Quadcopter Model

Automotive Systems Master

Software Based Automotive Systems

ASM2-SB / SS 2015

MODELLING OF QUADCOPTER



*Prof. Dr.Ing. Jörg Friedrich*

*Tutor: M.S. Vikas Agrawal*

Jha Vikas Kumar

Mahawar Nitesh Kumar

Table of Contents

**1. MATHEMATICAL MODEL 4**

1.1 Introduction 4

1.2 Mass Moment of Inertia Matrix 5

1.3 Initial Matrix Construction 6

1.4 Final Matrix Construction 7

1.5 State Equations 8

**2. DATA ANALYSIS PROGRAM 12**

**3. modelling gui 16**

**4** [**Introduction TO SIMULINK 19**](#_Toc257330270)

[1.1 Required skills and knowledge 19](#_Toc257330271)

[1.2 Required Materials 20](#_Toc257330272)

**5 AC QUADCOPTER SIMULATION 21**

List of Figures

Figure 1. Data Analysis Program…………………………………………………………………………………12

Figure 2. Modelling GUI……………………………………………………………………………………………16

Figure 3. AC Simulation Block…………………………………………………………………………………… 22

Figure 4. Attitude control block interior……………………………………………………………………. 23

Figure 5. Quadcopter control mixing overview……………………………………………………………24

Figure 6. Plus Configuration control mixing overview…………………………………………………28

Figure 7. Quadcopter Dynamics block…………………………………………………………………………29

1. **Mathematical Model:**

**1.1 Introduction**

Before jumping into the mathematic model, some discussion of notation is needed. Due to the complexity of a system with 6 degrees of freedom, various methods of notation have been developed and are required in order to sufficiently describe the critical variables. Shown below is an example of the notation we have chosen:

Here, the base variable is linear acceleration, or . As you can see, the variable also has two superscripts and one subscript to further delineate what we are describing. The top left superscript, b, tells us that the derivative taken was performed in the body frame of reference, while the top right superscript, , tells us that the acceleration is given in terms of body frame vector components, and the subscript, , tells us that this variable is referencing the center of mass with respect to the inertial frame.

Another important aspect of the math model is the coordinate system that is used. The chosen model and conventions will become very important as you work through your model, so be sure to keep your decisions in mind and clearly documented. The coordinate system will vary whether you use a plus (“+”) or “X” configuration.

The plus configuration we use is defined as having the X axis lie along the arm of motor 1 (which spins counter-clockwise from above, by our convention) with the Y axis set along the arm of motor 2 (spinning in the opposite direction of the adjacent motors) and the Z axis pointing upward. The value represents the distance from a given motor to the axis of rotation, and should be the same for every motor. This value will change if using an x configuration, which is defined as a rotation in the X-Y plane of 45 degrees in the positive yaw direction, which results in having the X axis lie between motor 1 and 2. In either configuration the x axis is assumed to be the positive forward direction for vehicle movement.

**1.2 Mass Moment of Inertia Matrix**

One element of the system of importance is the inertia matrix. The inertia matrix describes the quadcopters mass moment of inertia across the defined axes, and is important to the flight dynamics of the system. With some approximations, you can determine the mass moment of inertia across the X, Y, and Z axes in order to populate the required inertia matrix. This particular process is covered in more detail in the Mass Moment of Inertia documentation. Once determined using either the “+” or “X” configuration, the inertia matrix will appear as follows:

**(Mass Moment of Inertia Matrix)**

Here, is the inertia of the quadcopter relative to the body frame with , , and being the inertia of the quadcopter across each axis. Due to the symmetry of the system, the matrix is diagonal and **will be identical** for either a “+” or “X” configuration. The diagonal form of the matrix is convenient due to the need to invert the matrix for use in the angular velocity state equation.

**Thrust Coefficient**

The motors' thrust is the driving force behind all quadcopter maneuvers and thus is integral to control design and simulation. The thrust,, provided by a single motor/prop system can be calculated as follows:

Where is the thrust coefficient for a specific rotor, is the density of air, is the cross sectional area of the propeller's rotation, is the radius of the rotor, and is the angular velocity of the rotor. For simple flight modeling a lumped parameter approach can be used to simplify the characterization process:

**(Thrust Coefficient Relation)**

Here is the lumped parameter thrust coefficient that pertains to the individual motor/prop system. The thrust provided by the motor/prop provides a force perpendicular to the X-Y plane of the body frame in the positive Z direction.

**Torque Coefficient**

In order to understand motor effect on yaw, the torque force of the motor/prop system must also be determined, and can be done in a similar fashion to that of the thrust tests. The related lumped parameter equation is shown below:

**(Torque Coefficient Relation)**

In this case, is the torque created by the motor and is the torque coefficient for the motor/prop system. This torque provides a force that acts to yaw the system about the z-axis.

**1.3 Initial Matrix Construction**

After performing a range of tests with each of the test stands, the provided data analysis programs can help you calculate these coefficients for characterizing your system. With this information we can create a matrix describing the thrusts and torques on the system like that shown below:

**(“+” configuration)**

All of the present values have been explained thus far except for , which is simply the distance between the motors and the respective axes of rotation, where is the arm length from quadcopter hub center to motor/prop.

If using an x configuration, can instead be found by , as that would be the value for the distance between the motor/prop and the body's axes of rotation. Thus, experiences no change from this configuration adjustment, while the effect of will be distributed across all four motors for both pitch and roll.

**(“X” configuration)**

**Throttle Command Relation**

An important consideration here for control purposes is that the coefficients of thrust and torque are based on a relationship with RPM of the motors and not something directly determined by the control system (such as throttle command). Due to this, a linear regression is needed that will translate throttle command values (as percent throttle) to RPM values. The following regression was created for this purpose:

Here is the expected steady-state motor RPM, is the throttle percentage command, is the throttle % to RPM conversion coefficient, and is the y-intercept of the linear regression relationship. The linear regression can be performed using the provided data analysis program, which allows your controller to use the proper coefficients as determined by your motor testing for maximum accuracy and realism.

**Gyroscopic Forces**

There is one more set of forces to account for before we create our moment matrix, and those are the forces resulting from gyroscopic precession. Gyroscopic precession is a phenomenon that occurs when the axis of rotation of a rotating body is changed, and the results are typically non-intuitive to those unfamiliar with its effects. The gyroscopic forces resulting on the body are governed by the inertia of each motor’s rotating components (), the rolling and pitching rates ( and ), as well as the speed of each motor/prop system (). The gyroscopic torques created by the motors for pitch and roll action are shown below:

The π/30 term corresponds to the transition from RPM to radians that must occur for the gyroscopic force to be calculated.

**1.4 Final Matrix Construction**

With these motor/prop forces added in to the appropriate terms we can again organize the equations in matrix form for our simulation purposes. The resulting matrix will account for the mentioned aerodynamic, gyroscopic, and thrust moments created by the motor/prop systems on the quadcopter for a "+" configuration:

Here, refers to the moments present in the body frame resulting from the aerodynamics, thrusts, and torques on the system. The quadcopter body also experiences forces that act on it from gravity and the lift of the rotors. The lift force can be expressed as follows:

refers to the forces acting in the body frame on the quadcopter due to aerodynamics and thrust (assumed oriented strictly in the positive z direction). It should be noted that while we say there are acting aerodynamic forces, it is assumed that the static thrust and torque tests capture the elements of aerodynamics that we are interested in. Additional effects (such as blade flapping, frame aerodynamic drag, etc.) could be added to the model after additional research and testing.

**1.5 State Equations**

Now we move on to the state equations that define the dynamics model. The first we’ll discuss is the Angular Velocity State Equation.

**(Angular Velocity State Equation)**

This equation describes the change in roll (), pitch (), and yaw () rates of the quadcopter by taking into account the inertia, angular velocity, and the moments applied by the motor/prop systems. is the angular acceleration across each axis in the body frame with respect to the inertial frame, and can also be written as.

Having already the inertia matrix and moment matrices, we move on to which is a cross-product matrix for rotational velocity. The form of this matrix is shown below:

Here, P, Q, and R are again the rotation rates about the X, Y, and Z axis, respectively. The term is the rotational velocity of the quadcopter body within the body frame and is defined directly by P, Q, and R.

The next state equation defined is the Euler Kinematic Equation, which allows us to determine the rate of change of the Euler angles in the inertial frame.

Before discussing this equation we’ll discuss rotation matrices. According to the aerospace rotation sequence, the rotation of an aircraft is described as a rotation about the z-axis (yaw) then a rotation about the y-axis (pitch) followed by a rotation about the x-axis (roll). Each rotation is made based on a right-handed system and in a single plane.

Using these three rotations a composite rotation matrix can be created which can transform the motion of the aircraft from the body frame to a new reference frame. The resulting rotation matrix transforms rotations from the body frame with respect to the inertial frame and can be found using matrix multiplication. Below s, c, and t represent sine, cosine, and tangent functions respectively.

Following through with the matrix multiplication yields the rotation matrix from the inertial to the body frame using the aerospace rotation sequence:

**(ZYX Sequence Rotation Matrix)**

This rotation matrix is of particular importance in solving the velocity and position state equations. A complete discussion of rotation matrices is beyond the scope of this document.

Using sequential rotation matrices, the angular velocity of the aircraft in the body frame can be related to the changes in angle rotation as shown below, where the C matrices of are those from u*b*.

Performing matrix multiplication and addition and taking the derivative the Euler Kinematic Equation can be found (details not given here):

**(Euler Kinematic Equation)**

While this approach is effective, there is one very important drawback; a singularity occurs when *θ* is equal to ±. Due to this, the accuracy and numerical stability of a simulation can be compromised if the aircraft's pitch approaches or reaches ±90°. Considering the modest control design intentions of this simulation, this will not be an issue for most users. However, several approaches to avoiding this problem exist, including using quaternions for the simulation, and thus motivated users may choose to modify our simulation to utilize this or another approach to removing this singularity.

The next state equation we discuss is the Velocity State Equation, which describes the acceleration of the center of mass of the rigid body quadcopter model based on the forces and accelerations acting on the body.

**(Velocity State Equation)**

Here, is the linear acceleration of the center of mass in the body frame with respect to the inertial frame. The variable is the total mass of the quadcopter, while is the acceleration of gravity translated to act in the body frame by the rotation matrix .

Using this equations you can find the linear acceleration of the quadcopter in the X, Y, and Z directions of the body frame.

Finally, the last state equation to be covered is the Position State Equation, which describes the linear velocity of the center of mass of the quadcopter in the inertial frame.

**(Position State Equation)**

# Here, is simply the velocity of the quadcopter in the body frame that is rotated into the inertial frame using the transpose of , which is . This state equation allows us to determine the velocity of the quadcopter in the X, Y, and Z directions of the inertial frame.

1. **Data Acquisition & Analysis**

# Importing Data to the MATLAB Workspace

To obtain the performance parameters like CT, CQ, TC, CR we measure RPM, thrust, torque, and a first order model time constant for the motor, ESC, and prop system. The overall Motor Test System includes designs for the required hardware and circuits, an Arduino Uno program that facilitates easy data collection, and a data analysis program that makes data analysis simple and fast. Hence RPM, Thrust and Torque are tabulated at different time intervals in an Excel Spreadsheet.

**Step-by-step process of importing data from the Arduino COM window or an Excel spreadsheet, into the MATLAB workspace.**

**Step 1** Hold “Shift” (or equivalent for your computer) and click to select all the way up to the top of the data. As seen below in , make sure to include the column headings in the selection. This makes MATLAB correctly name the columns during import, saving you some time.

**Step 2** “Ctrl + C” (or equivalent) to copy the data, and Right-click + Paste into the MATLAB workspace window.

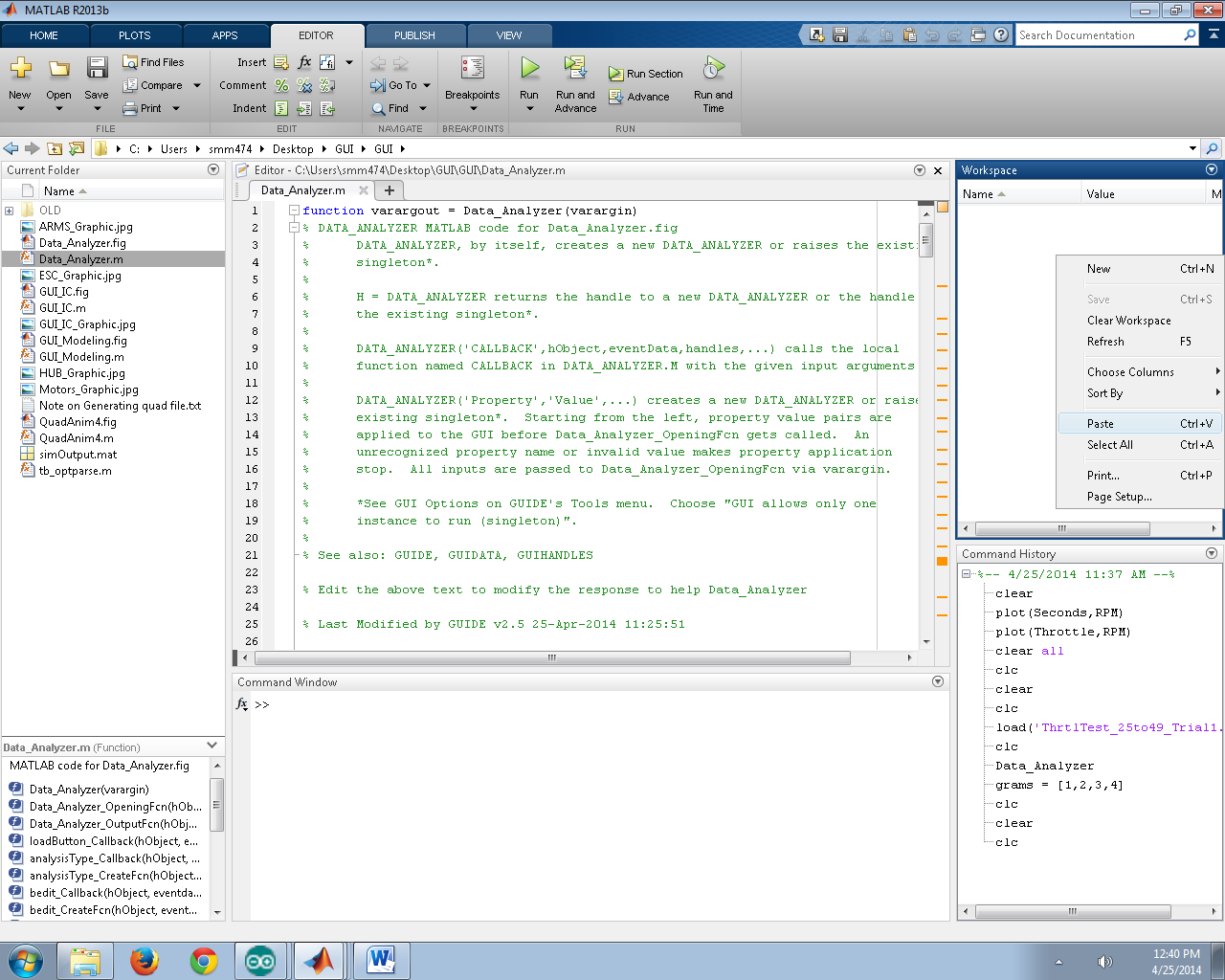


Figure 1: Pasting data into the MATLAB workspace

**Step 3**: The “Import” window will open, as seen below in . MATLAB is able to automatically populate the column headings as well (if not, columns must be labeled “**Seconds**”, “**Throttle**”, and “**RPM**” ). Select the green checkbox “Import Selection” at the top right of the Import window and MATLAB will import and parse the data ().

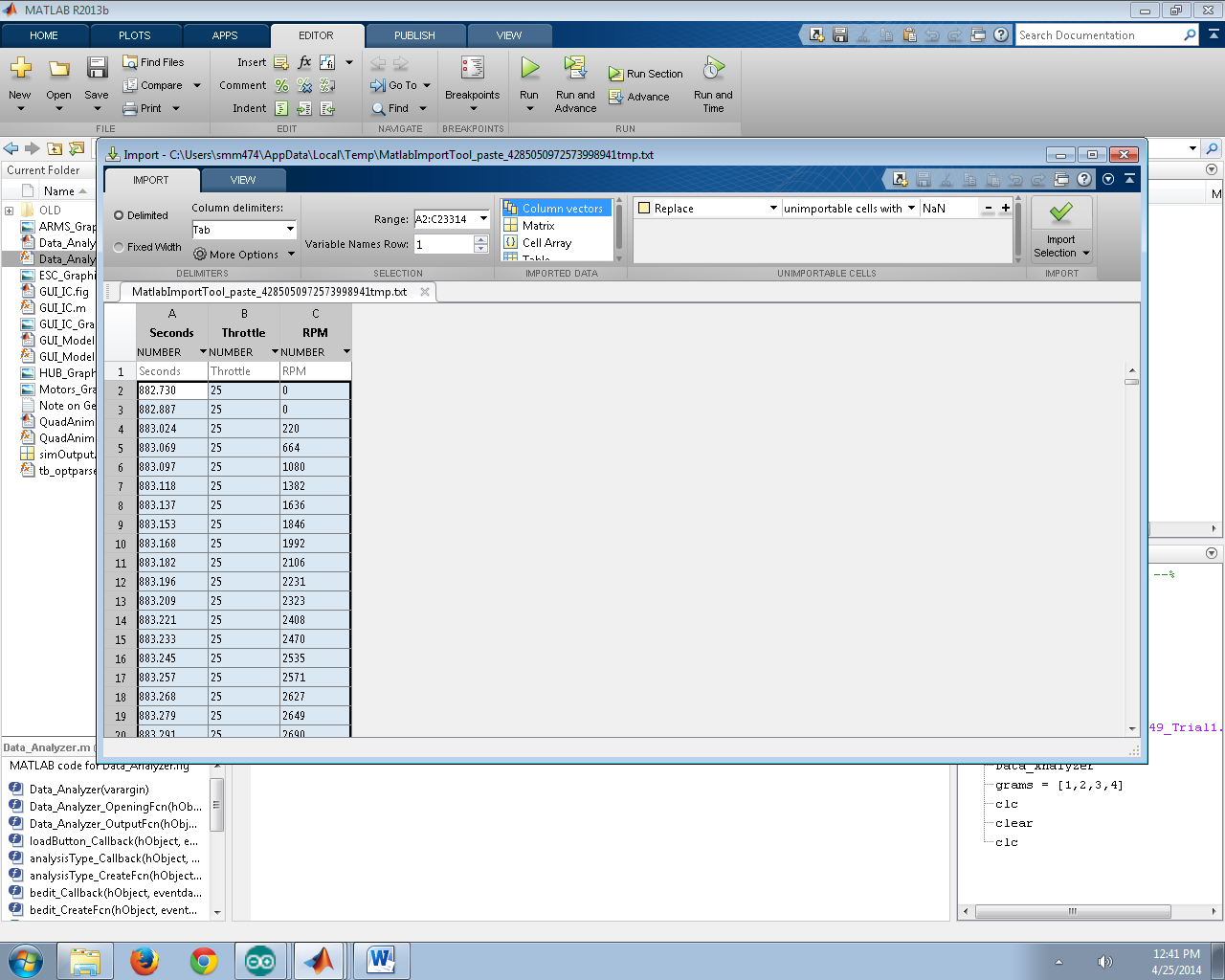


Figure 2. Importing the selection into the workspace

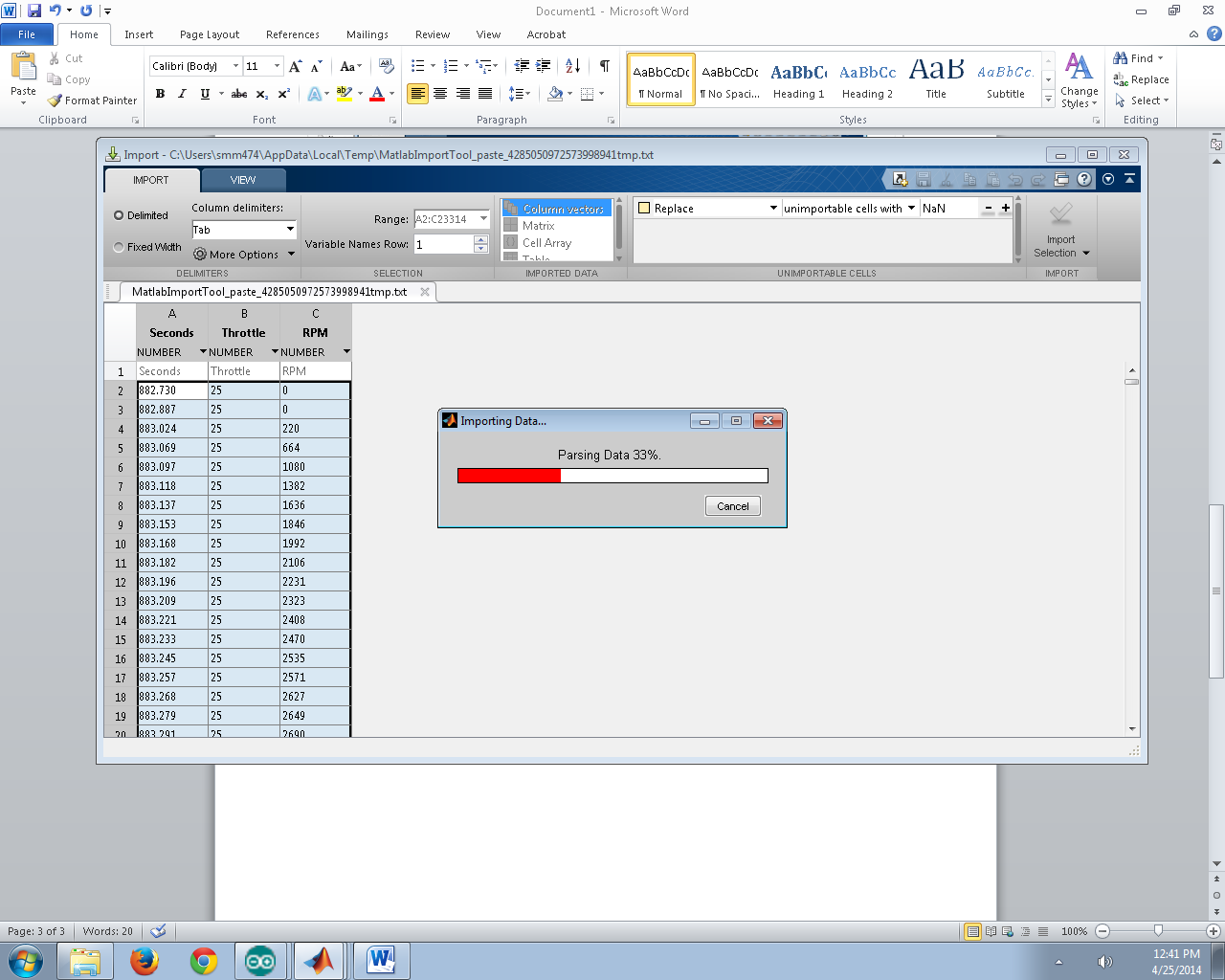


Figure 3. Parsing the data

The data is now imported to the MATLAB workspace (4) as individual column vectors.

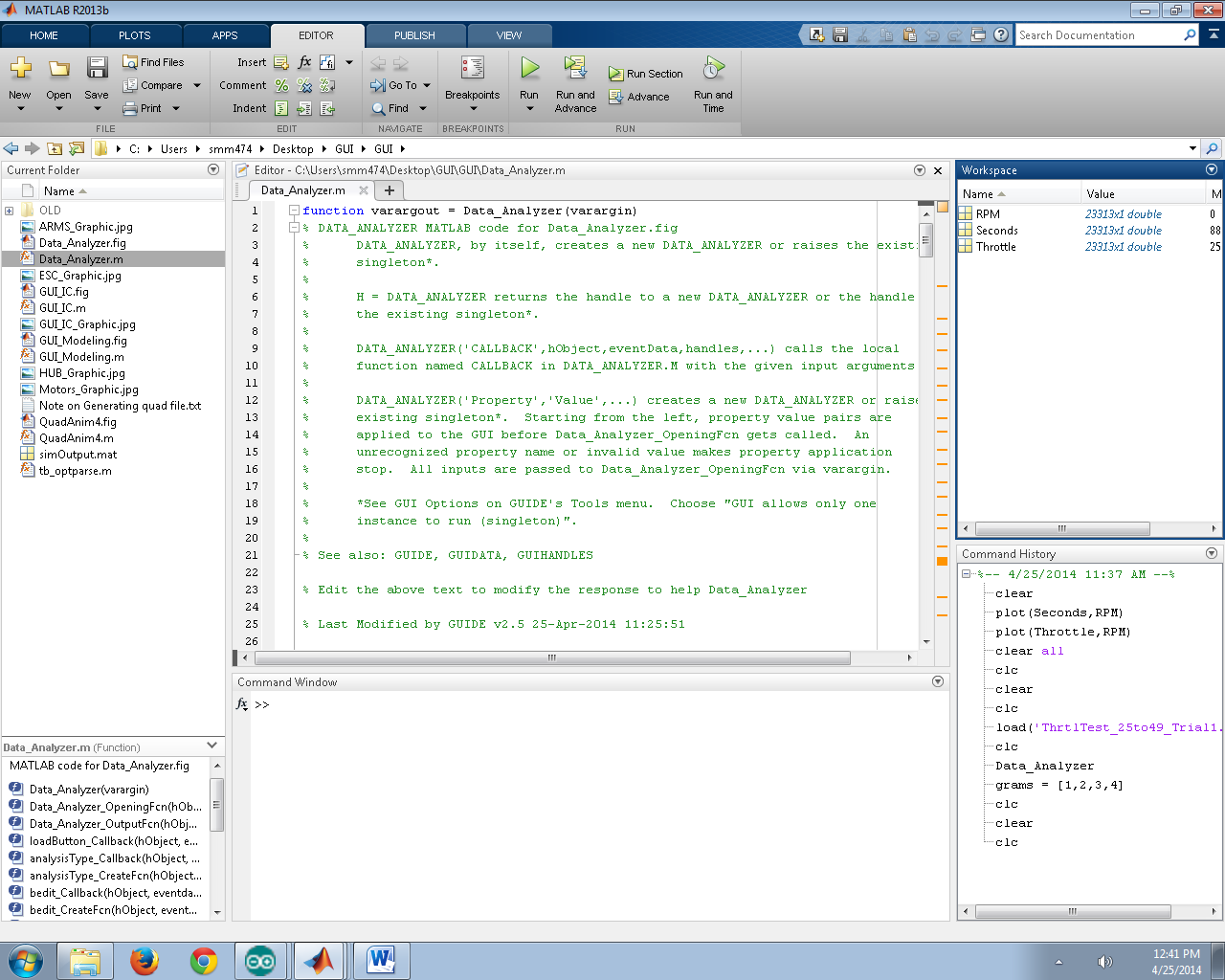


Figure 4. Data imported to workspace

**Step 4**: Copy raw scale measurements from an Excel sheet, as seen below in 5. If Excel was not used, you can manually enter a column vector of measurements.

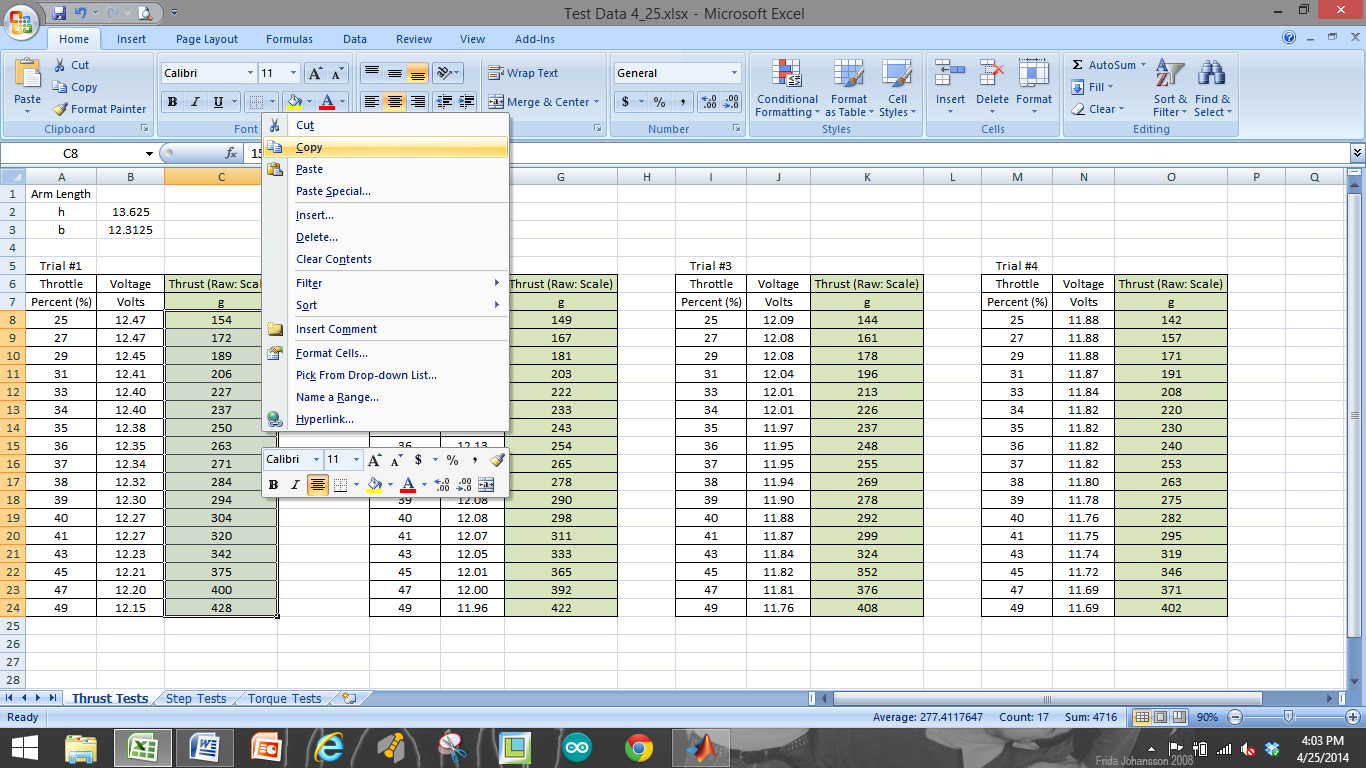


Figure 5. Copying data from Excel sheet

**Step 5**: This data from Excel can now be pasted into the MATLAB workspace along with the data from the Arduino COM window. This can be seen in 6.

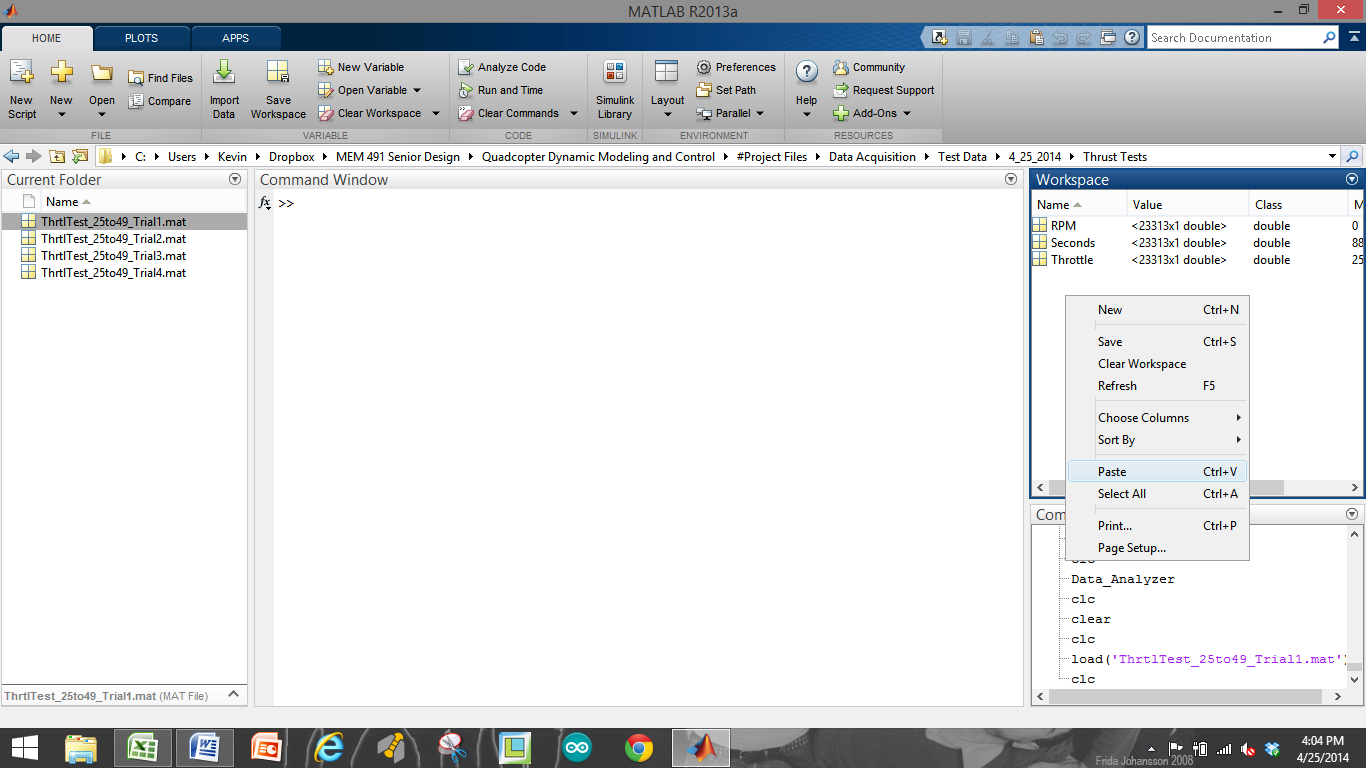


Figure 6. Pasting data into workspace

**Step 6**: This step is important, as the analysis tools we provide require that the measurements from the scale be stored as a vector with a specific name. After selecting “Paste” into the workspace, MATLAB’s Import window will once again open, except that MATLAB now doesn’t know what to call the column heading. For this case, we must call the column heading “**gramsMeas**”. Once the heading is labeled, you can select “Import Selection” at the top right of the screen and MATLAB will import this data from Excel into a vector called “**gramsMeas**”. This is seen below in 7.

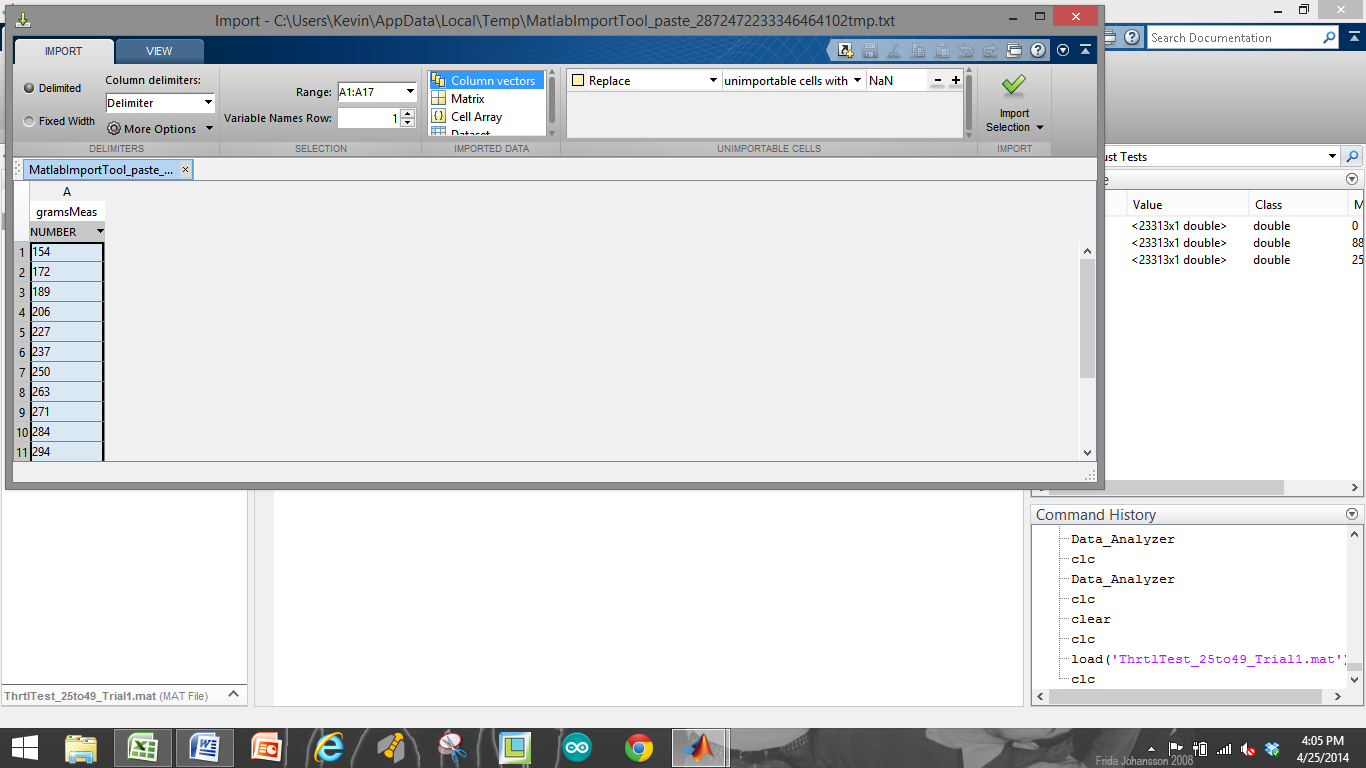


Figure 7. Importing the data into vector called "gramsMeas"

**Step 7**: Finally, we are able to save all of this workspace data as a .mat file into a location of our choice for later use in the data analysis programs. Select all (4) variables in the workspace window, right click, “Save As…” and Save the data into the desired folder with a meaningful name (date, tests performed, variables of interest, etc.). See below.

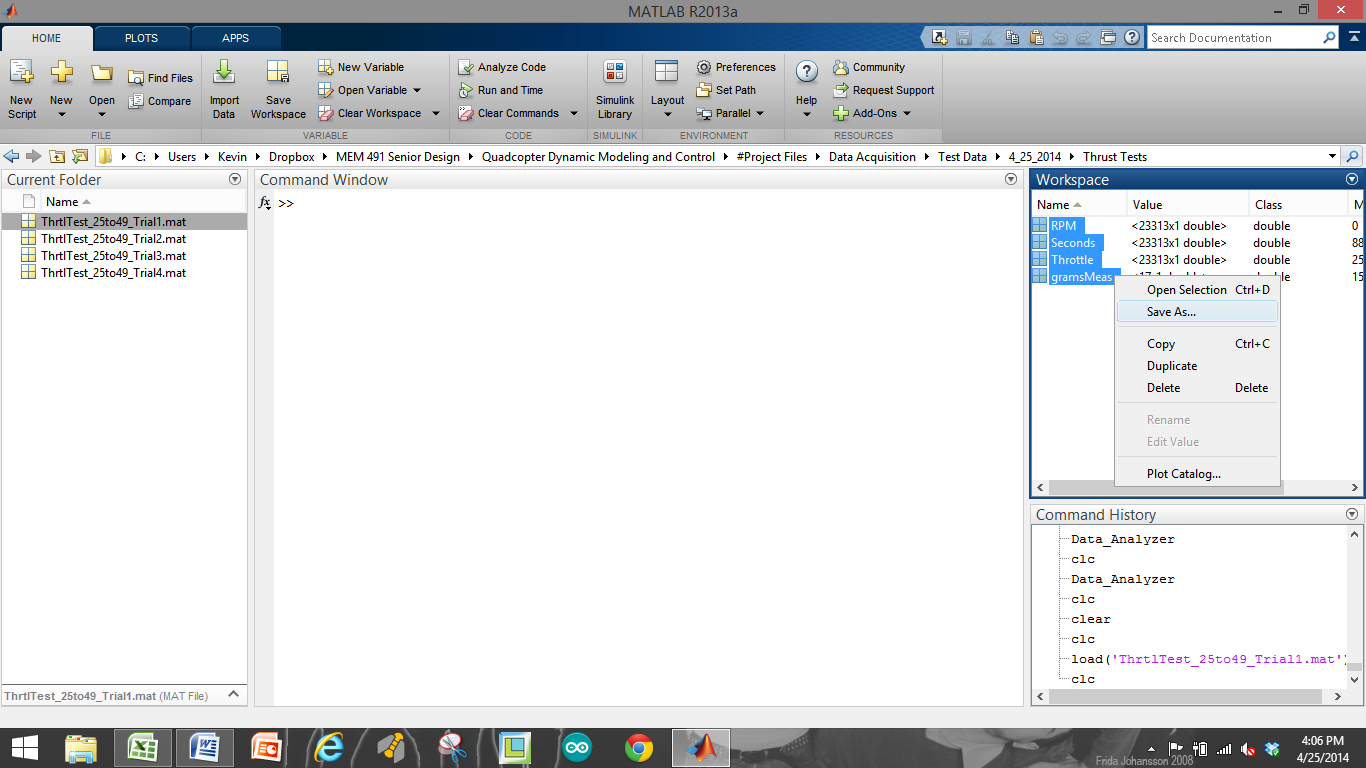
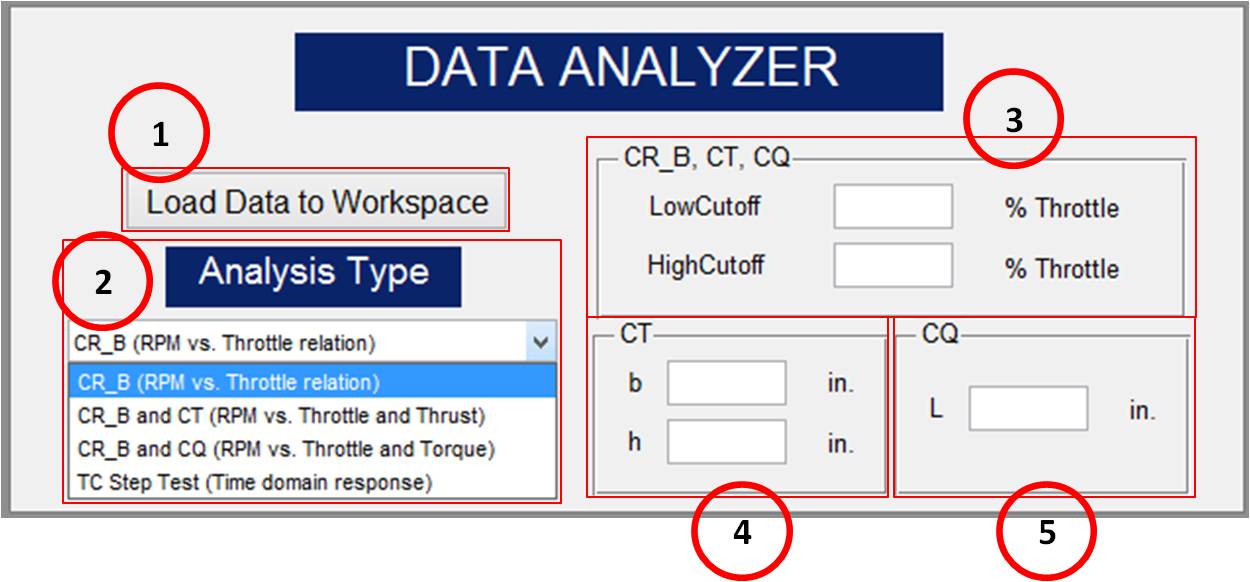


Figure 8. Saving the data

**DATA ANALYZER**

*Figure 9. Data Analysis Program*

The simulation requires us to estimate several quadcopter performance parameters that are acquired through experimental testing of the motors of the quadcopter vehicle. This GUI was made to help analyze this motor test data. The parameters this GUI is able to calculate are called: CR and b (Throttle to RPM conversion), CT (Thrust coefficient), CQ (Torque coefficient), and TC (motor time constant). By loading test data into the workspace the GUI can automatically calculate parameters and plot the data. The Data Analysis GUI can be seen below in 9.

**Description:**

1. Load button: this button loads in structures of data that were previously saved. The data loaded should correspond to the analysis to be performed (i.e. don’t try to use the TC Step Test analysis on Throttle Response Data!)
2. Analysis Type – there are 4 options accessible through the drop down box:
   * **CR\_B (RPM vs. Throttle relation)**

|  |  |
| --- | --- |
| Required User Input:  *Optional User Input:* | Seconds, Throttle, and RPM data *Low Cutoff, High Cutoff* |

* + 1. This type of analysis requires experimental data input of Seconds, Throttle, and RPM. It is the most general type of analysis this interface can perform. No raw scale readings of thrust or torque are needed. Once the “Run” button is selected, the program will analyze the RPM vs. Throttle data to obtain a fit that calculates the and parameters (linear relation with non-zero intercept between throttle percentage and RPM).
    2. The LowCutoff and HighCutoff for throttle percentage can also be included in this program. These values are entered as shown on figure 1 above. For example, if a test is run between throttle limits of 0 and 60%, you may still only want the program to analyze the data between values of 20 and 40% throttle. This is an important concept since having the fit be accurate around the anticipated normal operating range is often more important that having the fit be based on the full possible range of RPM values. Therefore, we recommend some iteration involving finding the approximate throttle required for hover and then performing additional testing and parameter fits around this range (+/- 5-10% throttle for example). This will produce more accurate simulation results, and presumably better control design, at near hover conditions, which are usually the conditions of primary interest. If these values aren’t entered, the program will automatically find the lowest (non-zero) and highest throttle values from the test data and use those instead.
    3. See the function “calculate\_CR\_B.m” for more details. It is strongly recommended that users spend some time understanding the function since the approach utilized might not be ideal for all users.
  + **CR\_B and CT (RPM vs. Throttle and Thrust)**

|  |  |
| --- | --- |
| Required User Input:  *Optional User Input:* | Seconds, Throttle, and RPM data  gramsMeas (raw scale), b, h  *Low Cutoff, High Cutoff* |

* + 1. This type of analysis requires data inputs of Seconds, Throttle, RPM, and grams measured (gramsMeas: raw scale readings from thrust test). The analysis also requires inputs for the dimensions of the thrust test rig, namely: b and h. “b” is the distance from the pin joint on the rig to the point of contact with the scale. “h” is the height of the center motor axis above the pin joint on the rig.
    2. NOTE: If the “Load Data to Workspace” button is used but the only data saved is Seconds, Throttle, and RPM, the gramsMeas (raw scale Thrust values) parameter must be added by another means, such as direct vector entry at the MATLAB command window or copy paste and import from a spreadsheet.
    3. As with the CR\_b analysis, this type of analysis will also accept user inputs of low and high cutoff for Throttle %. See discussion above.
    4. Once the data is loaded and the user inputs filled in, this program once again calculates the and parameters along with the coefficient (Thrust coefficient). is a relation between Thrust and RPM2.
    5. See the function “calculate\_CT.m” for more details. It is strongly recommended that users spend some time understanding the function since the approach utilized might not be ideal for all users.
  + **CR\_B and CQ (RPM vs. Throttle and Torque)**

|  |  |
| --- | --- |
| Required User Input:  *Optional User Input:* | Seconds, Throttle, and RPM data  gramsMeas (raw scale), L  *Low Cutoff, High Cutoff* |

* + 1. This type of analysis requires data inputs of Seconds, Throttle, RPM, and grams measured (gramsMeas: raw scale readings from torque test). The analysis also requires an input for a Torque rig dimension, L. “L” is the perpendicular distance between the axis of the motor and the axis of the arm that is in contact with the scale.
    2. NOTE: If the “Load Data to Workspace” button is used but the only data saved is Seconds, Throttle, and RPM, the gramsMeas (raw scale force values in grams) parameter must be added by another means, such as direct vector entry at the MATLAB command window.
    3. As with the and analysis, this type of analysis will also accept user inputs of low and high cutoffs for Throttle %.
    4. Once the data is loaded and the user input filled in, this program once again calculates the and parameters along with the coefficient (Torque coefficient). is a relation between Torque and RPM2.
    5. See the function “calculate\_CQ.m” for more details. It is strongly recommended that users spend some time understanding the function since the approach utilized might not be ideal for all users.
  + **TC Step Test (Time domain response)**

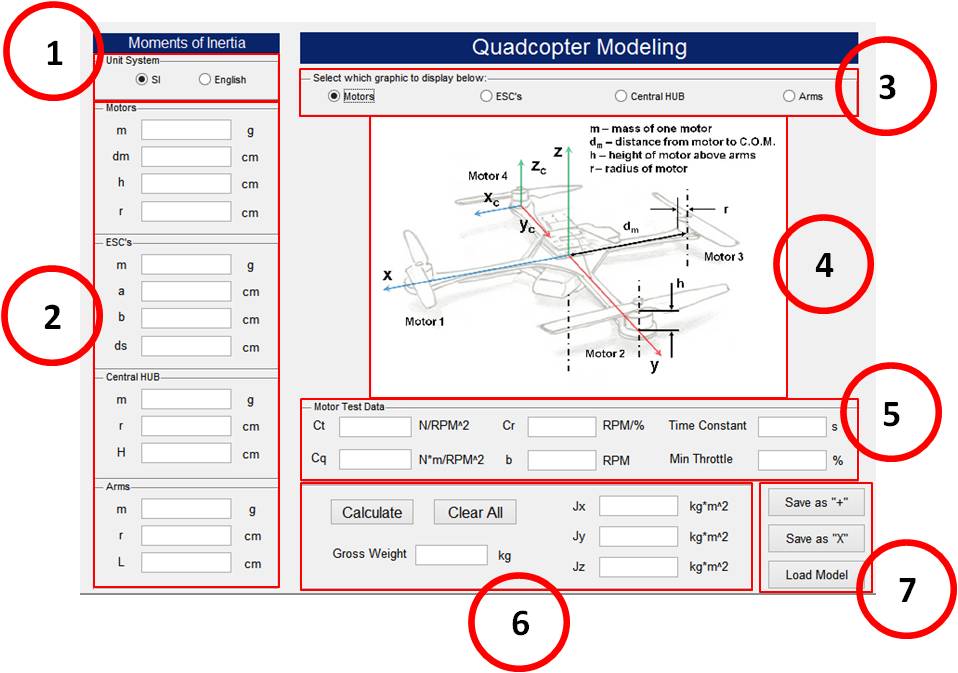
|  |  |
| --- | --- |
| Required User Input:  *Optional User Input:* | Seconds, Throttle, and RPM data  *N/A* |

* + 1. This type of analysis requires data inputs of Seconds, Throttle, and RPM. The program calculates the time constant () of the motor from a step change in throttle input.
    2. The program does not utilize the low and high cutoffs for Throttle %, because it automatically recognizes the two throttle settings that the step command is operating between.
    3. See the function “calculate\_TC.m” for more details. It is strongly recommended that users spend some time understanding the function since the approach utilized might not be ideal for all users.

1. **Modeling GUI**

**Purpose:**

This GUI accepts user input of physical measurements from their quadcopter vehicle. A primary purpose of this interface is to calculate the moment of inertia matrix for the vehicle based on specific dimensions and measurements of the quadcopter. The interface also accepts the performance coefficients and other parameters obtained through motor testing. Once all of the fields are populated, the interface saves a MATLAB “structure” that contains all the parameters required to run the simulation. A structure is a data type that gives us a way to combine multiple pieces of information into a single variable that can easily be passed around the MATLAB environment. The modeling GUI can be seen below in 1.



*Figure 11. Modeling GUI*

**Description:**

1. Unit System toggle: toggle units between SI and English based on the measurement tools being utilized. IMPORTANT: Whether inputs are in grams/centimeters or ounces/inches, the program will convert everything into the SI unit system (kilograms/meters) for use in the simulation.
2. Inputs for Motor, ESC, HUB, and Arm dimensions (either in g/cm or oz./in.).
   * Motors:
     + m: mass of 1 motor (with propeller) using scale
     + dm: distance from the center of mass (COM) of the motor to the COM of the quad vehicle (COM is assumed to coincide with geometric center!)
     + h: height of the motor above the arm (not including prop axel)
     + r: radius of a motor
   * ESC’s:
     + m: mass of 1 ESC
     + a: width of an ESC
     + b: length of an ESC
     + ds: distance from the COM of the ESC to the COM the quad vehicle
   * Central HUB:
     + m: total mass of the central HUB (including battery, controller, power distribution board, etc.). This value might be most easily obtained by subtracting the mass of the individual components mentioned herein by the total weight of the quad in a “ready-to-fly” state (i.e. Hub mass = Total mass – (Motors + ESC’s + Arms).
     + r: radius of the central HUB (modeled as a cylinder, estimate as needed)
     + H: height of the central hub (total height, modeled as a cylinder, see Hub diagram)
   * Arms:
     + m: mass of (1) arm (motor, ESC, etc. excluded)
     + r: radius of an arm (modeled as a cylindrical rod)
     + L: length of an arm (arm only up to attachment to HUB)
3. Toggle between graphics to display in center of GUI to assist in measuring parameters.
4. Motor, ESC, HUB, and Arm graphics displayed in center of GUI
5. Motor Test Data inputs: Coefficients obtained from motor test data are entered here. Also, “Min Throttle” (the minimum throttle setting for which actual motor rotation is achieved) can be entered. If this value is not known, simply enter “0”.
6. Select the “Calculate” button to calculate the Gross Weight of the vehicle, and the Jx, Jy, and Jz values of the inertia matrix. Or, select “Clear All” to clear all of the GUI fields.
7. Once the fields are populated, the user can select “Save as +” or “Save as X” to save the model in a “plus” configuration or in an “X” configuration. The parameters are saved inside a MATLAB “structure” from which the Simulink simulation extracts them during simulation. If a model is already saved, the user can select “Load Model” to select a desired structure and the GUI will populate the fields with these saved parameters. This is useful because if the user wishes to adjust 1 parameter of their model they don’t need to retype everything in again.

# INTRODUCTION TO SIMULINK

## 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 reasonableand 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 desireto 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.

## 1.1 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.

## 1.2 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

1. 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 (and altitude (Z) commands using a PID controller (that isn’t particularly well tunedeither). When you first open the model you should see something like this

(Figure 1).

*Figure 3. AC Simulation Block*

1. 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)
2. 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.
3. 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 anysource 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 4. Attitude control block interior

1. 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?
2. 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 5. Quadcopter control mixing overview

1. 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 6. Plus configuration control mixing

1. 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).

Next we’ll check out the Quadcopter Dynamics block. Inside you’ll see the diagram shown in Figure 5.



Figure 7. Quadcopter Dynamics block

1. 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 and 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.
2. 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.
3. 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.
4. 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).
5. 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.
6. 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?
7. 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.