ECE 3111: Systems Analysis Project

Quadcopter System Model

by Atharva Manjrekar

Dr. Ashwin Dani

TA: Iman Salehi

Collaborators: Ajani Blackwood, David Szymanowski, Tariq Ali

Table of Contents

Introduction

A quadcopter is an aerial mobile vehicle that has four identical rotors arranged at the corner of it's chassis. The four rotors are paired with propellers that have a fixed angle of attack. An example of a quadcopter configuration can be seen in *Figure 1* below. Here, we can see that the movement of the quadcopter and it's attitude can be described using both inertial and body frames. In this project, we would be designing a quadcopter system model whose position and attitude can be controlled. We would first start by mathematically modeling the quadcopter platform. This would involve using the provided non linear equations of motion (EOM) and linearizing them to form a linearized system. A linearized system would help us derive transfer functions and state space models. We would also study the open loop response of the quadcopter plant in Matlab, mirroring the same principles that we learned in class. Finally, we would design a proportional integral derivative (PID) controller to stabilize the quadcopter system using Simulink. Our controller would be designed such that it drives the quadcopter to a height of five feet with a distribution of ±0.01524 meters or ±0.05 feet.

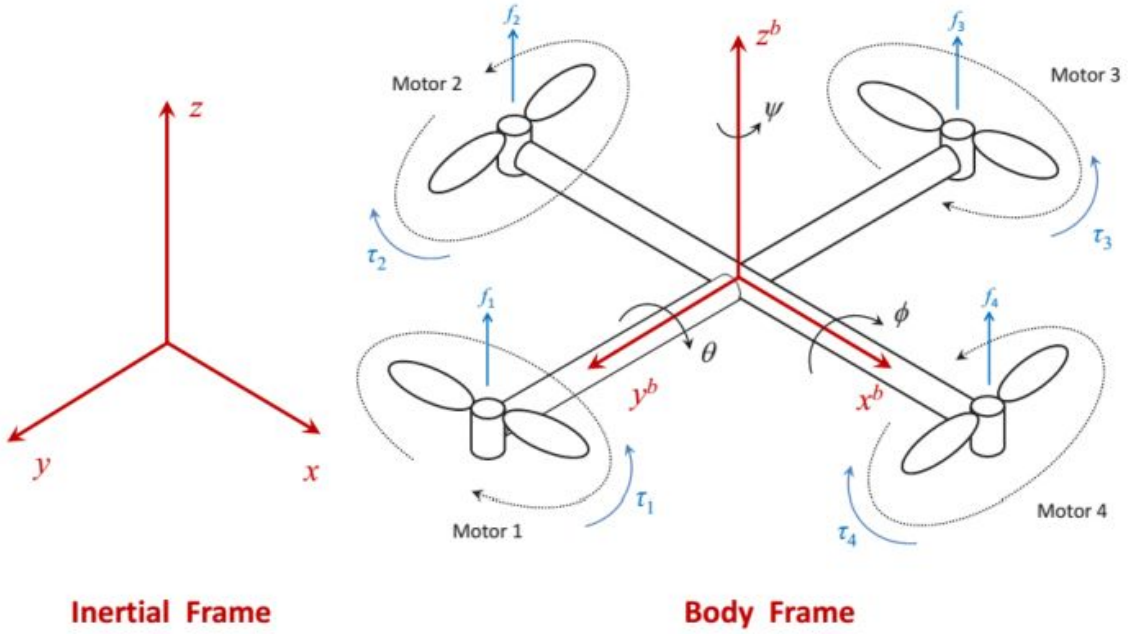

Figure 1: Quadcopter configuration

1. Linearization

1.1. Linear Approximation

The entire model of our quadcopter could be represented in state space form as shown in *Figure 1.1A*. Using the values of roll-φ, pitch-θ, and yaw-ψ which are provided as 0, we can efficiently linearize our equations. The x, y, and z values are also 0. For our controller design, we would linearize the provided equations so that we get *Figure 1.1B*. The non linear EOM were linearized using the definition of linearization that we learned in class. For example, X^{dot}, Y^{dot}, and Z^{dot} all stay the same as linearizing a constant yields the same variable. Since the trig identity inside the nonlinear equations of V_x^{dot} and V_y^{dot} result in a zero, the right portion of the equation ends up being multiplied by a zero and as such we are only left with the term unchanged in the left. This same principle is used to linearize the remainder of the equations which yields what we see in *Figure 1.1B* below. We would be using the provided parameters to simplify our linearized EOM so that we can plug in these values into our matrices for the next section. These parameters are shown in *Figure 1.1C* below. It should be noted that a voltage squared (v_1^2) is equivalent to an arbitrary variable called U_1.

$$u_1 = v_1^2,\ u_2 = v_2^2,\ u_3 = v_3^2 \text{ and } u_4 = v_4^2$$

$$\dot{x} = v_x \tag{5}$$

$$\dot{y} = v_y \tag{6}$$

$$\dot{z} = v_z \tag{7}$$

$$\dot{v}_x = -\frac{k_d}{m}v_x + \frac{kc_m}{m}\left(\sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta\right)\left(v_1^2 + v_2^2 + v_3^2 + v_4^2\right) \tag{8}$$

$$\dot{v}_y = -\frac{k_d}{m}v_y + \frac{kc_m}{m}\left(\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi\right)\left(v_1^2 + v_2^2 + v_3^2 + v_4^2\right) \tag{9}$$

$$\dot{v}_z = -\frac{k_d}{m}v_z - g + \frac{kc_m}{m}\left(\cos\theta\cos\phi\right)\left(v_1^2 + v_2^2 + v_3^2 + v_4^2\right) \tag{10}$$

$$\dot{\phi} = \omega_x + \omega_y\left(\sin\phi\tan\theta\right) + \omega_z\left(\cos\phi\tan\theta\right) \tag{11}$$

$$\dot{\theta} = \omega_y\left(\cos\phi\right) + \omega_z\left(-\sin\theta\right) \tag{12}$$

$$\dot{\psi} = \omega_y\left(\sin\phi/\cos\theta\right) + \omega_z\left(\cos\phi/\cos\theta\right) \tag{13}$$

$$\dot{\omega}_x = \frac{Lkc_m}{I_{xx}}\left(v_1^2 - v_3^2\right) - \left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right)\omega_y\omega_z \tag{14}$$

$$\dot{\omega}_y = \frac{Lkc_m}{I_{yy}}\left(v_2^2 - v_4^2\right) - \left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right)\omega_x\omega_z \tag{15}$$

$$\dot{\omega}_z = \frac{Lkc_m}{I_{zz}}\left(v_1^2 - v_2^2 + v_3^2 - v_4^2\right) - \left(\frac{I_{xx} - I_{yy}}{I_{zz}}\right)\omega_x\omega_y \tag{16}$$

Figure 1.1A: Quadcopter non linear model equations in state space form

```
% Linearized EOM
Xdot = Vx;                                                    %Equation 5

Ydot = Vy;                                                    %Equation 6

Zdot = Vz;                                                    %Equation 7

Vx_dot = (-Kd/Mass)*Vx + (g*Pitch);                           %Equation 8

Vy_dot = (-Kd/Mass)*Vy - (g*Roll);                            %Equation 9

Vz_dot = (-Kd/Mass)*Vz - g + (K*Cm/Mass)*(U1+U2+U3+U4);       %Equation 10

Roll_dot = Wx;                                                %Equation 11

Pitch_dot = Wy;                                               %Equation 12

Yaw_dot = Wz;                                                 %Equation 13

Wx_dot = ((L*K*Cm) / Ixx) * (U1-U3);                          %Equation 14

Wy_dot = ((L*K*Cm) / Iyy) * (U2-U4);                          %Equation 15

Wz_dot = ((L*K*Cm) / Izz) * (U1-U2-U3-U4);                    %Equation 16
```

Figure 1.1B: Linearized equations of motions

Parameter	Symbol	Value	Unit
Mass of the quadcopter	m	0.5	kg
Radius of the quadcopter	L	0.25	m
Propeller lift coefficient	k	$3 \cdot 10^{-6}$	$\mathrm{N\,s^2}$
Propeller drag coefficient	b	$1 \cdot 10^{-7}$	$\mathrm{N\,m\,s^2}$
Acceleration of gravity	g	9.81	$\mathrm{m/s^2}$
Air friction coefficient	k_d	0.25	kg/s
Quadcopter inertia about the x^b-axis	I_{xx}	$5 \cdot 10^{-3}$	$\mathrm{kg\ m^2}$
Quadcopter inertia about the y^b-axis	I_{yy}	$5 \cdot 10^{-3}$	$\mathrm{kg\ m^2}$
Quadcopter inertia about the z^b-axis	I_{zz}	$1 \cdot 10^{-2}$	$\mathrm{kg\ m^2}$
Motor constant	c_m	$1 \cdot 10^4$	$\mathrm{v^{-2}s^{-2}}$

Figure 1.1C: Parameters of the quadcopter

From *Figure 1.1B* and *Figure 1.1C* we can simplify Equation 10 for V_z^{dot} even further. We know that the sum of all rotor voltages need to be equal to one another in order for the quadcopter to gain some altitude. Since our design needs all rotor voltages to be equal to one another we can build a relationship as shown below.

$$\sum_{i=1}^{4} U * [\frac{K*Cm}{Mass}] = g \qquad \textit{Equation 1.1A}$$

$$\sum_{i=1}^{4} U = g/[\frac{K*Cm}{Mass}] \qquad \textit{Equation 1.1B}$$

Using the known values for K, Cm, Mass, and Gravity from Figure *1.1C*, we get the sum of all rotor voltages to yield 163.5 V as shown below.

$$\sum_{i=1}^{4} U = 9.81/[\frac{(3E-6)*(1E4)}{0.5}] = 163.5\ V \qquad \textit{Equation 1.1C}$$

Now that we know U_1, U_2, U_3, and U_4 are equal to 163.5V, we can use this value in our linearized equation for V_z^{dot} to simplify it as shown in *Figure 1.1D* below. Conveniently, the right portion of this equation is canceled once we plug the respective values.

```
Vz_dot = (-Kd/Mass)*Vz - g + (K*Cm/Mass)*(U1+U2+U3+U4);

Vz_dot = (-Kd/Mass)*Vz - 9.81 + ((3E-6)*(1E4)/0.5)*(163.5);

Vz_dot = (-Kd/Mass)*Vz - 9.81 + 9.81;

Vz_dot = (-Kd/Mass)*Vz;                                    %Equation 10
```

Figure 1.1D: Linearized Equation 10 for V_z^{dot}

2. System Modeling

2.1. State Space Model

Using our linearized EOM from the first problem, we can efficiently construct the matrices [A], [B], and [C]. Using the provided matrices [x], [u], and [y] we can use the relationship shown in *Figure 2.1A* below. It is assumed that D = 0.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$
$$\mathbf{y} = C\mathbf{x}$$

Figure 2.1A: Matrix equations for solving A, B, C

By numerically solving our linearized EOM and using the provided [12x1] matrix [x] we can construct Matrix A as shown in *Figure 2.1B*. The dimensions of Matrix A are [12x12].

```
Mat_X = transpose([x y z Vx Vy Vz Roll Pitch Yaw Wx Wy Wz])

Mat_A = [0   0   0   1   0   0   0   0   0   0   0   0   % X dot
         0   0   0   0   1   0   0   0   0   0   0   0   % Y dot
         0   0   0   0   0   1   0   0   0   0   0   0   % Z dot
         0   0   0 -0.5   0   0   0   g   0   0   0   0   % Vx dot
         0   0   0   0 -0.5   0  -g   0   0   0   0   0   % Vy dot
         0   0   0   0   0 -0.5   0   0   0   0   0   0   % Vz dot
         0   0   0   0   0   0   0   0   0   1   0   0   % Roll dot
         0   0   0   0   0   0   0   0   0   0   1   0   % Pitch dot
         0   0   0   0   0   0   0   0   0   0   0   1   % Yaw dot
         0   0   0   0   0   0   0   0   0   0   0   0   % Wx dot
         0   0   0   0   0   0   0   0   0   0   0   0   % Wy dot
         0   0   0   0   0   0   0   0   0   0   0   0]  % Wz dot
```

Figure 2.1B: Matrix A

Now we can construct our Matrix B using the provided [4x1] matrix [u] as well as our linearized EOM. This yields as *Figure 2.1C*. The dimensions of Matrix B are [12x4].

```
Mat_U = transpose([U1 U2 U3 U4]);

Mat_B = [ 0     0     0     0          % X dot
          0     0     0     0          % Y dot
          0     0     0     0          % Z dot
          0     0     0     0          % Vx dot
          0     0     0     0          % Vy dot
        .06    .06   .06   .06          % Vz dot
          0     0     0     0          % Roll dot
          0     0     0     0          % Pitch dot
          0     0     0     0          % Yaw dot
        1.5     0   -1.5     0          % Wx dot
          0   1.5     0   -1.5          % Wy dot
        .75  -.75   .75  -.75 ]        % Wz dot
```

Figure 2.1C: Matrix B

Matrix C can be constructed using the two provided [4x1] matrices [y] and [x]. This yields as *Figure 2.1C*. The dimensions of Matrix C are [6x12].

```
Mat_Y = transpose([x y z Roll Pitch Yaw])

Mat_C = [1 0 0 0 0 0 0 0 0 0 0 0          % X dot
         0 1 0 0 0 0 0 0 0 0 0 0          % Y dot
         0 0 1 0 0 0 0 0 0 0 0 0          % Z dot
         0 0 0 0 0 0 1 0 0 0 0 0          % Roll dot
         0 0 0 0 0 0 0 1 0 0 0 0          % Pitch dot
         0 0 0 0 0 0 0 0 1 0 0 0]         % Yaw dot
```

Figure 2.1D: Matrix C

Additionally, we can also solve for matrix [X^{dot}] using our newly found matrix [A] and [B]. The relationship as shown in *Figure 2.1A* can be used here and this yields a [12x1] matrix [X^{dot}] as shown in *Figure 2.1E* below.

```
Mat_Xdot =

                                                    Vx
                                                    Vy
                                                    Vz
                                    (981*Pitch)/100 - Vx/2
                                  - (981*Roll)/100 - Vy/2
(3*U1)/50 + (3*U2)/50 + (3*U3)/50 + (3*U4)/50 - Vz/2
                                                    Wx
                                                    Wy
                                                    Wz
                                       (3*U1)/2 - (3*U3)/2
                                       (3*U2)/2 - (3*U4)/2
                 (3*U1)/4 - (3*U2)/4 + (3*U3)/4 - (3*U4)/4

The size of the matrix Xdot is: 12   1
```

Figure 2.1E: Matrix Xdot

2.2. Transfer Functions

With the help of our matrices [A], [B], and [C] we can now construct our transfer function using the equation: $G_{ij}(s) = C_i(sI - A)^{-1} * B_j + D$. The transfer function would be computed from the inputs of U_1, U_2, U_3, and U_4 to our outputs of x, y, z, roll-φ, pitch-θ, and yaw-ψ. Plugging the transfer function equation in our Matlab script provided that the matrices [A], [B], and [C] are declared, would yield *Figure 2.2A*. The dimension of this matrix was [6x4] which makes sense since we had four inputs and six outputs.

```
Mat_Gij =

[                     0, 2943/(100*s^3*(2*s + 1)),                     0, -2943/(100*s^3*(2*s + 1))]
[-2943/(100*s^3*(2*s + 1)),                     0, 2943/(100*s^3*(2*s + 1)),                     0]
[       3/(25*s*(2*s + 1)),       3/(25*s*(2*s + 1)),       3/(25*s*(2*s + 1)),       3/(25*s*(2*s + 1))]
[               3/(2*s^2),                       0,              -3/(2*s^2),                       0]
[                       0,               3/(2*s^2),                       0,              -3/(2*s^2)]
[               3/(4*s^2),              -3/(4*s^2),               3/(4*s^2),              -3/(4*s^2)]

The size of the matrix Gij is: 6  4
>>
```

Figure 2.2A: Matrix G_{ij}

3. Control Conversion

3.1. Algebraic Conversion

In order to find the transfer function in terms of Z(s)/U(s) we would be using the linearized equations from problem one. We know that all four rotor voltages need to be equal to one another and as such we can conclude the relationship that $Z(s)/U(s) = Z(s)/U_1(s) = Z(s)/U_2(s) = Z(s)/U_3(s) = Z(s)/U_4(s)$. This means that our transfer function would have to be multiplied by a constant four at the end to account for the four rotor voltages.

We would be using the linearized equations of Z, Z^{dot}, V_z^{dot}. The linearized equation of Z^{dot} and V_z^{dot} can be seen in *Figure 1.1B.* We can use these three equations and isolate Z(s) and U(s) such that we would have a transfer function that accounts for Z(s) over all four rotor voltages in relation with U(s). A mathematical setup for this can be seen in *Figure 3.1B* below. We can also confirm that Matlab provides us with the same transfer function given the relationship as shown in *Figure 3.1A*.

```
% Four Rotor Voltages:
U1 = (9.8 / (K*Cm / Mass)) / 4
U2 = U1
U3 = U1
U4 = U1

syms Zs s Vzs Us

% Equation #2
Vzs = s*Zs;

% Equation #3
Vz_dot = -0.5 * Vzs + (K*Cm / Mass) * U1;

% Equation #2 into #3
Us = (1/(4*(K*Cm / Mass))) * (Vz_dot + 0.5*Vzs)
Zs = Us * (4*(K*Cm / Mass) / (s^2 + 0.5*s))

Transfer_func = (expand(Zs / Us))
```

```
Transfer_func =

     0.24
  -----------
  s^2 + 0.5 s

Continuous-time transfer function.

>>
```

Figure 3.1A: Matlab script and output for Z(s)/U(s)

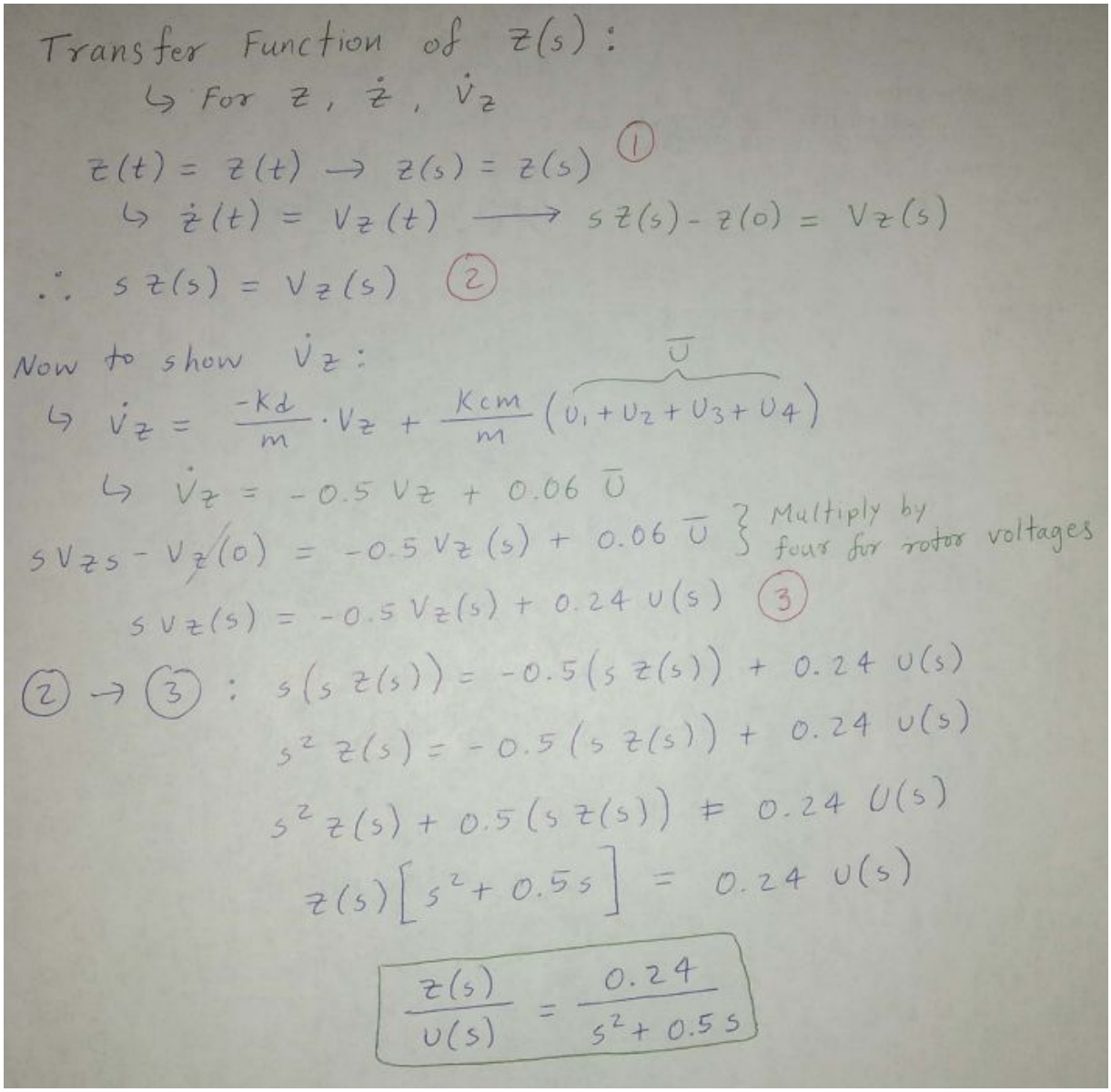

Figure 3.1B: Mathematical setup of Z(s)/U(s)

We know that the input vector is in terms of the squared voltages of the rotors, and therefore our control system should compute U_1, U_2, U_3, U_4 instead of just V_1, V_2, V_3, V_4. We do not want our motors to exceed a voltage that is over 10 volts. The voltage input constraints are shown in *Figure 3.1C* below. We would be solving for our voltage values in the next section when we plot it with time.

$$0 \text{ volt}^2 \leq u_1, u_2, u_3, u_4 \leq 100 \text{ volt}^2$$

Figure 3.1C: Input voltage range

4. PID Control Design and Analysis

An open loop step response of our transfer function can be plotted as shown in *Figure 4A.* The step response of a closed loop transfer function can be seen in *Figure 4B*.

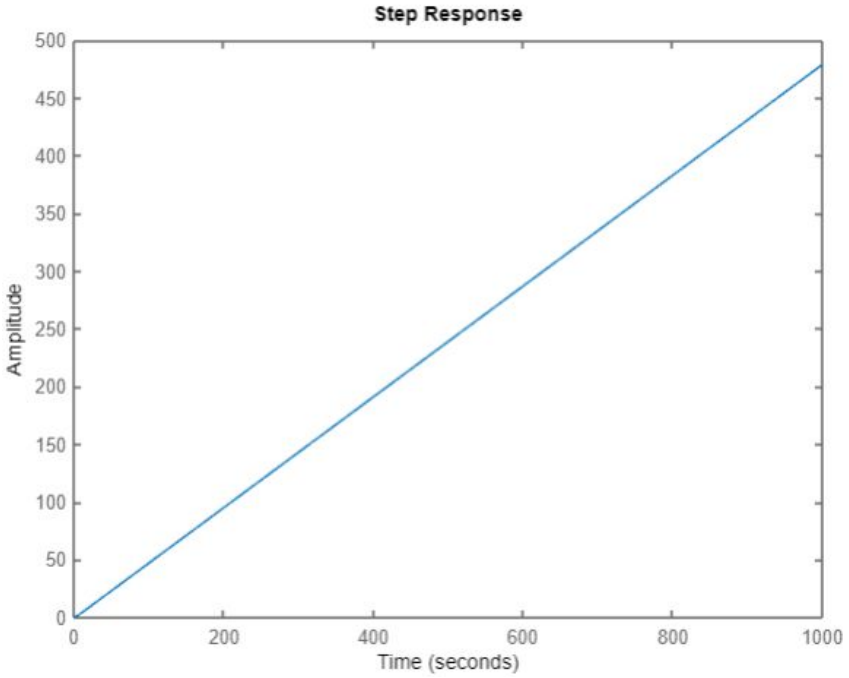

Figure 4A: Open loop step response of transfer function Z(s)/U(s)

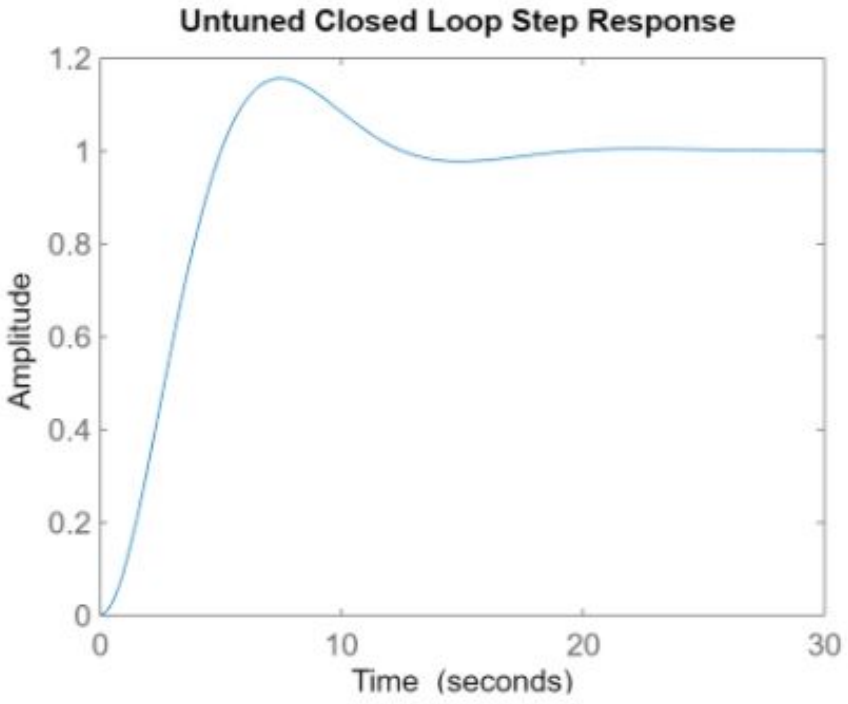

Figure 4B: Closed Loop step response without PID controller

4.1. Closed-loop PID system block diagram

Using our transfer function from the previous section we can design a PID controller in Simulink such that the quadcopter can fly up to an altitude of five feet with an expected deviation of ±0.05 feet. A disturbance input was included in this controller and multiplied by 5. A setup of this controller can be seen in *Figure 4.4A* below.

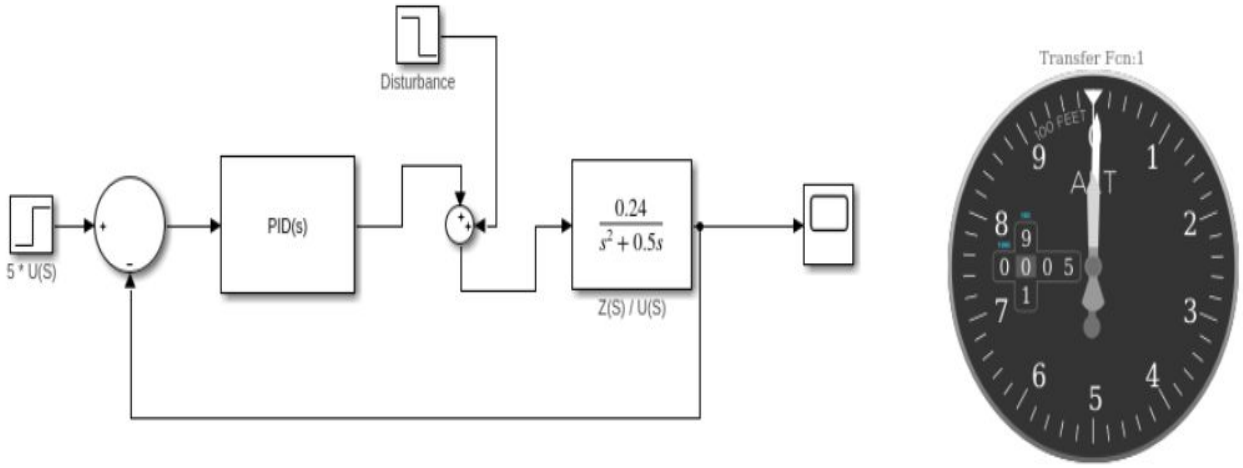

Figure 4.1A: PID controller design in Simulink

4.2. Root Locus

Since we know that all four rotor voltages are the same, we do not need to plot four separate root loci plots as they would all be the same. Instead we can plot the root loci for our transfer function from the previous problem. Shown in *Figure 4.2A* we see the root locus plot for our transfer function Z(s)/U(s).

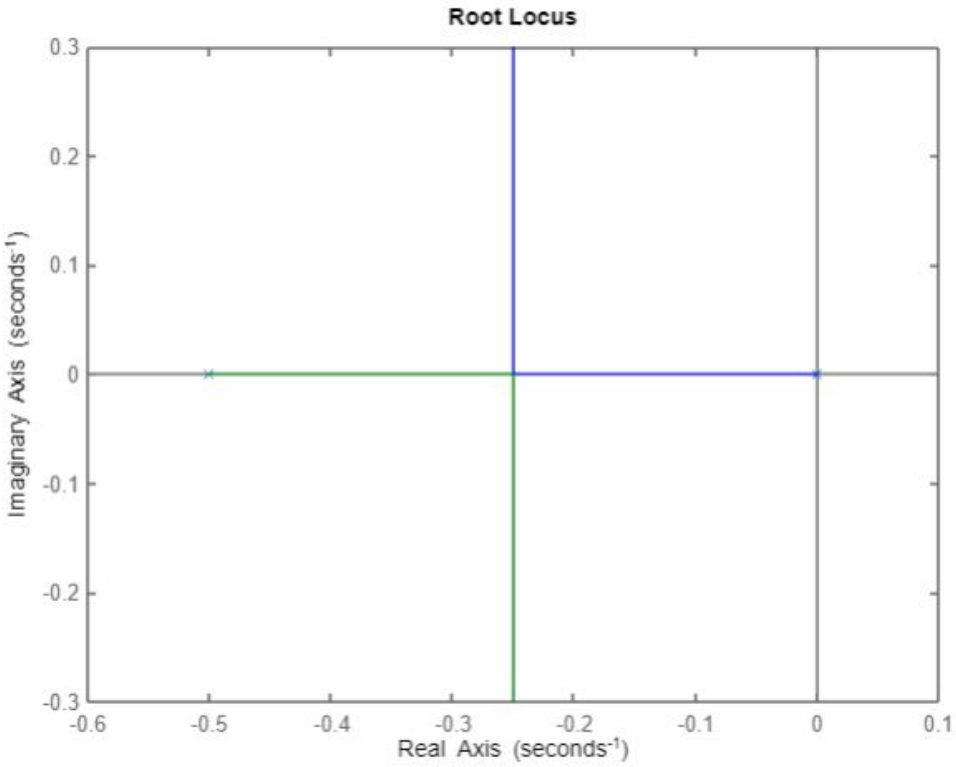

Figure 4.2A:Root Locus plot for Z(s)/U(s)

4.2.1. P controller and stability

A P controller can stabilize a first order system but for higher order gains the P controller is not sufficient to stabilize a system. Our transfer function is not a first order and as such a P controller may not be best suited for stability. From the root locus plot shown in *Figure 4.2A* above, we can see that the poles converge to a singular point on the real axis when gain approaches infinity. The imaginary points lead to infinity.

4.2.2. PI controller and stability

In a PI controller, the I value allows us to control any steady state error that could occur and as such we can reduce steady state errors at the risk of a negative effect on stability. A PI controller also reduces stability by decreasing the damping ratio, while increasing the peak overshoot and settling time. The roots from our root loci plot in the left hand side would also come closer to the imaginary axis. This means that our plant can be stabilized by a PI controller.

4.2.3. PID controller and stability

Our plant can be stabilized by a PID controller as here we can control all three gains of the proportional, integral, and derivative terms. It is important to note that if these gain values are not tuned correctly, this could lead to instability within our plant.

4.3. Bode Plot

The bode plot for our transfer function Z(s)/U(s) can be found from Matlab as shown in *Figure 4.3A* below. From this figure we can also find the gain and phase margins. Matlab tells us that the gain margin is at infinity while the phase margin is at 52.7° (at 0.382 rad/s).

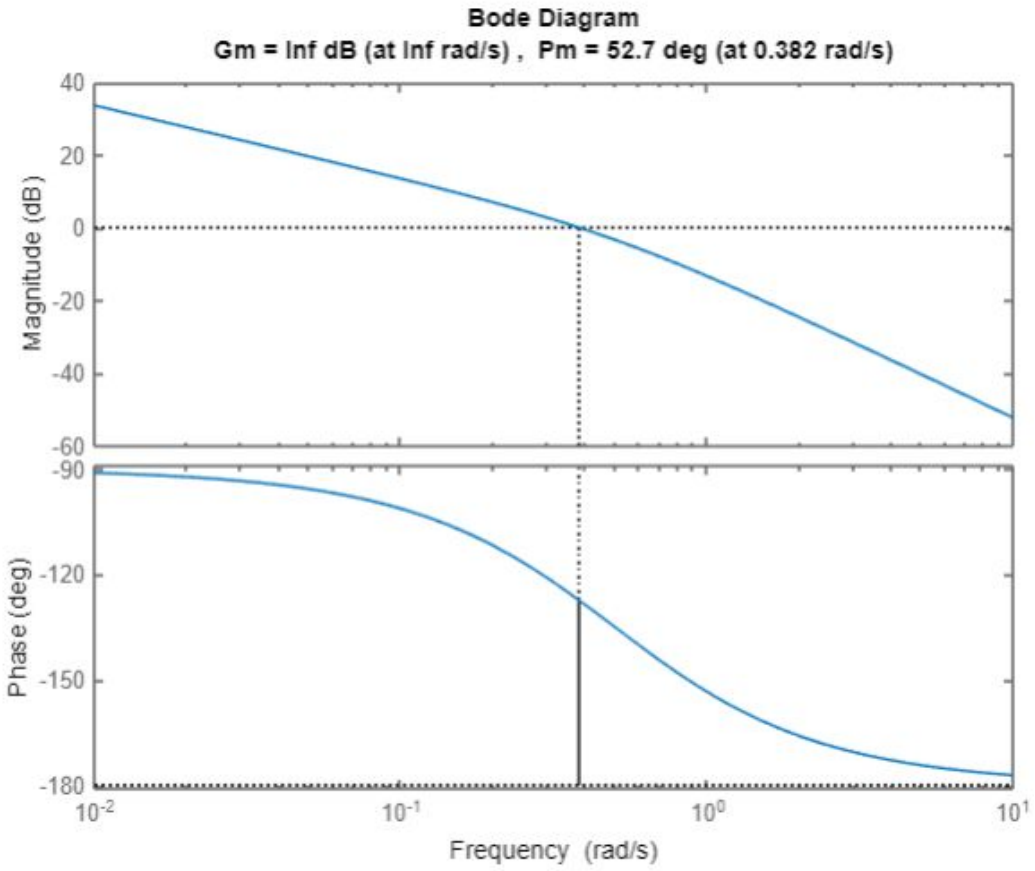

Figure 4.3A: Bode plot for transfer function Z(s)/U(s)

4.4. PID controller for altitude (Z vs Time)

The PID tuned values are shown in *Figure 4.4A* below. The PID tune command automatically sets our P, I, and D values for elevation of five feet. These values can be further tuned so that we can get an optimal graph for altitude versus time. The quadcopter altitude vs

time graph can be seen in *Figure 4.4B* below. The settling time was found to be 2.512 seconds at 5.1 feet. There was a slight overshoot value of under 3% in the beginning which is as expected.

The quadcopter reaches a stabilized 5 feet at around the 30 second mark and settles there for the remainder of time. The settling time for our transfer function (at 98% of steady state value) is around 2.5 seconds as we can see in *Figure 4.4B*. This value has been improved greatly from our initial closed loop step response in *Figure 4B*. The tuned parameters of our PID controller helped us greatly

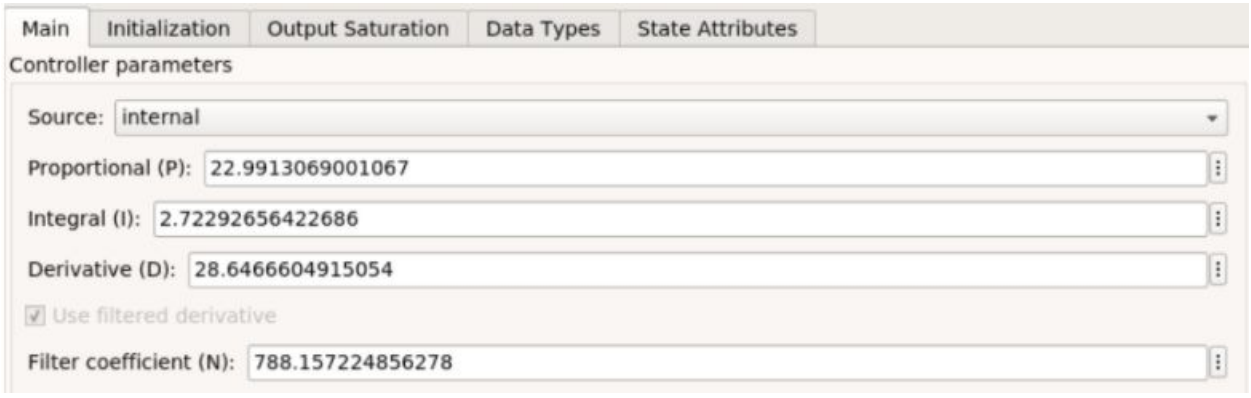

Figure 4.4A: PID controller tuned values

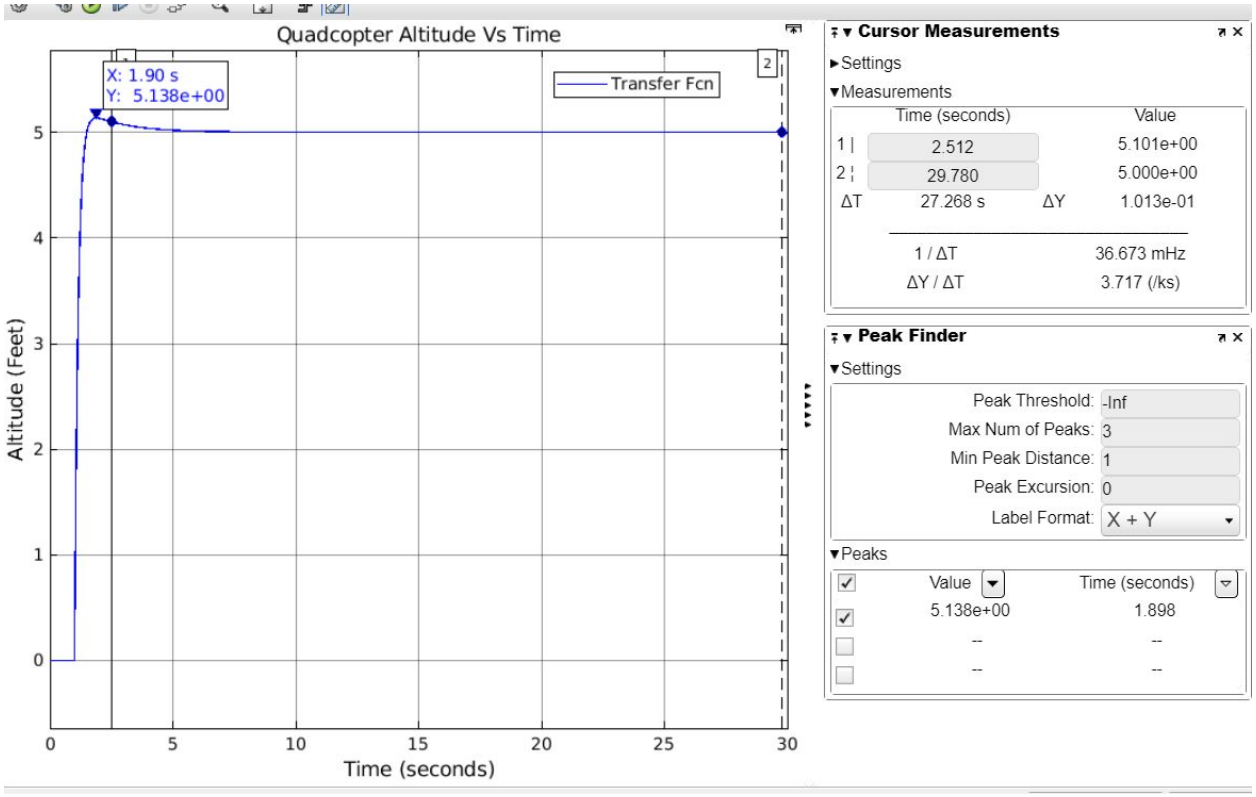

Figure 4.4B: PID controller graph with T_s=2.5sec at 5.1 feet

4.5. Rotor voltage plot (U vs Time)

We know that all rotor voltages were the same as the quadcopter needed equal voltages at all times to fly at a specified altitude. The sum of all four rotor voltages was calculated as shown in *Figure 4.5A* below. This sum was then divided by 4 to account for the same voltage that each rotor would have. This value was then square rooted to get V_1, V_2, V_3, V_4. These values can be seen in the figure below which yield ~6.39V and this is within the provided range as shown in *Figure 3.1C* above. The sum of all four rotor voltages over time (U vs Time) can be seen in *Figure 4.5B* below. The voltages of the four rotors here can be seen rapidly increase towards a sum of 163.5V and then gradually settle to a stabilized value.

```
% Four Rotor Voltages:
U = (9.8 / (K*Cm/Mass));            % U = 163.33 V
U1 = (9.8 / (K*Cm/Mass)) / 4;       % U1 = 40.833 V
U2 = U1
U3 = U1
U4 = U1

V1 = sqrt(U1)                       % V1 = 6.39 V
V2 = sqrt(U2)
V3 = sqrt(U3)
V4 = sqrt(U4)
```

Figure 4.5A: Four rotor voltage values

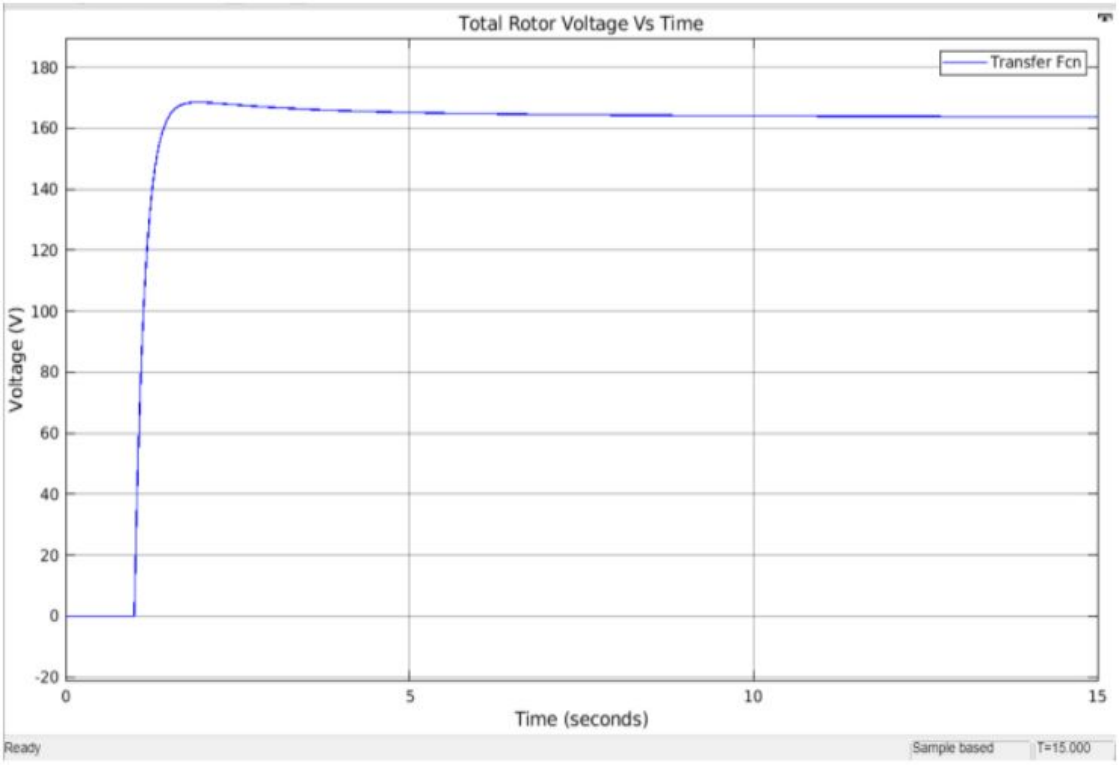

Figure 4.5B: Combined rotor voltage plot (U vs Time)

Conclusion

With the help of tools such as MATLAB and Simulink, we were able to successfully design a quadcopter system that can fly to an altitude of five feet within the provided deviation. Since there were no bounds provided for the settling time or overshoot percentages, our Z vs Time graph was tuned so that these values were reasonable enough. The peak value for our quadcopter is measured to be at 5.138 feet, and within 3 seconds there remains less than 2% deviation from a 5 feet altitude. The quadcopter reaches a stabilized height of 5 feet and thus prove that our system design is valid.

References

1. Bode. (n.d.). Mathworks. https://www.mathworks.com/help/ident/ref/bode.html Getting Started With Simulink. (n.d.). Mathworks.
2. https://www.mathworks.com/videos/getting-started-with-simulink-part-1-building-and-simulating-a-simple-simulink-model-1508442030520.html
3. Nise, N. S. (2019). Control Systems Engineering (8th ed.). Wiley.
4. Jain, Vipin. “Steady State Error & PI Controllers.” Electrical Technology, https://www.electricaltechnology.org/2019/06/steady-state-error-pi-controllers.html#:~:text=PI%20controller%20%28Proportional%20plus%20integral%20controller%29%20reduces%20the,reduces%20the%20stability%20also%2C%20it%20is%20the%20disadvantage.
5. Manisekaran, Nagendran. “Can P controllers be used to stabilize higher order processes?” ResearchGate, 13 March 2014, https://www.researchgate.net/post/Can-P-controller-be-used-to-stabilize-higher-order-process#:~:text=The%20P%20controller%20can%20stabilize%20a%20first%20order,destabilize%20the%20system%20by%20using%20very%20high%20.

Appendix

```
% ECE 3111 Final Project Matlab File
% Collaborators:
%Ajani Blackwood
%Tariq Ali
%David Szymanowski

clc
clear;
close all;
%% Problem 1
%Symbolic variables

% XYZ are frame coordinates
syms x y z;

% Linear Velocities
syms Vx Vy Vz;

% Acceleration V'(x), V'(y), V'(z)
syms Vx_dot Vy_dot Vz_dot;

% Angular Velocities
syms Wx Wy Wz;

% Angular Acceleration
syms Wx_dot Wy_dot Wz_dot;

% Roll Pitch Yaw
syms Roll Pitch Yaw;

% System Rotor Voltages
syms Vr1 Vr2 Vr3 Vr4;

% Constants for mass and other parameters
```

```
Mass = 0.5; %Unit:kg
L = 0.25; %Unit:m %symbol L
K = 3e-6; %symbol K
Kd = 0.25; %Unit Kg/s %symbol Kd
Drag_Coeff = 1e-7; %Unit Nms^2 %symbol b

g = 9.81; %Unit m/s^2 %symbol g
Ixx = 5e-3; %Unit Kgm^2 %Ixx
Iyy = Ixx; %Iyy
Izz = 1e-2; %Unit %Izz
Cm = 1e4; %Unit V^-2 S^-2 %Cm

% U values
% We know U1+U2+U3+U4 = (Vr1^2)+(Vr2^2)+(Vr3^2)+(Vr4^2)
syms U1 U2 U3 U4
% U1 = Vr1^2;
% U2 = Vr2^2;
% U3 = Vr3^2;
% U4 = Vr4^2;

% Linearized EOM
Xdot = Vx; %Equation 5

Ydot = Vy; %Equation 6

Zdot = Vz; %Equation 7

Vx_dot = (-Kd/Mass)*Vx + (g*Pitch); %Equation 8

Vy_dot = (-Kd/Mass)*Vy - (g*Roll); %Equation 9

% Vz_dot = (-Kd/Mass)*Vz - g + (K*Cm/Mass)*(U1+U2+U3+U4);
%
% Vz_dot = (-Kd/Mass)*Vz - 9.81 + ((3E-6)*(1E4)/0.5)*(163.5);
%
% Vz_dot = (-Kd/Mass)*Vz - 9.81 + 9.81;
```

```
Vz_dot = (-Kd/Mass)*Vz; %Equation 10

Roll_dot = Wx; %Equation 11

Pitch_dot = Wy; %Equation 12

Yaw_dot = Wz; %Equation 13

Wx_dot = ((L*K*Cm) / Ixx) * (U1-U3); %Equation 14

Wy_dot = ((L*K*Cm) / Iyy) * (U2-U4); %Equation 15

Wz_dot = ((L*K*Cm) / Izz) * (U1-U2-U3-U4); %Equation 16

%% Problem 2 Part 1
%Matrix pieces
Mat_X = transpose([x y z Vx Vy Vz Roll Pitch Yaw Wx Wy Wz])

Mat_A = [0 0 0 1 0 0 0 0 0 0 0 0 % X dot
0 0 0 0 1 0 0 0 0 0 0 0 % Y dot
0 0 0 0 0 1 0 0 0 0 0 0 % Z dot
0 0 0 -0.5 0 0 0 g 0 0 0 0 % Vx dot
0 0 0 0 -0.5 0 -g 0 0 0 0 0 % Vy dot
0 0 0 0 0 -0.5 0 0 0 0 0 0 % Vz dot
0 0 0 0 0 0 0 0 0 1 0 0 % Roll dot
0 0 0 0 0 0 0 0 0 0 1 0 % Pitch dot
0 0 0 0 0 0 0 0 0 0 0 1 % Yaw dot
0 0 0 0 0 0 0 0 0 0 0 0 % Wx dot
0 0 0 0 0 0 0 0 0 0 0 0 % Wy dot
0 0 0 0 0 0 0 0 0 0 0 0] % Wz dot

fprintf('The size of the matrix A is: %s', num2str(size(Mat_A)))

Mat_U = transpose([U1 U2 U3 U4]);
Mat_B = [ 0 0 0 0 % X dot
```

```
0 0 0 0 % Y dot
0 0 0 0 % Z dot
0 0 0 0 % Vx dot
0 0 0 0 % Vy dot
.06 .06 .06 .06 % Vz dot
0 0 0 0 % Roll dot
          0     0     0     0          % Pitch dot
          0     0     0     0          % Yaw dot
        1.5     0  -1.5     0          % Wx dot
          0   1.5     0  -1.5          % Wy dot
        .75  -.75   .75  -.75 ]        % Wz dot

  fprintf('The size of the matrix B is: %s', num2str(size(Mat_B)))


  Mat_Xdot = (Mat_A*Mat_X) + (Mat_B*Mat_U)
  fprintf('The size of the matrix Xdot is: %s', num2str(size(Mat_Xdot)))

  %Output Matrix y = transpose([x y z φ θ ψ])


  %Mat_Y = transpose([x y z Roll Pitch Yaw])

  Mat_C = [1 0 0 0 0 0 0 0 0 0 0 0          % X dot
          0 1 0 0 0 0 0 0 0 0 0 0           % Y dot
          0 0 1 0 0 0 0 0 0 0 0 0           % Z dot
          0 0 0 0 0 0 1 0 0 0 0 0           % Roll dot
          0 0 0 0 0 0 0 1 0 0 0 0           % Pitch dot
          0 0 0 0 0 0 0 0 1 0 0 0]          % Yaw dot


  fprintf('The size of the matrix C is: %s', num2str(size(Mat_C)))
  % Mat_Y = Mat_C * Mat_X
  %% Problem 2 Part 2


  % Matrix GIJ
  % Gij (s) = Ci(sI -A)-1Bj + D
  syms s


  SI_minusA = s* eye(12) - Mat_A;


  Mat_Gij = (Mat_C *(SI_minusA)^-1 *(Mat_B))
  fprintf('The size of the matrix Gij is: %s', num2str(size(Mat_Gij)))
  %% Problem 3


  % Four Rotor Voltages:
  U = (9.8 / (K*Cm/Mass));            % U = 163.33 V
```

```
U1 = (9.8 / (K*Cm/Mass)) / 4;        % U1 = 40.833 V
U2 = U1
U3 = U1
U4 = U1


V1 = sqrt(U1)                        % V1 = 6.39 V
V2 = sqrt(U2)
V3 = sqrt(U3)
V4 = sqrt(U4)


syms Zs s Vzs Us


% Equation #2
Vzs = s*Zs;


% Equation #3
Vz_dot = -0.5 * Vzs + (K*Cm / Mass) * U1;


% Equation #2 into #3
Us = (1/(4*(K*Cm / Mass))) * (Vz_dot + 0.5*Vzs)
Zs = Us * (4*(K*Cm / Mass) / (s^2 + 0.5*s))


Transfer_func = (expand(Zs / Us))


% Transfer function to plot


TF = tf([0.24], [1 0.5 0])


%% Problem 4 PID controller
figure(1)
margin(TF)


figure(2)
rlocus(TF)


figure(3)
step(TF)
```