Controls Final Project Report

Mike Hennessy, Kennedy Necoechea, Jack Michaelis

Control Systems

Mechanical Engineering Department

Loyola Marymount University

Los Angeles

December 13th, 2024

Design Philosophy

One of our primary design priorities was effective defense, focused on preventing the opposing drone from scoring. To achieve this, we programmed our drone to position itself strategically between the opposing drone and its target basket, aiming to intercept or block shots. This defensive maneuver disrupts the opponent's scoring opportunities, forcing them into less favorable positions and reducing their efficiency. The tradeoff for prioritizing this defensive capability was a slight reduction in our drone's speed and maneuverability, as resources were allocated toward precision positioning and tracking algorithms. However, the benefits of consistently interfering with opposing shots outweighed the cost, as strong defense is critical to controlling the game and limiting the opponent's scoring potential.

Another key goal was ensuring smooth movement and accurate shot execution, particularly from a distance. By optimizing the drone's mechanics and programming, we ensured it could maintain proper spacing from the basket, enabling it to calculate and execute high-percentage shots. This positioning not only improved shooting accuracy but also increased the likelihood of scoring three-pointers, maximizing point potential. The tradeoff for this approach was a reduced focus on close-range agility, as emphasis was placed on precise long-distance calculations and consistent mechanics. Despite this, the ability to reliably score higher-value shots provided a significant competitive advantage, making this tradeoff worthwhile.

Dynamics Controller Design [Q1-7]

Question 1:

$$Y(s) = G_y(s) \cdot T(s) \rightarrow G_y(s) = \frac{Y(s)}{T(s)}$$

$$T(t) = m \cdot \ddot{y}(t) \rightarrow T(s) = m \cdot s^2 \cdot Y(s)$$

$$G_y(s) = \frac{Y(s)}{m \cdot s^2 \cdot Y(s)} = \boxed{\frac{1}{ms^2} = G_y(s)} \rightarrow \text{vertical movement}$$

$$T(s) = G_T(s) \cdot T_d(s) \rightarrow G_T(s) = \frac{T(s)}{T_d(s)}$$

$$\dot{T}(t) = \frac{1}{\tau}[T_d(t) - T(t)]$$

$$s \cdot T(s) = \frac{1}{\tau}[T_d(s) - T(s)]$$

$$s \cdot T(s) \cdot \tau = T_d(s) - T(s)$$

$$T(s)[s \cdot \tau + 1] = T_d(s)$$

$$\frac{T(s)}{T_d(s)} = \frac{1}{s \cdot \tau + 1}$$

$$\boxed{G_T(s) = \frac{1}{s\tau + 1}} \rightarrow \text{rotor spin-up}$$

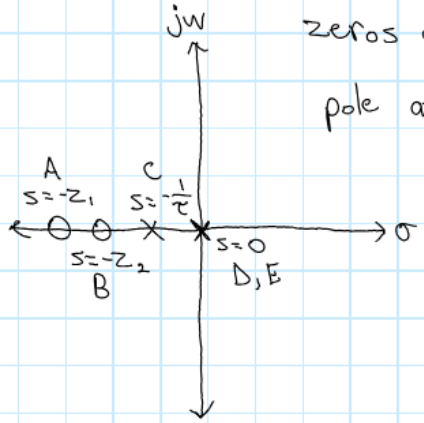
$$K(s) = \frac{K_D \cdot s^2 + K_P \cdot s + K_I}{s}$$

$$\boxed{K(s) = \frac{K(s + z_1)(s + z_2)}{s}} \rightarrow \text{PID controller}$$

$$T_{OL,y}(s) = K_y(s) \cdot G_r(s) \cdot G_y(s) = \left[\frac{K(s+z_1)(s+z_2)}{s}\right]\left[\frac{1}{s\cdot\tau+1}\right]\left[\frac{1}{m s^2}\right]$$

zeros at $s = -z_1, -z_2$ (A, B) poles at $s = 0, s = -\frac{1}{\tau}$ (C, D)

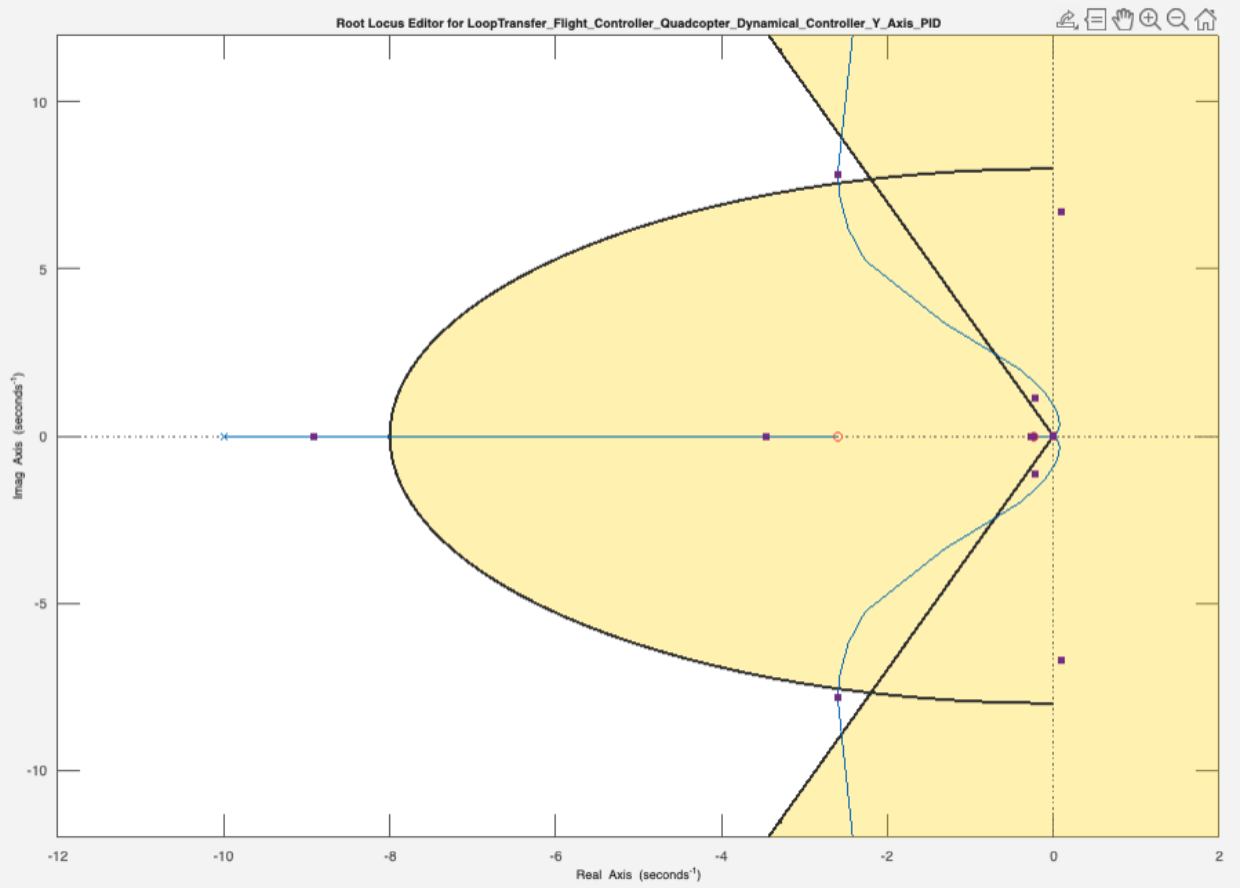
pole at $s = 0$ (E)



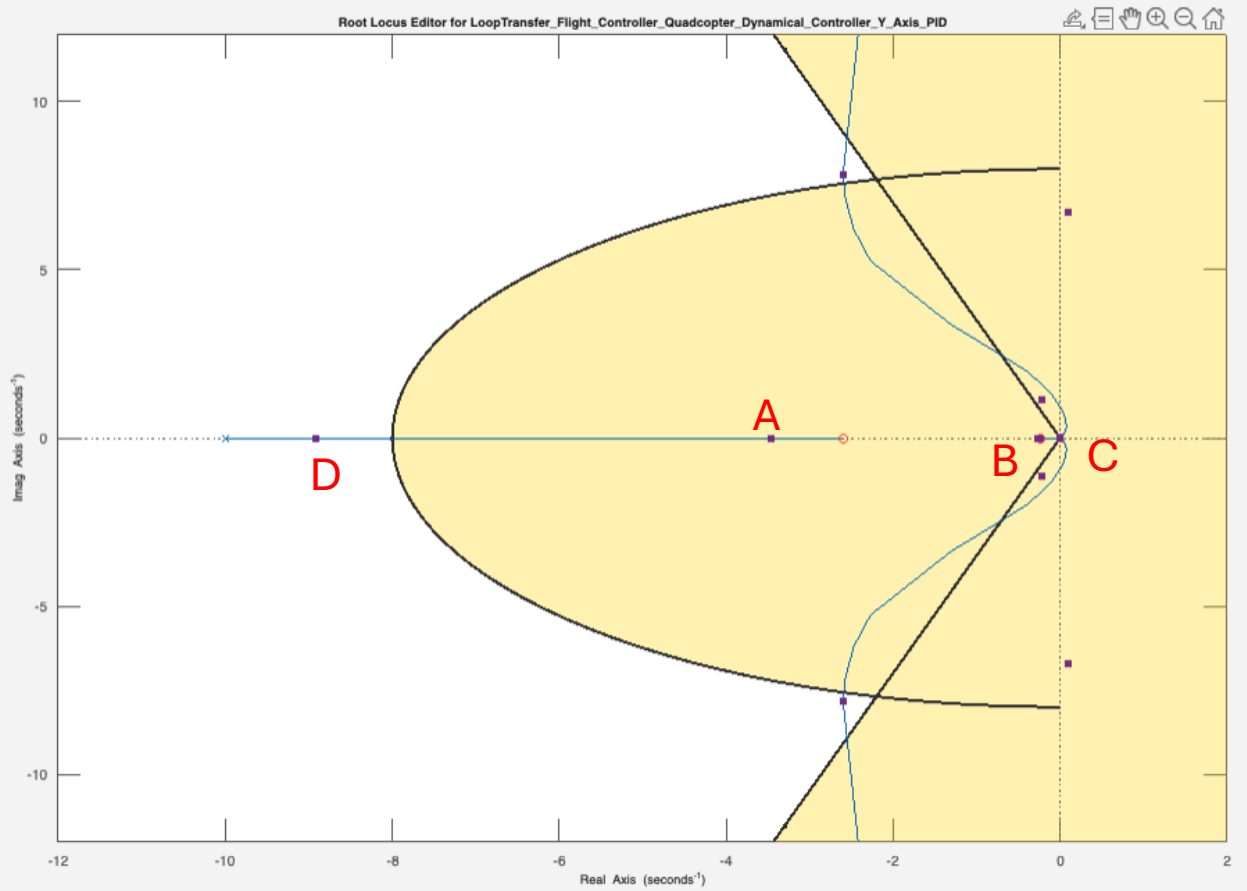
A, B: PID controller zeros
C: rotor spin-up pole (thrust dynamics)
D: vertical movement pole
E: PID controller pole

Question 2:

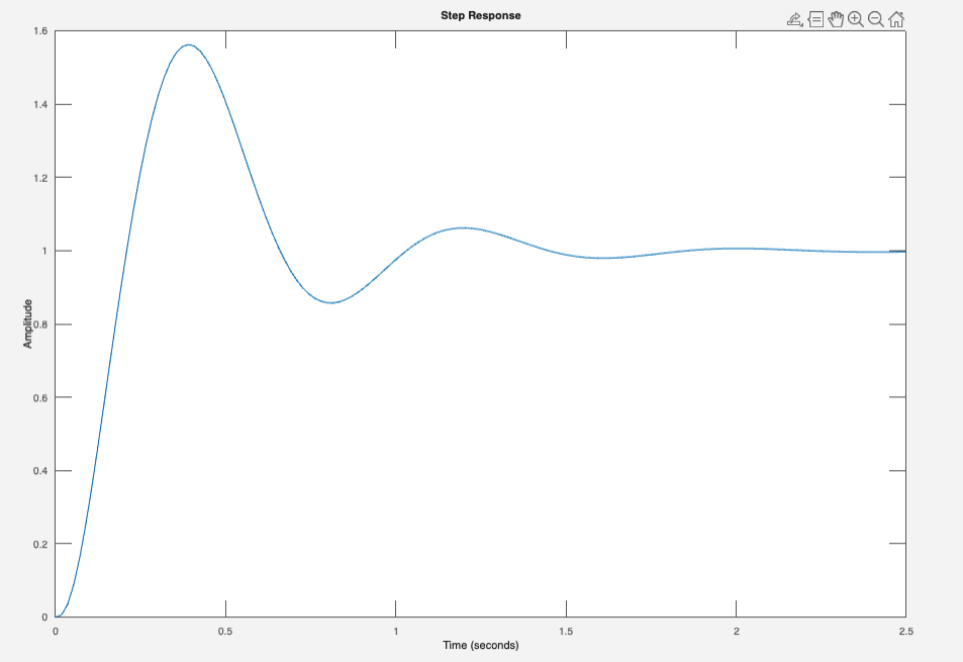
a. Root Locus Plot $\zeta > 0.3$ and $\omega_n > 8\, rad/s$



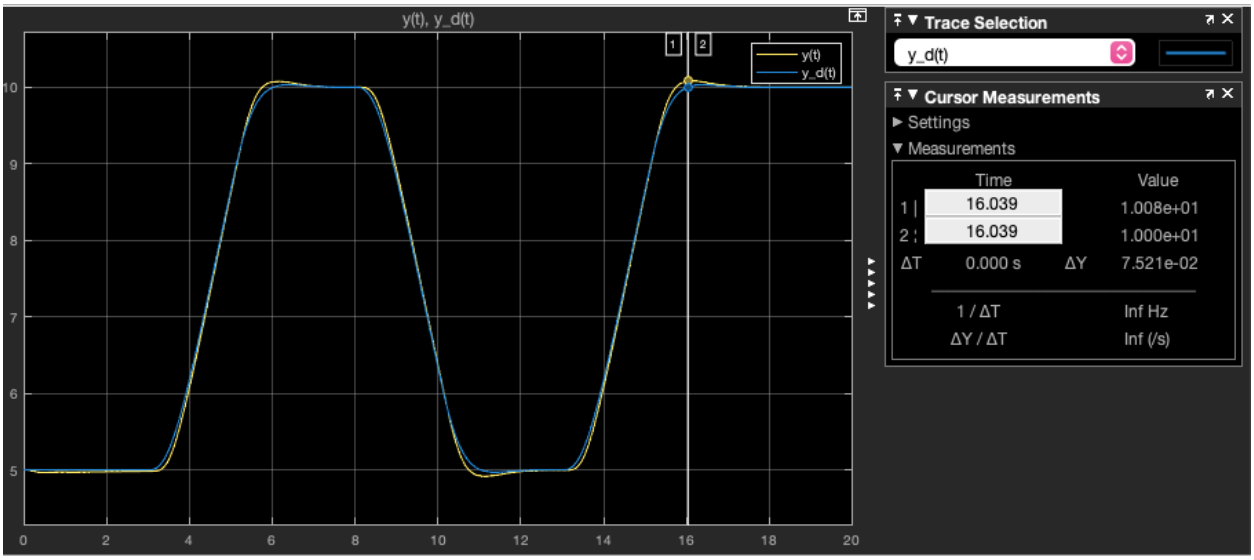
b. Root locus plot with labeled X's and O's



c. Theoretical response step plot



d. Actual response plot “Y-Axis”



Question 3:

$\Theta(s) = G_\theta(s) \cdot M(s) \rightarrow G_\theta(s) = \frac{\Theta(s)}{M(s)}$

$M(t) = J \cdot \ddot{\Theta}(t)$

$M(s) = J \cdot s^2 \cdot \Theta(s)$

$G_\theta(s) = \frac{\Theta(s)}{J \cdot s^2 \cdot \Theta(s)} = \boxed{\frac{1}{J \cdot s^2} = G_\theta(s)}$ → pitch angle

$M(s) = G_M(s) \cdot M_d(s) \rightarrow G_M(s) = \frac{M(s)}{M_d(s)}$

$\dot{M}(t) = \frac{1}{\tau}[M_d(t) - M(t)]$

$s \cdot M(s) = \frac{1}{\tau}[M_d(s) - M(s)]$

$s \cdot M(s) \cdot \tau = M_d(s) - M(s)$

$M(s)[s \cdot \tau + 1] = M_d(s)$

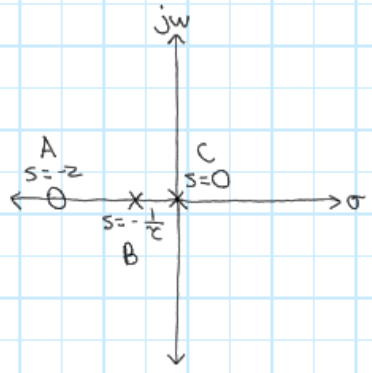
$\frac{M(s)}{M_d(s)} = \boxed{\frac{1}{s \cdot \tau + 1} = G_M(s)}$ → rotor spin-up

$K_\theta(s) = K_\theta \cdot s + K_s$

$\boxed{K_\theta(s) = K(s + z)}$ → PD controller

$T_{OL,\theta}(s) = K_\theta(s) \cdot G_M(s) \cdot G_\theta(s) = [K(s+z)]\left[\frac{1}{s \cdot \tau + 1}\right]\left[\frac{1}{J \cdot s^2}\right]$

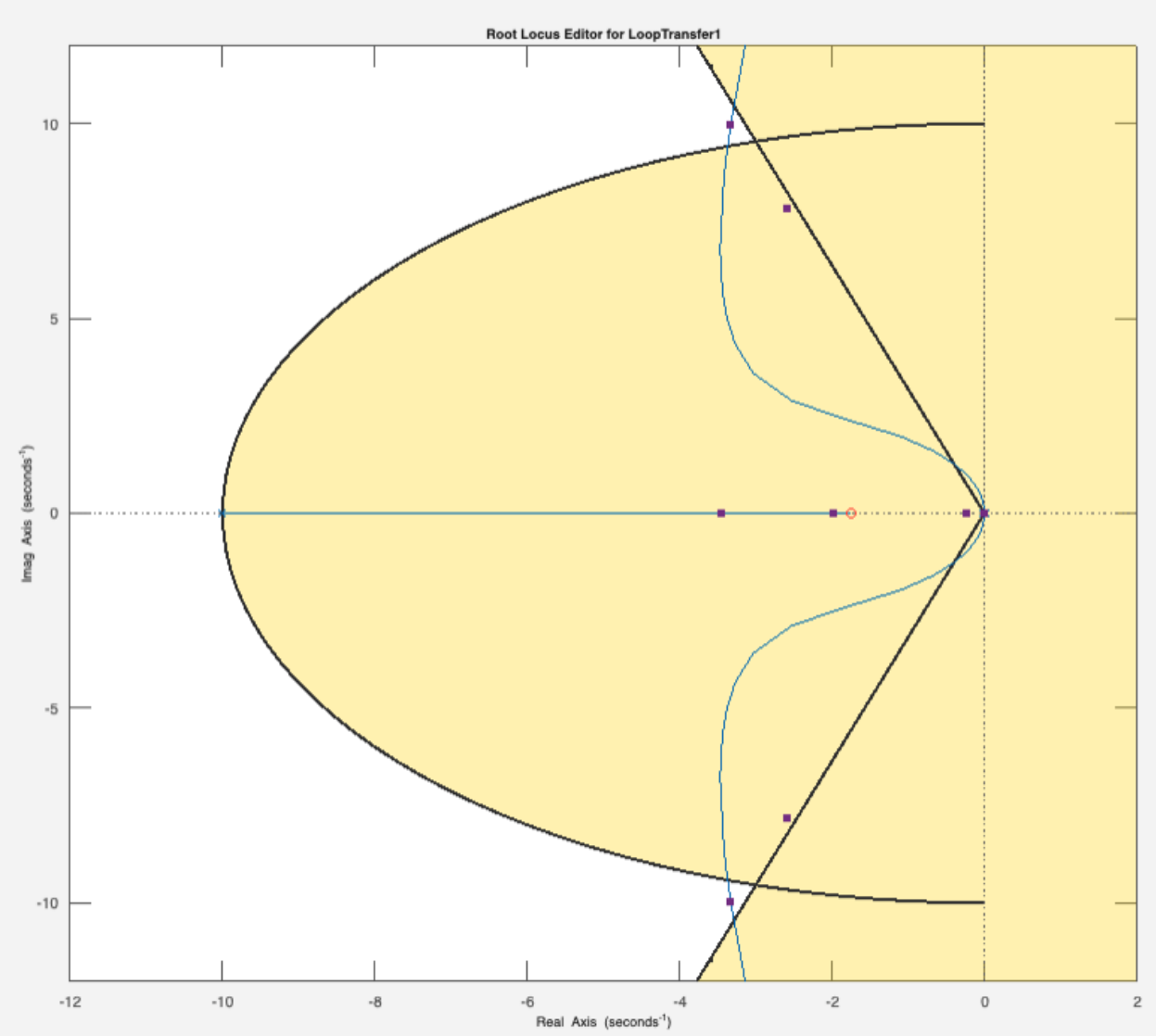
zero at $s = z$ (A); pole at $s = -\frac{1}{\tau}$ (B); pole at $s = 0$ (C)



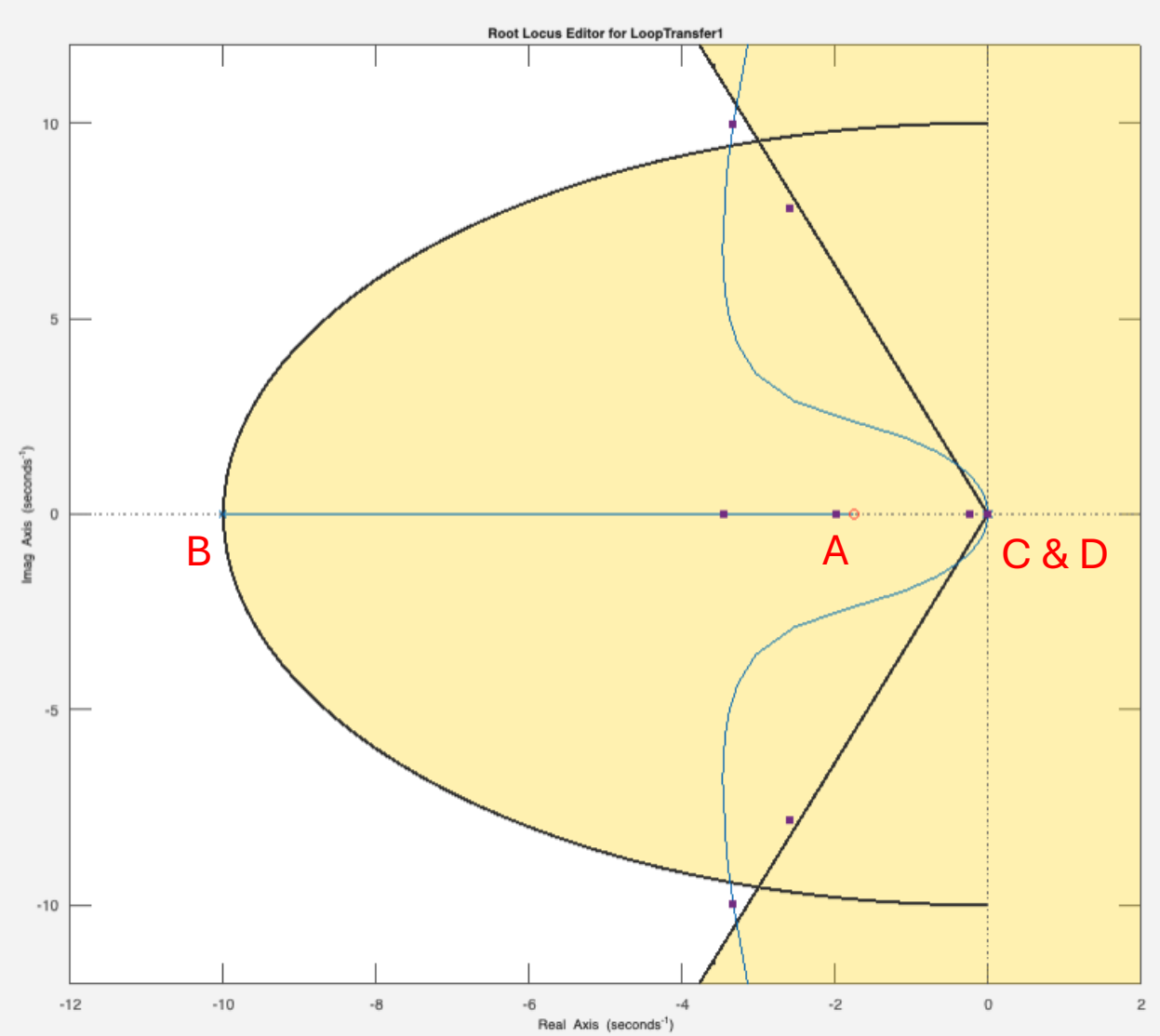
A: PD controller zero
B: rotor spin-up pole (thrust dynamics)
C: pitch angle pole

Question 4:

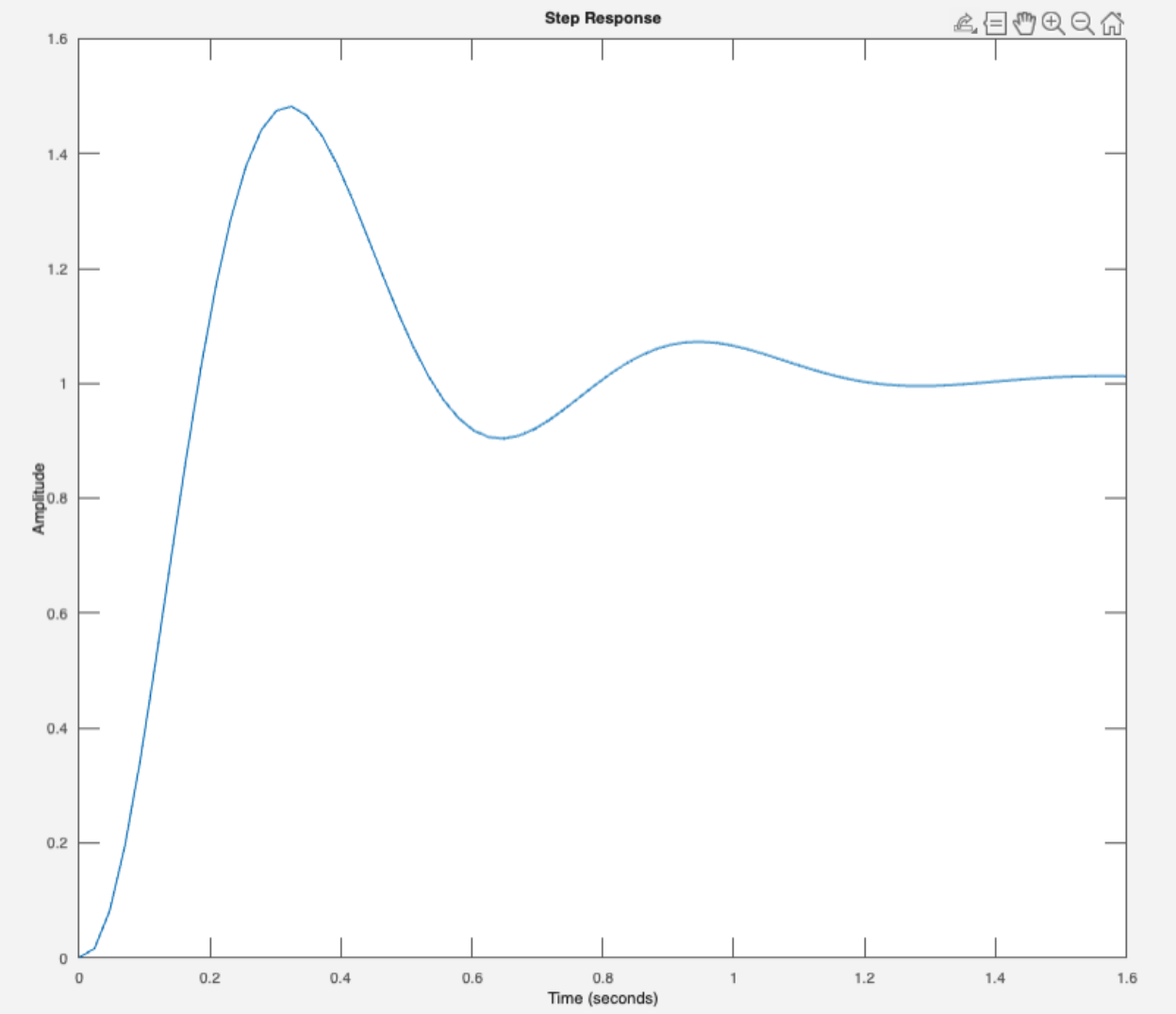
a. Root Locus Plot $\zeta > 0.3$ and $\omega_n > 10\ rad/s$



b. Root locus plots with labeled X's and O's



c. Theoretical step response


Step Response
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

Question 5:

$$X(s) = G_x(s) \cdot \Theta(s) \rightarrow G_x(s) = \frac{X_s}{\Theta(s)}$$

$$m \cdot \ddot{x}(t) = T_o \cdot \theta(t)$$

$$m \cdot s^2 \cdot X(s) = T_o \cdot \Theta(s)$$

$$\frac{X(s)}{\Theta(s)} = \boxed{\frac{T_o}{m \cdot s^2} = G_x(s)} \rightarrow \text{horizontal movement}$$

$$K_x(s) = \frac{K_D \cdot s^2 + K_P \cdot s + K_I}{s}$$

$$\boxed{K_x(s) = \frac{K(s+z_1)(s+z_2)}{s}} \rightarrow \text{PID controller}$$

$$T_{CL,\theta}(s) = \frac{K_\theta(s) \cdot G_m(s) \cdot G_\theta(s)}{1 + K_\theta(s) \cdot G_m(s) \cdot G_\theta(s)} = \frac{[K(s+z_3)]\left[\frac{1}{s \cdot \tau + 1}\right]\left[\frac{1}{J \cdot s^2}\right]}{1 + [K(s+z_4)]\left[\frac{1}{s \cdot \tau + 1}\right]\left[\frac{1}{J \cdot s^2}\right]}$$

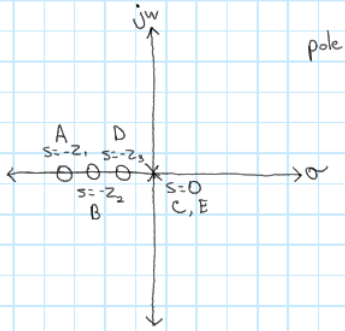
closed-loop transfer function

$$= \boxed{\frac{K(s+z_3)}{[s \cdot \tau + 1][J \cdot s^2] + [K(s+z_4)]} = T_{CL,\theta}(s)}$$

$$T_{OL,x}(s) = K_x(s) \cdot T_{CL,\theta}(s) \cdot G_x(s) = \left[\frac{K(s+z_1)(s+z_2)}{s}\right]\left[\frac{K(s+z_3)}{[s \cdot \tau + 1][J \cdot s^2] + [K(s+z_4)]}\right]\left[\frac{T_o}{m \cdot s^2}\right]$$

zeros at $s = -z_1, -z_2$ (A, B); zero at $s = -z_3$ (D); pole at $s = 0$ (E)

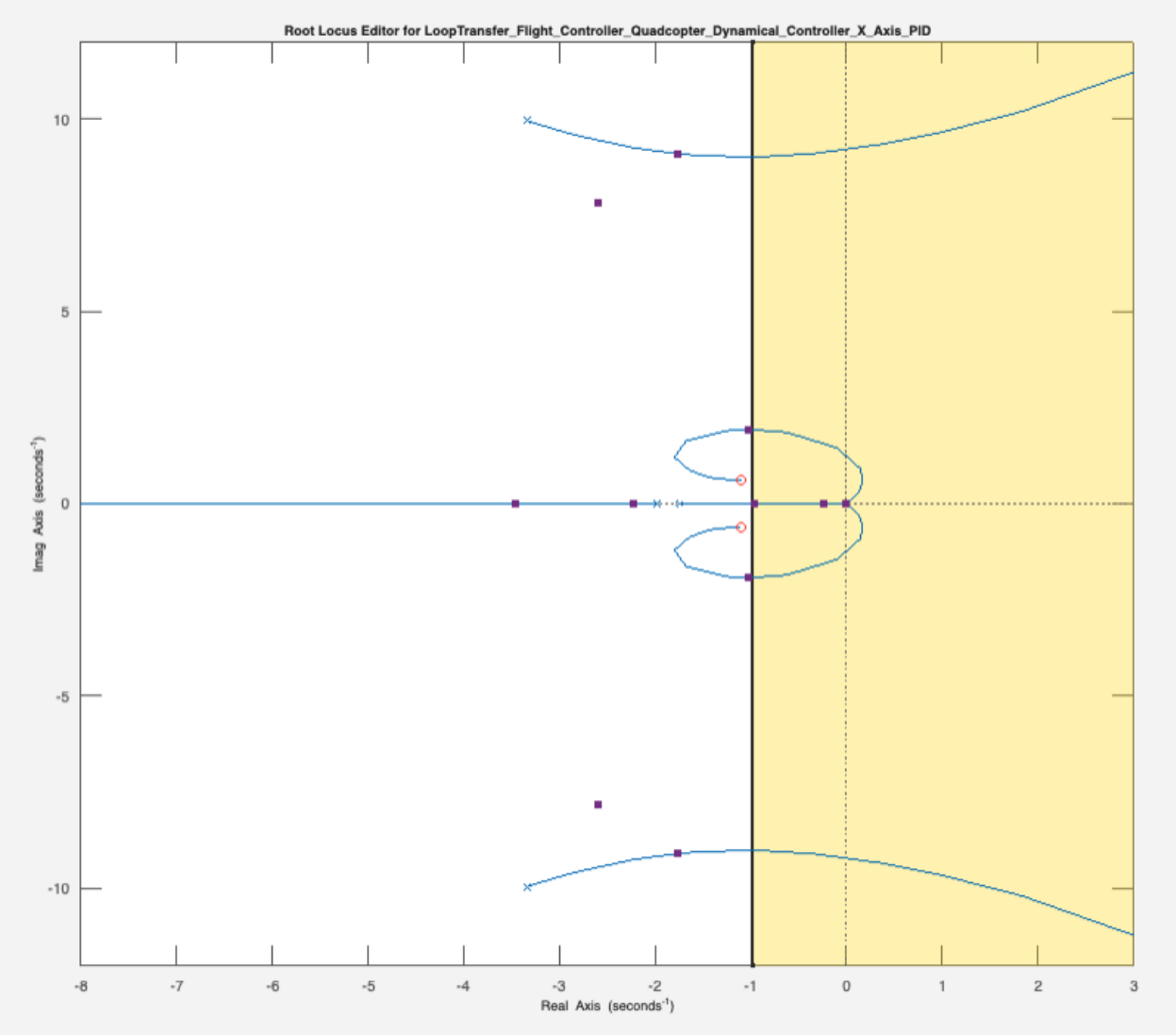
pole at $s = 0$ (C)



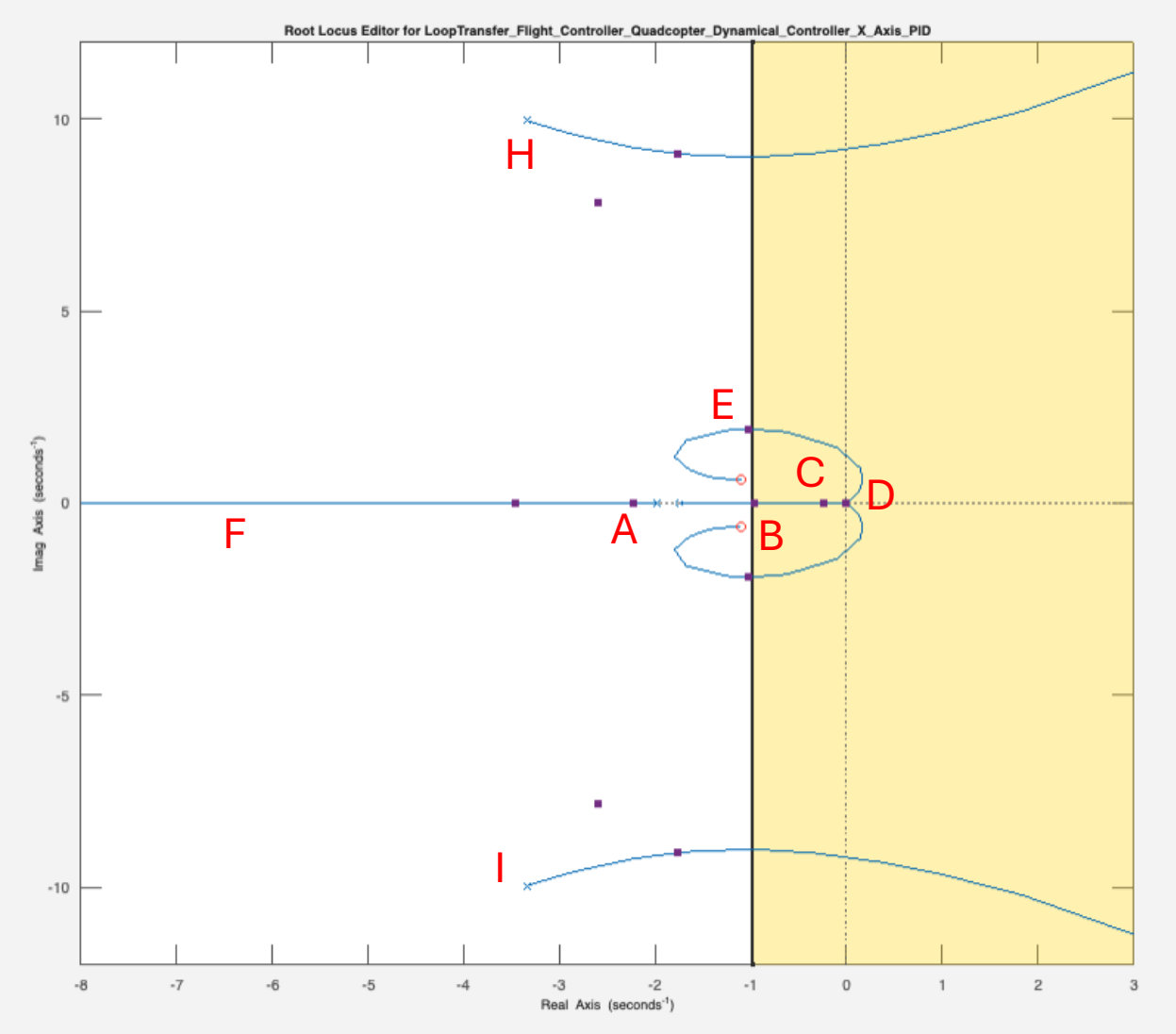
A, B: PID controller zeros
C: PID controller pole
D: closed-loop zero of pitch control system
E: horizontal movement pole

Question 6:

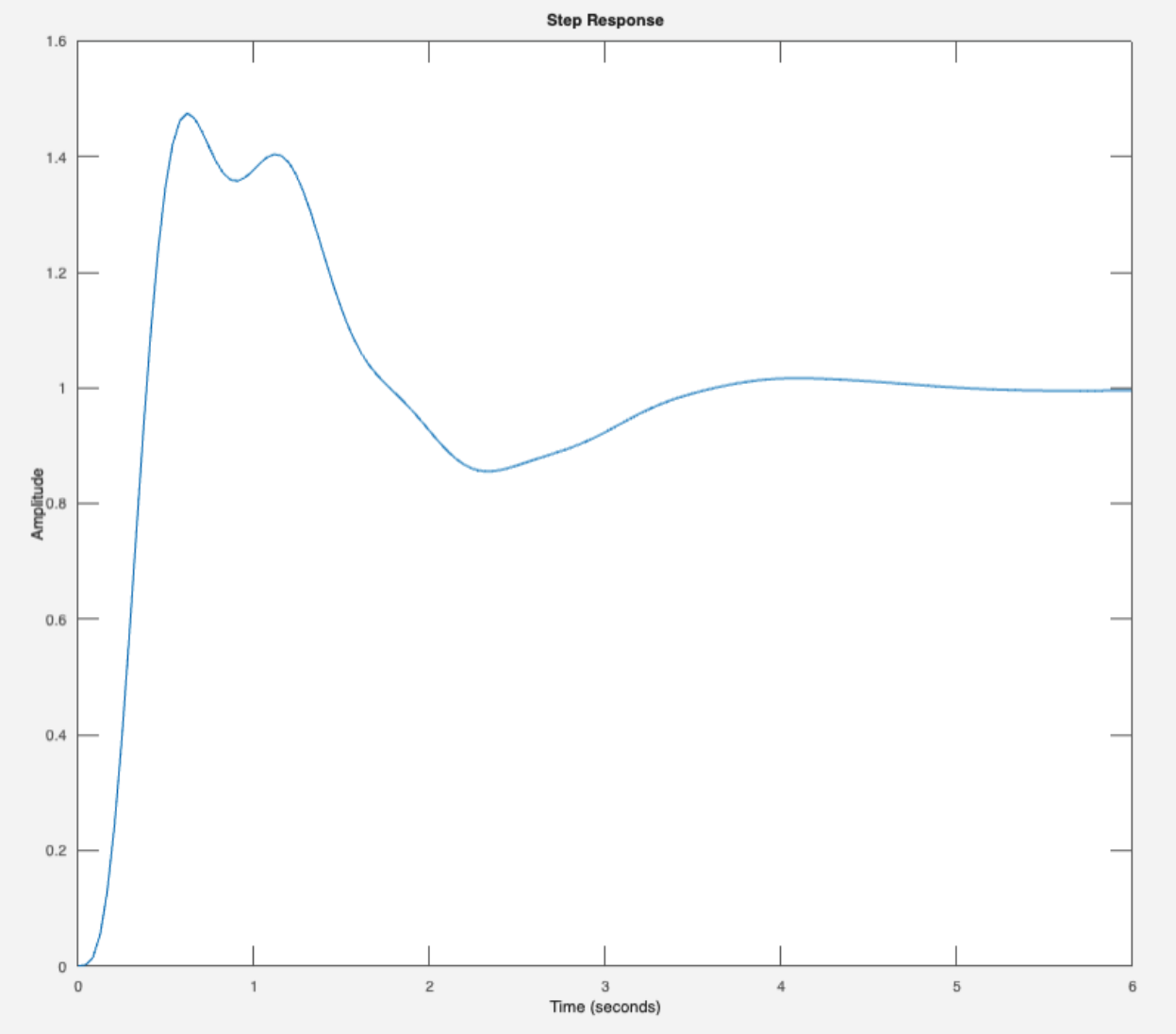
a. Root locus plot with $t_s < 4\ s$



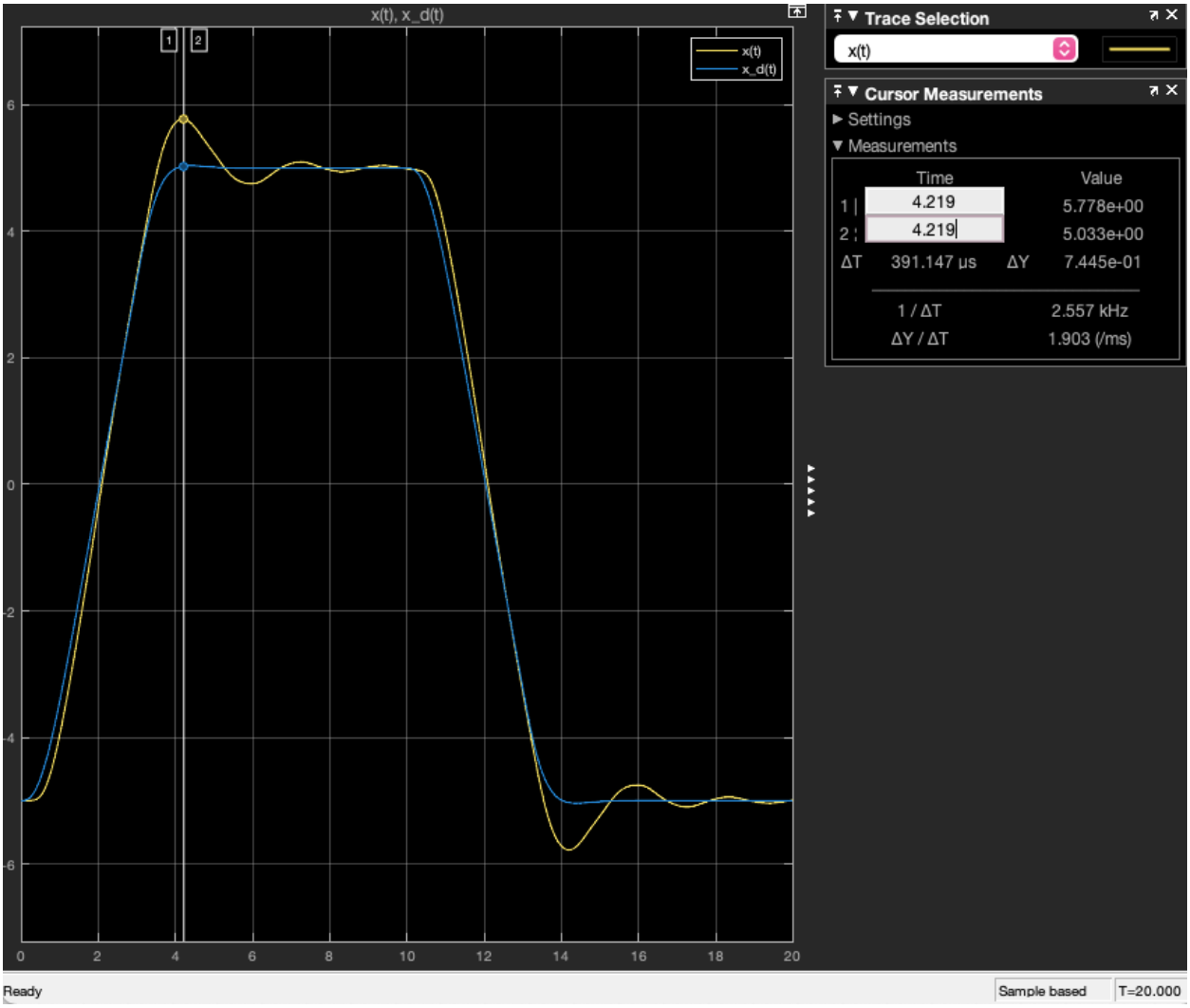
b. Root locus plot with labeled X's and O's



c. Theoretical step response

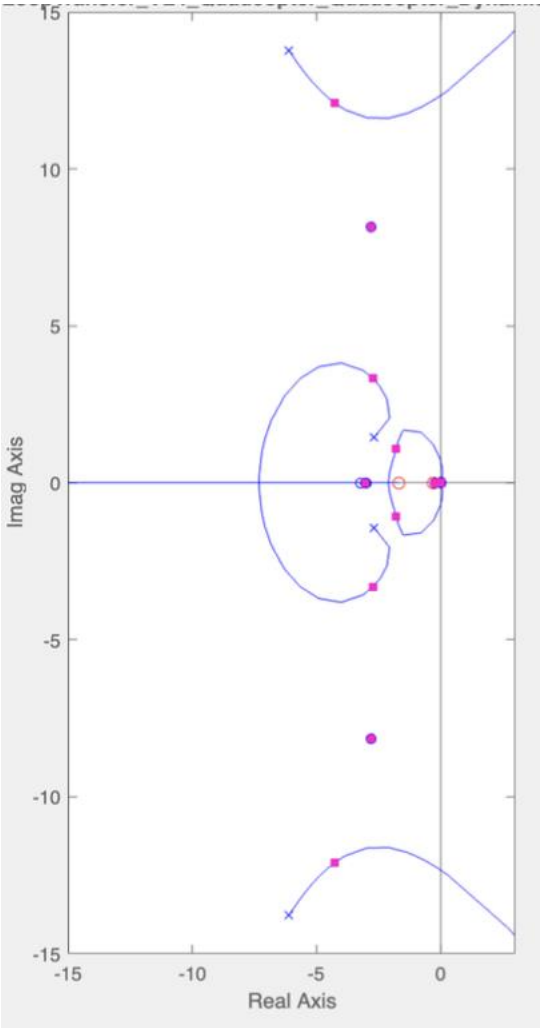


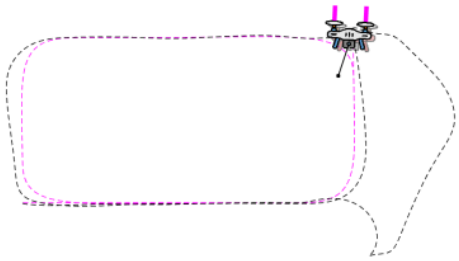
d. Actual response plot “X-Axis”


x(t), x_d(t)
x(t)
x_d(t)
Trace Selection
x(t)
Cursor Measurements
Settings
Measurements
Time
Value
1 | 4.219 5.778e+00
2 ; 4.219 5.033e+00
ΔT 391.147 µs ΔY 7.445e-01
1 / ΔT 2.557 kHz
ΔY / ΔT 1.903 (/ms)
Ready
Sample based
T=20.000

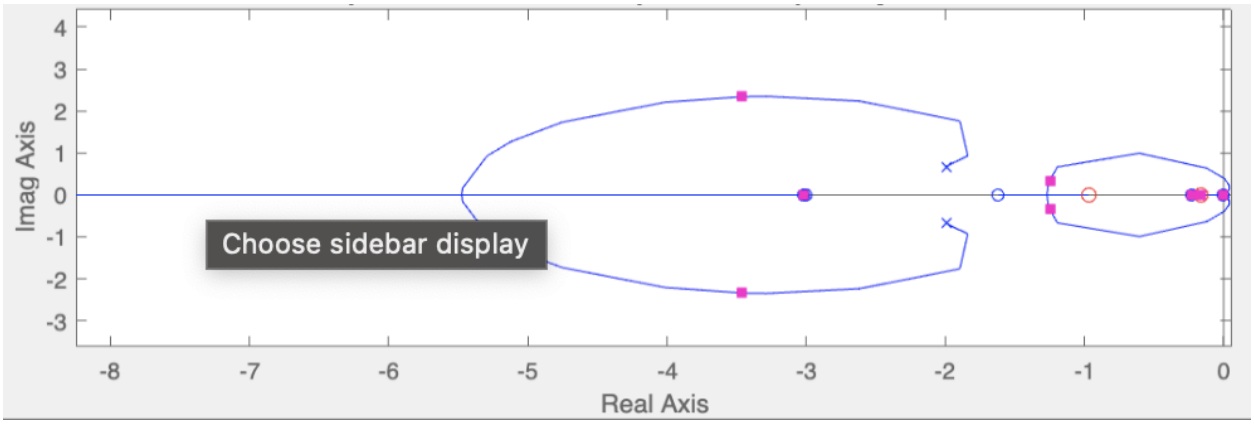
Question 7:

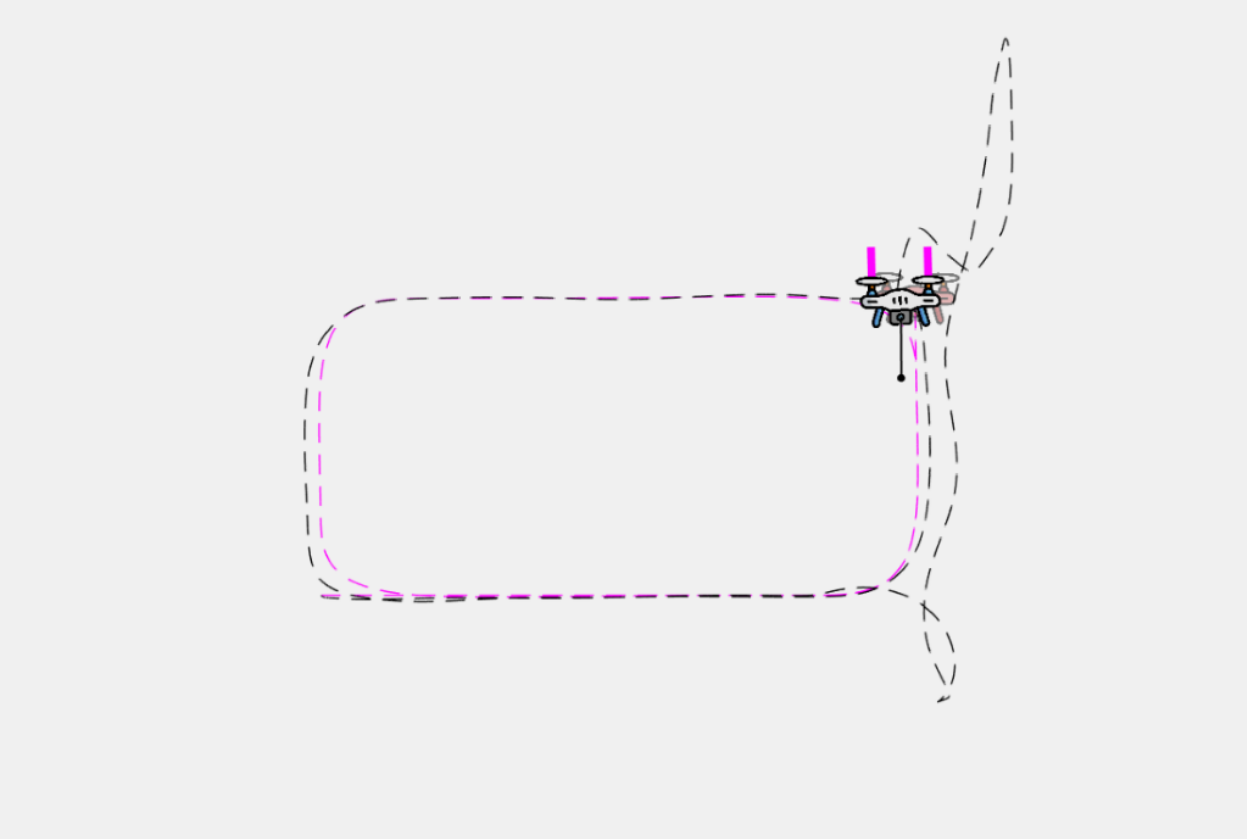
First Tuning





Second Tuning


Imag Axis
Real Axis
Choose sidebar display
4
3
2
1
0
-1
-2
-3
-8
-7
-6
-5
-4
-3
-2
-1
0



Finite State Machine

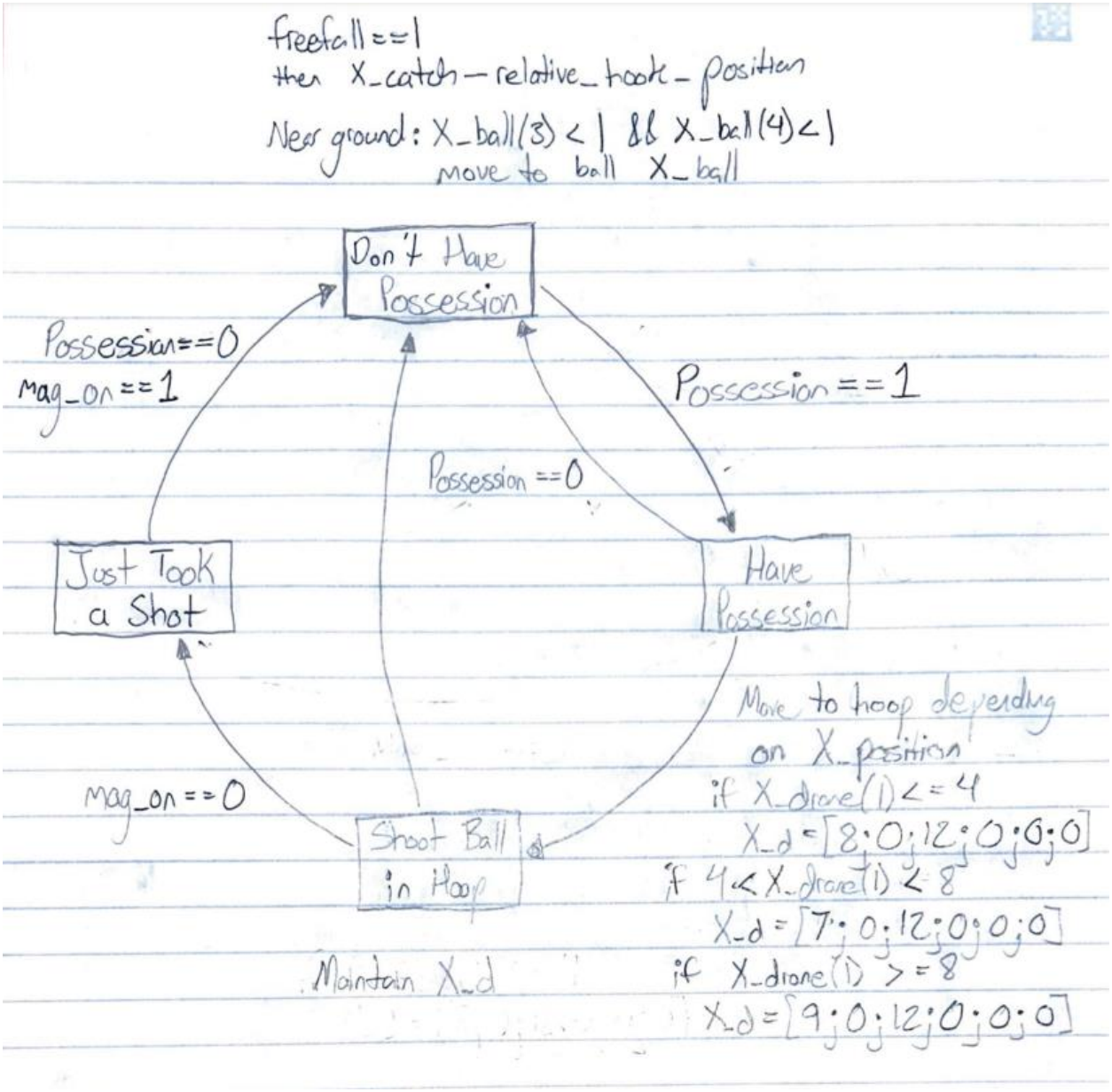


Figure 1: Finite State Machine Diagram.

Ball Release Logic

Projectile Motion Calculations

To determine when the drone should release the ball, we calculated the horizontal time to the hoop and the ball's vertical position at that time. The hoop's position, [8;8], is a fixed and known value, representing its center at x = 8 (horizontal position) and y = 8 (vertical height). This information was given as part of the problem setup and allowed us to consistently calculate the ball's trajectory relative to the hoop. Using the ball's initial position and velocity, we predicted its path based on projectile motion equations. Below is a hand-drawn diagram and step-by-step calculations:

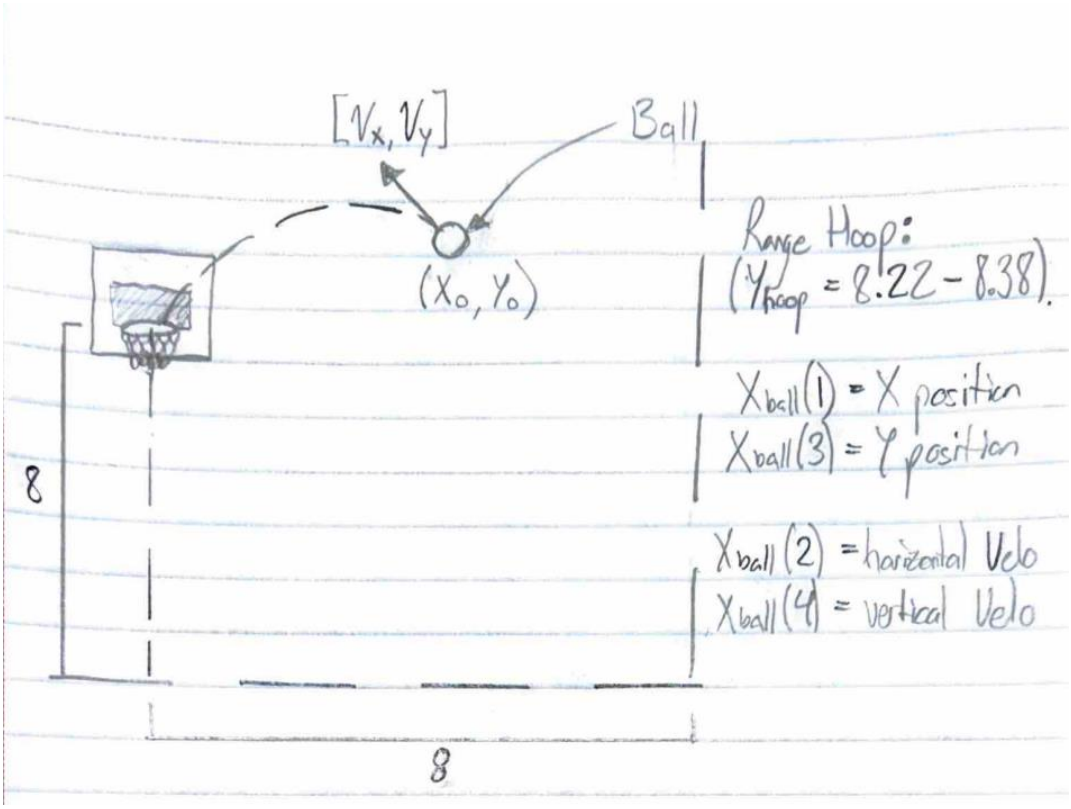


Figure 2: Hand-drawn Diagram.

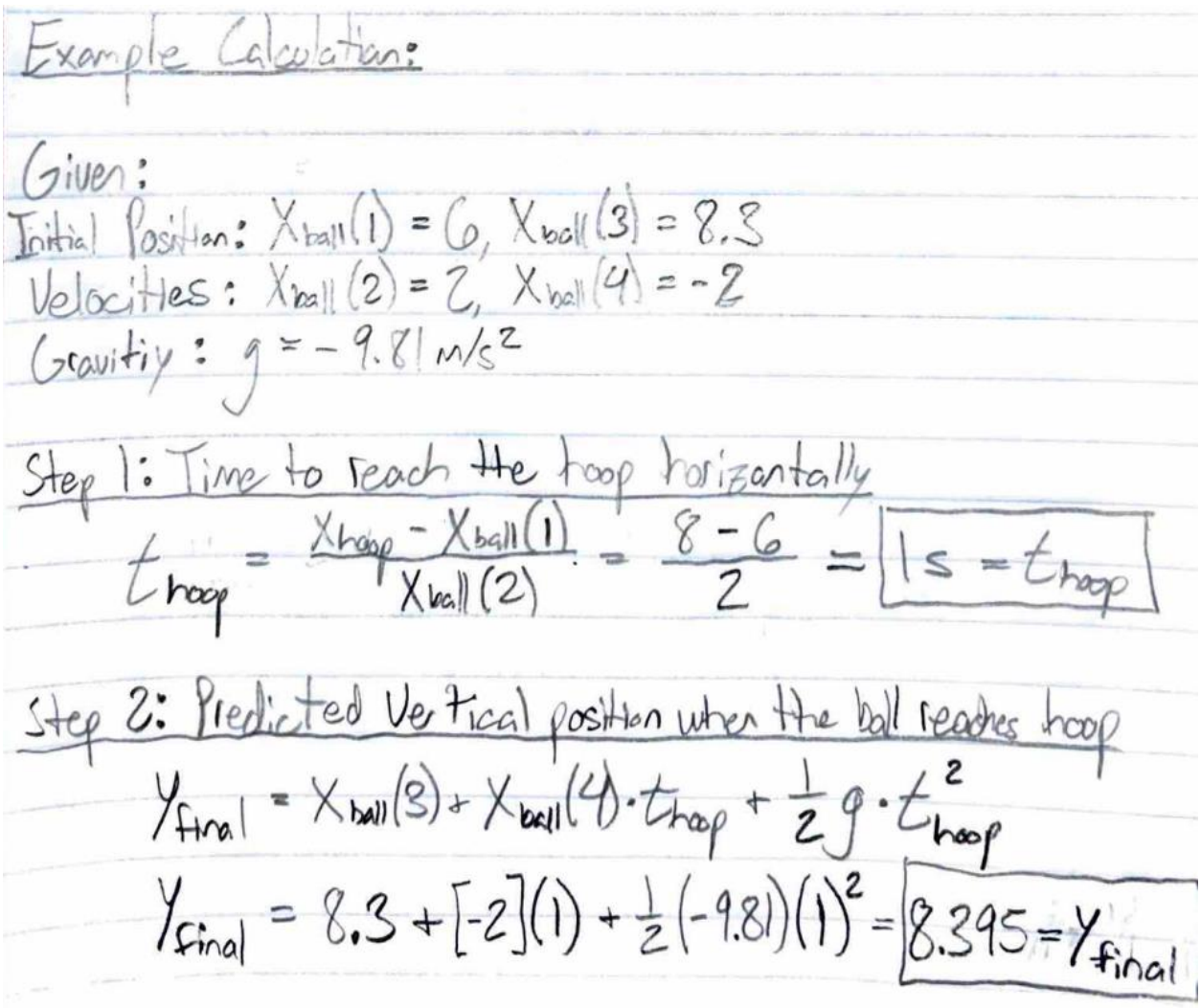


Figure 3: Example Projectile Calculations [1].

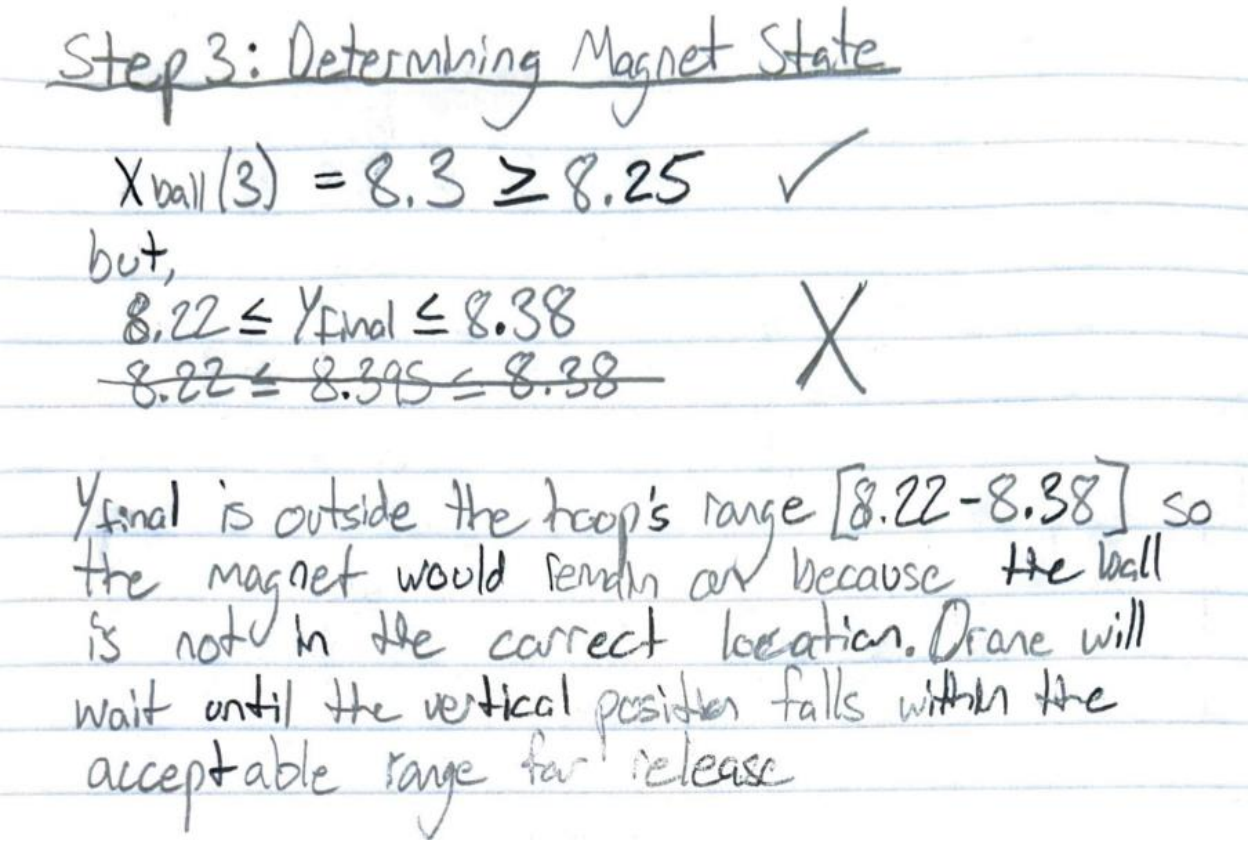


Figure 4: Example Projectile Calculations [2].

Release Conditions Calculations

The function begins by calculating two key factors to determine when the drone should release the ball: the time to reach the hoop and the ball's vertical position at that time.

1. **Horizontal Time to Reach the Hoop:**

   The horizontal time to reach the hoop (t_{hoop}) is calculated by dividing the horizontal distance between the ball's current position and the hoop by the ball's horizontal velocity:

$$t_{hoop} = \frac{8 - X_{ball}(1)}{X_{ball}(2)}$$

   This gives the time it will take for the ball to travel to the hoop's horizontal position at x = 8.

2. **Vertical Position at Time t_{hoop}:**

   The function calculates the vertical position (Y_{final}) at the time the ball reaches the hoop. This is done using the physics equation for projectile motion, which incorporates the ball's initial vertical position, vertical velocity, and gravitational acceleration (g = -9.81 m/s^2):

$$Y_{final} = X_{ball}(3) + X_{ball}(4) * t_{hoop} + \frac{1}{2} g * t_{hoop}^2$$

   Here, $X_{ball}(3)$ and $X_{ball}(4)$ represent the initial vertical position and vertical velocity, respectively. This equation gives the predicted height of the ball when it reaches the hoop.

Conditional Checks in the Code

The code also incorporates conditional checks to ensure the ball is in the correct position to be released:

1. **Ball Above the Hoop:**

   The condition X_ball(3) >= 8.25 ensures that the ball is above the hoop before considering release.

2. **Vertical Position within the Hoop's Range:**

   The condition 8.22 <= Y_final <= 8.38 ensures that the predicted vertical position of the ball is within the acceptable range for scoring.

If both conditions are met, the magnet is deactivated, and the ball is released. If the conditions are not met, the magnet remains on, holding the ball in place.

Magnet Activation or Deactivation

The function then checks if the ball's predicted vertical position at the time it reaches the hoop falls within a valid scoring range. If the vertical position is within the range of $8.22 \leq Y_{final} \leq 8.38$ and the ball is above the hoop at $Y_{ball} \geq 8.25$, the magnet is deactivated to release the ball.

- **Magnetic Deactivation:**

  If the vertical position is within the acceptable range and the ball is above the hoop, the magnet is deactivated:

  ```
  mag_on = 0; % deactivate the magnet to release the ball
  ```

  This ensures the ball is released at the optimal time to score

- **Hold Time Management:**

  The t_hold variable ensures that the magnet will stay deactivated for at least 1 second before it can be reactivated. This helps avoid immediate reactivation after the ball is released:

  ```
  t_hold = t + 1.0; % plan to reactivate the magnet in 1.0 seconds
  ```

```
function [mag_on,t_hold] = fcn(X_ball,t,t_hold_prev)
    % OUTPUTS
    % mag_on: 1 = magnet activated; 0 = magnet deactivated
    % t_hold: time when the magnet will be reactivated

    % INPUTS
    % X_ball: [6x1] state of ball
    % t: simulation time
    % t_hold_prev: output from previous time this function ran

    % constants
    g = -9.81;
    hoop_position = [8; 8]; % define the center of the hoop
    t_hold = t_hold_prev; % keep track of when to reactivate the magnet
    mag_on = 1; % assume the magnet should be on

    % projectile calculations
    time_to_hoop = (hoop_position(1) - X_ball(1))/(X_ball(2));
    y_final = X_ball(3) + X_ball(4)*time_to_hoop+0.5*g*(time_to_hoop)^2;

    % check
    if (X_ball(3) >= 8.25) && (8.22 <= y_final) && (y_final <= 8.38)
        mag_on = 0; % deactivate the magnet to release the bal
        t_hold = t + 1.0; % and plan to reactivate the magnet in 1.0 seconds
    elseif t < t_hold_prev % if the hold time has not yet elapsed
        mag_on = 0; % keep the magnet deactivated
    end
end
```

Figure 5: Ball Release Code.

Two Design Improvements

Finite State Machine: Defense, Improvement 1

To improve the drone's defense, we modified the finite state machine to better handle situations where the opponent has possession of the ball. Instead of the drone chasing the opponent directly, the drone now positions itself between the opponent and the hoop. This adjustment was made to ensure the drone stays in a better defensive position, preventing it from getting too close to the opponent and improving its ability to intercept the ball. The code implements this by calculating a position that is halfway between the ball and the opponent's hoop:

```
% Improvement: when opponent has ball, catch them between hoop and ball instead of chasing the ball
X_d = (X_ball - opponent_hoop_position) / 2 + X_ball - relative_hook_position;
```

This tactic ensures that the drone stays in a more strategic spot rather than blindly chasing the ball. After gaining possession, the FSM transitions to the offense state to try for a shot. Testing showed that this defensive positioning resulted in fewer missed interceptions and more successful transitions to offense.

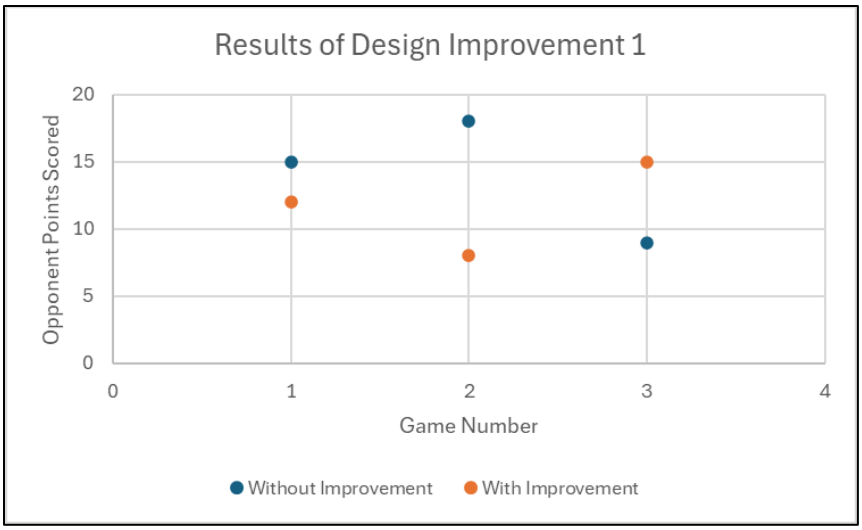


Figure 6: Defensive results of design improvement 1.

```
if state_prev == 1 % don't have possession, try to get possession
    if freefall % i.e., ball is free, fly to estimated intercept point
        X_d = X_catch - relative_hook_position;
    % Improvement: mitigates the glitch when the ball is on the ground by going directly to the ball
    elseif X_ball(3) < 1 && X_ball(4) < 1
        X_d = X_ball;
    else % i.e., opponent has posesition, fly to ball
        % Improvement: when opponent has ball, catch them between hoop and ball instead of chasing the ball
        X_d = (X_ball - opponent_hoop_position) / 2 + X_ball - relative_hook_position;
    end
    if possession % if gain possession
        state = 2; % try to score basket
    end
```

Figure 7: Finite State Machine Improvement 1.

Finite State Machine: Offense, Improvement 2

To improve the drone's shooting accuracy, we adjusted the finite state machine to optimize its target position during shot attempts. Initially, the default height was set to 11 meters, but testing showed better results when increasing it to 12 meters. This change not only improved the shooting percentage by allowing shots from a higher release point but also helped avoid collisions with the basket, especially when the drone retrieved the ball from directly below or slightly beyond the basket. We observed that when the drone attempted a shot too close to the basket, it would often collide with the rim or release the ball incorrectly, resulting in missed

points. To address this, we repositioned the target shot to a slightly shorter distance—2 meters from the basket—providing better clearance and preventing crashes.

The finite state machine now adjusts the drone's x-position based on its location relative to the court:

- **Near the basket** (X_drone(1) <= 4): [8; 0; 12; 0; 0; 0].
- **Mid-range** (4 < X_drone(1) < 8): [7; 0; 12; 0; 0; 0].
- **Beyond the basket** (X_drone(1) >= 8): [9; 0; 12; 0; 0; 0].

These changes ensure proper spacing from the basket and prevent the drone from crashing, while maintaining a high shooting percentage. During State 3: Shooting the Ball, the drone's higher shot height provides a cleaner release, reducing the likelihood of interference with the basket. After the shot, the FSM transitions to State 4: Just Took a Shot (Wait for Rebound)**.** If possession is lost during State 2: Have Possession (Position for Shooting) or State 3**,** it reverts to State 1: Don't Have Possession (Try to Gain Possession)**.** Testing confirmed that these adjustments significantly improved performance without compromising shot accuracy across the court, especially when starting from directly below the basket.

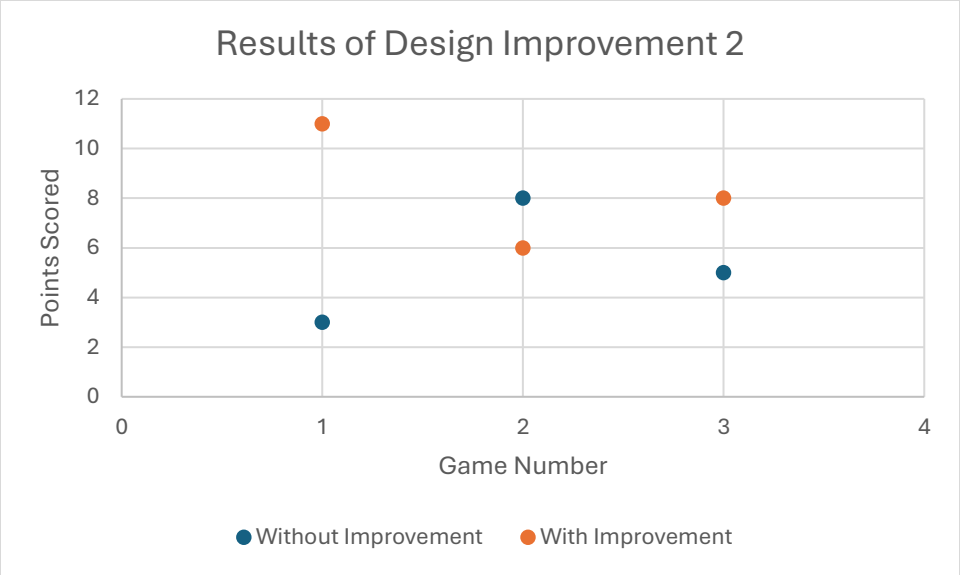


Figure 8: Offensive results of design improvement 2.

```
    % Improvement: prevents drone from running into basket. Gets better set up depending on the position that the drone is at
    % Improvement: desired y position is 12 instead of 11 because the drone makes more shots at a greater height
    elseif state_prev == 2 % have possession, try to score basket
        if X_drone(1) <= 4
            X_d = [8;0;12;0;0;0];
            if mag_on == 0 % if have taken a shot
                state = 3; % wait for possible rebound
            elseif possession == 0 % otherwise, if lose possession
                state = 1; % try to regain possession
            end
        elseif ((X_drone(1) < 8) && (X_drone(1) > 4))
            X_d = [7;0;12;0;0;0];
            if mag_on == 0 % if have taken a shot
                state = 3; % wait for possible rebound
            elseif possession == 0 % otherwise, if lose possession
                state = 1; % try to regain possession
            end
        elseif (X_drone(1) >= 8)
            X_d = [9;0;12;0;0;0];
            if mag_on == 0 % if have taken a shot
                state = 3; % wait for possible rebound
            elseif possession == 0 % otherwise, if lose possession
                state = 1; % try to regain possession
            end
        end

    elseif state_prev == 3 % just took a shot, wait on possible rebound
        X_d = X_ball - relative_hook_position;
        if mag_on % wait for the magnet to reactivate
            state = 1; % try to regain possession (ball may have reset)
        end
    end
end
```

Figure 9: Finite State Machine Improvement 2.