Two Degree of Freedom Ball Balancer

California State University, Chico

Dustin Adams
Max Bauerle
Alex Lopez
Will Powers

MECA 482 – Control Systems Design
May 18, 2020

Department of Mechanical and Mechatronic Engineering and Sustainable Manufacturing
California State University, Chico, CA 95929-0789

Introduction:

Balancing a ball on a plate with two degrees of freedom can be done by stabilizing a hollow ball on a flat plate. This is done by using two rotary servo motors that are attached to the bottom of the plate to control the X-Y position of the ball. The position of the ball is measured by an overhead camera that tracks the position of the ball in real time. A mathematical model of the ball balancer system and a model of a designed PD system are integrated into MATLAB/Simulink. This PD controller measures the amount of error ($X(s) - X_d(s)$), in the system and considers the position of the ball and the velocity of the ball.

Modeling:

To begin, it is important to mathematically model the physical system by creating a transfer function of the kinematics that the system is experiencing at any given time. *Figure 1* shows a free body diagram of one servo motor in the system. Since both servo motors and their effects on the system are nearly identical (just acting on different axes), only one mathematical model is needed.

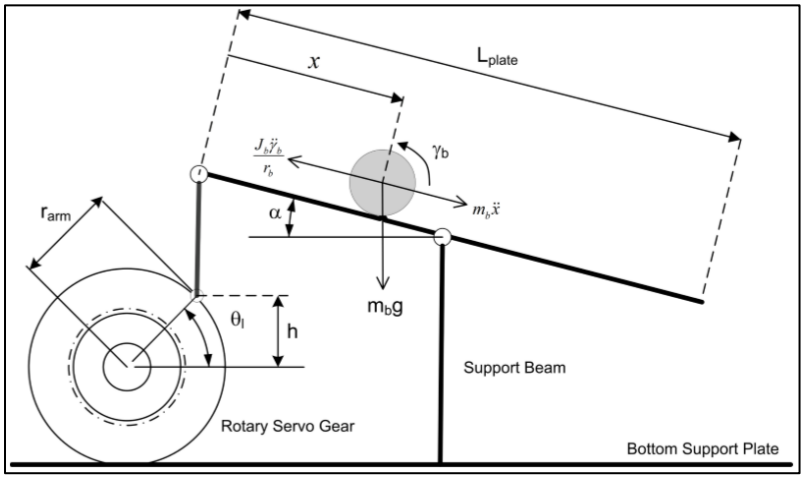

Figure 1: Free Body Diagram of one Rotary Servo Motor acting on the System

To stay stationary on the plate, the gravity acting on the ball must be equal to the force from the ball's momentum. Therefore, the force in the x-direction can be calculated as:

$$F_{x,t} = mg\sin(\alpha(t))$$

The force caused by the torque of the ball on the plate is:

$$F_{x,r} = \frac{\tau_b}{r_b} \quad \text{where} \quad \tau_b = J_b\ddot{\gamma}_b(t)$$

Therefore, the rotational force is:

$$F_{x,r} = \frac{J_b\ddot{x}(t)}{{r_b}^2} \quad \text{where} \quad x(t) = \gamma_b(t)r_b$$

Combining the equations for the force caused by rotation and the force counteracting gravity yields:

$$F_x = m_b\ddot{x}(t) = F_{x,t} - F_{x,r}$$

$$m_b\ddot{x}(t) = m_b g \sin(\alpha(t)) - \frac{J_b\ddot{x}(t)}{{r_b}^2}$$

Solving the equation for linear acceleration gives:

$$\ddot{x}(t) = \frac{m_b g \sin(\alpha(t)){r_b}^2}{m_b{r_b}^2 + J_b}$$

Next, it is important to represent the position of the ball to the angle of the servo motor. This is done by first considering the beam and servo angles that are required to change the height of the beam. Taking the sine of the beam angle and the servo load shaft angle, respectively yields:

$$\sin(\alpha(t)) = \frac{2h}{L_{plate}}$$

$$\sin(\theta_l(t)) = \frac{h}{r_{arm}}$$

Forming a relationship between these two equations gives:

$$\sin(\alpha(t)) = \frac{2r_{arm}\sin(\theta_l(t))}{L_{plate}}$$

To find a linearized equation of motion for the ball's motion and the servo angle, combine the relationship between beam angle and servo shaft load angle with the equation for linear acceleration. Since the equation must be linear, it is important to use the small angle approximation ($\sin(\theta_l(t)) \approx \theta_l(t)$).

$$\ddot{x}(t) = \frac{2m_b g r_{arm}{r_b}^2}{L_{plate}(m_b{r_b}^2 + J_b)}\theta_l(t)$$

Since many of the terms in this combined equation will be constant, the equation can be simplified to yield:

$$\frac{\ddot{x}(t)}{\theta_l(t)} = K_{bb} \quad \text{where} \quad K_{bb} = \frac{2m_b g r_{arm}{r_b}^2}{L_{plate}(m_b{r_b}^2 + J_b)}$$

Now, the moment of inertia for the hollow ball must be applied to the equation of motion. The equation for the moment of inertia of a hollow ball is:

$$J_b = \frac{2}{3}m_b{r_b}^2$$

Applying the moment of inertia to the linearized equation of motion and simplifying yields:

$$\frac{\ddot{x}(t)}{\theta_l(t)} = \frac{6gr_{arm}}{5L_{plate}}$$

It can be deduced that the movement of the hollow ball on the plate does not depend on the size or mass of the ball. Therefore, the constant K_{bb} for a hollow ball can be simplified and equated to be:

$$K_{bb} = \frac{6gr_{arm}}{5L_{plate}} = 1.087$$

Plugging in the calculated value for K_{bb} into the linearized equation of motion and applying a Laplace transform yields:

$$P_{bb}(s) = \frac{x(s)}{\theta_l(s)} = \frac{1.087}{s^2}$$

This calculation and process was done in MATLAB as shown in *Figure 2* and a step plot and pole-zero map were generated as shown in *Figure 3* and *Figure 4*, respectively.

```
%Basic Principals of 2DOFBB
clc
Lt=.275;                    %length of table
g=9.81;                     %gravity
rarm=.0254;                 %radius of arm
mball=.003;                 %Mass of ball
rball=39.25 / 2 / 1000;     %radius of ball
jball=2/3*mball*rball^2;    %Inertia of ball
Tmin=-.523;                 %Theta min
Tmax=.523;                  %Theta max

K_bb = 6/5*g*rarm/Lt;

%Postion Dynamics of Ball w.r.t plate angle
num=K_bb
den=[1 0 0]
Pball=tf(num,den)

pzmap(Pball)
step(Pball) %Plot of position with 1 radian step input
axis([0 6 0 30])
ylabel('Position')
```

Figure 2: MATLAB Code for the Transfer Function

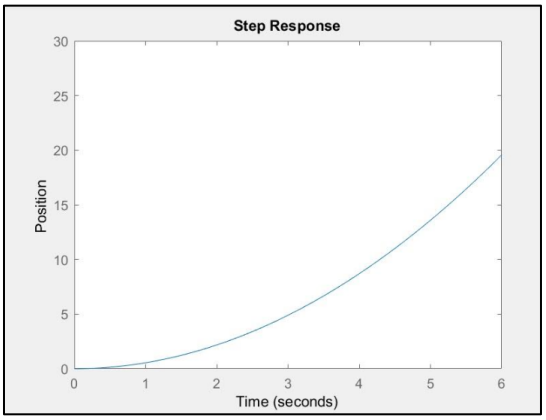

Figure 3: Open Loop Step Response with a 1 Radian Step

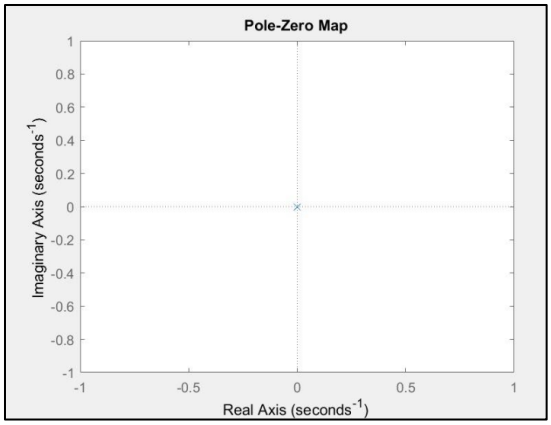

Figure 4: Pole-Zero Map Showing the System will be Unstable

Figure 3 shows that with a step angle input, the ball begins to accelerate and eventually will roll off the plate.

The servo plant system function takes a voltage and converts this to an angle change in radians:

$$P_s(s) = \frac{\theta_l(s)}{V_m(s)}$$

The servo plant function was given as:

$$P_s(s) = \frac{K}{s(\tau s + 1)}$$

Now the servo plant function must be combined with dynamics of the position with respect to servo angle. Here, it is shown that the total plant function P(s) is:

$$P(s) = P_{bb}(s)P_s(s)$$

Where the linearized equation of motion is:

$$P_{bb}(s) = \frac{x(s)}{\theta_l(s)} = \frac{K_{bb}}{s^2}$$

Combining the servo plant function and the linearized equation of motion is simplified to be:

$$P(s) = \frac{x(s)}{V_m(s)} = \frac{K_{bb}K}{s^3(\tau s + 1)}$$

Modeling the total plant function in MATLAB is shown in *Figure 5* and the generated step plot is shown in *Figure 6.*

```
%Tau and k calculation variables
Beq= 15e-3;
Rm=2.6;
eta_g=.9;
eta_m=.69;
km=.804/1000*60/(2*3.14);
kt=1.088*.278*.0254;
Kg=14*5;
jrotor=5.23e-5*.278*.0254
jtach=1e-5*.278*.0254
jm=jrotor+jtach
jg=3*.03*(1.5/2*.0254)^2/2
jext=.5*.04*(.05)^2
jl=jg+jext
Jeq=Kg^2*jm*eta_g+jl
Beq_v = ( Beq*Rm + eta_g*eta_m*km*kt*Kg^2 ) / Rm;% Viscous damping relative to motor
Am = eta_g*eta_m*kt*Kg / Rm;% Actuator gain

K = Am / Beq_v      % Steady-state gain (rad/s/V)
tau = Jeq / Beq_v   % Time constant (s)

nums=K
dens=[tau 1 0]
Ps=tf(nums,dens) %Transfer function for servo plant

P=tf(Pball*Ps)   %Transfer Function for servo plant and ball position
pzmap(P)
Da=20            %Angle input in degress
Ra=Da*pi/180     %Angle input in radians
step(Ra*P)       %Step input at inputed angle in radians
axis([0 7 0 30])
ylabel('Position')
```

Figure 5: Total Plant Function MATLAB Code

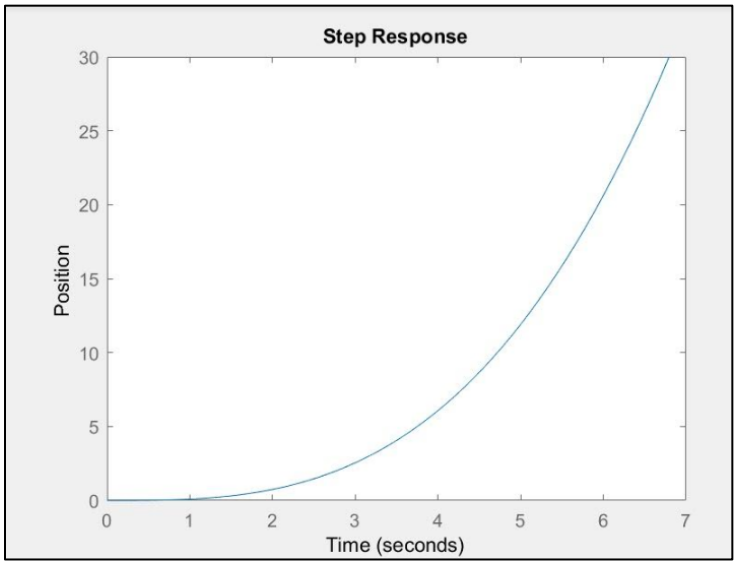

Figure 6: Total Plant Function Step Response of 20 degrees

In the MATLAB code shown in *Figure 5*, τ can be calculated to be .0239 seconds. τ is the time that it takes for the first order system's response to equal 62.3% of the steady state value. Since this number is so small, it can be neglected. Therefore, the servo plant function is negligible since the response is very fast. Thus, we can assume that the desired load angle will equal the actual load angle.

$$\theta_l(t) = \theta_d(t)$$

Controller Design:

Figure 7 below shows the block diagram control design for the Ball Balancer system. The outer ball loop finds the error between the desired position and takes into account the velocity of the ball with respect to time.. The inner loop controls the position of the servo motor using the desired angle.

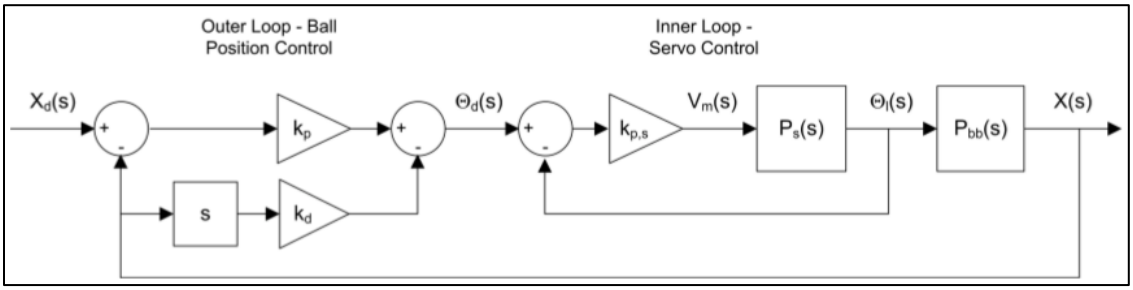

Figure 7: Control System Design for One Axis of the Ball Balancer

Putting this block diagram into an equation form gives the following:

$$\theta_d(s) = K_p\big(x_d(s) - x(s)\big) - K_d s x(s)$$

Solving the linearized equation of motion for $\theta_l(s)$ yields:

$$\theta_l(s) = \frac{x(s)s^2}{K_{bb}}$$

Since the assumption stated earlier that $\theta_l(t) = \theta_d(t)$ these equations can be combined to form the relationship:

$$\frac{x(s)s^2}{K_{bb}} = K_p\big(x_d(s) - x(s)\big) - K_d s x(s)$$

Simplifying the above equation to equate $\frac{x(s)}{x_d(s)}$ gives the transfer function of the PD system:

$$\frac{x(s)}{x_d(s)} = \frac{K_p K_{bb}}{s^2 + K_{bb}K_d s + K_{bb}K_p}$$

This system specifications were designed to achieve a 2% settling time of 3 seconds and a percent overshoot of less than 10%. To achieve this, the general form of a second order system was implemented.

$$\frac{Y(s)}{R(s)} = \frac{{\omega_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2}$$

To begin, the percent overshoot equation can be leveraged to find the damping ratio (ζ):

$$\%OS = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} * 100$$

Which can be rearranged and solved to yield:

$$\zeta = -\ln(\%OS)\sqrt{\frac{1}{\ln(\%OS)^2 + \pi^2}} \quad = \quad .5912$$

Next, the settling time equation is used to find the damping ratio (ω_n):

$$t_s = -\frac{\ln(C_{ts}\sqrt{1-\zeta^2})}{3\omega_n}$$

Which is rearranged and solved to yield:

$$\omega_n = -\frac{\ln(C_{ts}\sqrt{1-\zeta^2})}{3t_s} \quad = \quad 2.32 \text{ rad/s}$$

Now, if the inner loop is considered ideal, the second order form of the system can be assumed to be:

$$\frac{x(s)}{x_d(s)} = \frac{K_{bb}K_p}{s^2 + K_{bb}K_d s + K_{bb}K_p}$$

To solve for K_p and K_d, the following is done:

$$K_p = \frac{{\omega_n}^2}{K_{bb}} \quad = \quad 4.98$$

$$K_d = \frac{2\zeta\omega_n}{K_{bb}} \quad = \quad 2.53$$

Figure 8 shows the MATLAB code used to find the state space representation of the proportional-derivative control.

```
107    %State Space Representation of PD control
108 -  [A,B,C,D]=tf2ss(numc,denc)  %Controller canonical form
109
110
111
112
```

```
Command Window

A =

   -2.7513   -5.4153
    1.0000         0


B =

     1
     0


C =

         0    5.4153


D =

     0
```

Figure 8: MATLAB Code for State Space PD Control

Step response was tested with both proportional control and proportional-derivative control to see the effect of the addition of the derivative componet. *Figure 9* shows the MATLAB code for each.

```
%Step Response with just proportional control
num1=kp*K_bb
den1=[1 0 kp*K_bb]
T=tf(num1, den1)
step(15*T)
axis([0 7 0 30])
ylabel('Position')

%Step response with proportional and derivative control
numc=kp*K_bb
denc=[1 K_bb*kd K_bb*kp]
Y=tf(numc,denc)
step(Y)
ylabel('Position')
```

Figure 9: MATLAB Code for Proportional Control and Proportional-Derivative Control

The MATLAB code shown in *Figure 9* was used to generate graphs to show the output of a step response for each system. The response of proportional control is conveyed in *Figure 10* and the response of proportional-derivative control is shown in *Figure 11*.

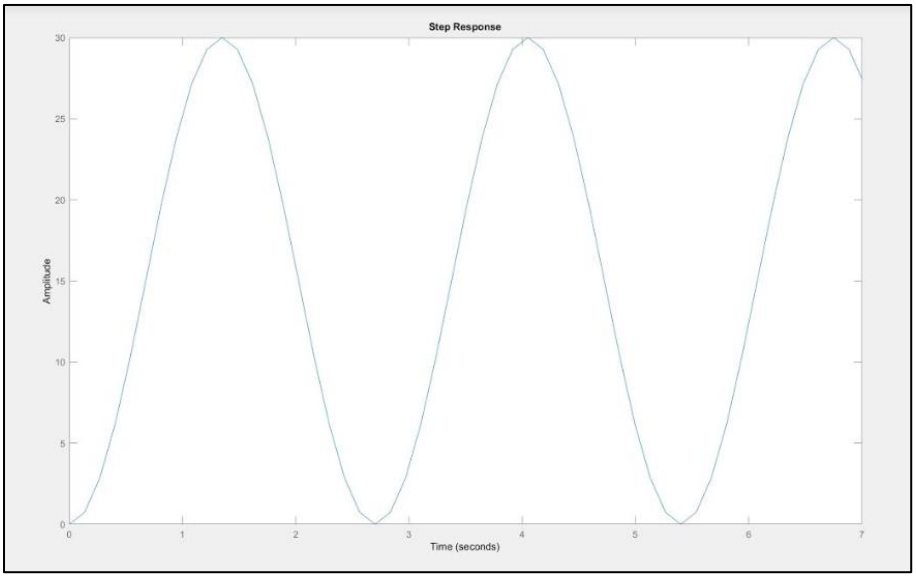

Figure 10: Graph of the Step Response of Proportional Control

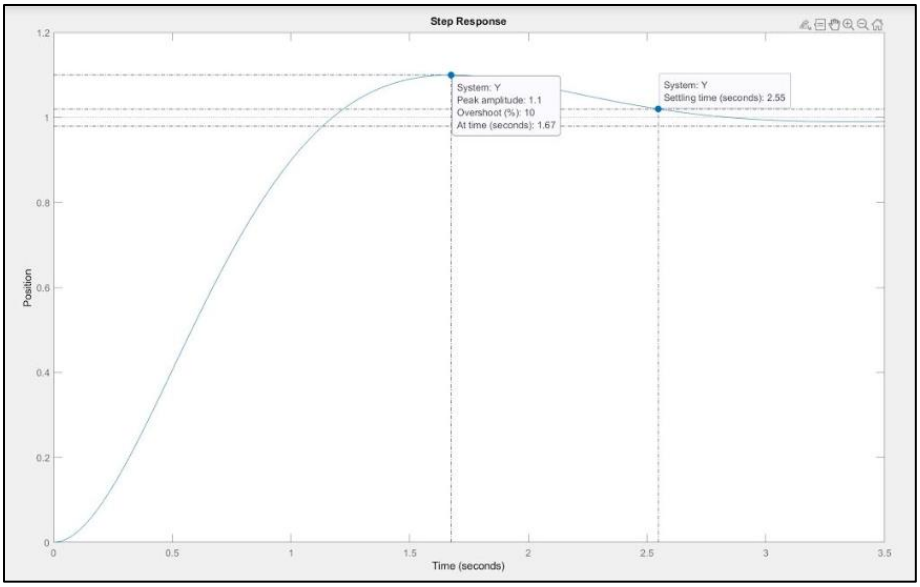

Figure 11: Graph of the Step Response of Proportional-Derivative Control

Figure 10 shows that without the addition of the derivative control that there is no dampening. This is revealed through the oscillations of ball position.

Sensor Calibration:

The overhead camera measures the position of the ball with the origin, (0,0), being at the bottom left. As shown in *Figure 12*, the resolution (which is a user defined variable, res) extends along the x and y axes, which are inverted. A resolution of 200 was used for this design.

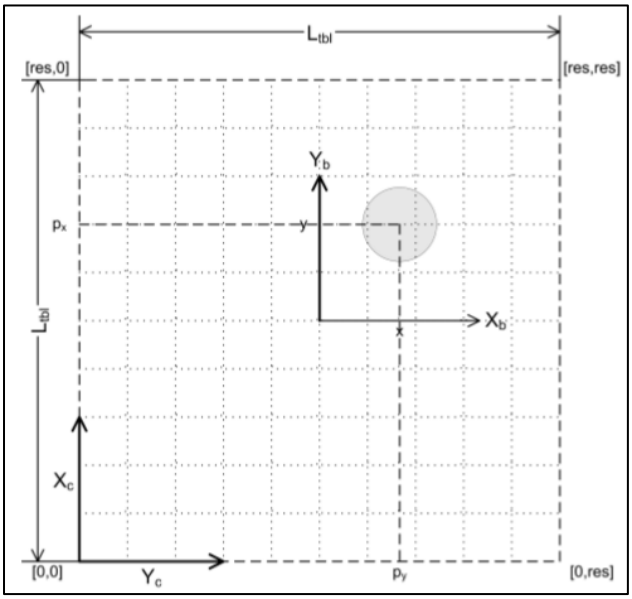

Figure 12: Schematic of the Measurements of the Ball Position Recorded by the Overhead Camera

In order for the controller to appropriately respond to the ball's position, the origin must be in the center of the plate. The coordinates of the ball's position can be changed into this format with the equations shown below.

$$x = L_{tbl}\left(\frac{P_y}{res} - \frac{1}{2}\right)$$

$$y = L_{tbl}\left(\frac{P_x}{res} - \frac{1}{2}\right)$$

In the simulation section, *Figure's 15* and *16* show a test where p_y is recognized by the camera as 190 pixels while p_x is 140 pixels with the resolution set to 200. This translates to an x-position of 12.375 cm and a y-position of 5.5 cm.

Simulation:

The simulation model shown in *Figure 13* was first created in SolidWorks and then uploaded into Coppelia Sim for use. *Figure 14* shows the block diagram for both the X and Y axes that was generated in Simulink.

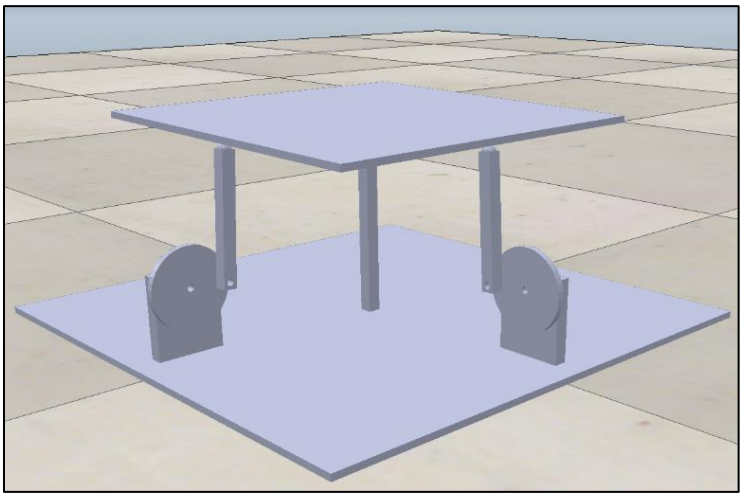

Figure 13: Simulation Model of the Ball Balancer System in Coppelia Sim

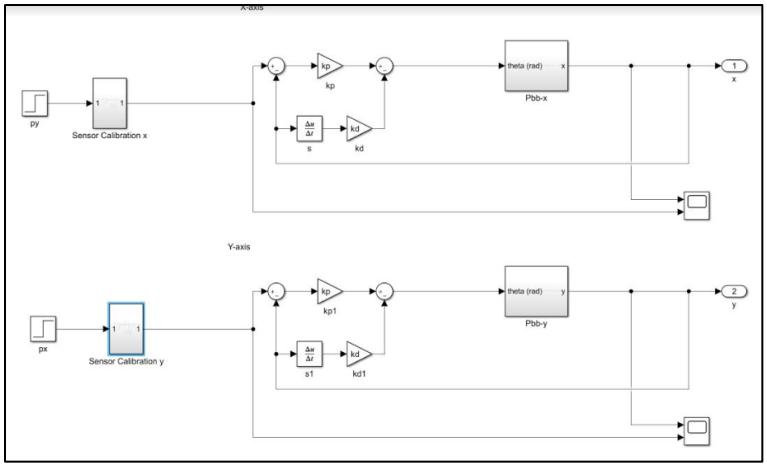

Figure 14: Block Diagram of X and Y Axes, generated in Simulink

Shown below in *Figure 15* is the scope of a x input of 12.375 cm ($p_y = 190$ pixels) at t=2 seconds. *Figure 16* shows the scope of a y input of 5.5 cm ($p_x = 140$ pixels) at t=2 seconds.

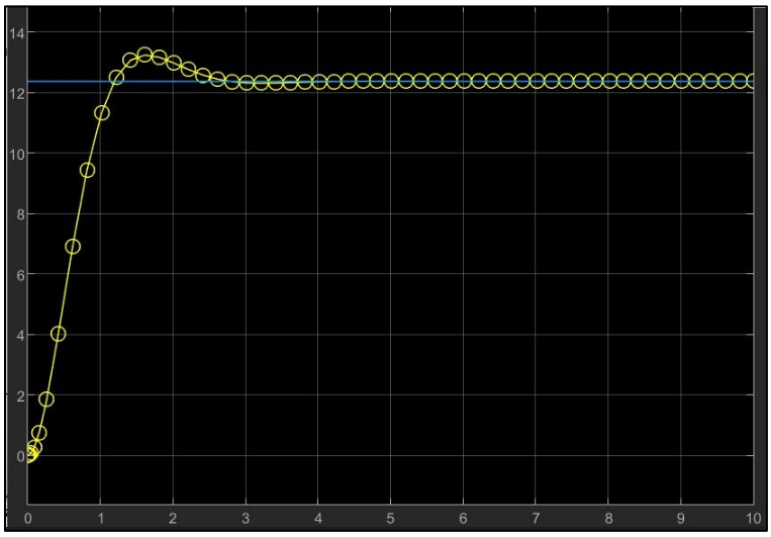

Figure 15: Scope of x input of 12.375cm at t=2 seconds

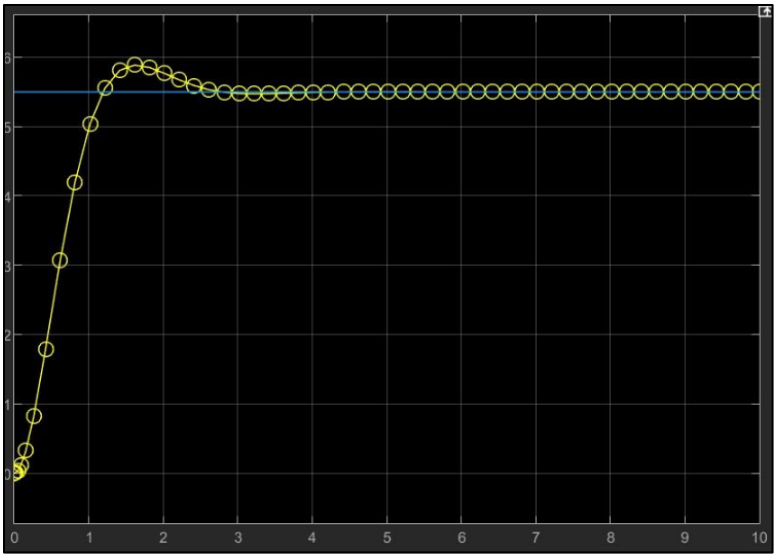

Figure 16: Scope of y input of 5.5cm at t=2 seconds

Simulation testing was done inside Simulink and good results were achieved. Unfortunately, Coppelia Sim proved to be difficult to leverage together with the MATLAB/Simulink codes.

Conclusion:

Proper observation on both the x and y axis were achieved as shown in *Figures 14* and *15*. The results met the previously stated specifications. The Coppelia Sim testing could not be completed, as there were problems with the sensor inputs and joints. The system was modeled in both MATLAB and Simulink, and both measured proper functionality in response to a step input.

References:

2D Ball Balancer Control Using QUARC." *Matlab Instruction Manuel, Quanser Innovate Educate,* https://Nps.edu/Documents/105873337/0/56+-+2D+Ball+Balancer+Control+-+Instructor+Manual.pdf/709c97d2-0fae-426c-9e2a-4b36e8411edf?t=1436282347000

"Ball & Beam: Digital PID Controller Design." *Control Tutorials for MATLAB and Simulink - Ball & Beam: Digital Controller Design,* http://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=ControlDigital.

"2 DOF Ball Balancer." *Quanser,* www.quanser.com/products/2-dof-ball-balancer/.