Systems Analysis Project

ECE 3111

Spring 2015

Zac Sutton, David Paquette, Andrew Budd, Dylan Hammerman

Contents

Objective

Control position and attitude of a quadcopter using the rotor's angular velocities, and design a PID controller to stabilize the vehicle at a specified altitude.

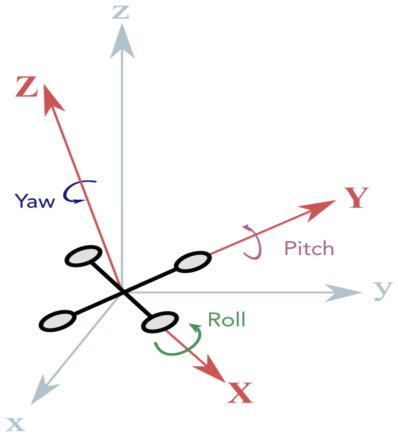
Background

Hovering is achieved when the total thrust generated by the four rotors is equal to the force of gravity, the quadcopter will not spin in place when the torque of the two clockwise spinning motors is equal and opposite to the torque of the counter-clockwise spinning motors.

The first part of designing a control system is to model the quadcopter mathematically using nonlinear equations of motion. These equations of motion are based upon the known system parameters. By linearizing these equations we were able to derive transfer functions and state space models of the system. In order to correctly model the response of the quadcopter, two reference frames must be modeled. One is the inertial frame (in reference to the ground) and the other is the body frame which is in reference to the quadcopter. The body reference frame will be parallel to the inertial frame when hovering, but must be separately modeled for roll and pitch to be properly represented.
For the end of this project, the motor driver system is assumed, and did not play a part in the formation of the transfer functions. This motor control system controls the quadcopter motors based on the inputs of altitude, roll, pitch and yaw.

Linearization

A simple diagram of a quadcopter in its two reference frames is shown below. The grey axes define the inertial reference frame, and the red axes define the body reference frame.



The state of a quadcopter can be described with 6 values: the x, y, and z location of the center of the device in the inertial frame, and the roll, pitch, and yaw angles. Roll corresponds to rotation about the x body axis and is expressed as the angle of the y body axis with the y inertial axis as a reference. Similarly, pitch corresponds to rotation about the y body axis and is expressed as the angle of the z body axis with the z inertial axis as reference. And finally, yaw corresponds to rotation about the z body axis and is expressed as the angle of the x body axis with the x inertial axis as reference. In the model we refer to roll, pitch, and yaw as φ, ϑ, and ψ respectively. Our state-space model includes 6 other states as well: the velocities in the x, y, and z inertial directions, and the angular velocities about the x, y, and z body axes referred to as ω_x, ω_y, and ω_z respectively. Our inputs for the state-space model correspond to the voltage inputs to each of the 4 motors. The inputs have the form $u_i=v_i^2$ where v_i is the voltage input to the i'th motor. For our application, the state equations need to be linear. So the set of nonlinear equations that describe the total dynamics of the quadcopter were linearized around a chosen operating point. It turns out that the simplest operating point to linearize around is all state variables and inputs equal to 0. This is a reasonable choice for an operating point since it corresponds to the quadcopter sitting in a level position. The resulting linear equations are:

(1) $$x' = v_x$$

(2) $$y' = v_y$$

(3) $$z' = v_z$$

(4) $$v_x' = \frac{-k_d}{m} v_x$$

(5) $$v_y' = \frac{-k_d}{m} v_y$$

(6) $$v_z' = \frac{-k_d}{m} v_z + \frac{-k \cdot c_m}{m} (u_1 + u_2 + u_3 + u_4) - g$$

(7) $$\phi' = \omega_x$$

(8) $$\theta' = \omega_y$$

(9) $$\psi' = \omega z$$

(10) $$\omega_x' = \frac{k \cdot L \cdot c_m}{I_{xx}} (u_1 - u_3)$$

(11) $$\omega_y' = \frac{k \cdot L \cdot c_m}{I_{xx}} (u_2 - u_4)$$

(12) $$\omega_z' = \frac{b \cdot L \cdot c_m}{I_{xx}} (u_1 - u_2 + u_3 - u_4)$$

Once the system was linearized, a few things became apparent. First of all, from equations (4) and (5) above, we can see that acceleration in the x and y directions depends only on the velocities in the x and y directions. These terms are simply due to air resistance. The lack of any other state variable or input in these equations suggests that our model will be uncontrollable in the x and y directions.

Another observation from equation (6) is that there is a constant term g in the model due to gravity. Since a constant can't be included in a state space model, the term was ignored under the assumption that when the system is implemented, feedback would correct for the acceleration of gravity.

As an interesting side note, we found that if the operating point for linearization was changed slightly, our model changed significantly. If we find the value of our u input, call it u_0, such that u_0 applied to all 4 motors results in upward acceleration equal to g, we can linearize around $u_i = u_0 \; ; \; i = 1,2,3,4$. If we redefine the input to our system as $r_i = u_i - u_0$, then an input of $r_{1,2,3,4} = 0$ results in

hover. When the system is linearized around the new input equal to 0, equations (4), (5) and (6) become respectively:

$$v_x' = \frac{-k_d}{m} v_x + g_\theta$$

$$v_y' = \frac{-k_d}{m} v_y + g_\phi$$

$$v_z' = \frac{-k_d}{m} v_z + \frac{-k \cdot c_m}{m} (r_1 + r_2 + r_3 + r_4)$$

The constant g is multiplied by state variables so can be included in the model. Theoretically, x and y would be controllable in this model. No further simulation or testing was done on this model so it's limitations when it comes to implementation are not known.

System Modeling

The 12 linear equations above were set up in a state_space model with the state vector x containing the 12 state variables. The **A** matrix contains any terms from the state equations that involve state variables. The **B** matrix contains terms that involve the 4 X 1 input u vector. The 6 state variables x, y, z, φ, ϑ and ψ are the meaningful output of the system since those variables that can be measured in our particular implementation. The state matrix **C** is such that the output vector $y = [x\ y\ z\ \varphi\ \vartheta\ \psi]$. The dimensions of the state matrices are:

$$dim(\boldsymbol{A}) = 12\ X\ 12$$
$$dim(\boldsymbol{B}) = 12\ X\ 4$$
$$dim(\boldsymbol{C}) = 1\ X\ 12$$

To design a PID controller, the system needs to be modeled in the s domain. The formula for conversion from state_space to transfer function was implemented in MATLAB. Since the state space model has 4 inputs and 6 outputs, a total conversion will result in 24 transfer functions; one relating each output to each input. Since x and y are not controllable, the 8 transfer functions involving these variables are 0. The remaining 16 transfer functions for state variables z, φ, ϑ and ψ are calculated with MATLAB as follows. The indexing in

the code was chosen so that the resulting matrix was in a useful form for the next step.

```
for j = 1:4
     t = ((s*eye(12)-A)^-1)*B(:,j);
     transferFunctions(1, j) = sym(t(3));
     transferFunctions(2, j) = sym(t(9));
     transferFunctions(3, j) = sym(t(8));
     transferFunctions(4, j) = sym(t(7));
      %transfers(i, j) =  sym(t);
end
```

MATLAB Sample 1. Computes the four diagonalized transfer functions from our linearized state space model.

Control Conversion

The 4 X 4 matrix, **T**, containing transfer functions can be used in the relation between input and output as:

$$\boldsymbol{Y}(s) = \boldsymbol{T}^{*}\boldsymbol{U}(s)$$

where $\boldsymbol{Y}$ is the Laplace transform of output vector $y = [z\ \varphi\ \vartheta\ \psi]$ and $\boldsymbol{U}$ is the Laplace transform of the input vector $\boldsymbol{u} = [u_1\ u_2\ u_3\ u_4]$. We were given a transformation matrix, $\boldsymbol{C}$, that relates a higher level input command vector, $\boldsymbol{u'} = [u_z\ u_\psi\ u_\vartheta\ u_\varphi]$ to the motor input vector as:

$$\boldsymbol{u'} = \boldsymbol{C}^{*}c_m^{*}\boldsymbol{u} \Rightarrow \boldsymbol{u} = \boldsymbol{C}^{-1*}1/c_m^{*}\boldsymbol{u'}$$

To obtain transfer functions for the output versus the new inputs we make the substitution:

$$\mathbf{Y}(\mathrm{s}) = \mathbf{T}^{*}\mathbf{C}^{-1*}1/\mathrm{c_m}^{*}\mathbf{U'}(\mathrm{s})$$

```
motorTransformations = [-.5 .5 .5 .5
             .5 -.5 0 0;
             1 0 -.5 -.5;
             0 0 .5 -.5];
mappedTransferFunctions = transferFunctions*motorTransformations^-1 ;
mappedTransferFunctions = (1/cm*mappedTransferFunctions);
```

MATLAB Sample 2. Maps motor speeds to angle and altitude control commands.

The 4 X 4 matrix $\boldsymbol{T}^{*}\boldsymbol{C}^{-1*}1/c_m$ contains the transfer functions. To further simplify the problem, we considered only the transfer function of the ith output to the ith input for $i = 1,2,3,4$. These are the expressions on the diagonal of the matrix.

$$\frac{Z}{U_z} = \frac{3}{125000s^2 + 62500s} \tag{13}$$

$$\frac{\Psi}{U\Psi} = \frac{1}{200000s^2} \tag{14}$$

$$\frac{\Theta}{U_\Theta} = \frac{3}{200000s^2} \tag{15}$$

$$\frac{\Phi}{U_\Phi} = \frac{-3}{200000s^2} \tag{16}$$

These 4 functions are treated as the plant transfer function for 4 separate unity feedback systems.

PID Control Design and Analysis

Can a P controller stabilize the system?

Without any proportional gain our closed loop system response is marginally stable due to one or more poles being located on the *jw* axis for each system, which is shown in the root locus plots in figure 10. Because each of our transfer functions are of type 1 or type 2, a proportional gain controller is enough to stabilize the system. For *Z* translation, the root locus shows that as the proportional gain increases to infinity the poles move together to some point on the real axis while the imaginary component moves to infinity, this shows that the closed loop system is stable with only a proportional controller. For each of the angles root locus plots we can see that the system remains only marginally stable even as the proportional gain is increased to infinity. Because the proportional controller only allows for marginal stability either an I or D term may need to be added. The proportional controller will also not account for any steady state error that may arise during stabilization.

Can a PI controller stabilize the system?

Because a P controller can stabilize the system, a PI controller can also stabilize the system. Adding the I term allows the system to handle any steady

state error that may occur. It would also allow the system to operate at some specified damping ratio by adding a pole to the system along the damping angle, if required.

Can a PID controller stabilize the system?

Adding the D term to the PI controller allows us to control the transient response of the system. By modifying the K_d term we can adjust the speed at which the system reaches its set point. As we increase K_d, the overshoot of the system reduces. In this application it is desirable to keep the vehicle from overshooting its set point and to reduce any unnecessary oscillations. By using the MATLAB PID tuning toolbox we noticed that altering the K_d gain had little effect on stability of the system, so we selected a response that seemed to fit our desired transient response.

PID Design

Figure 5 shows the Z translation response with a step input of 5 ft (1.524 m). As shown, the system overshoots by about 0.1 m and takes about 13 seconds to settle. Our goal while tuning the PID gains was to keep the transient response fairly calm, so we lowered our derivative gain for motion in the Z direction. Lowering the Kd term increased the stabilization time giving us the desired slower response. We were also looking to decrease the amount of overshoot in our Z direction system response. To accomplish this we decreased our integral gain, the result can be seen in figure 5. After the system stabilized in the Z direction we applied a disturbance of magnitude 1.5 at 15 seconds. This implies that the vehicle should stabilize and hover at 5 feet. All the gains for each PID controlled feedback system can be seen in table 1. As shown in figure 5, the simulated feedback system was able to compensate for the disturbance, showing a negligible response in the output. Figure 6, 7 and 8 show the response of our yaw, pitch and roll feedback system with a set point of 0 radians. During the first 15 seconds, the output of each angular response is shown to be stable at 0 radians. At 15 seconds, we introduced a disturbance of 1.5 radians to each of the angular systems. The simulated disturbance caused a max error of about 0.00017 radians above the set point in the yaw response, as time increased the feedback system continually corrected the error pushing the response back to zero. The two other angular feedback systems had a very similar response to a disturbance of the same magnitude and the results can be seen in figures 7 and 8. To tune our gains we

used the PID tuning toolbox. This tool allowed us to simulate and view the updated response against the current response.

The open loop frequency domain transfer functions for each of the controllable outputs can be seen in equations 13 through 16. These plots show that each open loop system response is, at least, marginally stable. In each bode plot, the magnitude is less than 0 db when the phase plot crosses the 180 $^\circ$. For each system, the phase and gain margins are infinite, this implies that each open loop transfer function is either stable or marginally stable.

	$\mathbf{K_p}$	$\mathbf{K_i}$	$\mathbf{K_d}$
Z-Translation	7260.07	0.00269	0.94081
Yaw	6018.33	0.006127	34.1758
Pitch	441.06	0.00757	15.374
Roll	-200.61	-1.22918	-6856.06

Table 1. Numerical gains for each PID controller, tuned using MATLAB.

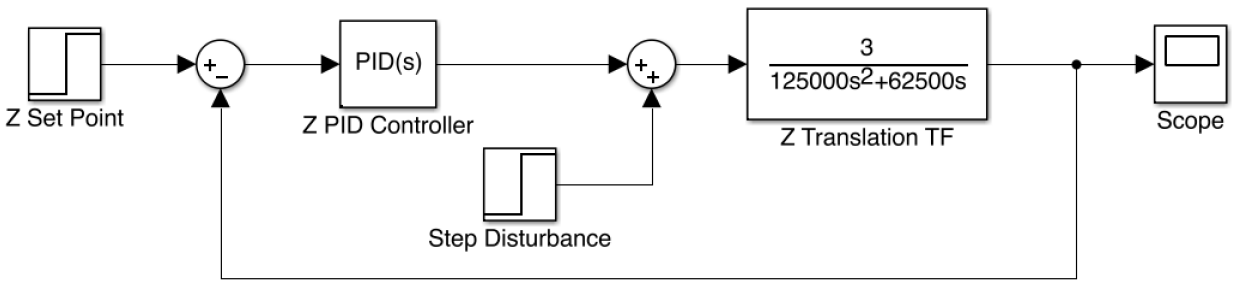


FIG 1. Z-translation block diagram.

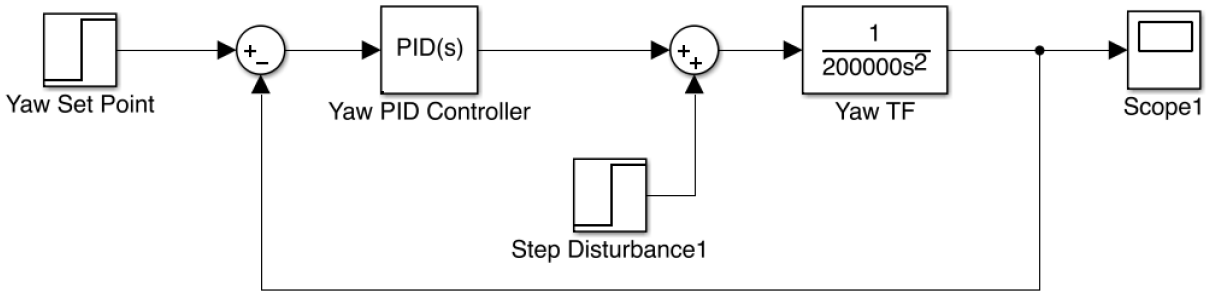


FIG 2. Yaw block diagram

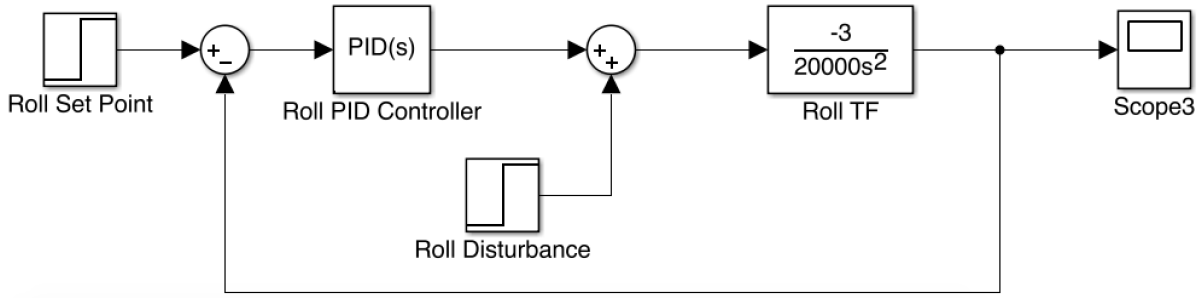


FIG 3. Roll block diagram

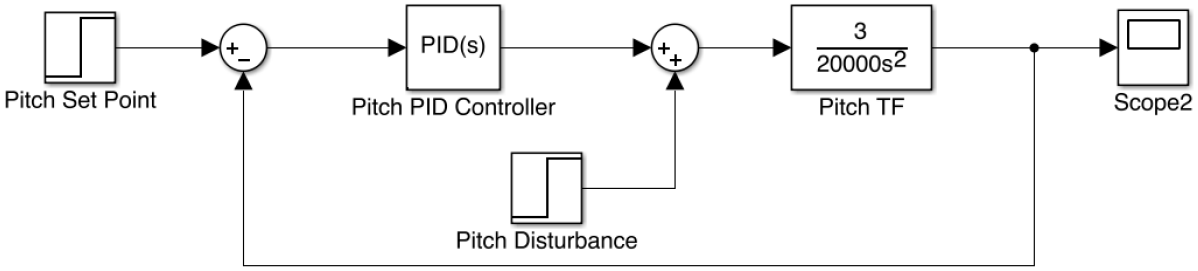


FIG 4. Pitch block diagram

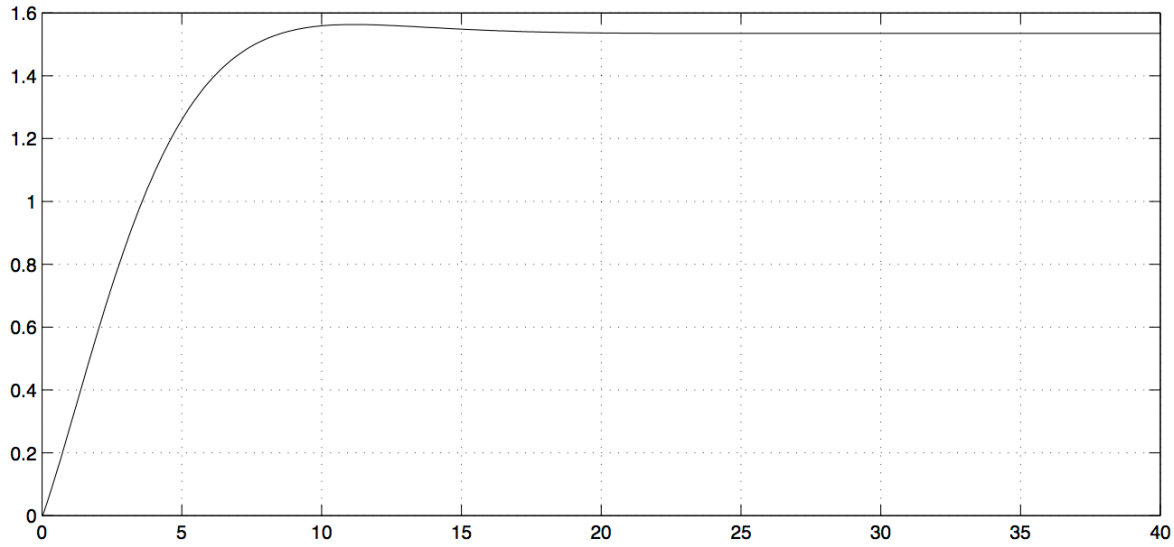


FIG 5. Z response to unit step input with disturbance at 15 s, set point at 1.526 meters.

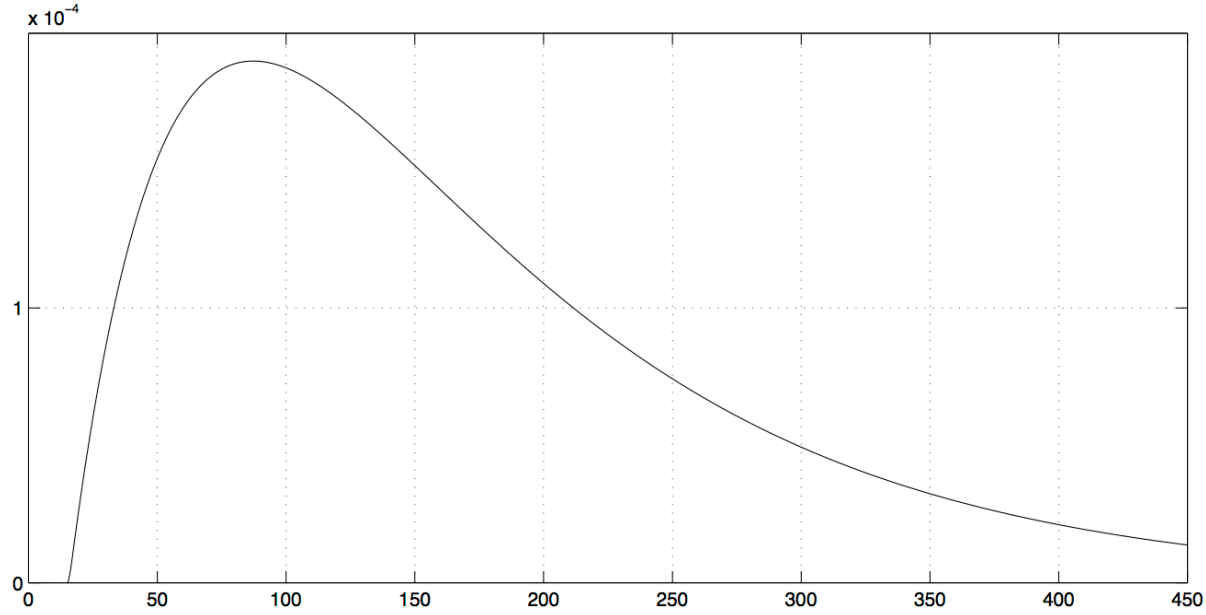


FIG 6. Yaw response to unit step input with disturbance at 15 s, set point at 0 degrees

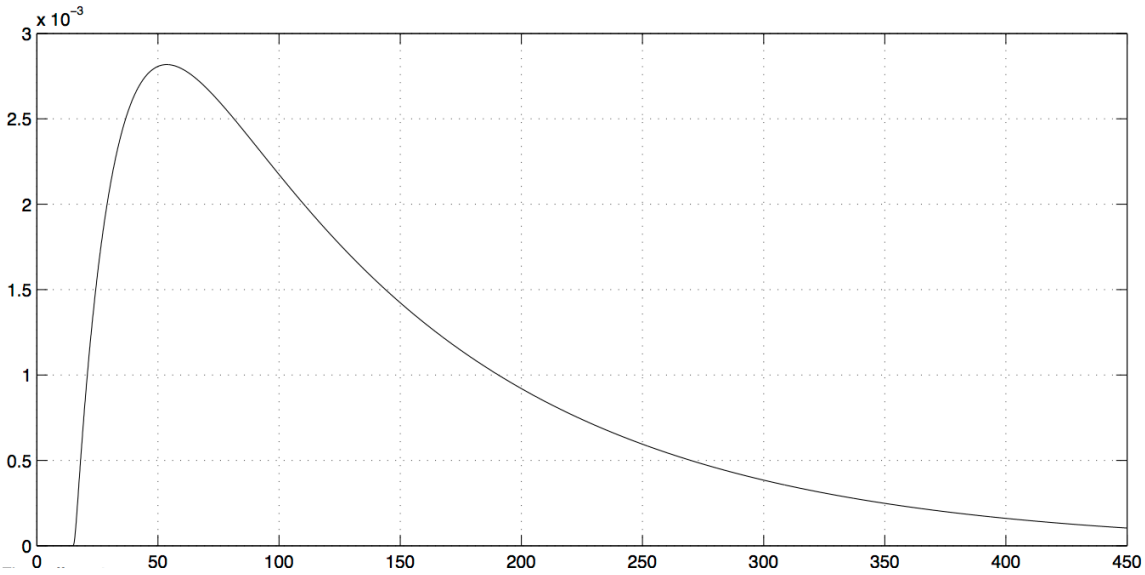


FIG 7. Pitch response to unit step input with with disturbance at 15 s, set point at 0 degrees

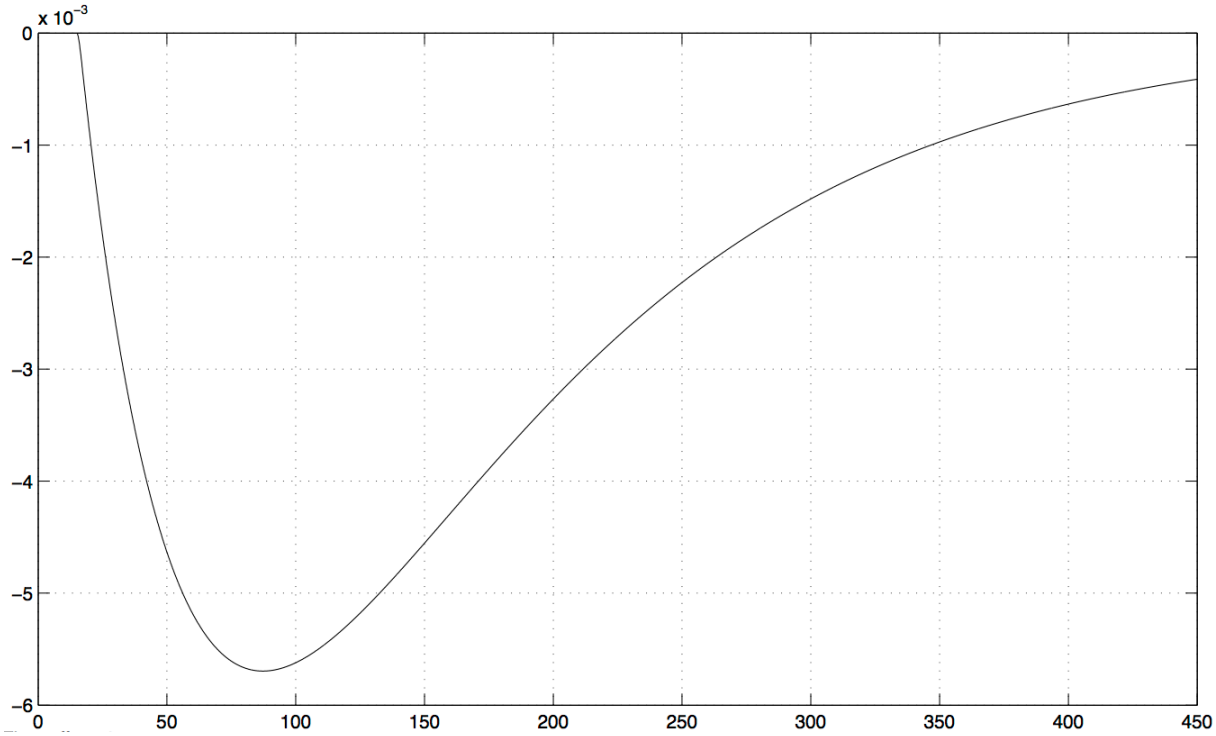


FIG 8. Roll response to unit step input with with disturbance at 15 s, set point at 0 degrees

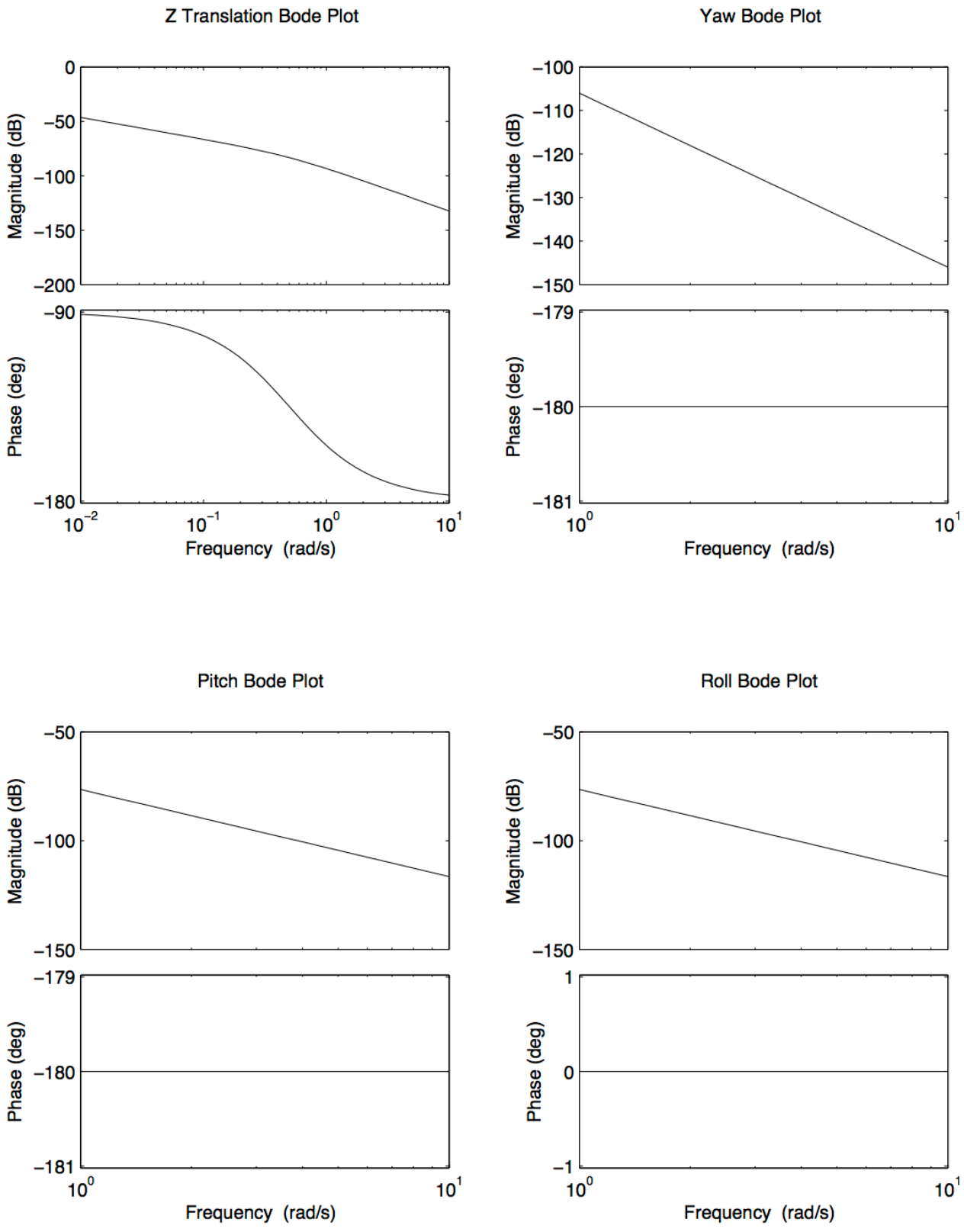


FIG 9. Bode plots

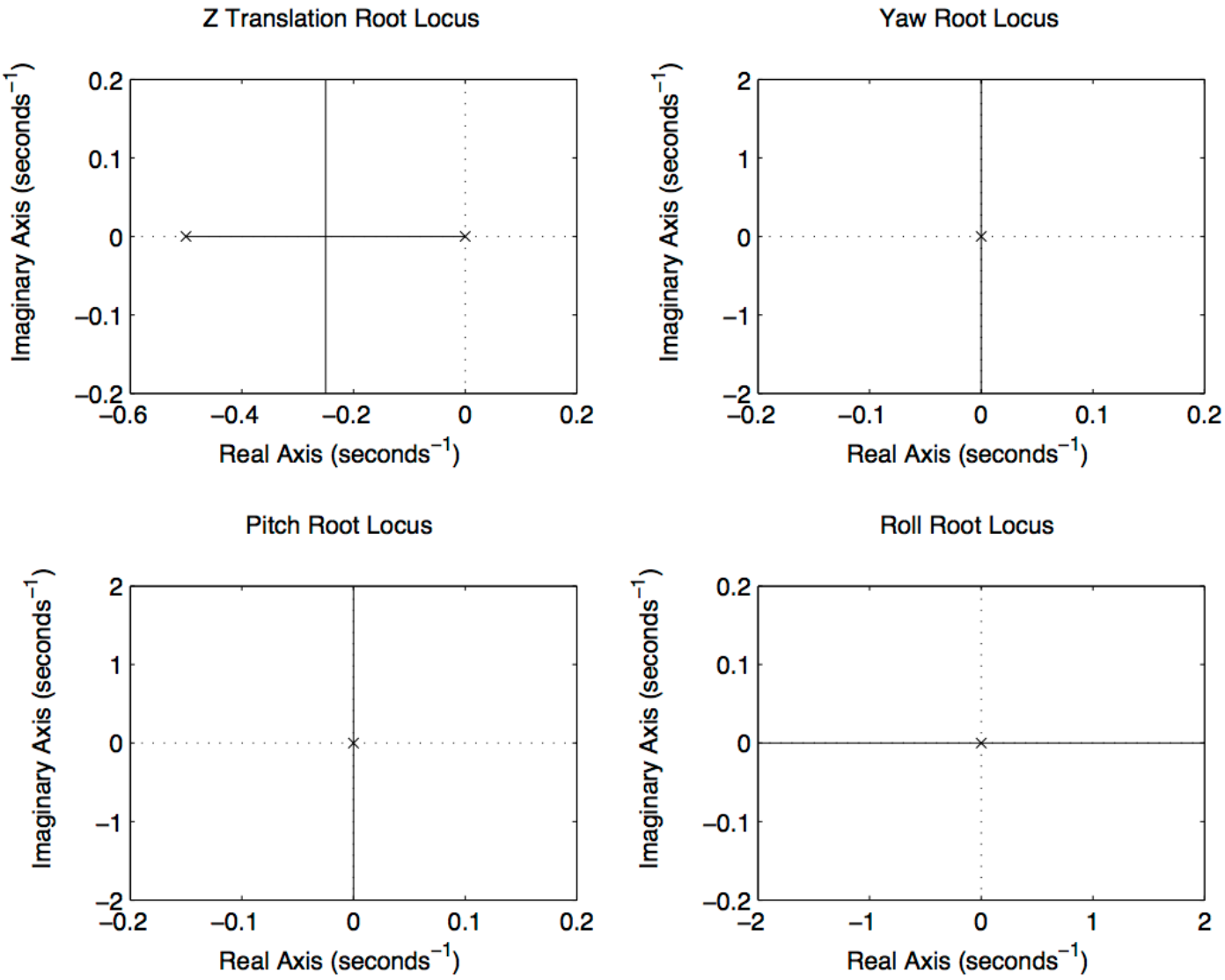


FIG 10. Root locus plots