Simulation and Control

Purpose:

The Simulink based quadcopter simulation is intended to get you up and running with a simulation representative of your vehicle's dynamic performance. We hope you will find the simulation to be understandable and to have a reasonable and intuitive layout. The simulation is intended to act as a sort of starter kit to help accelerate you towards the goals of your project and provide you with tools to explore quadcopter control design techniques. The simulation represents a synthesis of several author's approaches to modeling the behavior of a quadcopter, with a little bit of additional modeling added by us. Many important effects (most obviously aerodynamic effects such as blade flapping) are ignored or dramatically simplified, so you will need to make sure you understand the limitations of the model in order to get reliable service out of it. Since it is written in Simulink and MATLAB code, we hope you will feel comfortable modifying or expanding the simulation in any way you desire to fit the needs of your project. To that end we've tried to provide understandable and adequately commented code and models to make modification by users a reasonable task.

Required Skills/Knowledge:

No special skills or knowledge are explicitly required to use this simulation, but for best results you'll want a bit of the following:

- Experience in problem solving within the Simulink environment and running simulations
- Experience in problem solving within the MATLAB environment and writing scripts and functions
- A basic understanding of the mathematics associated with aircraft dynamics
- At least *some* familiarity with feedback control concepts (If' you've never heard of PID control, you've got some serious studying to do) including both classical

feedback control design and state-space modeling from a mathematics perspective. Really the more you know the more cool stuff you can do.

- Curiosity, patience, and determination. Seriously.

Required Materials:

- PC or Mac with MATLAB 2013a or later. May work on some earlier versions, but this is untested.
  (Tested and developed using Simulink Version 8.1 (R2013a) 13-Feb-2013)
- The contents of the downloadable zip file containing the documentation for this project along with the entire contents of the *Quadcopter Simulation* folder added to your MATLAB path (this is important!)
- If you want to simulate your own vehicle you will also need
  - A quadcopter model saved somewhere you can find it. (Normally this is saved from the Moment of Inertia GUI and should be a .mat file)

AC_Quadcopter_Simulation.slx Basic Demonstration and Walkthrough:

Good news: the best way to learn how the simulation works is to play with it! So instead of trying to explain every nuance to you (which would be tedious for both of us), I'll provide a sort of guided demonstration of the simulation and some of its features and typical operation. I'll do this in steps, but you should feel free to explore in any way you like.

1. Open up a quadcopter Simulink model
   a. Start with the AC_Quadcopter_Simulation.slx file.
2. This is an attitude-command-only model. In other words, there is no control system to track position. Instead the controller only tries to track attitude (ϕ, θ, ψ) and altitude (Z) commands using a PID controller (that isn't particularly well tuned either). When you first open the model you should see something like this (Figure 1).

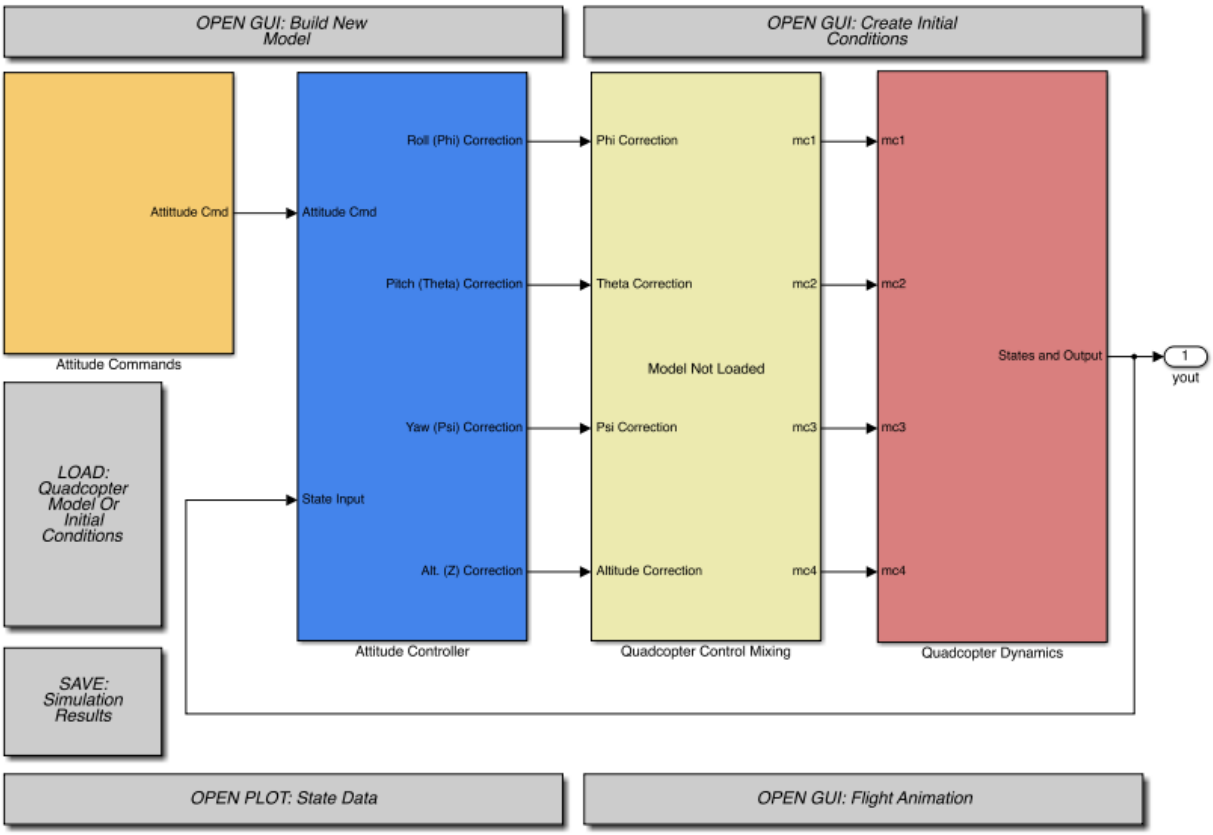

Figure 1. AC Simulation Overview

3. Read over what all the blocks say they do. Notice the grey blocks; these are buttons. Double clicking on any of these will do what the button says. (These blocks are empty subsystems with their "OpenFcn" set to execute code, making them act like buttons)
4. Double click the LOAD button and load an initial condition (IC structure), such as "Hover.mat". Double click the button again and load a quadcopter model (quadModel structure), such as "quadModel_+". Note that these structures now appear in the MATLAB workspace and can be interacted with or changed just like any MATLAB structure.
5. With these loaded the simulation can be run, but let's check out a few things first. Open the Attitude Commands block and look at the signal source blocks. These can be changed to any source you want, and what we've provided is just an example. Note that the simulation uses radians for angles and meters for distance except where otherwise noted (many interfaces and plots do a conversion internally and

will have different units labeled). Next check out the Attitude Controller block. Looking in here you'll see the following (Figure 2).

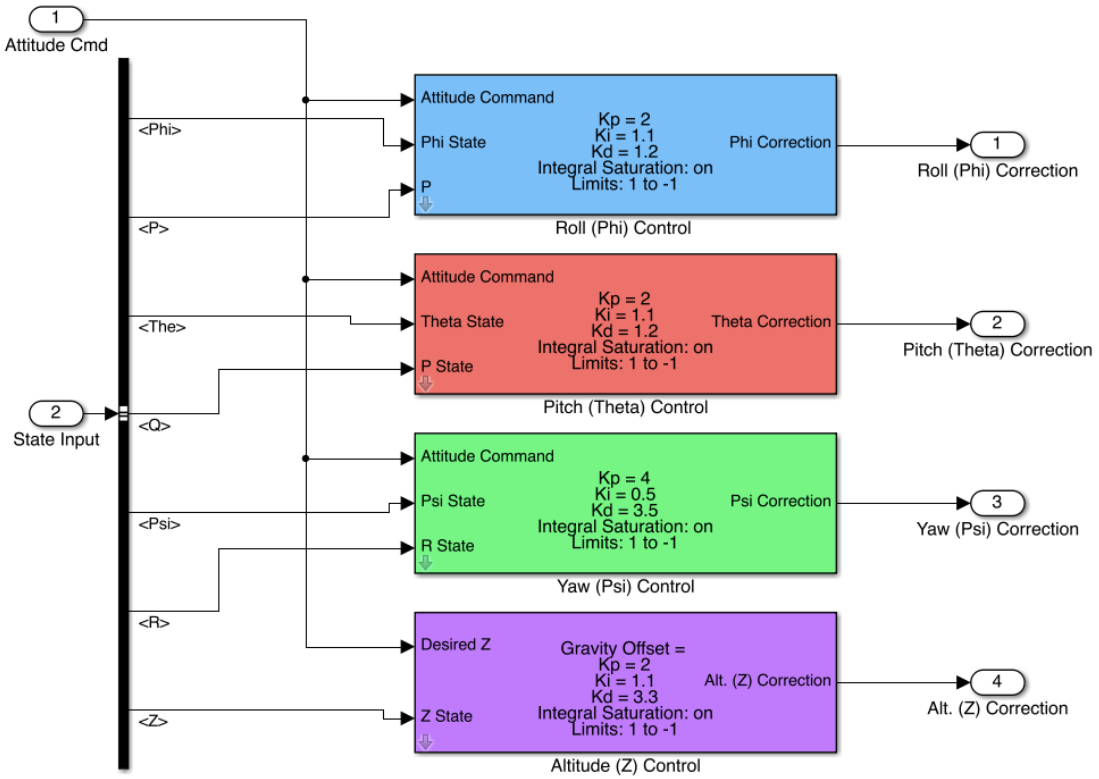

Figure 2. Attitude control block interior

6. The text on the front of these blocks is changed to reflect the mask parameter values. Click on one of the blocks and look at the interface for changing the PID gains. If you want to look at the PID control signal path inside, either click the little arrow in the bottom left corner of the block or right click and select ("Mask-Look under mask"). Notice anything unusual about the PID controller?
7. When you've explored that, back out and look inside the next block entitled "Quadcopter Control Mixing." This block takes the correction commands for the Phi, Theta, Psi and Z and "mixes" them by letting each correction be sent to the correct motor (sign is very important, consult our Math Modeling Document for the conventions we're using). It should say "Configuration "+"" on the front, indicating that it has recognized that you loaded a "+" configuration model. You should see something like this inside.

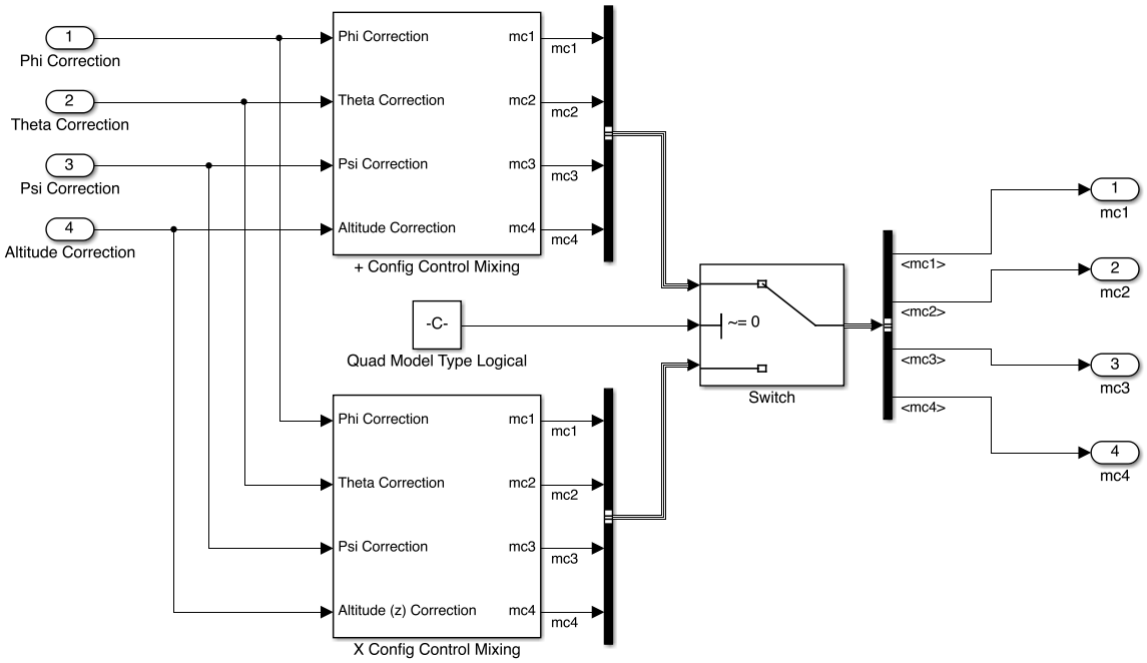

Figure 3. Quadcopter control mixing overview

8. The purpose of the switch mechanism is to allow both "X"-configuration and "+"-configuration vehicles to be controlled correctly without having to use a different block. Take a look inside both the "+" and "X" blocks. The plus block is the one that will actually be used right now since we loaded a "+" quadcopter model. The equations are written next to each output for reference (see Figure 4).

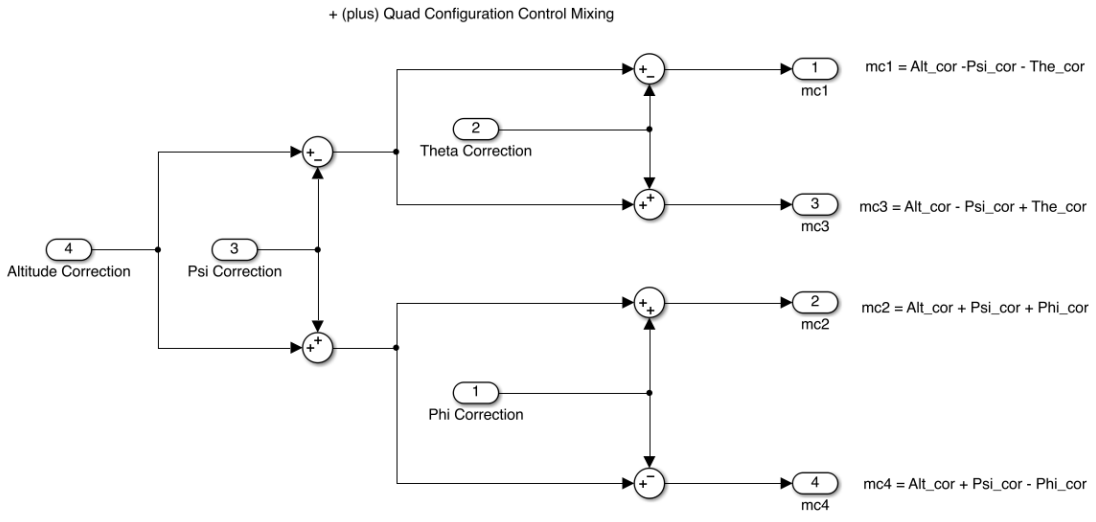

Figure 4. Plus configuration control mixing

9. It should be mentioned that these output signals are in terms of percent throttle. This is very important, as it allows us to tune the controller and expect the same performance from the simulated quadcopter for a given control system set of gains as we would get from a real-world quadcopter control system implementation (as long as the real implementation also computed response in terms of percent throttle, which is easy to achieve).
10. Next we'll check out the Quadcopter Dynamics block. Inside you'll see the diagram shown in Figure 5.

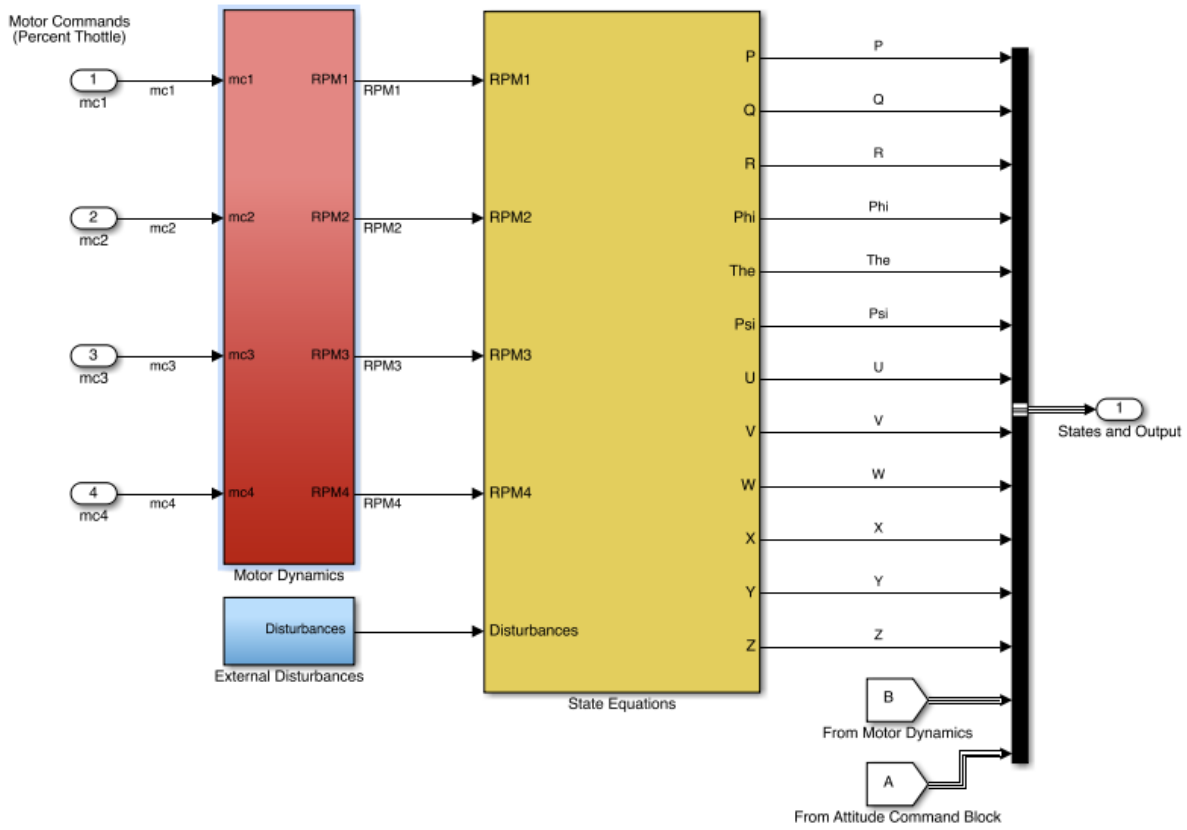

Figure 5. Quadcopter Dynamics block

11. The motor dynamics block restricts input commands to between 0% and 100% throttle (which is obviously the maximum possible range of throttle command signals), simulates the motor cutoff behavior at very low throttle, and most importantly applies the C_R and b linear relation to the percent throttle signal and simulates the first order delay. The output of the block is the RPM for each motor at any given moment in time.

12. The disturbance block is probably not something you'll use very much. It was added in order to make it easier for you to add external disturbance effects (such as wind forces on the vehicle) to the simulation. Use it as a simple example, and develop it to suit your needs. Otherwise, just leave all the disturbance inputs as zero.
13. Finally we have the heart of the simulation. The State Equations block executes a Level 2 S-Function written in the MATLAB language. This code can be viewed by clicking "edit" from the mask of the block (or opening the code file directly from MATLAB). This is where we simulate the state equations describing the dynamic behavior of the vehicle. Take a look at the code sometime and work through what it does. Having a working example should make it possible for you to relatively easily customize your simulation or add additional features and account for dynamic effects we have neglected. There is a LOT that can be done with S-Functions, so taking some time to look over our example in order to improve our model or build your own S-Function may well be a worthwhile effort. Notice on the mask of the S-Function block that there are parameter inputs called "quadModel" and "IC". This is where the structures from the workspace are passed to the S-function.
14. Okay, so if you have actually been patient and not already done this, let's run a simulation. If you have an "IC" and "quadModel" structure already loaded, try clicking the Simulink run button. You can fool around with what solver, step size, etc. you want to use later, but for now ode45 with the usual parameters set to "auto" will suffice (a fixed step solver may be preferable, but you can make that determination on your own).
15. After the simulation runs (should only take a few seconds), you can take a look at the simulation output by clicking the OPEN PLOT button. You'll get two figures that should be pretty self-explanatory. Notice that on the first plot, the attitude and altitude commands are shown with a dotted black line, while the response of the simulated behavior of the vehicle is shown with a blue, red, or green line. The other figure shows motor commands and motor speeds (in RPM). Any other plots you might be interested in during your work can either be added to the functions we have provided, or coded from scratch to your liking. Much of the simulation output

data is saved in 'yout', and any additional values or variables you want to save can be added using what we've done as an example.

16. Click on OPEN GUI: Flight Animation. See the documentation for this GUI for more info, and try watching the simulation results and clicking on stuff. You're bound to find a few bugs now and again (and again, and again…), but it beats only having those plots to look at, right?
17. That completes our basic walkthrough! Play with the simulation, experiment with it, change stuff, etc. Maybe you can find some flaws (there's bound to be at least a few), or a better way to do something. We really hope you find it helpful.

Additional Models:

1. **PC_Quadcopter_Simulation**

   Most autonomous quadcopter control projects hope to have their vehicle do more than just follow attitude commands. Using [1] as a guide, we developed a basic position tracking control system simulation model. The main difference between this model and the AC_Quadcopter_Simulation is the "Position Controller" block. This block accepts Timeseries inputs from the workspace, and uses a PD controller to command the attitude of the vehicle in order to fly the quadcopter to the desired location. Further information on the operation of this controller can be gained by exploring the functionality of the block, which is labeled and commented in hopes of helping you along. Note that this particular model is in no way optimized (none of the models we've provided are), so adjusting gains or changing control structures may very well improve the performance. This is left for you to explore. To show you how to make a Timeseries, we've included a demo script called "Path_Construction_Demo_Script" that shows you a relatively painless approach. You can even try drawing a picture this way if you like (Figure 6).

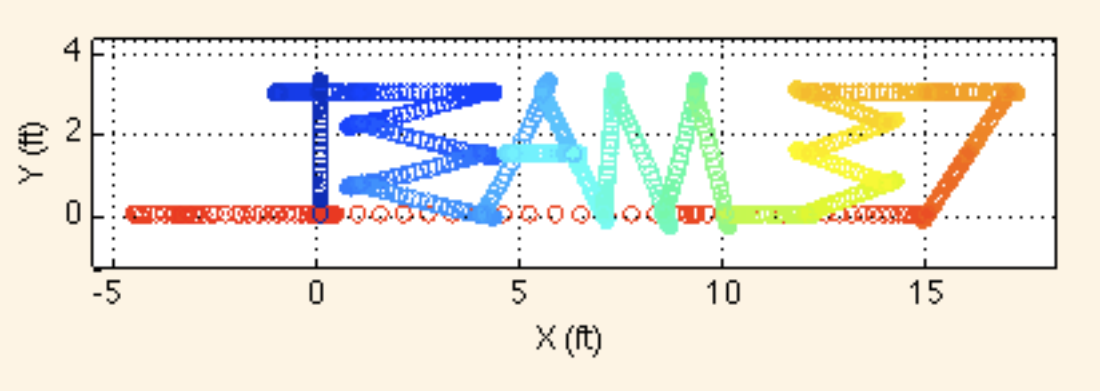

Figure 6. Customized path demonstration, (Shown on a modified version of the QuadAnim GUI)

2. Team37_Quadcopter_Simulation

Here you'll find a model representing part of our attempt at making a simulation to represent many of the real-world issues we faced during our implementation of a digital control system using a real microprocessor (Arduino Due) and sensor (MPU6050) onboard our vehicle. THIS IS NOT MEANT TO TEACH YOU HOW TO DO ANYTHING THE "RIGHT" WAY. We aren't even close to experts of this sort of thing, but we did achieve some success tracking attitudes with our quadcopter, and we think this simulation helped us work out reasonable PID gains, simulate filter effects, and predict the stability or instability of certain control system setups. Major changes from the continuous time simulations provided by the other two models include the following:

- A State Estimation block simulating the behavior of the onboard sensor data processing routine we used. This allowed us to explore our state estimation equations and also to see the effects of the discrete low-pass filter we used. Not included is random sensor noise present in the signal or (more importantly) vibration effects, which we found were very substantial in our investigation and affected our controller's performance.

- Multiple sampling points, such as the onboard state estimation block (about 400 Hz), the attitude command block (measured as 54 Hz, but not very vital to the simulation), and the sampling process between the control system and the ESC's (50 Hz, standard servo signal output rate using the Arduino servo library).
- Quantization effects: Due to data type and Arduino Servo library limitations, our percent throttle signals had to be rounded to the nearest 0.1% throttle. This lack of continuous signal capability has an effect on stability, and we felt it was an important effect to add to the model (See Figure 7).

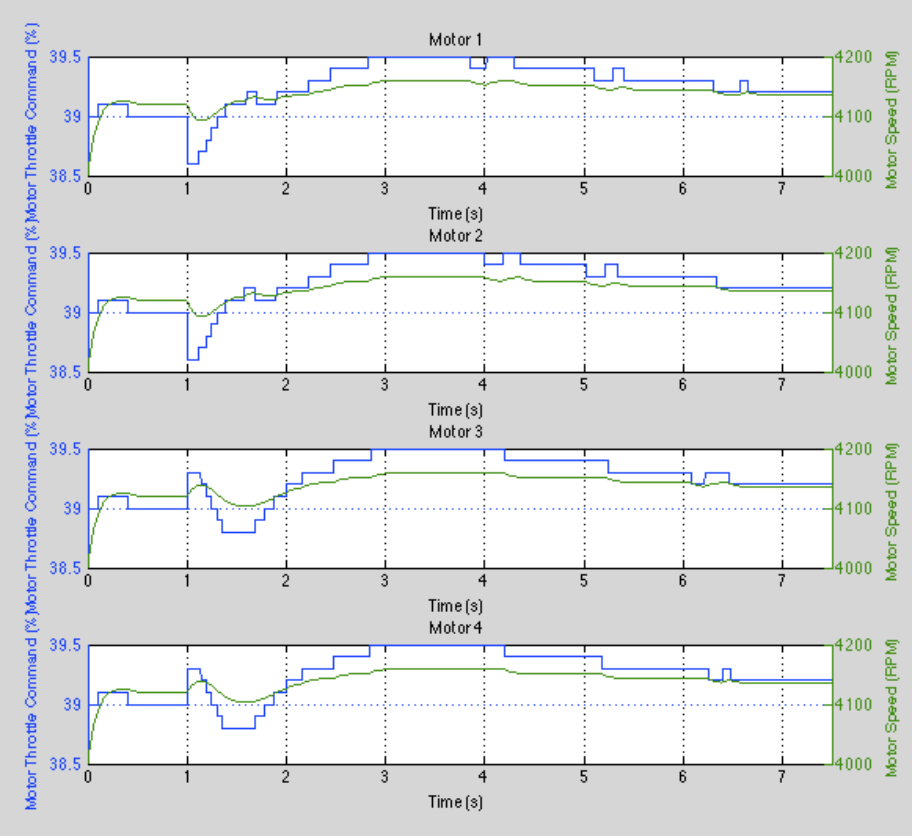

Figure 7. Quantized motor commands

That's most of the important stuff. When you dive into actual control system implementation, you can check out this model and perhaps it will give you a good idea or two for how to approach some of the challenges you will face. Note that this aspect of the project is the least well tested or explored, so we can guarantee there are many

improvements that could be made to this model to get more realistic performance or model effects in a better way.