}(\omega_{b/i} \times I\omega_{b/i})$

$\Sigma M^b = M^b_T + M^b_R + \cancel{M^b_G}$ ignore

M^b_T = moment due to thrust
M^b_R = " " " rotating motors
M_G = " " " gyroscopic motion

$M^b_T = \sum_{j=1}^{4}(r \times F^b_{T_j})$ r is distance from center of mass to motor

$M^b_R = k(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2)\begin{bmatrix}0\\0\\1\end{bmatrix}$ since yaw is only about the $\hat{k}$ axis.

(2) $\frac{d}{dt}\begin{bmatrix}p\\q\\r\end{bmatrix} = I^{-1}\left(\underbrace{\sum_{j=1}^{4}(r \times F^b_{T_j})}_{\text{Thrust Moment}} + \underbrace{k(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2)\begin{bmatrix}0\\0\\1\end{bmatrix}}_{\text{Yaw moment}}\right) - I^{-1}\left(\begin{bmatrix}p\\q\\r\end{bmatrix} \times I\begin{bmatrix}p\\q\\r\end{bmatrix}\right)$

$\frac{d}{dt}\Phi = H(\Phi)\omega_{b/i}$ where $H(x)$ is a transformation matrix

(3) $\frac{d}{dt}\begin{bmatrix}\phi\\\theta\\\psi\end{bmatrix} = \begin{bmatrix}1 & \tan\theta\sin\phi & \tan\theta\cos\phi\\ 0 & \cos\phi & -\sin\phi\\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\sin\theta}\end{bmatrix}\begin{bmatrix}p\\q\\r\end{bmatrix}$

$\frac{d}{dt}r^i_{b/i} = C_{i/b}V^b$

(4) $\frac{d}{dt}\begin{bmatrix}P_i\\P_j\\P_k\end{bmatrix} = \begin{bmatrix}c\theta c\psi & c\theta s\psi & -s\theta\\ -c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi + s\phi s\theta s\psi & s\phi c\theta\\ s\phi s\psi + c\phi s\theta c\psi & -s\phi c\psi + c\phi s\theta s\psi & c\phi c\theta\end{bmatrix}^T\begin{bmatrix}u\\v\\w\end{bmatrix}$

① $\frac{d}{dt}\begin{bmatrix}u\\v\\w\end{bmatrix}^b = \frac{1}{m}\left(mg\begin{bmatrix}-\sin\theta\\ \cos\theta\sin\phi\\ \cos\theta\cos\phi\end{bmatrix} + \begin{bmatrix}0\\0\\T\end{bmatrix}^b\right) - \begin{bmatrix}p\\q\\r\end{bmatrix} \times \begin{bmatrix}u\\v\\w\end{bmatrix}$

$= g\begin{bmatrix}-\sin\theta\\ \cos\theta\sin\phi\\ \cos\theta\cos\phi\end{bmatrix} + \frac{1}{m}\begin{bmatrix}0\\0\\T\end{bmatrix} - \begin{bmatrix}qw - rv\\ ru - pw\\ pv - qu\end{bmatrix}$

$\frac{d}{dt}\begin{bmatrix}u\\v\\w\end{bmatrix}^b = \begin{bmatrix}-g\sin\theta - qw + rv\\ g\cos\theta\sin\phi - ru + pw\\ g\cos\theta\cos\phi + \frac{1}{m}T - pv + qu\end{bmatrix}$

$\dot{u} = rv - qw - g\sin\theta$
$\dot{v} = pw - ru + g\cos\theta\sin\phi$
$\dot{w} = qu - pv + g\cos\phi\cos\theta + \frac{b}{m}(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$

[linearize]

$u = u_0 + \Delta u \quad u_0 = 0$
$v = v_0 + \Delta v \quad v_0 = 0$
$w = w_0 + \Delta w \quad w_0 = 0$
$p = p_0 + \Delta p$
$q = q_0 + \Delta q$
$r = r_0 + \Delta r$
$\theta = \theta_0 + \Delta\theta$
$\phi = \phi_0 + \Delta\phi$
$\psi = \psi_0 + \Delta\psi \quad \psi_0 = 0$

[trig identity]

$\cos(s+t) = \cos s\cos t - \sin s\sin t$
$\sin(s+t) = \sin s\cos t + \cos s\sin t$

$\sin(x_0 + \Delta x) \to \sin\Delta x \to \Delta x$
$\cos(x_0 + \Delta x) \to \cos\Delta x \to 1$

$\dot{u} = (r_0 + \Delta r)(v_0 + \Delta v) - (q_0 + \Delta q)(w_0 + \Delta w) - g\sin(\theta_0 + \Delta\theta)$
$= (r_0 v_0 + r_0\Delta v + v_0\Delta r + \Delta r\Delta v) - (q_0 w_0 + q_0\Delta w + w_0\Delta q + \Delta q\Delta w)$
$- g(\sin\theta_0\cos\Delta\theta + \cos\theta_0\sin\Delta\theta)$

$= \underbrace{\Delta r\Delta v}_{\text{small}} - \underbrace{\Delta q\Delta w}_{\text{small}} - g\sin\Delta\theta \to \Delta\theta$

$\therefore \dot{u} = -g\sin\Delta\theta = -g\Delta\theta$

$\dot{v} = \underbrace{\Delta p\Delta w}_{\text{small}} - \underbrace{\Delta r\Delta u}_{\text{small}} + g\cos\Delta\theta\sin\Delta\phi \quad \therefore \dot{v} = g\cos\Delta\theta\sin\Delta\phi$
$= g\Delta\phi$

$\dot{w} = g\cos\Delta\phi\cos\Delta\phi + \frac{b}{m}(6804.94 + 164.984\Delta\omega_1)$