MECA 482: Ball and Plate

May 13, 2020

Sessions: 30
Total Hours worked: 100

Group:
Nicholas Chenevey
Janette Calvillo Solis
Joseph Mount
Jason Fry
Jonathan Okonkwo

Introduction

A controller for a ball and plate system must be designed using Matlab, Simulink and Coppelia. The system is to keep a ball balanced at the center of the plate and must react according to the ball's position to avoid the ball from falling off the plate.

Modelling

The Ball and Plate system being used is represented in Figure 1. The nonlinear equation of motion, Eq (1) was obtained using the diagram [3]. Equations Eq (2) and Eq (3) were also found through the geometry shown in the figure below.

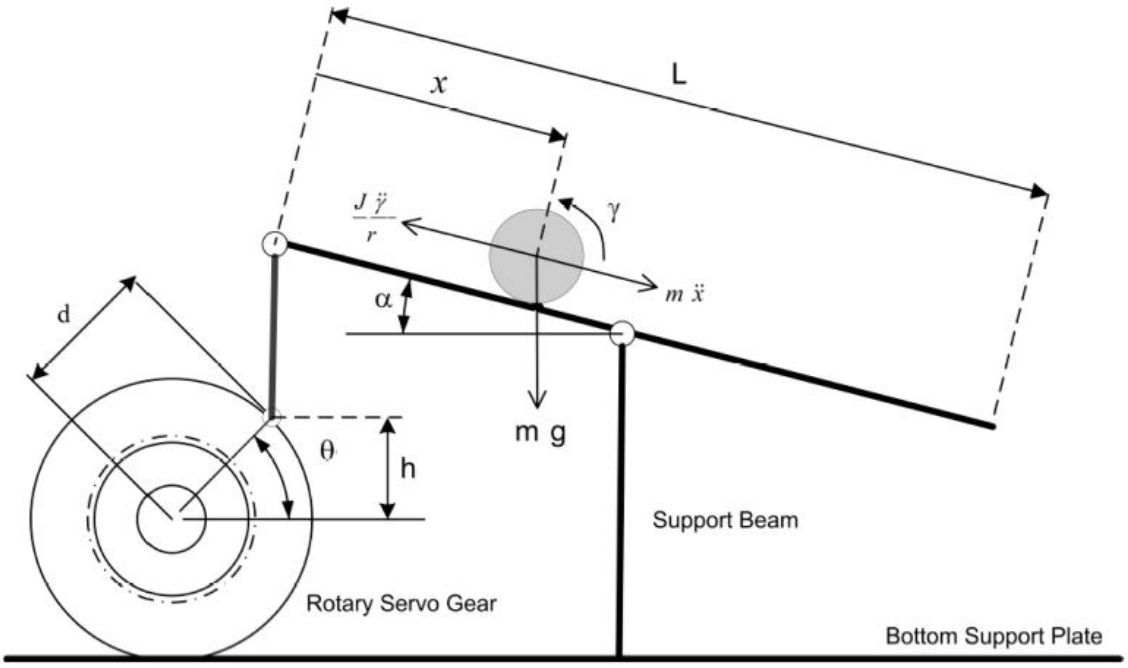

Figure 1. Ball and Plate Diagram

$$m\ddot{x} = mg\sin\alpha(t) - \frac{J\ddot{x}(t)}{R^2} \qquad \text{Eq (1)}$$

$$\sin\alpha(t) = \frac{2h}{L} \qquad \text{Eq (2)}$$

$$\sin\theta(t) = \frac{h}{d} \qquad \text{Eq (3)}$$

$$\ddot{x}(t)\left(m+\frac{J}{R^2}\right)=\frac{2mgd\sin\theta(t)}{L} \qquad \text{Eq(4)}$$

The nonlinear equation of motion shown in Eq (4) was then linearized by $\sin\theta(t) = 0$, thus, $\sin\theta(t)$ can be rewritten as $\theta(t)$ as shown in Eq (5). The transfer function, Eq (7), for one motor was then found by taking the Laplace Transform of the linearized equation, Eq (5). This transfer function equation was used for the second motor as well.

$$\ddot{x}(t)\left(m+\frac{J}{R^2}\right)=\left(\frac{2mgd}{L}\right)\theta(t) \qquad \text{Eq (5)}$$

$$X(s)\left(m+\frac{J}{R^2}\right)s^2=\left(\frac{2mgd}{L}\right)\theta(s) \qquad \text{Eq (6)}$$

$$\frac{X(s)}{\theta(s)}=\frac{2mgd}{s^2L\left(m+\frac{J}{R^2}\right)} \qquad \text{Eq (7)}$$

Controller Design and Simulations

The design was done with the criteria of 5% overshoot and 4% settling time. With this knowledge, Eq (8) and Eq(9) were used to find the natural frequency,w_n, and damping ratio,ζ. Since Root Locus was used to design the controller, the function rlocus(TF) was used in Matlab to obtain the graphs shown in Figure 2, which displays the roots and poles of the transfer function [2]. As shown in the figure, the system contains two poles on the imaginary axis that do not fall within the design requirements of a damping ratio and natural frequency of 0.7 and 1.45 respectively. In order to solve this problem a lead controller was added to shift the root locus allowing the poles to fall within the criteria. The pole for the lead controller was selected to be 0.01 to cancel the pole at the origin and the pole was selected to be 4 through trial since it was the smallest number which shifted the root locus as shown in Figure 3.

$$T_s=\frac{4}{\zeta w_n} \qquad \text{Eq (8)}$$

$$\%\,OS=\left(e^{-\zeta\pi/\sqrt{1-\zeta^2}}\right)100\% \qquad \text{Eq (9)}$$

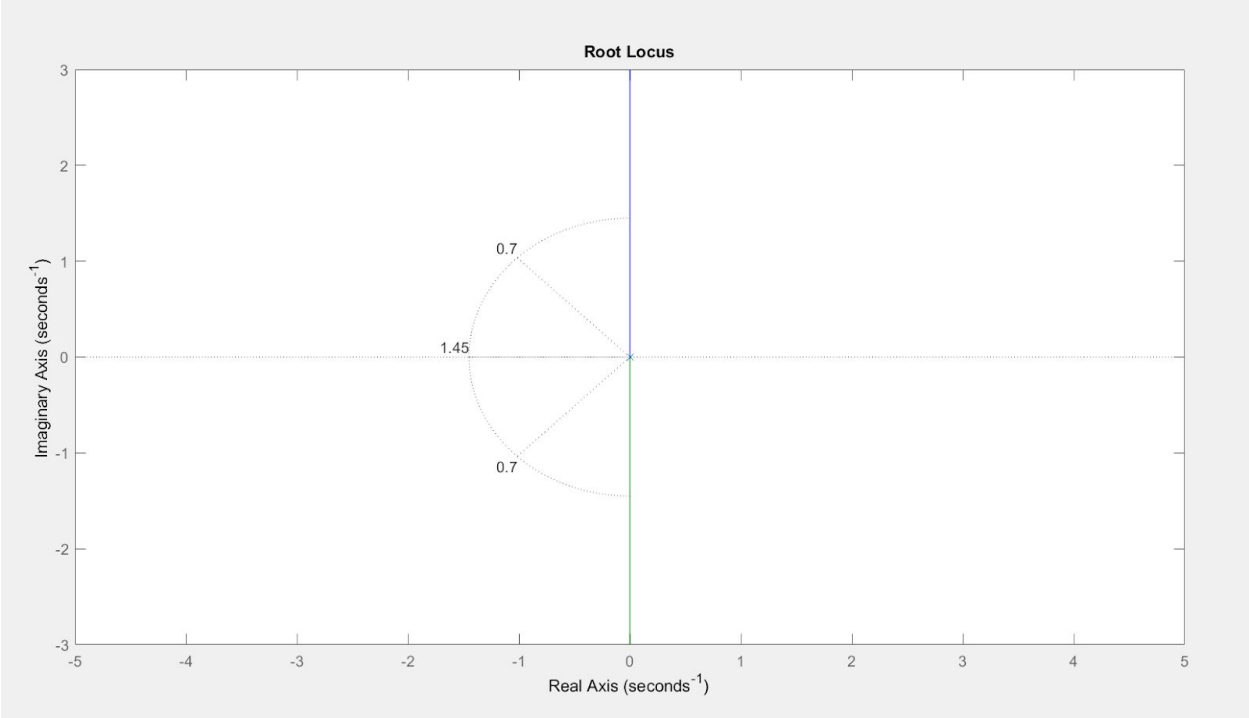

Figure 2. Poles are in the imaginary axis and not within design criteria

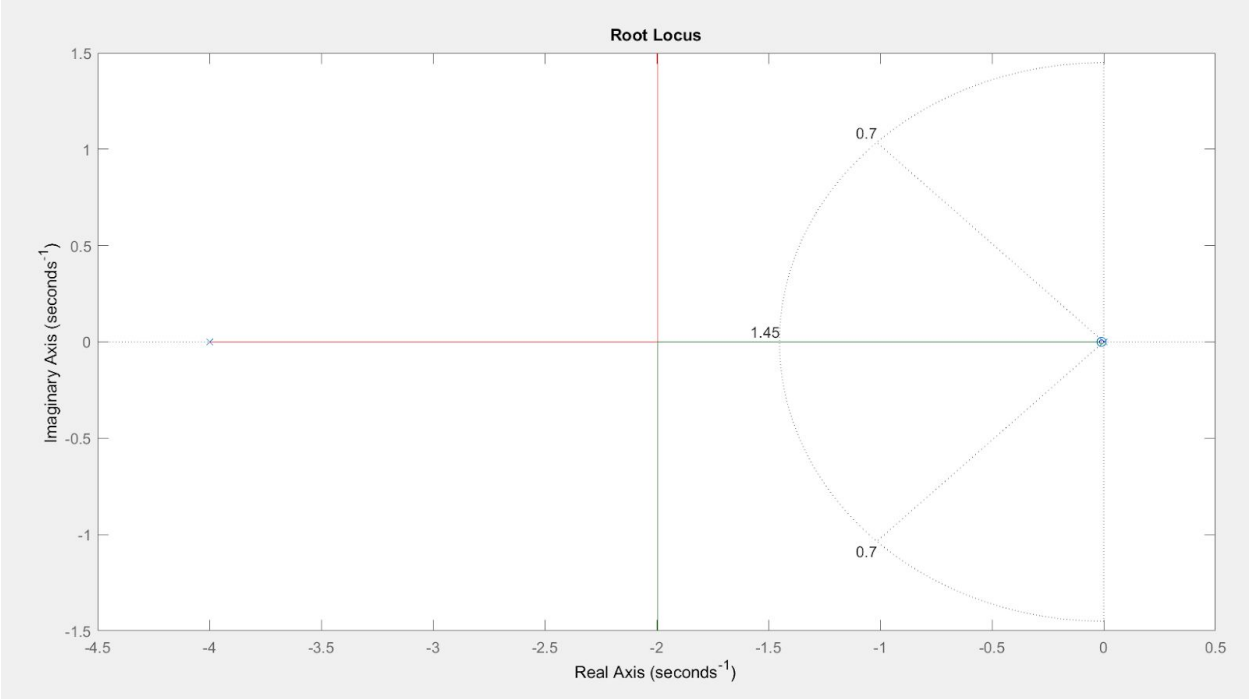

Figure 3. Poles are within design criteria after adding a Lead Controller

After these values were selected the gain was found by using the rlocfind function in matlab. As shown in Appendix A Figure 4, the gain found through Matlab was k = 9.9642

Simulink

Using Simulink, a mathematical model of the ball was developed for use in testing the lead compensator. Appendix A Figure 5 shows the complete simulink block diagram, including the lead compensator and the ball and plate model. After using the lead compensator to produce a target angle, it is passed into the ball and plate model, shown in Appendix A Figure 6, for testing. The model simulates the ball's position and outputs the expected position. Should the angle be correct, the position graph should stabilize at the originally targeted position.

After using the mathematical model of the ball to test the effectiveness of the gain and lead compensator, the Ball and Plate Model block could be removed. The resulting block diagram is displayed in Appendix A Figure 7. There are two separate resulting angles, one for the servo operating the X-axis and another for the servo operating the Y-axis. The first value is the desired coordinate, which then has the current coordinate of the ball subtracted from it. The resulting value is passed through the gain and lead compensator before being converted from degrees to radians. Finally, the resulting targeted angle is sent to the active MATLAB file.

Coppelia & MATLAB

Coppelia was then used with Matlab and Simulink to create a simulation using the Simulink models shown in Appendix A Figure 4 and Figure 5. Appendix A Figure 8 shows the code used by Coppelia [1] to communicate the ball's current coordinates to MATLAB while Appendix A Figure 9 displays the code used in Matlab to communicate with Coppelia.

The code within the while loop is used for testing purposes. The ball's current coordinates are found using the vision sensor's blob detection and then shown on a debug console. To use the coordinates within MATLAB, MATLAB calls the CoordCalc function and stores the returned values in new variables. A video of the simulation can be found in the following GitHub page.
https://github.com/MECA-482-Ball-and-Plate-Project/Ball-and-Plate

Optional: Controller Implementation

Due to COVID-19, the team was unable to implement the code into a real system.

References

[1] Mechatronics Ninja, 2020, “Mechatronics Ninja.” from www.youtube.com/channel/UCX6c67E5PTUq5W8n8POeO_w.

[2] Messner, Tillbury, 2017, “Control Tutorials for MATLAB and Simulink,” ctms.engin.umich.edu/CTMS/index.php?example=BallBeam.

[3] Quanser, 2020, “2 DOF Ball Balancer,” www.quanser.com/products/2-dof-ball-balancer/.

Appendix A: Simulation Code

```
>> Root_Locus
Select a point in the graphics window

selected_point =

  -1.9983 + 0.4060i


k =

    9.9642


poles =

  -1.9950 + 0.4060i
  -1.9950 - 0.4060i
  -0.0101 + 0.0000i
```

Figure 4. Gain, k, given by the Matlab function rlocfind

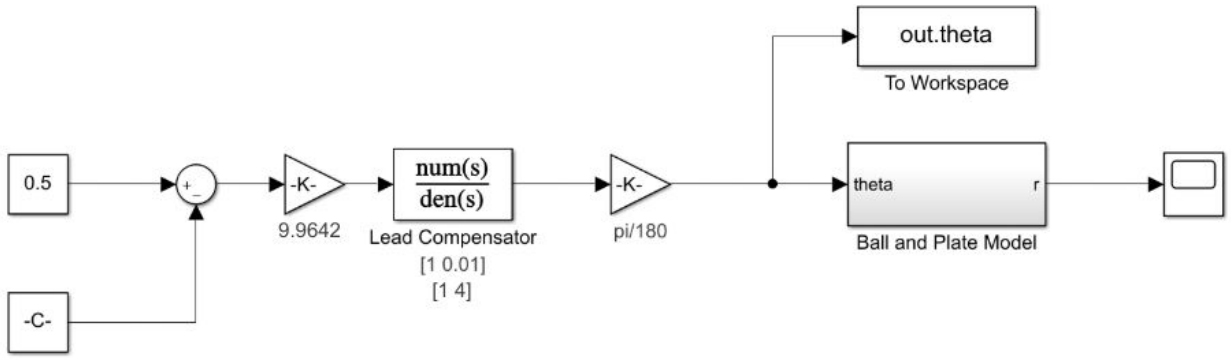

Figure 5. Simulink model

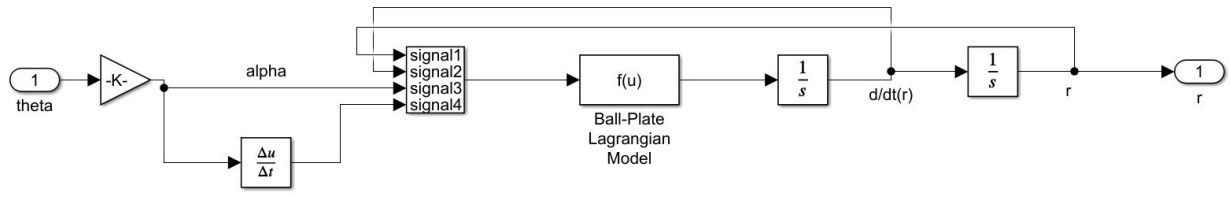

Figure 6. Inside the Ball and Plate block shown in Figure 5

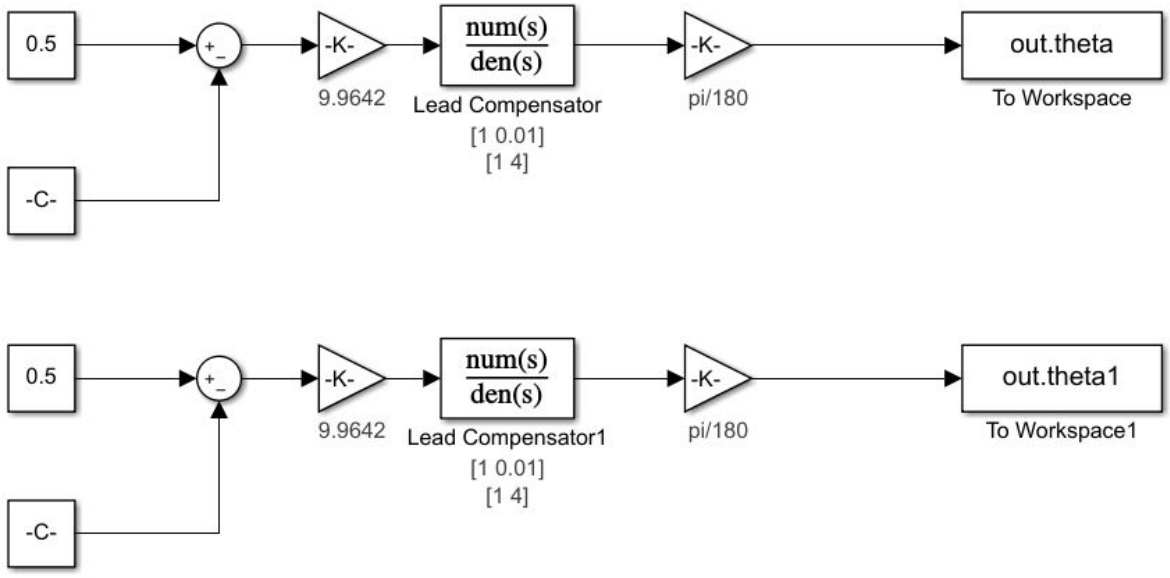

Figure 7. Ball and Plate Simulink Loop

```
function sysCall_threadmain()
    -- Put some initialization code here

    simRemoteApi.start(19999)

    out=sim.auxiliaryConsoleOpen("Debug", 8, 1)

    cam=sim.getObjectHandle("Cam")


    while (sim.getSimulationState()~=sim.simulation_advancing_abouttostop) do
    --     local p=sim.getObjectPosition(objHandle,-1)
    --     p[1]=p[1]+0.001
    --     sim.setObjectPosition(objHandle,-1,p)
    --     sim.switchThread() -- resume in next simulation step
    -- end

    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")

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=unpack1[5]
    ycoord1=unpack1[6]
    return {}, {xcoord1,ycoord1}, {}, ''

end
```

Figure 8. Coppelia Perspective Vision Sensor Threaded Code

```
clear all
close all
clc
coppelia=remApi('remoteApi');
coppelia.simxFinish(-1);
clientID=coppelia.simxStart('127.0.0.1',19999,true,true,5000,5);
if (clientID>-1)
     disp('Connected to remote API server');

     set_param('ball_and_beam', 'SimulationCommand', 'start')

     %joints
     h=[0,0];
        [r,h(1)]=coppelia.simxGetObjectHandle(clientID, 'RotateY0',coppelia.simx_opmode_blocking);
        [r,h(2)]=coppelia.simxGetObjectHandle(clientID, 'RotateX',coppelia.simx_opmode_blocking);

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

     [r,jposx]=coppelia.simxGetJointPosition(clientID, h(2),coppelia.simx_opmode_blocking);
     [r,jposy]=coppelia.simxGetJointPosition(clientID, h(1),coppelia.simx_opmode_blocking);


      r_mat=xcoord;
      set_param('ball_and_beam/Constant','Value',num2str(r_mat));
      pause(.01);

      theta=get_param('ball_and_beam/To Workspace','RuntimeObject');
      angle1= (theta.InputPort(1).Data * 10000);

      coppelia.simxSetJointTargetPosition(clientID,h(2),angle1,coppelia.simx_opmode_streaming)

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

    disp('Program ended');
```

Figure 9. Code used in Matlab to help Coppelia communicate with Simulink