1. Theory and design

1.1. State space model of the inverted pendulum

In order to demonstrate how an inverted pendulum operates we must implement state feedback control. Starting with the non-linear differential equation that describes the inverted pendulums kinematics, we have.

$$(I + ml^2)\frac{d^2\theta}{dt^2} + \mu\frac{d\theta}{dt} = mgl\ sin\ \theta + ml\frac{d^2x_p}{dt^2}\cos\theta$$

If we linearize the expression above around the unstable equilibria of the pendulum we obtain. The following equation allows us to derive control, if we use cart velocity/acceleration as the control input.

$$(I + ml^2)\frac{d^2\theta}{dt^2} + \mu\frac{d\theta}{dt} = mgl\ \theta + ml\frac{d^2x_p}{dt^2}$$

θ is the angle to the vertical.
μ is the coefficient of viscosity.
m is the mass of the pendulum.
I is the moment of inertia around the centre of mass.
l is the length to the centre of the mass.
x_p is the displacement of the cart.

Writing the equation with the highest state related term on the left-hand side gives.

$$\frac{d^2\theta}{dt^2} = -\frac{\mu}{(I + ml^2)}\frac{d\theta}{dt} + \frac{mgl}{(I + ml^2)}\theta + \frac{ml}{(I + ml^2)}\frac{d^2x_p}{dt^2}$$

As we are using the cart's velocity as the control input, write the acceleration control term on the right-hand side as follows. v_c is the velocity control input.

$$\frac{d^2x_p}{dt^2} = \frac{dv_c}{dt}$$

The coefficients used to represent the constant terms are as follows.

$$a_1 = \frac{\mu}{(I + ml^2)}$$

$$a_2 = -\frac{mgl}{(I + ml^2)}$$

$$b_0 = \frac{ml}{(I + ml^2)}$$

This leads to the equation of dynamics once the above are substituted in.

$$\frac{d^2\theta}{dt^2} = -a_1\frac{d\theta}{dt} - a_2\theta + b_0\frac{dv_c}{dt}$$

The state variables are as follows.

$$x_1 = \theta$$

$$x_2 = \frac{d\theta}{dt} - b_0\frac{dv_c}{dt}$$

$$\frac{d\theta}{dt} = x_2 + b_0 v_c$$

X1 is just the pendulum angle but X2 is the angular velocity along with the time differential of the control velocity term. X2 is then represented with the control and state considered.

$$\dot{x}_1 = \frac{d\theta}{dt} = x_2 + b_0 v_c$$

$$\dot{x}_2 = \frac{d^2\theta}{dt^2} - b_0\frac{dv_c}{dt}$$

Through rearranging XDot2 and substituting these into the dynamics equation you get.

$$\dot{x}_2 + b_0\frac{dv_c}{dt} = -a_1(x_2 + b_0 v_c) - a_2 x_1 + b_0\frac{dv_c}{dt}$$

Then, through cancelling terms and rearranging others, we obtain clearer understandings of the values of XDot1 and XDot2.

$$\dot{x}_1 = 0x_1 + 1x_2 + b_0 v_c$$

$$\dot{x}_2 = -a_2 x_1 - a_1 x_2 - a_1 b_0 v_c$$

This all leads to the following matrix form.

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -a_2 & -a_1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}b_0\\ -a_1 b_0\end{bmatrix} v_c$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

The above matrices correspond to two equations below that show how each individual matrix is used and what for. The output y is the pendulum angle theta.

$$\dot{X} = AX + BU$$

$\dot{X}$ is the time derivative of the state vector, A is the system matrix, X is the state vector, B is the control input matrix and U is the control input.

Below is how I calculated A and B in MATLAB.

```
l = 0.64; %length of pendulum rod (m)
m = 0.314; %mass of pendulum rod (kg)
mu = 0.05; %coefficient of viscous
g = 9.8; %acceleration due to gravity
I = (m*l^2)/12; %pendulum inertia of rod about the midpoint

%writing down the state space model
%these parameters complete these equations
a1 = (mu / (I + m*(l^2)));
a2 = (-m*g*l/(I + m*(l^2)));
b0 = (m*l/(I + m*(l^2)));

%this all leads to the matrix
%treat each matrix as an individual (A, B and C)
A = [0, 1; -a2, -a1];
B = [b0; -a1*b0];
C = [1 0];

%add poles provided
P = [-10, -11];
```

Figure 1: State Space Matrices

```
A =

         0    1.0000
   14.1346   -0.3589


B =

    1.4423
   -0.5176


C =

     1     0


P =

   -10   -11
```

Figure 2: Numeric Evaluation of State Space Matrices

We can see above and too the right what A, B and C equals, I was able to output this by removing the semicolons from the code or using the disp function. The values are obtained using the differential equation to derive the state space model. The coefficients are solved by inserting the parameters provided into the equations. The state space variables themselves allow for analysis of stability, controllability and observability.

$$Y = CX + DU$$

Y is the output, C is the output/observation matrix, X is the state vector, D is the transmission matrix and U is the control input.

1.2. Design a Luenberger observer to estimate state

A Luenberger observer is essentially a state space model of the plant that is used to create an approximation of the full state vector by using the parameters available. The model used is essentially taken from A and C, with the output providing a correction term. The equation used to create the state estimation is as follows.

$$\dot{\hat{X}} = A\hat{X} + BU + L(Y - C\hat{X})$$

L, the observer's gain vector, is found through ensuring the eigenvalues produced (the solutions being lambda) are both negative and real in the characteristics equation below.

$$|(A - LC - \lambda I) = 0$$

We use this feedback controller to reduce the prediction error, which we then use to correct its state estimate. This is done by driving the prediction error towards zero. The stability is determined by the eigenvalues by doing as above. We need to measure the output as X2 cannot be measured itself.

```
%define poles
P = [-10, -11];
%find observers values
L = place(A, C', P); %L is the observer gain
%place function computes a matrix for the gain
```

```
ans =

   20.6411    8.2583
```

Figures 4 and 5: Luenberger Observer and L Values

I have included a screenshot of the numeric values of the Luenberger gain vector (L) just above and to the right. X1 = 20.64, X2 = 8.26. These values could have been obtained by using the Ackermann formula, but MATLAB's place function makes this process much simpler.

1.3. Augment positional state into your state space model

Along with the angle of the pendulum and the angular velocity, I wish to control the cart position so I can bring the cart itself to a standstill where the rod is balanced upright again. In order to keep the cart at its initial position I add a third state (X3). The differential of (X3) is given by the input velocity control signal, due to the control signal itself being the cart velocity.

By integrating the velocity control signal, we can obtain the cart position. In order to get integral feedback, I need to create another state that can compute the integral of the error signal.

Adding this integral feedback by using an integrator to reduce the zero steady state error. Once an output is obtained from the plant, it is compared against the reference input (0). I then take the resulting error, integrate it and then weight it by the integral gain. This is then added to the state feedback.

Finally, the augmented state space system matrices become the following.

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ -a_2 & -a_1 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}b_0\\ -a_1 b_0\\ 1\end{bmatrix} + v_c$$

$$Y = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$$

In my MATLAB code, they now appear as below.

```
A = [0 1 0;
     -a2 -a1 0;
     0 0 0];
B = [b0; -a1*b0; 1];
C = [1 0 0];
```

1.4. Add integral action to the state space model

In order to reduce steady state positional error of the pendulum cart location by adding another state (x4) to compute the integral of the positional error. To obtain this, we augment the matrices by adding another row, assuming the reference input to be 0.

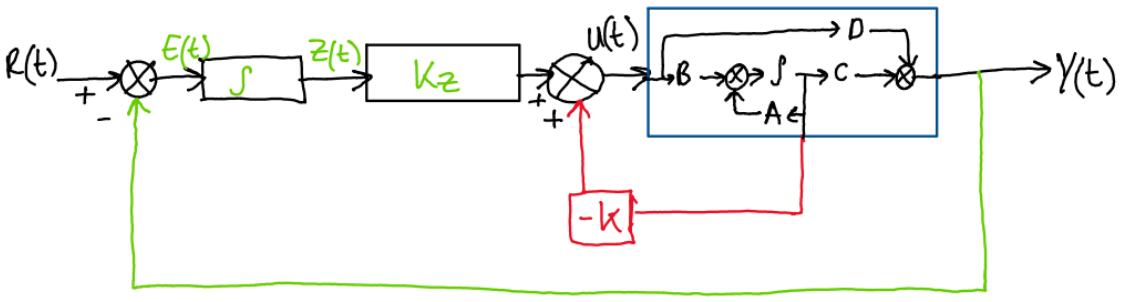

Figure 3: Illustration of the System

State Z is added to compute the integral of the error signal. This error is the difference between the desired output and the actual output that the system obtains. By using this feedback control, we obtain the following equation 1.

$$U = -KX - K_Z Z$$

Equation 2 is standard space equation that leads us to equation 3 being augmented with the new state Z.

$$\frac{d}{dt}[X] = [AX + BU]$$

$$\frac{d}{dt}\begin{bmatrix} X \\ Z \end{bmatrix} = \begin{bmatrix} AX + BU \\ y - r \end{bmatrix} = \begin{bmatrix} AX + BU \\ CX - r \end{bmatrix}$$

From this point on we have four states. The matrices become as follows, (X4) represents the integrated cart position error.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -a_2 & -a_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_0 \\ -a_1 b_0 \\ 1 \\ 0 \end{bmatrix} + v_c$$

$$Y = [1 \ 0 \ 0 \ 0] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

1.5. Design a state feedback controller

In order to implement full feedback state control, I must use the gain vector K as below. This is a result of the closed-loop stability relying on the eigenvalues (lambda).

$$|(A - BK - \lambda I)| = 0$$

This uses the values of X1 and X2 I have previously estimated, these were obtained through the Luenberger observer and are going to be referred to as XHat1 and XHat2.

If I substitute the feedback terms into the previous state space equations I obtain.

$$\dot{X} = AX + BU = AX - BKX = (A - BK)X$$
$$Y = CX + DU = CX - DKX = (C - DK)X$$

In my code, I used the place function again but this time on A and B.

```
%this all leads to the matrix
%treat each matrix as an individual (A, B and C)
A = [0 1 0 0; -a2 -a1 0 0; 0 0 0 0; 0 0 1 0];
B = [b0; -a1*b0; 1; 0];
C = [1 0 0 0];

%add poles provided
P = [-10 -11 -12 -14];
%calculate gain
K = place(A, B, P); %place function to find the gain
disp(K);
```

Figure 6.1: Implementing Pole Placement

The values that I obtained for K are below. The feedback gain is computed as a matrix.

```
 1.0e+03 *

  0.4095    0.1138   -0.4851   -1.3074
```

Figure 6.2: Values of K

I can experiment with the location of the eigenvalues by changing K through pole placement. For example, more aggressive poles will cause the pendulum to be put back to its upright position must faster.

I used my own poles this time, multiplying them all by the same factor to illustrate how their values effect the output of the how the cart moves to recover from a disturbance.

```
%SFC gain: K
P = 4*[-1 -1.1 -1.2 -1.3];
K = place(A,B,P);
disp(K);
%K = [44.11, 12.05, -32.36, -35.82]
%1* is rather slow to stabilise
%3* is much better, allows for a better demonstration
%as it corrects quicker
%5* has the cart travel more but i think it is too aggresive
%has to go back and forth until it stabilises far too much
%4* is inbetween the above, quick to recover and
%still shows plenty of cart movement and a large change in the angle
%of the rod still being balanced
```

Figure 6.3: Exploring Pole Placement

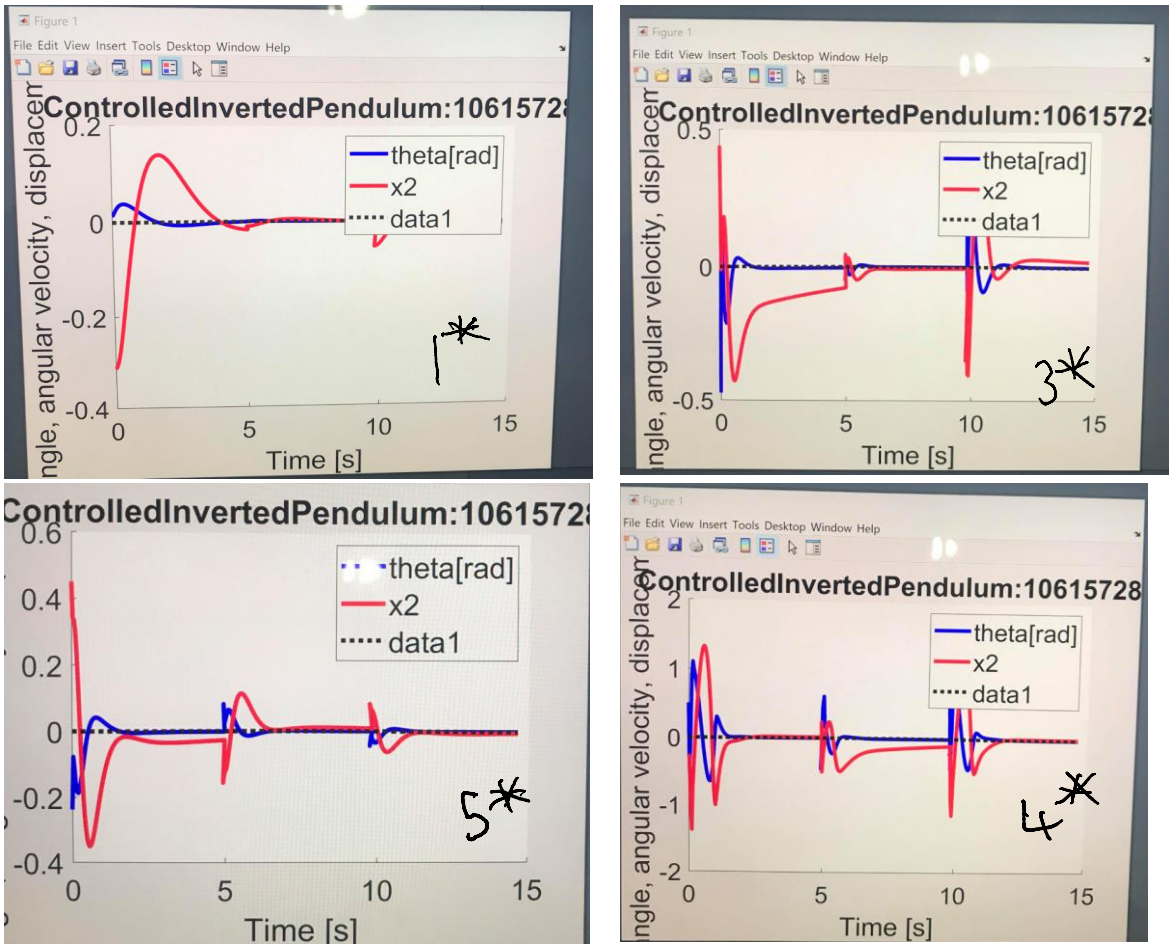

Figure 7: Table of Graphs from Pole Experimentation

1* was far too slow to stabilise, 3* was much better and corrected itself rather quickly. I then jumped to 5* and this was far too aggressive. The cart was travelling more than it needed, which you can see from the graph.

I decided on using 4* in the end, it's between 3* which gave a good demonstration of the cart moving to recover but 5* moved through a much larger range. I feel like settling for in between allows for plenty of movement from large disturbances, but the rod is still balanced relatively quickly by opting for slightly more aggressive poles.

2. Implementation and simulation

2.1. Implement the controller system using Euler integration

```
% compute x1Dot
xDot(1) = x(2)  + b0 * cos(x(1)) * u;

% compute x2Dot
xDot(2) = -a2 * sin(x(1)) - a1 * x(2) + (b0 * sin(x(1)) -a1 * b0 * cos(x(1))) * u;

% compute x3Dot
xDot(3) = cos(x(1)) * u;

% compute x4Dot
xDot(4) = x(3);
```

Figure 8: Updated Function: State Derivatives

Inside CBNLVCPend I added in xDot3 (cart velocity) and xDot4 (cart position, X3) in order to calculate them through calling the function inside StateSpaceIntegrator.

```
len = length(t) - 1; % get signal length
y = zeros(1,len); % init output
xout = zeros(4,len); %updated from 2 to 4

% record the initial state
xout(:, 1, :, :) = x0;
x  = x0;
xHat = x0;

% no control
u = 0;

% for all remaining data points, simulate state-space model
%using C-language compatible formulation
    for idx = 1:len
    tout(idx) = t(idx); % record time
    h = t(idx+1) - t(idx); % get the duration between updates

    %observer estimated state feedback rule to compute U
        %U = -KX
        %this uses sfc gain K and both XHats, X3 and X4
    u = -K(1)* xHat(1) -K(2)* xHat(2) -K(3)*x(3) -K(4)*x(4);

    %real output of the non-linear system
    %y = C*x + D*u;
    %updated as below: for c compatibility
    y(1) = (C(1) * x(1)) + (D(1) * u);
    y(2) = (C(2) * x(2)) + (D(2) * u);
    y(3) = (C(3) * x(3)) + (D(3) * u);
    y(4) = (C(4) * x(4)) + (D(4) * u);
```

Figure 9: Part 1 of the State Space Integrator

As you can see above, I have written the code to be c compatible. I have started the loop in which I have augmented the state feedback controller, through Euler integration. The use of a loop allows for iterative operation. xHat1 and xHat2 are calculated through the Luenberger's estimation.

```
    % calculate state derivative from non-linear equations
    xDot = VCPendDotCB(a1, a2, b0, x, u); %added x(3) and x(4)
    %to this function
    %directly use control velocty from the input
    xHatDot = VCPendDotCB(a1, a2, b0, xHat, u);

    %update the simulated non-linear system state X using
    %Euler integration over time step h
    x(1) = x(1) + h * xDot(1);
    x(2) = x(2) + h * xDot(2);

    %calculate the output from theta and thetaDot state
    %this being X(1) and XDot(1) = X(2)
    y(idx) = C(1)*x(1) + C(2)*x(2) + D(1)*u;

    %update the remaining simulated state X
    %again usuing Euler integration over h
    x(3) = x(3) + h*xDot(3);
    x(4) = x(4) + h*xDot(4);

    %calculate the observer correction term
    ycorr = L*(y-C*xHat);

    xHat(1) = xHat(1) + h * (xHatDot(1)+ycorr(1)); %first state:angle
    xHat(2) = xHat(2) + h * (xHatDot(2)+ycorr(2)); %second state:
    xHat(3) = xHat(3) + h * (xHatDot(3)+ycorr(3)); %third state:position
    xHat(4) = xHat(4) + h * (xHatDot(4)+ycorr(4)); %fourth state:integrated position

    % record the state
    xout(:, idx) = xHat;

    % calculate output from theta and thetaDot state
    y(idx)= (C(1)*xHat(1))+(C(2)*xHat(2))+(C(3)*xHat(3))+(C(4)*xHat(4))+(D(1)*u);

    end
end
```

Figure 10: Parts 2 and 3 of the State Space Integrator

The new positional state (x3) has been computed using integration over the specified time step h. I then calculate the observer correction term using all the values I have computed. This is again done in a c style. I record the state and then calculate the output.

```
% consider the following parameters and equations
l = 0.64;
m = 0.314;
u = 0.05;
g = 9.81;
I = (1/12)*m*l^2;

b0 =  m*l/(I+m*l^2);
b1 = 0;
a0 = 1;
a1 = u/(I+m*l^2);
a2 = -m*g*l/(I+m*l^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% setup time points
dt =  0.04;
%changed from 0.04
Tfinal = 5;
t = 0:dt:Tfinal;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build linearized state space matrices from differential equation
% for velocity control
A = [0   1  0  0;
     -a2 -a1 0  0;
      0   0  0  0;
      0   0  1  0];
B = [b0; -a1*b0; 1; 0];
C = [1 0 0 0];
D = [0 0 0 0];

    %Luenberger observer: L
% A = [0 1; -a2 -a1];
% C = [1 0];
%I used these matrices to calculate L: less values
% P = [-10 -11]; %define only 2 poles
% L = place(A, C', P); %ensure use of inverse of C
% disp(L); %produces 20.69 9.45
L = [20.69; 9.45; 1; 1]; %add ones to match matrix sizes

    %SFC gain: K
P = 4*[-1 -1.1 -1.2 -1.3]; %define 4 poles
K = place(A,B,P); %place function to compute gain
disp(K);
%K = [44.11, 12.05, -32.36, -35.82]
```

Figures 11 and 12: Parts 1 and 2 of Controlled Pendulum Animation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
titleMessage = 'ControlledInvertedPendulum:10615728';
disp(titleMessage)

% initialize arrays
xData = [];
yData=[];
tData=[];
kickFlag=[];

% for sub-loop runs
runs = 3;
%less runs, easier to test
for kick=1:runs

    % for each run randomly perturb intial condition
    x0 = [0; 1 * (rand - 0.5); 0; 0];

    % run Euler integration
    [y, t, x] = StateSpaceIntegrator(@CBNLVCPend, a1, a2, b0, C, D, K, L, t, x0);

    % get time
    newTime = (kick-1) * t(end) + t;

    % just show kick arrow for short time after kick
    frames = length(t);
    kickFlagK = zeros(1,frames);
    if(x0(2) > 0)
        % scale arrow to size of kick
        kickFlagK(1: floor(frames/4)) = -abs(x0(2));
    else
        % scale arrow to size of kick
        kickFlagK(1: floor(frames/4)) = abs(x0(2));
    end
```

```
% plot out the state variables
PlotStateVariable2x2(xData, tData, titleMessage);

% for all time point animate the results
figure
range=1;

% cart not moving so set distance to zero
distance = xData(3,:); %this has been updated to xData

% use animate function: updated xData(1,:)
AnimatePendulumCart((xData(1,:)+pi), distance, l, tData, range, kickFlag, titleMessage);
```

Figures 23 and 14: Part 3 and 4 of Controlled Pendulum Animation

Inside the main file, I add my own paremeters and compute L and K again. I took these values from previous tasks as I already knew how those poles effected the outcome. For the sake of testing, I reduced the number of runs so I could watch through the animation faster and more efficiently. I changed distance to xData as the cart now needs to move. I made a change to x0, adding 2 more values to make the matrices sizes match.

2.2. Run the MATLAB simulation

With the number of runs set back up to 10, I screen recorded the full animation and uploaded it to the following link.

https://youtu.be/mHgj3-hxrNU

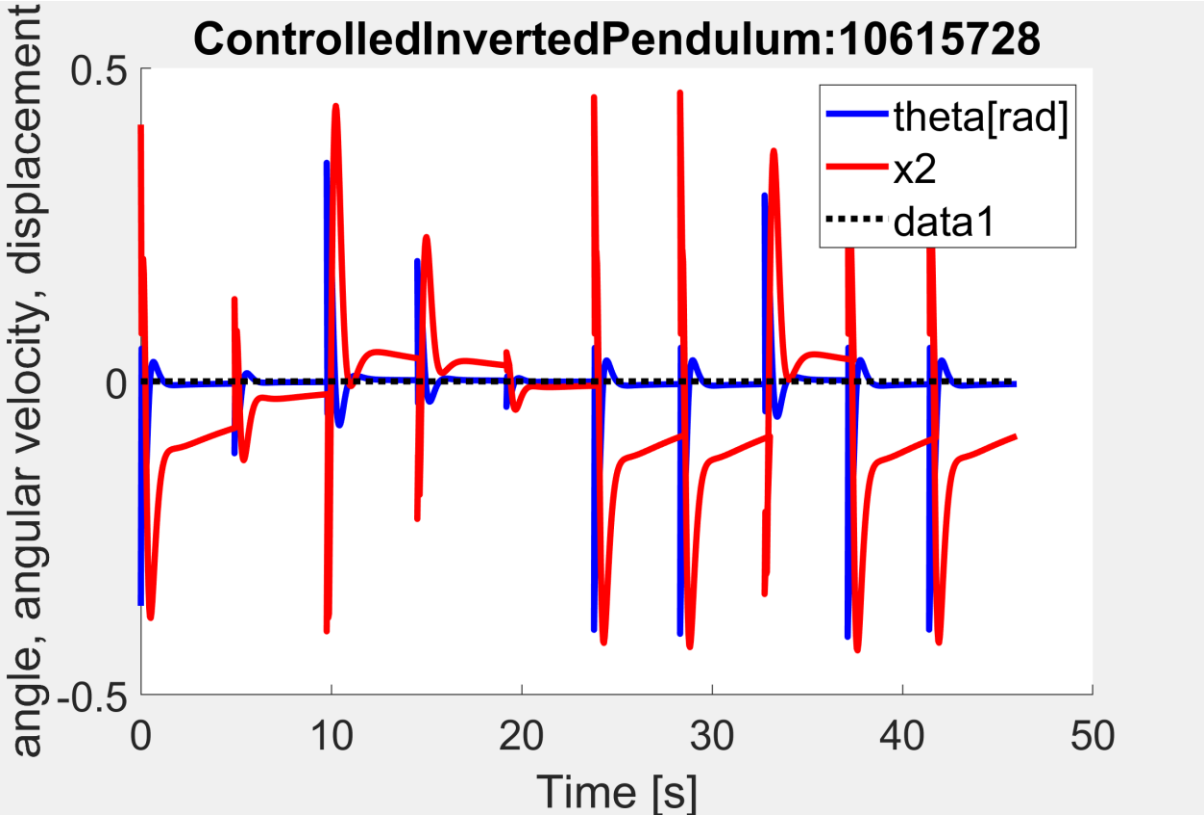

Figure 15: Angle, Angular Velocity and Displacement of the Inverted Pendulum plotted against Time