# Multi-Variable Control Systems Project Update 1

Alek Krupka

February 2026

## 1 Introduction

This project update will mainly be focused on discussing the project description along with a modification that will allow the project to be used for a SISO controller design. Additionally, a SISO controller is proposed to control this modified system.

## 2 Project Motivation

The goal for the semester-long project is to develop a control system for a 2-propeller drone. This is a simplification made to the standard four propeller drone which reduces the degrees of freedom experienced by the input. While this simplication confines the drone to a single plane (the plane formed by the z-axis and the axis along which the drone arms form), it still allows for the design of a multi-variable control system. This project was selected to gain experience with controlling a slow-reacting and non-linear system. The details of the system's non-linearity will be discussed later in 3.1. Even with only having two propellers, the system still maintains these properties, meaning the simplification to a two-propeller drone does not negate this motivation.

## 3 Project Definition

### 3.1 System Description

To begin, we define a drone with just two propellers. This allows us to ignore pitch (rotation around the x-axis) and yaw (rotation around the z-axis). However, this restricts the drone to just rotating around its y-axis and moving along the x and z-axis. For this project, that will be sufficient though. Below, we derive the equations for the output based on the input (speed of propeller for each drone).

We treat this drone as a rigid body with two motors along the x-axis with distance $r$ away from the center of mass. Each motor $i$ produces a linear acceleration in the $z$ direction from the drone's perspective. This thrust force is equivalent to the following.

$$F_{thrust} = k_f \omega^2 \tag{1}$$

where $k_f$ is the thrust coefficient for all propellers.

Analyzing the impact on the body frame, we see that the acceleration from the body frame is the following.

$$a_{body} = \begin{bmatrix} 0 \\ 0 \\ \frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} \end{bmatrix} \tag{2}$$

Now we must calculate the torque applied to the drone. To calculate the angular acceleration due to the thrusts from each propeller, we need to calculate the moment of inertia for the entire drone. For this section, we will solve this symbolically and will list its assumed value in section 3.2. For roll, we say that the acceleration around the y-axis is the following.

$$\alpha_y = \frac{l * k_f(\omega_1^2 - \omega_2^2)}{I} \tag{3}$$

To get the real-world directional accelerations though, we need to translate the body-frame accelerations into the real-world coordinate system. This is done using the rotation matrix listed below,

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{4}$$

Multiplying this rotation matrix by the directional accelerations produces the real-world acceleration value. This gives us the acceleration space definition shown below.

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R_y(\theta) a_{z,body} \tag{5}$$

$$= \begin{bmatrix} \sin(\theta) \\ 0 \\ \cos(\theta) \end{bmatrix} \frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} \tag{6}$$

Afterwards, gravity accounts for a downward acceleration of $-9.81m/s^2$ giving us the final real-world frame definition of the accelerations.

$$a_x = \sin(\theta)\frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} \tag{7}$$

$$a_z = \cos(\theta)\frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} - 9.81 \tag{8}$$

Using these equations, we are able to generate a space with six states. These states are the directional positions along with rotation and each of their first derivatives. The equations for each state are below.

$$\dot{x} = v_x \tag{9}$$

$$\dot{v_x} = a_x = \sin(\theta)\frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} \tag{10}$$

$$\dot{z} = v_z \tag{11}$$

$$\dot{v_z} = a_z = \cos(\theta)\frac{\sum_{k=0}^{1}(k_f \omega_i^2)}{m} - 9.81 \tag{12}$$

$$\dot{\theta} = \omega_y \tag{13}$$

$$\dot{\omega_y} = \alpha_y = \frac{l * k_f(\omega_1^2 - \omega_2^2)}{I} \tag{14}$$

However, this system is nonlinear due to the acceleration being proportional to the squared motor speed. This linearization will be covered later due to being able to linearize differently depending on the operating point selected at the time. The final state space will look something like this though.

Start by defining the state space $x$ as follows,

$$x = \left[x, v_x, z, v_z, \theta, \omega_y\right]^T \tag{15}$$

$$\tag{16}$$

Thus, our state space will look something like the following.

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \Phi_1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Phi_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ \rho_1 & \rho_2 \\ 0 & 0 \\ \rho_3 & \rho_4 \\ 0 & 0 \\ \rho_5 & \rho_6 \end{bmatrix} \begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \tag{17}$$

where $\rho_k$ and $\Phi_j$ are linearization constants.
These variables are filled in once an operating point has been determined.

### 3.2 Model Uncertainty

The main uncertainty in the model comes mainly from the moment of inertia along with the value of the thurst coefficient $k_f$. There is additional uncertainty with the mass of the drone. While the mass can be measured, there will be some error and the mass will not always be the same

For the thrust coefficient a reasonable value we will be assuming is

$$k_f = 10^{-5} \pm 5 \tag{18}$$

For this project, we will be assuming a mass of

$$0.5 \pm 0.1kg \tag{19}$$

We will assume the moment of inertia for this two-propeller drone is a rod. This is because the two-propeller drone is just a rod with motors on the end While there is slight weighting issues we will account for this in the model uncertainty. The moment of intertia for the rod is

$$I_{rod} = \frac{4}{3}ml^2 \tag{20}$$

For this project, assume each motor is roughly $10^{-2}m$ away from the center. Using these measurements, we get a moment of

$$I_{rod} \approx \frac{2}{3} \times 10^{-4} \tag{21}$$

This moment is assumed to have an error of around $10^{-4}$ when moment error and previously defined mass errors are taken into consideration.

### 3.3 Exogenous Input

Characteristics of exogenous input to the system include wind and measurement noise from the GPS.

For the GPS, real-world sensors typically do not measure to more than 1.5m accuracy. In this project, we will assume that the noise coming from every measurement is uniformly distributed and that all RVs are i.i.d.

Wind noise in the system can be simplified to cause acceleration of the drone in a random direction. While in the real-world wind patterns can be predicted and are not random, for this project we will assume that wind pushes the drone in a random direction every sample. The acceleration in each direction is normally and identically distributed with a variance of $1m/s^2$.
For the final SISO system, the voltage to motor speed will not be known and a separate control loop will need to be used for keeping the motor at the correct speed.

### 3.4 Control Objective

‘

The objective for developed controller will be to have the drone go from an initial position $p_i$ to a final position $p_f$ while keeping the following constraints in mind.

- Overshoot: The controllers should not overshoot the magnitude of the final position by greater than 2% of the destination. For example, if going to a final position of $(x = 1000, z = 1000)$ going from point $(x = 0, z = 0)$, let $X_t$ be the amount it goes past 1000 on the $x - axis$ and $Z_t$ be the amount it goes past 1000 on the z-axis at time $t$. In this case, at no time $t$ should $\sqrt{X_t^2 + Z_t^2}$ be greater than $\sqrt{1000^2}$.
- Steady state error: The steady state error for the $z$ position should be no greater than 0.1% of the z-setpoint. Meanwhile, the steady state error for the $x$ position should be at most $2m$ from the target.
- Steady State Noise: The noise should be at most 0.25% on the z-axis and at most $4m$ from the target on the x-axis.
- Phase Margin: Any controller for this system should have at least $45°$ of phase margin.
- Max rotation: The drone should not rotate by more than $30°$ in either direction at any point in it's flight.
- Max velocity: $||v_z + v_x||$ will be limited to $9m/s \approx 20mph$ in the final design.

These are all control objectives for the final design of the final controller.

## 4 SISO Subsystem Description for Project

The Single-Input-Single-Output modification can be made to the drone. Assume that both propellers spin at exactly the same speed. Now, the only output is the z-position measurement and the only input is the desired propeller speed. With these restrictions, the drone can be treated as a SISO control system which can be modeled. In this report, a SISO controller will be designed for this subsystem.

### 4.1 SISO Simplification and Description

For this subsystem, we will use the thrust equations derived earlier. Since we do not need to worry about the drone's rotation, we can simplify our state space in the following way. We can base this off of the following equations.

$$a_z = \frac{2k_f\omega_i^2}{m} - 9.81 \tag{22}$$

Again, we are assuming the value of $k_f$ is $10^{-5}$ and $m = 0.5kg$. This makes the acceleration of the drone the following.

$$a_z = 4 \times 10^{-5}\omega_i^2 - 9.81 \tag{23}$$

As we can see, the equation we arrive at is nonlinear. This non-linearity will pose a problem for any controller we apply to this system as concepts such as bode plots and state spaces depend on the principle of linearity. Thus, we must linearize the system around an operating point. Due to the constant acceleration caused by gravity, we will want to linearize around an equilibrium which is the propeller speed for which the drone experiences 0 acceleration. We find this propeller speed through the following calculations,

$$\sqrt{\frac{9.81}{4 \times 10^{-5}}} \approx 495rad/s \tag{24}$$

Linearizing around this operating point we arrive at the linearized equation.

$$a_z = \frac{d}{dt}a_z|_{\omega=495}\omega + (4 \times 10^{-5}495^2 - 9.81) \tag{25}$$
$$= 0.0792\omega + 9.801 - 9.81 \tag{26}$$
$$\approx 0.08\omega \tag{27}$$

For the SISO control design, we will assume no noise to start. This allows the state space to be described as follows.

$$\begin{bmatrix} \dot{v} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0.08 \\ 0 \end{bmatrix} u \tag{28}$$
$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ z \end{bmatrix} \tag{29}$$

Additionally, we can find the $z - posiition$ in terms of transfer function. As we can see by the state-space, the system has two poles which are both at 0 resulting in the following transfer function.

$$Y(s) = \frac{0.08}{s^2}U(s) \tag{30}$$
$$H(s) = \frac{0.08}{s^2} \tag{31}$$

## 5 Proposed Controller Designs for SISO Subsystem

Three different controllers were designed for this subsystem. These subsystems are just designing a discrete time controller with the desired poles of G(s), using full state feedback, and using a PID controller. As shown in 6 full state feedback shows the most promising results of the designed controller.

### 5.1 Designing a G(s)

For this subsystem, we design a controller $G(s)$ which is fed a error and gives a control signal to the propellers whose transfer function is $H(s)$. This allows us to say that the output $Y(s)$ can be represented as the following.

$$Y(s) = G(s)H(s)E(s) \tag{32}$$
$$E(s) = r - Y(S) \tag{33}$$
$$E(s) = r - \frac{0.056G(s)}{s^2}E(s) \tag{34}$$
$$r = E(s)\left(\frac{0.08G(s)}{s^2} + 1\right) \tag{35}$$
$$E(s) = \frac{rs^2}{0.08G(s) + s^2} \tag{36}$$

This allows us to derive $Y(s)$ as the following,

$$Y(s) = G(s)H(s)(r - Y(s)) \tag{37}$$
$$= \frac{G(s)H(s)r}{1 + G(s)H(s)} \tag{38}$$
$$Y(s) = \frac{0.08G(s)r}{s^2 + 0.08G(s)} \tag{39}$$

Suppose we let $G(s)$ be a second order control law with a single zero and four poles. In this case we say,

$$G(s) = \frac{z_1 + s}{a_1 s^3 + a_2 s^2 + a_3 s + a_4} \tag{40}$$

$$\tag{41}$$

This gives us the transfer function for the error as,

$$E(s) = \frac{r(a_1 s^5 + a_2 s^4 + a_3 s^3 + a_4 s^2)}{0.08s + 0.08z_1 + a_1 s^5 + a_2 s^4 + a_3 s^3 + a_4 s^2} \tag{42}$$

For this feedback, lets try placing the poles for the closed loop error at the following

$$-0.1 \pm 0.001j, -0.2 \pm 0.001j, -0.25$$

Using this controller also allows us to say that $Y(s)$ is the following,

$$Y(s) = \frac{0.08rs + 0.08rz_1}{a_1 s^5 + a_2 s^4 + a_3 s^3 + a_4 s^2 + 0.056s + 0.056z_1} \tag{43}$$

The open loop transfer function can also be stated as the following,

$$G(s)H(s) = \frac{z_1 + s}{a_1 s^5 + a_2 s^4 + a_3 s^3 + a_4 s^2} \tag{44}$$

## 5.2 Full State Feedback Controller Design

Another possible controller for this SISO system is a full state feedback controller. This solution involves feeding back the states of the observed system back into the controller. In this case we say that our input $u$ is the following,

$$u = -KB \tag{45}$$

However, we only have a sensor in this design to tell the position of the drone, not velocity. If we could design a full state feedback control loop using just the position then no observer would be needed. This would require our gain vector $K$ to be the form $K = \begin{bmatrix} 0 & k \end{bmatrix}$. Lets see if the poles can be placed using just this controller though. We know that the poles are the on the eigenvalues of the matrix $A - BK$. Using this we find this matrix and its eigenvalues.

$$(A - BK) = \begin{bmatrix} 0 & -0.08k \\ 1 & 0 \end{bmatrix} \tag{46}$$

This gives the system eigenvalues of $\pm\sqrt{0.08k}$. This outcome is not desired since for $k > 0$ at least one pole will be greater than 0. For $k < 0$ both poles will be fully imaginary with zero real components. This result is also undesirable. This shows we must use both states in our feedback loop. Using both states requires and observer whose design will be shown below.

First, assume we do have access to both states. This allows our gain vector $K$ to be in the form $K = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$ Using this gain vector allows us to place the poles of our closed loop system at the eigenvalues the matrix $A - BK$ again as shown below.

$$(A - BK) = \begin{bmatrix} -0.08k_1 & -0.08k_2 \\ 1 & 0 \end{bmatrix} \tag{47}$$

This gives the characteristic polynomial

$$\chi\lambda = \lambda^2 + 0.08k_1\lambda + 0.056k_2 \tag{48}$$

This polynomial is far better as it allows us to place the closed loop poles almost anywhere. However, this also requires us to design an observer for the system using the state space which was also done. When designing a system it is known that the poles for the observer and controller can be placed separately so that the observer design does not revolve around the controller design. Thus, we can design our observer using the following equation.

$$\hat{x_{k+1}} = A\hat{x_k} + L(y - C\hat{y_k}) + Bu \tag{49}$$
$$\hat{y_k} = C\hat{x} + Du \tag{50}$$

where $L = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix}$.

It is also known that the poles of the closed loop observer are equivalent to the eigenvalues of $A - LC$. We show this matrix below.

$$A - LC = \begin{bmatrix} 0 & -l_1 \\ 1 & -l_2 \end{bmatrix} \tag{51}$$

This gives the characteristic poly

$$\lambda^2 + l_2\lambda + l_1 \tag{52}$$

For the feedback controller, we can just use the direct output of the observer.

Using this observer and controller design the poles and zeros of the closed loop system can be placed as desired.

### 5.3 PID Controller Design

The final controller to test on the simplified SISO system was a PID controller. The PID controller uses gains $k_p$, $k_i$ and $k_d$ to adjust the proportional, integral, and derivative gains respectively. This controller allows the gains to be tuned to see if some combination produces and optimal output.

## 6 Results

### 6.1 Plant Design Results

The nonlinear plant was designed in MATLAB using the equations. Since these equations are non-linear it is easiest to just write the code for the plant then test a discrete time controller on the linearized model. Below, Figures 1 and 2 show the position and velocity of the drone respectively over time when no input is given. This is a easy condition to check since the velocity should decrease linearly over time while the position decreases as the integral of the velocty.

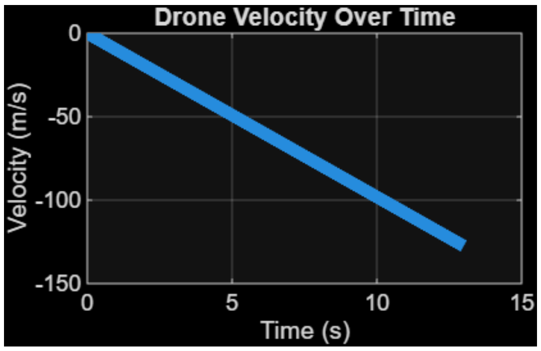

Figure 2: Drone Plant Model - Velocity Over Time

As we can see by these tests, the model clearly works as expected and will be good to use for testing our controllers.

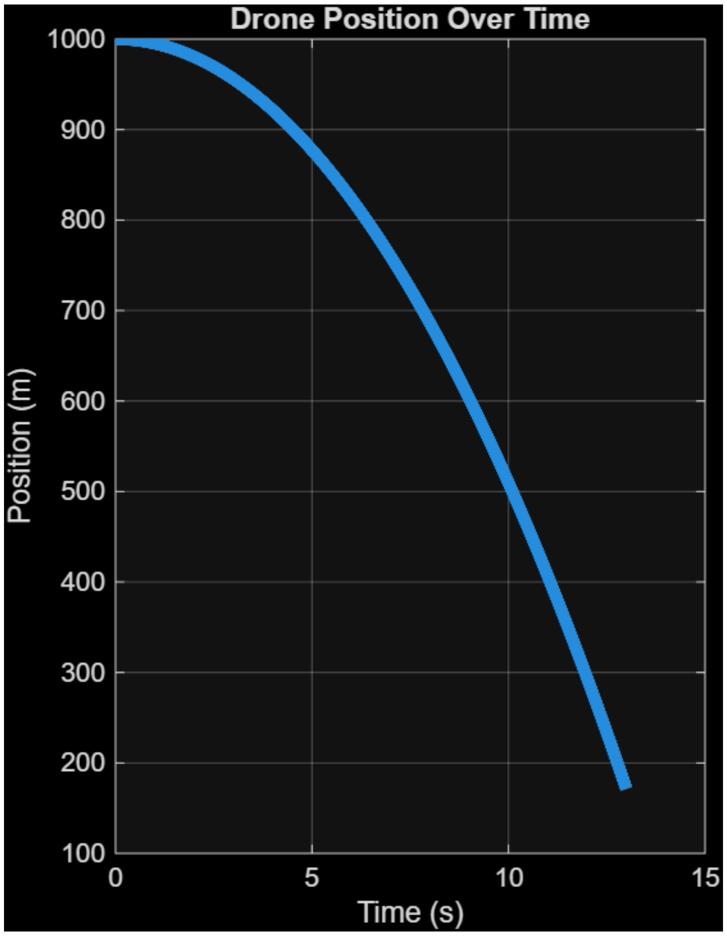

Figure 1: Drone Plant Model - Position Over Time

### 6.2 Designing G(s) Results

We designed a controller $G(s)$ as specified and used it for our controller. The root locus for this controller is shown below in 3

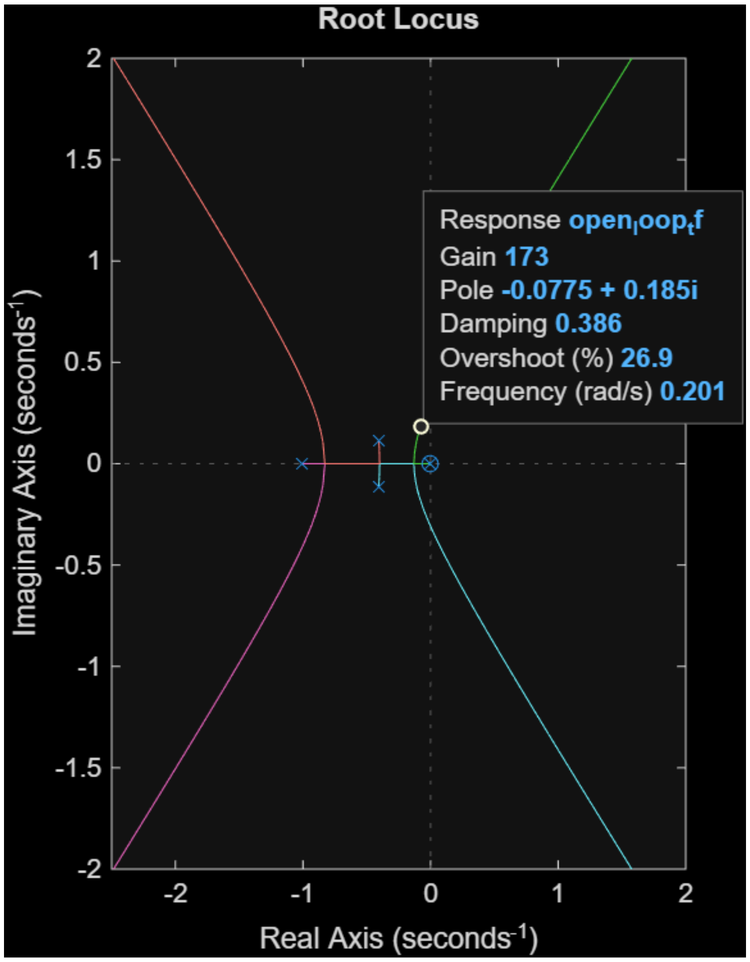

Figure 3: Designed G(s) Controller Root Locus

As we can see by the root locus, we would need a very high gain in order for the system to go unstable. This is a desired characteristic in the developed controller. Next, we investigate the bode plot of the closed loop controller and analyze the phase margin of the closed loop system. The bode plot is shown below in 4.

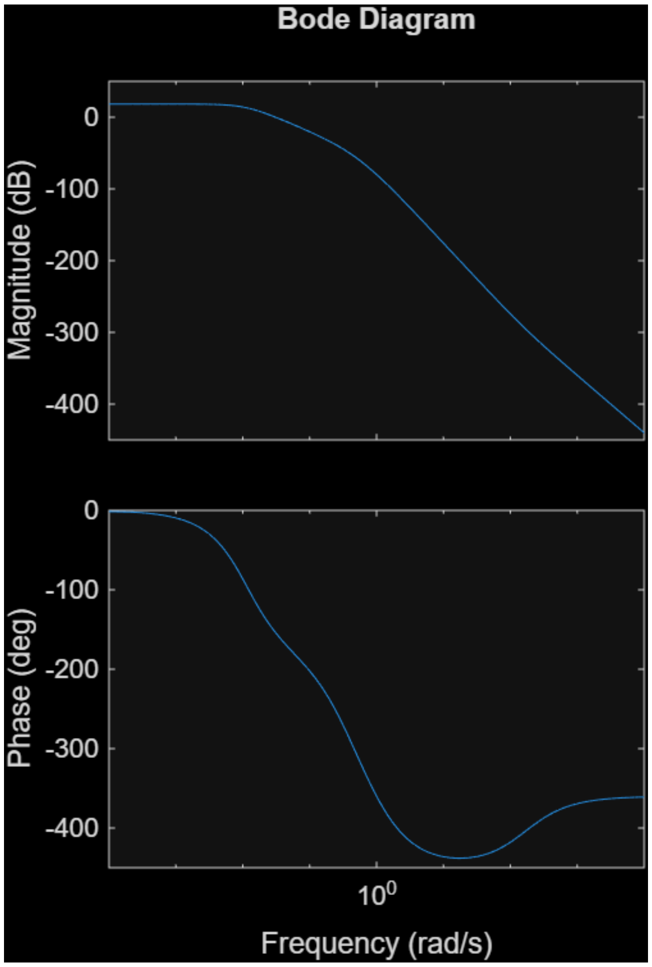

Figure 4: Designed G(s) Controller Bode Plot

As seen by the bode plot for the closed loop controller, we have about 30° of phase margin. While this is less than desired, it is still enough so that the controller should not go unstable. However, as we will see in the results, this controller does not work so well. The plots of velocity, position, and input of the system when using the controller are shown below in Figures, 5, 6, 7.

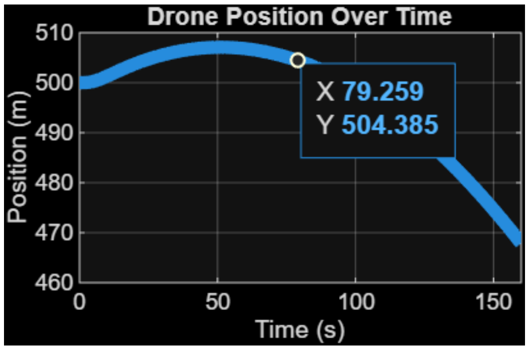

Figure 5: Designed G(s) Results - Position Over Time

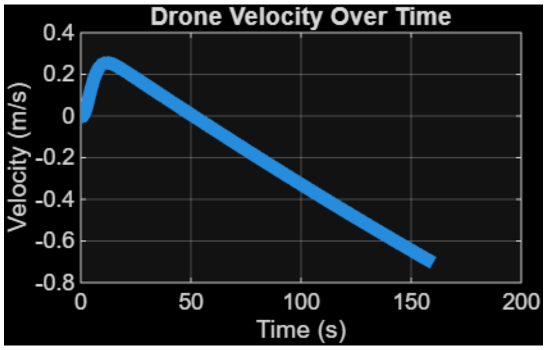

Figure 6: Designed G(s) Results - Velocity Over Time

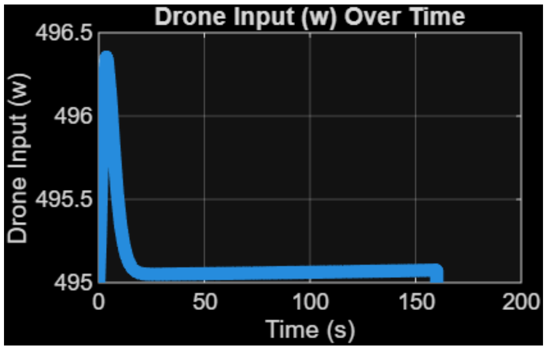

Figure 7: Designed G(s) Results - Input Over Time

As seen by the results above, this controller does not really work well for this system. The drone was set to go to a position of 900m, and the controller did not perform well at all in this task. Additionally, if I move the poles more negative, it results in overshoot of the controller which is undesirable. This is all desipte having proper gain and phase margin designs meaning these results are liekly due to nonlinearities within the system. One other problem is the massive spike in input at the start which should be handled in the future even if the controller worked better.

### 6.3 Full State Feedback Results

The next design tested in my work done on the controller was a full state feedback controller design. This design has the benefit of not only using position as a state to control with, but also velocity. As discussed previously, this required the use of an observer to estimate the velcoity which is unable to be measured. However, as we will not later, this observer performed extremely well in tracking the systems position and velocity. We first though will analyze the bode plot of the final close loop state space using full state feedback.

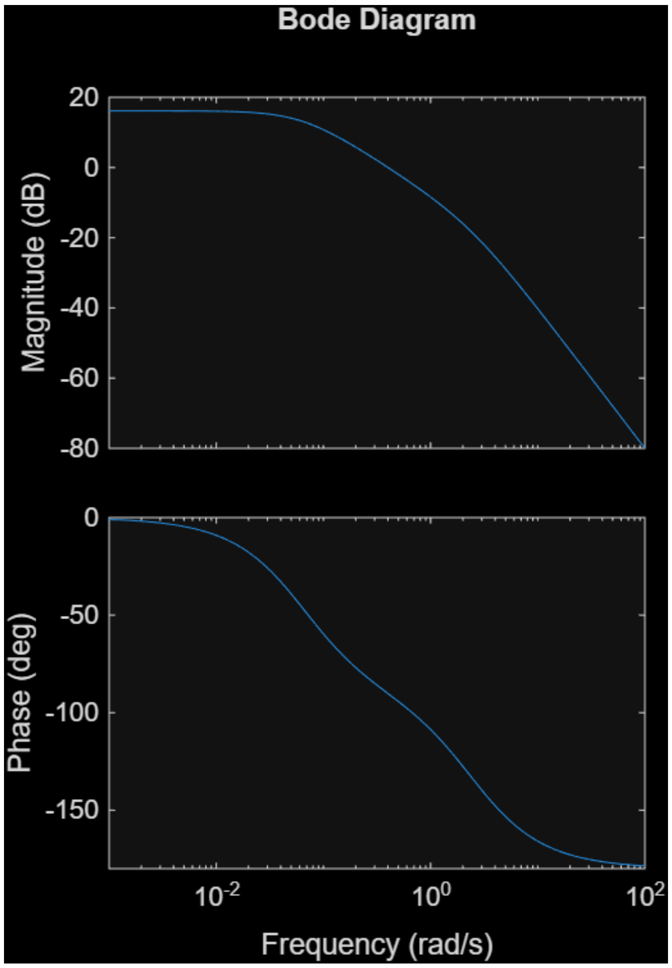

Figure 8: Full State Feedback Bode Plot

As seen by the bode plot, our system incorporates robust design and has plenty of phase margin (roughly 80°). This is a desirable characteristic to have for our closed loop controller. Of all controller designs for this SISO system, this controller has the best frequency response.

Next, we show the results of the controller when using this controller in

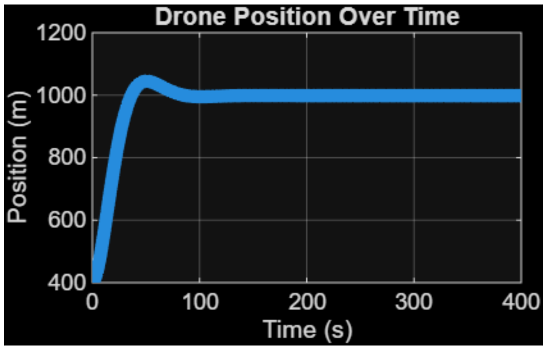

Figure 9: Full State Feedback Controller Position over Time

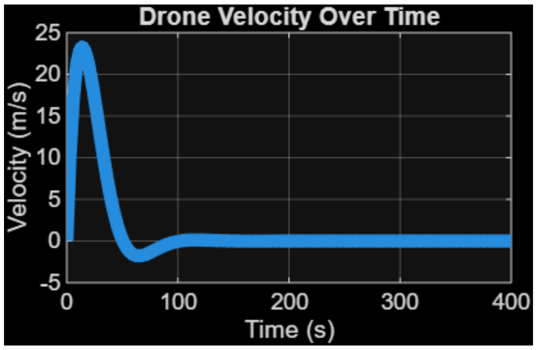

Figure 10: Full State Feedback Controller Velocity Over Time

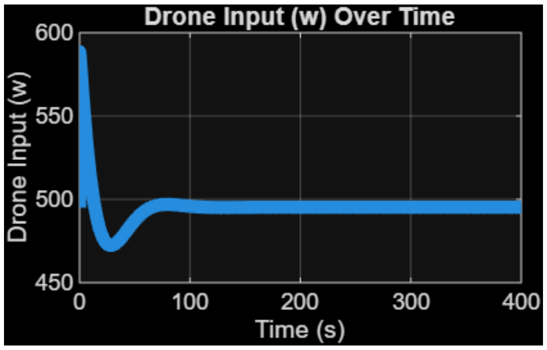

Figure 11: Full State Feedback Controller Input Over Time

As seen by these results, this controller took the drone from a position of 400m to 1000m with roughly 20m of overshoot (well under the 2% spec stated earlier). Again, the issue with this controller is that the input spikes at the start, Additionally, the controller does not meet the velocity limiting spec stated at the start of the report. However, a design to handle that specification will be designed later in the semester.

The greatest success coming from this controller design was the developed observer though. As seen in Figure 12, the observed position and velocity match the actual position and velocities extremely well. Given the system's nonlinearities, this observer works extremely well.

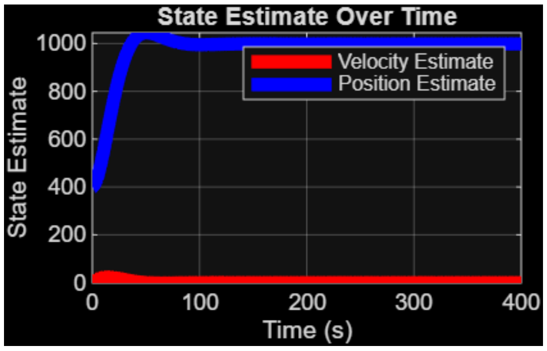

Figure 12: Full State Feedback Observer Results

### 6.4 PID Controller Design Results

Again, we want to analyze the frequency response and robustness of the system. This is again done by finding the root locus of the open loop controller and the bode plot of the closed loop feedback controller. These results are shown below in Figures 13 and 14 respectively.

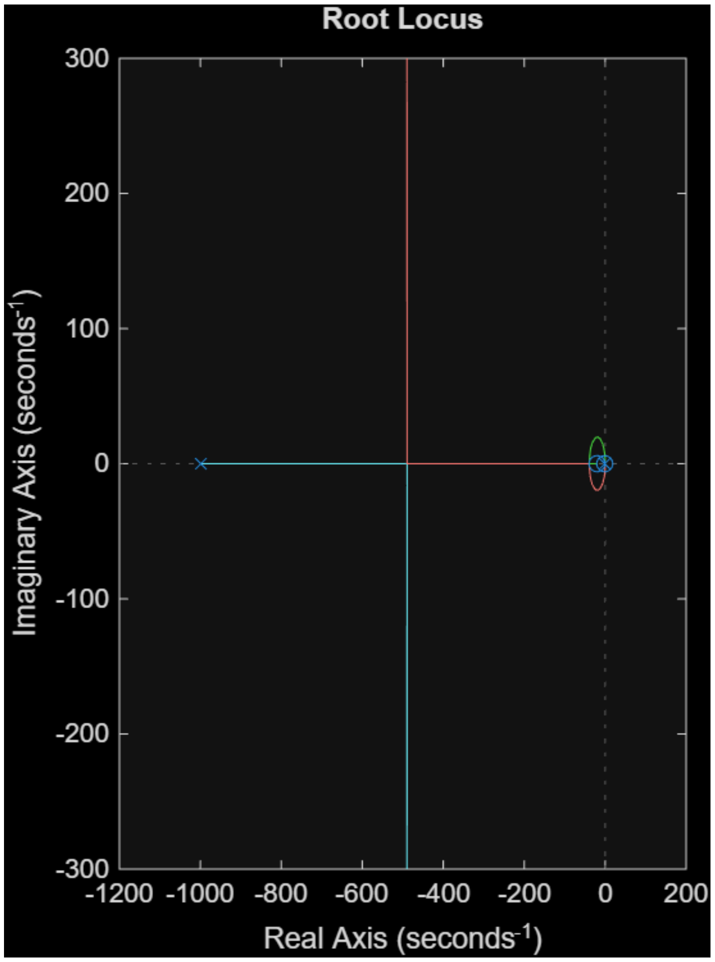

Figure 13: PID Root Locus Results

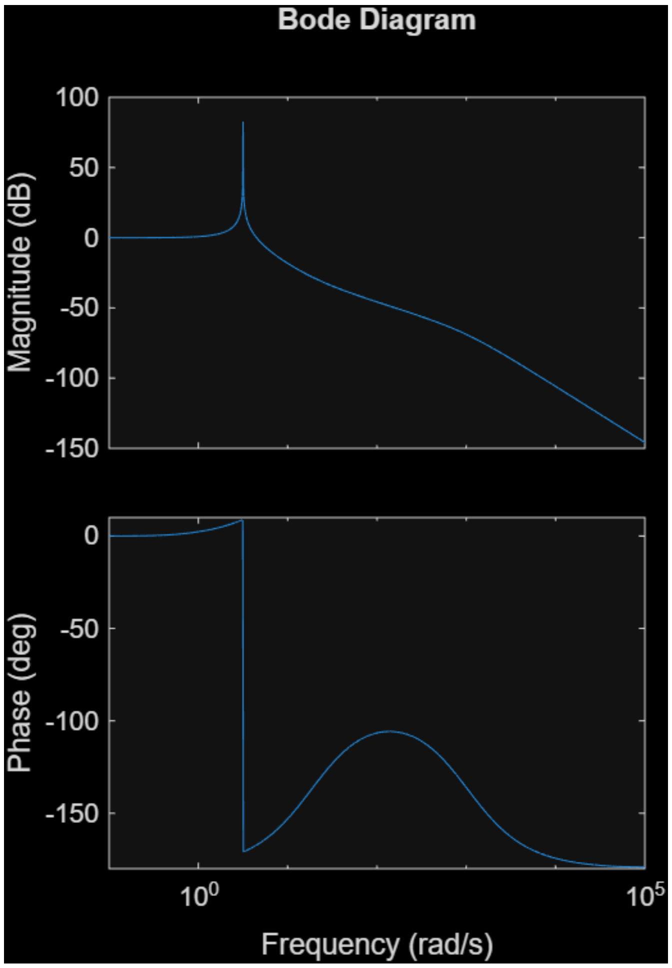

Figure 14: PID Bode Plot Results

As seen by the root locus and bode plots using just a PID controller does not provide for a stable system When we look at the open loop transfer function though this is obvious. The transfer function for the open loop found through MATLAB is the following.

$$G(s)H(s) = \frac{510s^2 + 10005s + 5000}{s^4 + 1000s^3} \tag{53}$$

Unfortunately, the pid controller does not provide enough poles and zeros make some of the poles in the denominator nonzero. Having these three, unmovable poles in the denominator will prevent this controller from working.

As we can see below by the results of this controller on our drone, the PID controller performs worst in this scenario. This can be seen in Figures, 15, 16, 17

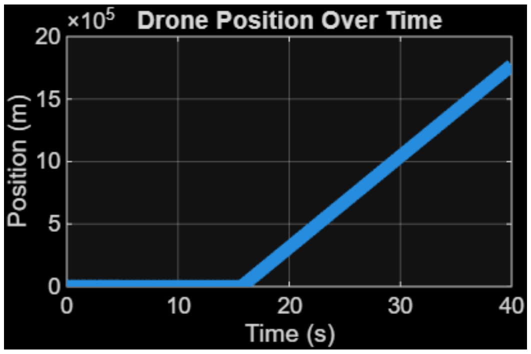

Figure 15: PID Controller Position Over Time

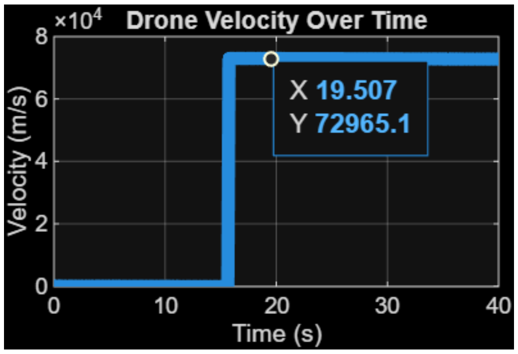

Figure 16: PID Controller Velocity Over Time

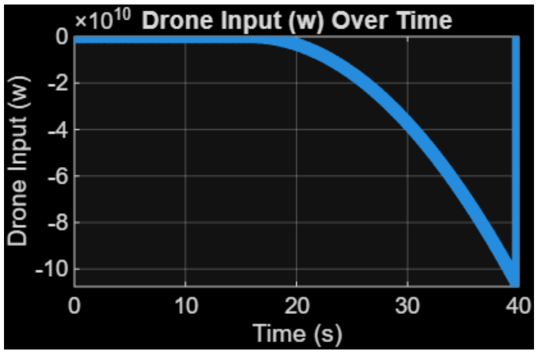

Figure 17: PID Input Results Over Time

## 7 Conclusion

As seen by the controllers developed, the full state feedback controller by far performed the best out of all of them. It may be possible to design a better G(s) function to make as our controller with a different form, but using the observer to find velocity most likely made the biggest impact. Whether or not the final design makes use of full state feedback, the developed observer should heavily be considered within the final controller design.

## 8 More Information

More information about the code and live scripts can be seen at my GitHub Page at the link below.

https://github.com/NightHydra/Multivariable-Control-Systems-Drone-Controller-Project