# Control Design: Ball on Plate

Presented by:

Carlos Villasenor

Brandon Briseno

Grant Patterson

Katherynne Estrada

Pachia Vang


MECA 482 Control System Design

Fall 2021: Section 01

Sinan Bank

California State University, Chico

Department of Mechanical and Mechatronics Engineering and Advanced Manufacturing

Chico, CA 95929

December 17, 2021

## A. Introduction

Control system theories are applied to many applications to help improve productivity and enhance today's technology. This report will display the importance and necessary components of control system design through mathematical and simulation modeling. The goal of the project is to develop a comprehensive solution to control a ball's position on a self-stabilizing flat plate. The ball on plate system consists of a plate that will be tilted using two servo motors to control the plate across the x and y directions as well as a camera to act as a sensor. The models provided in this report will not include the sensor. All models and controls are based off of the operational viewpoint in Figure 1. The design is simple but effective. The camera works as a sensor to detect the position of the ball. The rotary servos will be connected to the plate allowing for movement of the plate in all directions. The main support beam, which is stationary, provides support to the platform to ensure a leveled base. The servos will be attached to the base and slightly under the plate to keep the design compact as well as aid in sufficient support for the platform assembly.

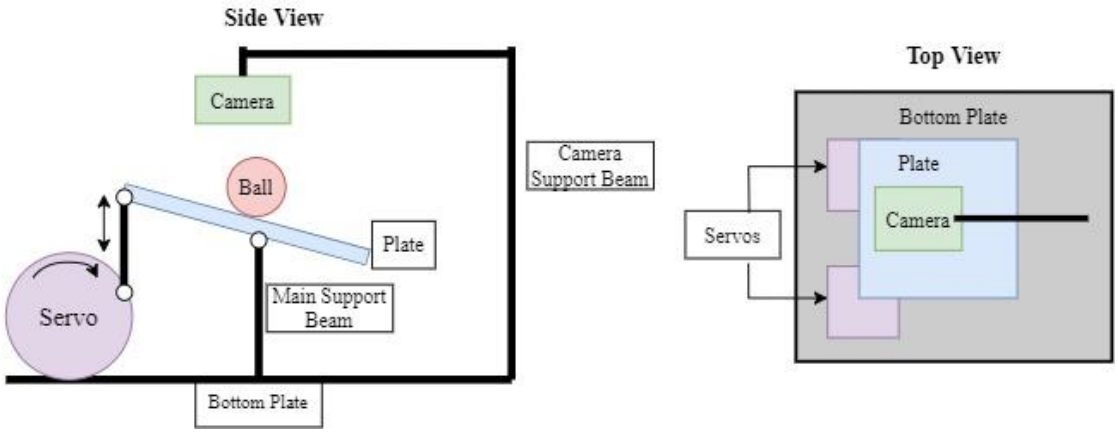



**Figure 1**. Side view and top view of ball on plate system.

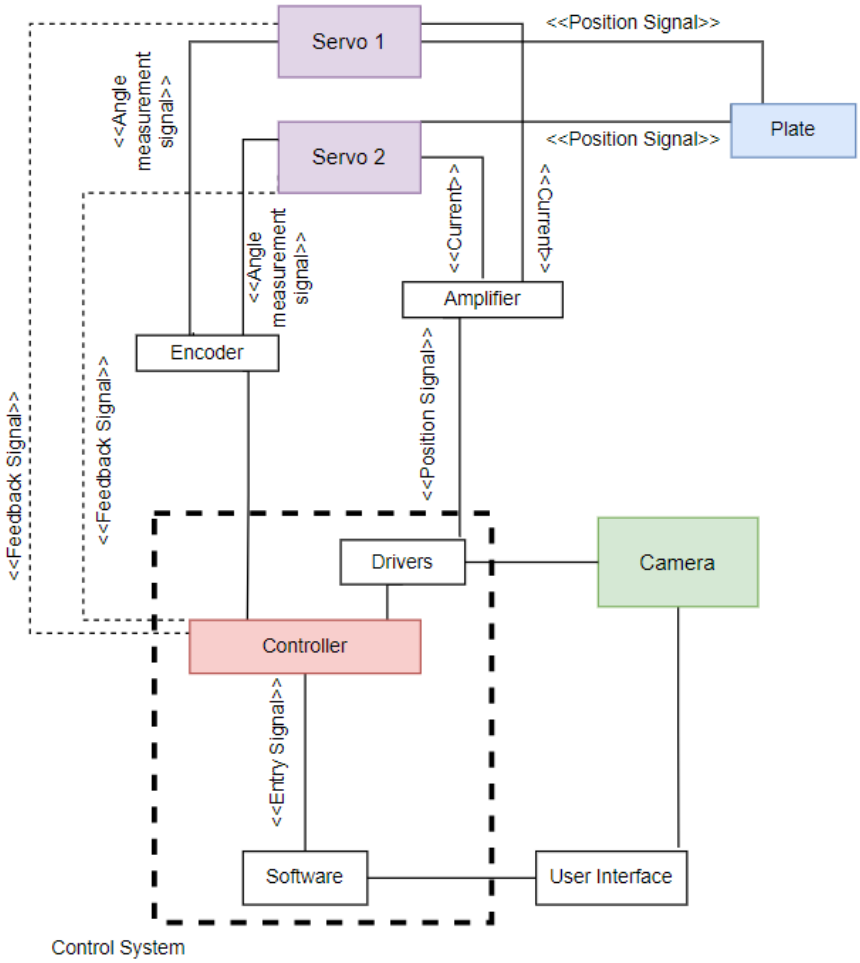



**Figure 2**. Functional/logical diagram of ball on plate system.

## B. Modeling

In this section, the mathematical model of the system will be explained. Figure 3 shows the model from which the motion equations were acquired. The following is assumed of the system:

1. The ball is not slipping.
2. The ball is perfectly symmetrical and smooth.
3. All friction is neglected.
4. The ball and plate are in contact with each other at all times

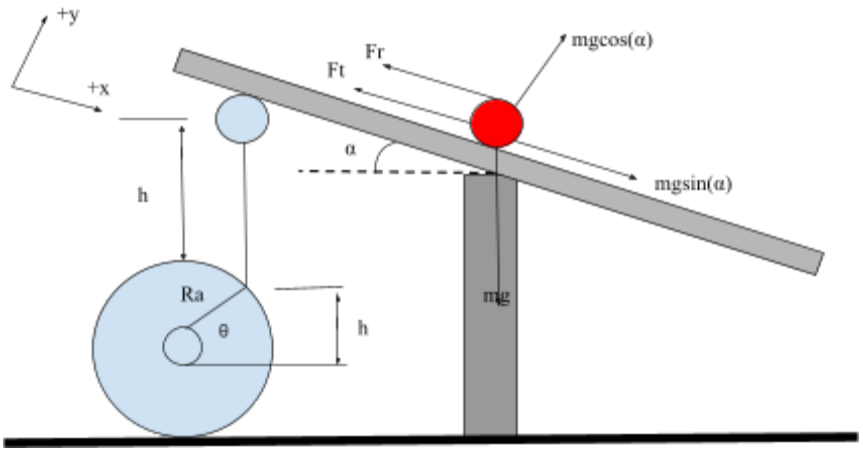



**Figure 3.** Free body diagram of the system.

Based off of Figure 3, the linear acceleration equation was found to be

$$x'' = \frac{R^2(mgsin(\alpha))}{mR^2 + J} \quad \text{Eq. (1)}$$

To find the equation of motion found in Eq (4), Eq (2) and Eq (3) were used to evaluate the magnitude of the linear movement in correlation with the rotational movement.

$$sin(\theta) = \frac{h}{R_a} \quad \text{Eq. (2)}$$

$$sin(\alpha) = \frac{2h}{L} \quad \text{Eq. (3)}$$

Assuming $\theta \leq 10^{\circ}\ to\ 15^{\circ}$ the nonlinear equation of motion was found to be:

$$x'' = \frac{6gR_a\alpha}{5L} \quad \text{Eq. (4)}$$

To find the transfer function P(s), the following equations are needed.

$$\frac{x(s)}{\theta(s)} = \frac{6gR_a\alpha}{5Ls^2} \quad \text{Eq. (5)}$$

$$\frac{\theta(s)}{V(s)} = \frac{K}{s(\tau s+1)} \quad \text{Eq. (6)}$$

By multiplying Eq (5) and Eq (6) together, one can find the final symbolic transfer function.

$$P(s) = \frac{x(s)}{\theta(s)} \cdot \frac{\theta(s)}{V(s)} = \frac{6gR_a\alpha}{5Ls^2} \cdot \frac{K}{s(\tau s+1)} \quad \text{Eq. (7)}$$

$$P(s) = \frac{6gR_a k}{5Ls^3(\tau_s+1)} \quad \text{Eq. (8)}$$

The transfer function in Eq. (8) was used for both motors in the simulation.

In the system transfer function, the variable values were plugged in to obtain the simplified transfer function. Shown in Equation 9 is the final transfer function for the ball on plate system. It is important to note that this function has a gain of 4.68, no existing zeros, and one double pole. Considering this transfer function for the ball on plate system, the poles and zeros must be manipulated in order to achieve a critically damped system behavior as response.

$$P(s) = \frac{4.68}{s^2} \quad \text{Eq. (9)}$$

The fact that the function contains a double pole made it slightly simpler to modify the system to behave as a critically damped system. Since critically damped second order systems must have a damping ratio of exactly 1, and real equal poles, the group used the Root Locus tool from Simulink in order to get the poles that were desired.

**C. Controller Design**

The approach taken to create the Simulink model was utilizing the PID controller block. Figure 4 demonstrates the entire block diagram which consists of the location of the ball in Coppelia, the target position of the ball, PID controller, transfer function derived from Figure 3, an output reference variable, and an oscilloscope. Following the method used in the provided PID videos, the Control System Designer application within Simulink includes a root locus editor that allows the user to manually move the poles and zeros to obtain the desired PID gains. The resulting step response graph is shown in Figure 5. Using

the root locus editor and a step response graph, it was possible to obtain a critically damped situation. The tool utilized can be found in Appendix C.

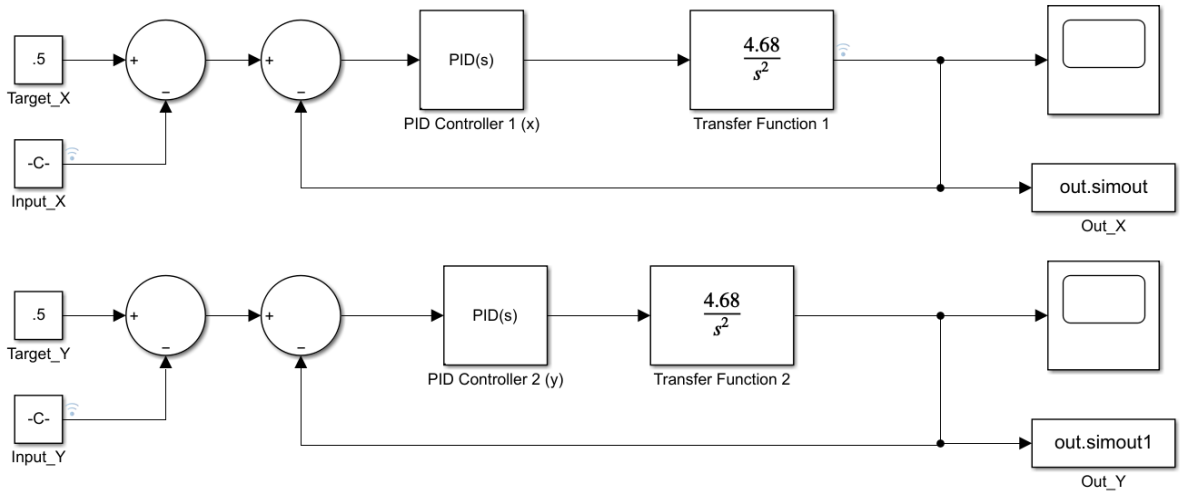



**Figure 4.** Simulink design of the PID controller.

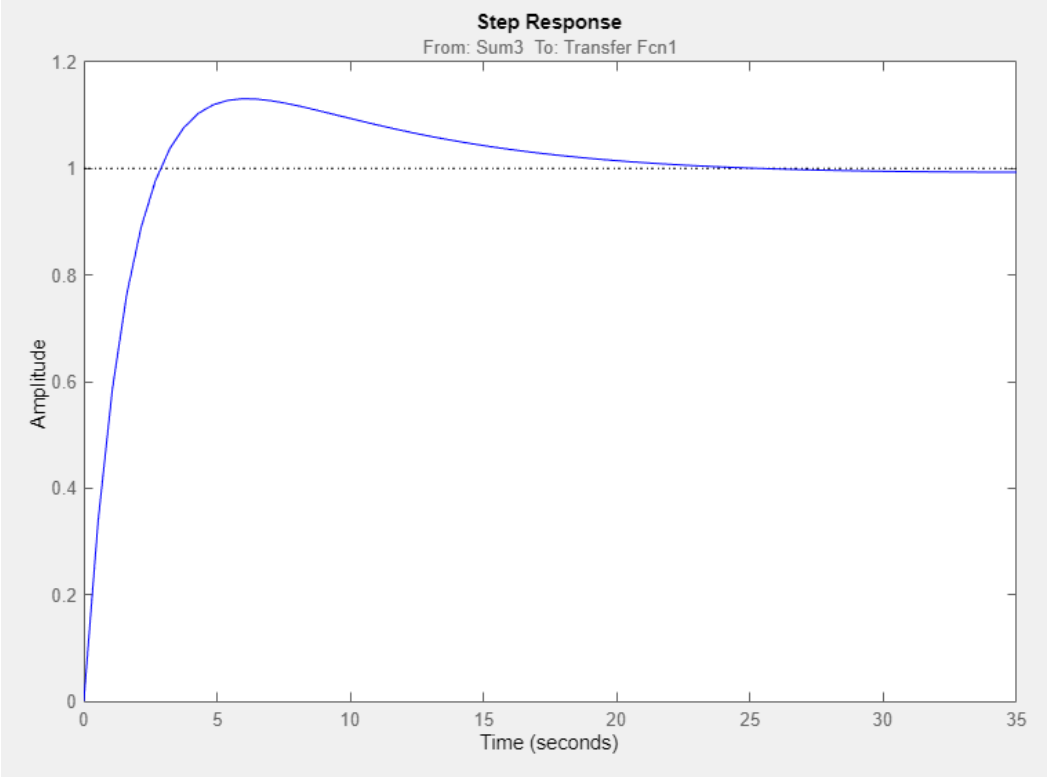



**Figure 5.** Step response from root locus editor.

**D. CoppeliaSim & MATLAB**

The CoppeliaSim model consists of basic geometries such as rectangular prisms, cuboids,and cylinders. The purple cylinders are intended to simulate a motor that controls the yellow rectangular prisms that act as beams connected to the plate. In order to allow for movement a series of joints was utilized. The black center rod has a prismatic joint and a spherical joint towards the top to allow for translation . The motors have rectangular prisms attached to them that serve as arms that then connect to the yellow beams that hold the plate. The motor implementation was done by adding two joints. The first, is a revolute joint that is placed in the center of the motor. The next one, is another revolute joint that connects the motor arm to the beam. A camera vision code was then implemented and allows the camera to sense the position of the ball and send input signals to the motors to adjust their rotation until equilibrium is met. Figure 6 depicts the model in CoppeliaSim, of the ball and plate system while Appendix B-1 contains code for the CoppeliaSim model.

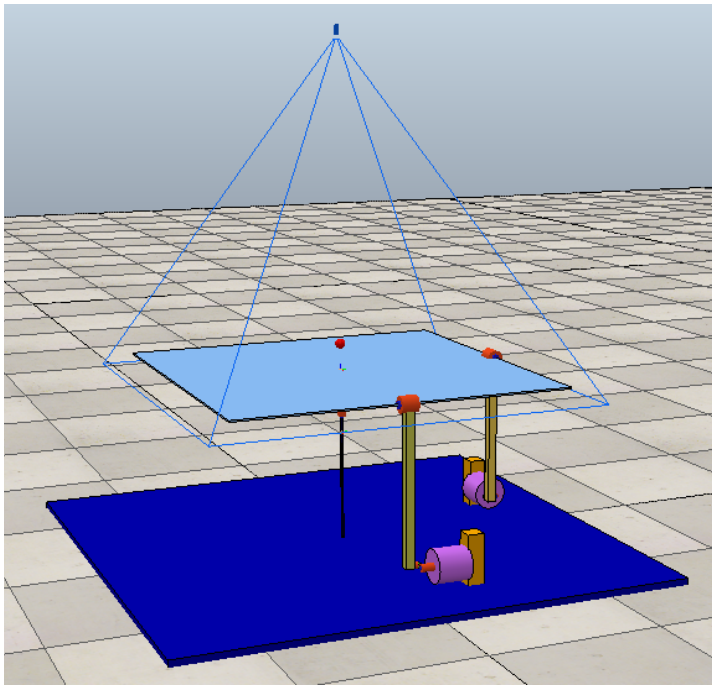


**Figure 6.** CoppeliaSim model of the ball and plate

Using the camera vision code the ball position is sensed using blob detection (see Appendix B-2), and the coordinates are sent to MATLAB. MATLAB then calls the CoordCalc function and stores the

returned values in new variables in order to use the coordinates. Appendix A depicts the MATLAB code implementation.

**E. Controller Implementation**

To tie the components together for simulation, MATLAB code and the Simulink controller were compiled. After starting the CoppeliaSim simulation, the MATLAB script is started shortly after to begin the simulation. The camera reads the position of the ball through code in CoppeliaSim, sends the coordinates to MATLAB which passes the coordinates through Simulink to get the desired motor movement to correct the ball's position. MATLAB takes this desired position and sends it to the servo motors in CoppeliaSim which adjust the plate accordingly to bring the ball back to center. The camera reads the ball's new position after this change, and the cycle repeats.

Unfortunately we could not get our simulation working properly. All programs communicate with each other but the ball simply does not go to the center of the plate. We believe there is an error in the output of our controller as it is supposed to output an angle for the motors.With controller refinement the simulation should properly retain the ball to the center of the plate.

A video of the simulation can be found in the following GitHub repository:

**https://pvang40.github.io/MECA-482-Group-3-Ball-on-Plate/**

**Appendix A: MATLAB Code**

```
clear; clc; close all;
coppelia=remApi('remoteApi');
coppelia.simxFinish(-1);
clientID=coppelia.simxStart('127.0.0.1',19999,true,true,5000,5);
open_system("meca482.slx")

    % PID gains
     P=0.0225594367101972;
     I=0.000899688669450741;
     D=0.158914516635466;

if (clientID>-1)
     disp('Connected to remote API server');
     disp('Starting Simulation')
     set_param('meca482','SimulationCommand','start')

     % Simulation joints
     h=[0,0];

[r,h(1)]=coppelia.simxGetObjectHandle(clientID,'Servo_X',coppelia.simx
_opmode_blocking);

[r,h(2)]=coppelia.simxGetObjectHandle(clientID,'Servo_Y',coppelia.simx
_opmode_blocking);

     while true

[res,retInts,retFloats,retStrings,retBuffer]=coppelia.simxCallScriptFu
nction(clientID,'Cam',coppelia.sim_scripttype_childscript,'CoordCalc',
[],[],[],'',coppelia.simx_opmode_blocking);
     xcoord=retFloats(1);
     ycoord=retFloats(2);

     r_matx=xcoord;
     set_param('meca482/Input_X','Value',num2str(r_matx));
     pause(.005);

     r_maty=ycoord;
     set_param('meca482/Input_Y','Value',num2str(r_maty));
     pause(.005);

     thetaX=get_param('meca482/Out_X','RuntimeObject');
     angleX= (thetaX.InputPort(1).Data*10000);

     thetaY=get_param('meca482/Out_Y','RuntimeObject');
     angleY= (thetaY.InputPort(1).Data*10000);
```

```
coppelia.simxSetJointTargetPosition(clientID,h(1),angleX,coppelia.simx
_opmode_streaming);

coppelia.simxSetJointTargetPosition(clientID,h(2),angleY,coppelia.simx
_opmode_streaming);


     end
else
      disp('Failed connecting to remote API server');
end
coppelia.delete(); % call the destructor!

close_system("meca482.slx")

disp('Program ended');
```

**Appendix B: CoppeliaSim**

### B-1. CoppeliaSim model code

```
simRemoteApi.start(19999)
function sysCall_init()
    sim.handleSimulationStart()
    sim.openModule(sim.handle_all)
    sim.handleGraph(sim.handle_all_except_explicit,0)
end

function sysCall_actuation()
    sim.resumeThreads(sim.scriptthreadresume_default)
    sim.resumeThreads(sim.scriptthreadresume_actuation_first)
    sim.launchThreadedChildScripts()
    sim.handleChildScripts(sim.syscb_actuation)
    sim.resumeThreads(sim.scriptthreadresume_actuation_last)
    sim.handleCustomizationScripts(sim.syscb_actuation)
    sim.handleAddOnScripts(sim.syscb_actuation)
    sim.handleSandboxScript(sim.syscb_actuation)
    sim.handleModule(sim.handle_all,false)
    sim.handleIkGroup(sim.handle_all_except_explicit)
    sim.handleDynamics(sim.getSimulationTimeStep())
end

function sysCall_sensing()
    -- put your sensing code here
    sim.handleSensingStart()
    sim.handleCollision(sim.handle_all_except_explicit)
    sim.handleDistance(sim.handle_all_except_explicit)
    sim.handleProximitySensor(sim.handle_all_except_explicit)
    sim.handleVisionSensor(sim.handle_all_except_explicit)
    sim.resumeThreads(sim.scriptthreadresume_sensing_first)
    sim.handleChildScripts(sim.syscb_sensing)
    sim.resumeThreads(sim.scriptthreadresume_sensing_last)
    sim.handleCustomizationScripts(sim.syscb_sensing)
    sim.handleAddOnScripts(sim.syscb_sensing)
    sim.handleSandboxScript(sim.syscb_sensing)
    sim.handleModule(sim.handle_all,true)
    sim.resumeThreads(sim.scriptthreadresume_allnotyetresumed)

sim.handleGraph(sim.handle_all_except_explicit,sim.getSimulationTime()
+sim.getSimulationTimeStep())
end

function sysCall_cleanup()
    sim.resetCollision(sim.handle_all_except_explicit)
    sim.resetDistance(sim.handle_all_except_explicit)
    sim.resetProximitySensor(sim.handle_all_except_explicit)
```

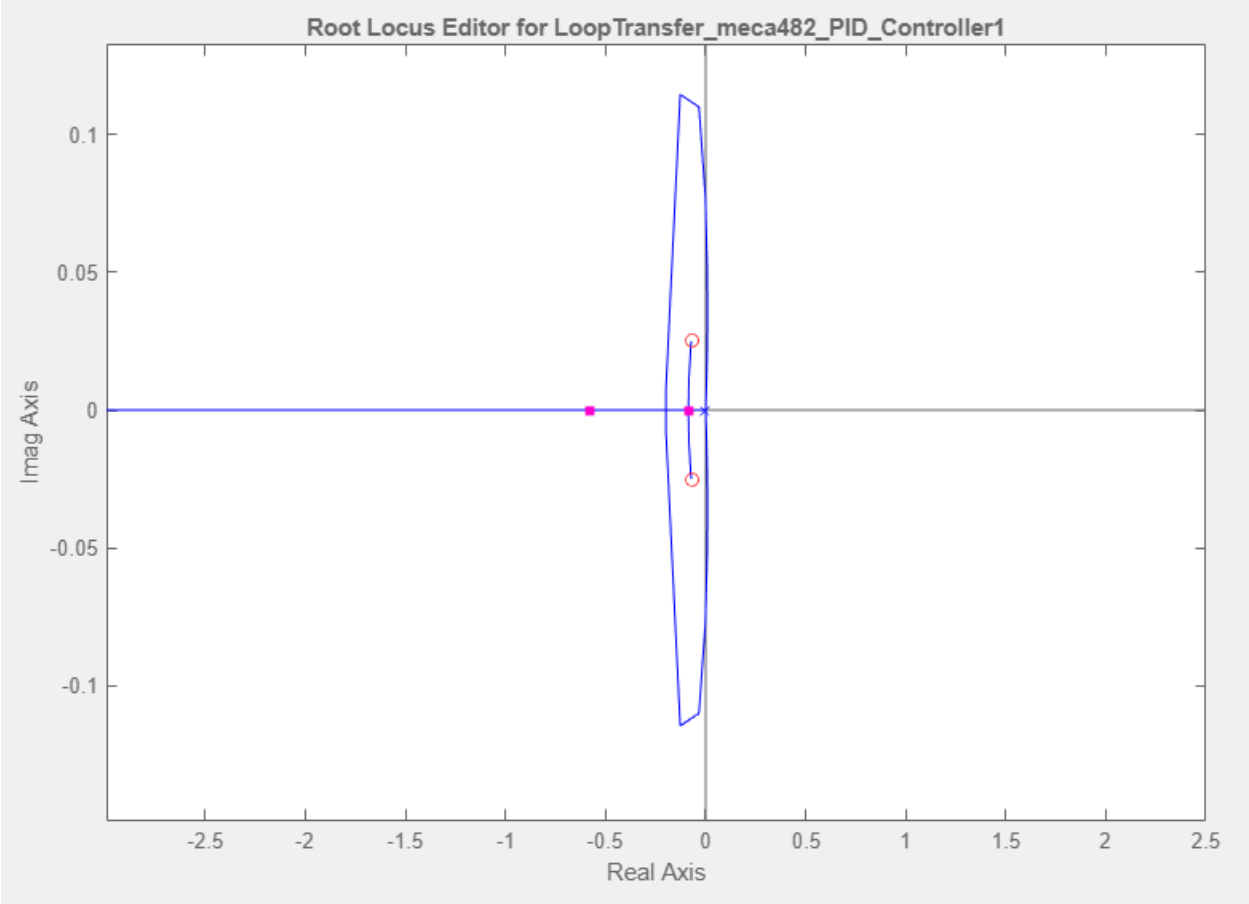
```
    sim.resetVisionSensor(sim.handle_all_except_explicit)
    sim.closeModule(sim.handle_all)
end

function sysCall_suspend()
    sim.handleChildScripts(sim.syscb_suspend)
    sim.handleCustomizationScripts(sim.syscb_suspend)
    sim.handleAddOnScripts(sim.syscb_suspend)
    sim.handleSandboxScript(sim.syscb_suspend)
end

function sysCall_suspended()
    sim.handleChildScripts(sim.syscb_suspended)
    sim.handleCustomizationScripts(sim.syscb_suspended)
    sim.handleAddOnScripts(sim.syscb_suspended)
    sim.handleSandboxScript(sim.syscb_suspended)
end

function sysCall_resume()
    sim.handleChildScripts(sim.syscb_resume)
    sim.handleCustomizationScripts(sim.syscb_resume)
    sim.handleAddOnScripts(sim.syscb_resume)
    sim.handleSandboxScript(sim.syscb_resume)
end
```

**B-2. Camera vision code**

```
function sysCall_threadmain()

    simRemoteApi.start(19999)
    out=sim.auxiliaryConsoleOpen("Debug", 8, 1)
    cam=sim.getObjectHandle("Cam")

    while (true) do

        simVision.sensorImgToWorkImg(cam)
        unused,pack1=simVision.blobDetectionOnWorkImg(cam, 0.1, 0,
false, nil)
        unpack1=sim.unpackFloatTable(pack1,0,0,0)
        xcoord=unpack1[5]
        ycoord=unpack1[6]
        sim.auxiliaryConsolePrint(out,ycoord)
        sim.auxiliaryConsolePrint(out," ")
        sim.auxiliaryConsolePrint(out,xcoord)
        sim.auxiliaryConsolePrint(out,"\n")
        sim.setStringSignal("distance", pack1)

    end
end
```

```
function sysCall_cleanup()
    -- Put some clean-up code here
end

function CoordCalc(inInts, inFloats,inStrings,inBuffer)
    cam1=sim.getObjectHandle("Cam")
    simVision.sensorImgToWorkImg(cam1)
    unused2,pack2=simVision.blobDetectionOnWorkImg(cam1, 0.1, 0,
false, nil)
    unpack2=sim.unpackFloatTable(pack2,0,0,0)
    xcoord1=unpack2[5]
    ycoord1=unpack2[6]
    return {}, {xcoord1,ycoord1}, {}, ''

end
```

**Appendix C: Root locus editor.**



**References**

*2 DOF Ball Balancer*. Quanser. (2021, April 20). Retrieved October 20, 2021, from https://www.quanser.com/products/2-dof-ball-balancer/.

MATLAB. (2018). *Understanding PID Control, Part 1: What Is PID Control?* Retrieved September 2, 2021, from https://www.youtube.com/watch?v=wkfEZmsQqiA.

Nise, “Control Systems Engineering”, Wiley, 2015 7th Ed.