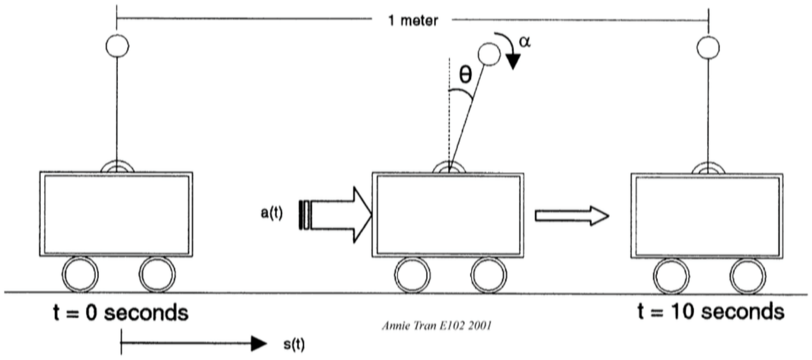
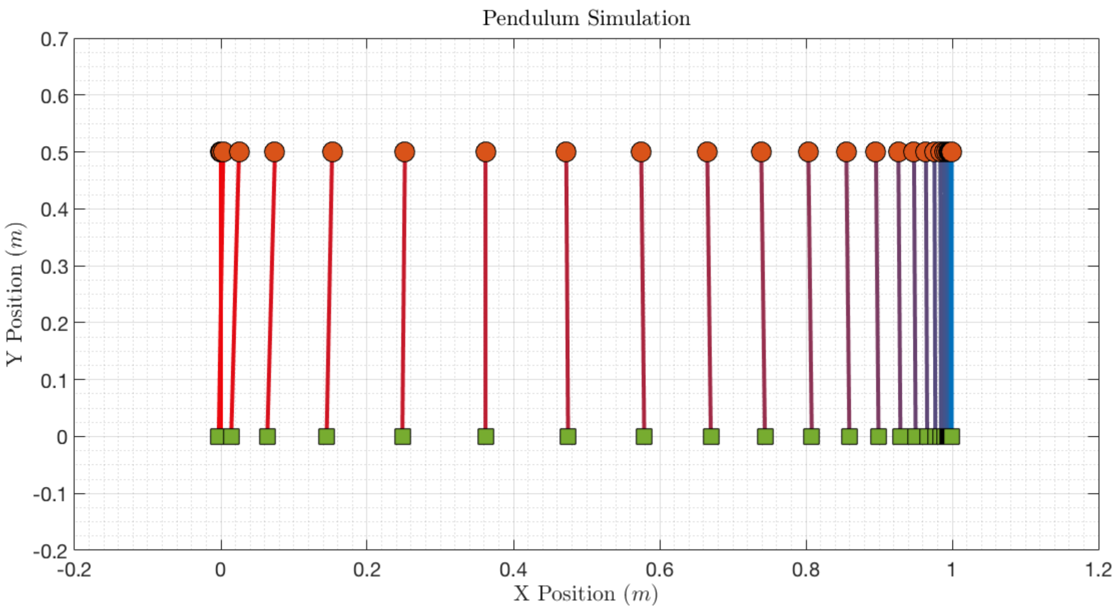
Advanced Systems Engineering II Final Simulation Study: State-Space Control of an Inverted Pendulum

Eyassu Shimelis

Abstract

The purpose of this final project is to design and validate a simulation of a state-space controller for a one-degree-of-freedom inverted pendulum on a massless, moving cart. From rest, the cart must move one meter and reach steady-state without overshooting or dropping the pendulum. All of this must be done in under ten seconds. This report details the design of a state-feedback controller consisting of an observer and integral action. The observer estimates the unknown state of the system and the integral action eliminates any steady state error.





I. Introduction

The inverted pendulum is a simple, yet highly unstable system, making it a fitting candidate for feedback control. Pendulums are mechanical systems composed of a mass at the end of a pinned rod. The pin allows the pendulum to rotate freely with minimum friction. Pendulums have only two stable points: when the mass is directly above the hinge, or when the mass is directly below the hinge. The latter is robust to small disturbances, while the former is not. The primary force acting on the mass is gravity, which exerts a torque over the rod's length, causing the pendulum to fall towards the positively stable equilibrium point. Like a ball on top of a hill, any small disturbances will cause the inverted pendulum to fall.

One method of controlling an inverted pendulum involves placing the pinned side onto a moving cart. The moving cart can be used to actively control for small deviations in the pendulum's angle, thereby stabilizing the system, as shown in Figure 1.

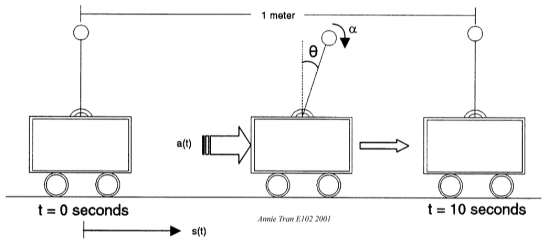


Fig. 1: Illustration of the project statement. The cart and inverted pendulum must move one meter under ten seconds.

The system of equations describing the system is shown below:

$$L\ddot{\theta}(t) - g\sin\theta(t) = -a(t)\cos\theta(t) + L\alpha,$$
$$\ddot{s} = a(t),$$

where

$\theta(t)$ – Pendulum Angle (rad)	$s(t)$ – Cart Position (m)
$a(t)$ – Cart Acceleration (m/s^2)	g – Gravitational Acceleration (m/s^2)
L – Pendulum Length (m)	α – Pendulum Angular Disturbance (rad/s^2)

II. State-space Design

The first step to designing a state-space controller is first linearizing the set of equations. The small angle assumption is one way of linearizing equations with sines and cosines. The assumptions states that for very small angles, $\sin\phi \to \phi$ and $\cos\phi \to 1$, resulting in the following set of linearized equations:

$$\ddot{\theta}(t) = \frac{g}{L}\sin\theta(t) - \frac{1}{L}a(t)\cos\theta(t) + \alpha$$
$$\ddot{s}(t) = a(t),$$

Using the state vector, $\boldsymbol{x} = \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \\ s(t) \\ \dot{s}(t) \end{bmatrix}$, control input $u = a(t)$, and output vector $\boldsymbol{y} = \begin{bmatrix} \theta(t) \\ s(t) \end{bmatrix}$, we can represent the linearized system as a combination of the form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u$$
$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}u$$

Replacing the state variables and state matrices,

$$\begin{bmatrix} \dot{\theta}(t) \\ \ddot{\theta}(t) \\ \dot{s}(t) \\ \ddot{s}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g}{L} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \\ s(t) \\ \dot{s}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{L} \\ 0 \\ 1 \end{bmatrix} a(t)$$

$$\begin{bmatrix} \theta(t) \\ s(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \\ s(t) \\ \dot{s}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} a(t)$$

Before designing the state-space system, we must evaluate whether the system is stable, controllable, and observable. The stability of the system can be found by examining the eigenvalues of the system matrix, $\boldsymbol{A}$. Using MATLAB's `eig` function, we find that the eigenvalues of the system matrix are $\lambda_n = 4.4288, -4.4288, 0, 0$. A system is stable only if the real parts of the eigenvalues are all less than zero. We see that this system is unstable.

The system is controllable only if the system matrix, $\boldsymbol{A}$, can be transformed into control canonical form, and the controllability matrix has full rank. Using MATLAB's `ctrb` function, we find that the controllability matrix has a rank of four; therefore, this inverted pendulum system is controllable.

Lastly, the system is observable only if the system matrix, $\boldsymbol{A}$, can be transformed into observer canonical form, and the observability matrix has full rank. Using MATLAB's `obsv` function, we find that the observability has a rank of 4; therefore, this system is observable.

A. Observer Design

State-space control assumes that we have full access to the state of the system; however, in some cases we are limited in the scope of what we can measure. In that case, if a system is fully observable, we can design an observer, which can estimate the state of the system. Given the control effort (u) and output (y), the observer outputs the predicted state, $\hat{\boldsymbol{x}}$. Namely,

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u + \boldsymbol{L}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$
$$\hat{\boldsymbol{y}} = \boldsymbol{C}\hat{\boldsymbol{x}} + \boldsymbol{D}u,$$

where the hats indicate the estimated states and outputs. As seen above, the observer also requires an observer gain, $\boldsymbol{L}$, which is set using MATLAB's `lqr` (Linear-Quadratic Regulator) function, which outputs a control gain that minimizes a quadratic cost function we specify. Based on the ratio of R and Q, we can specify the relative importance of the control effort versus the output error, respectively.

```
>> R_Qratio = 10;
   Q = R_Qratio*C'*C;
   R = 1;
  [K] = lqr(A,B,Q,R)

K =

  -34.2533   -7.7013   -3.1623   -3.9706
```

The observer is usually designed to have a significantly faster response than the plant. Using MATLAB's `place`, the poles were placed ten times faster than the system poles at $\lambda_n = -50, -51, -52, -53$

```
>> p = [-50, -51, -52, -53];
   L = place(A',C', p)'

L =

   1.0e+03 *

    0.1030    0.0011
    2.6709    0.0560
    0.0009    0.1030
    0.0469    2.6517
```

Lastly, the integral control was estimated at $K_i = 1$ (ideally we can use an augmented matrix to place both K and K_i using the `LQR` command; however, the command kept throwing an error and was left unresolved) and adjusted to fit the constraints of the project. Once the control gains are determined, the system can be created and simulated in MATLAB's Simulink.

III. Simulation

MATLAB's Simulink is a graphical programming environment suitable for designing our plant and controller, and simulating the system's responses to varying inputs. The overall block diagram of the system is shown below in Figure. 2.

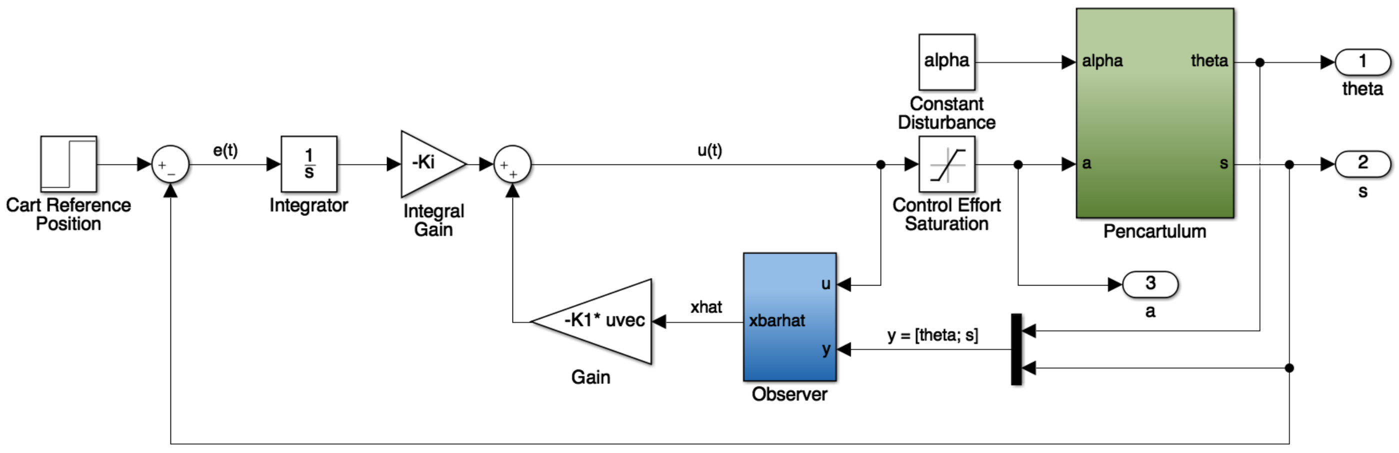


Fig. 2: Simulink block diagram of the entire system. Closed loop feedback is applied to the plant block, aptly named the Pencartulum. The control effort and system output are fed into the observer that estimates the state of the system.

The plant and observer systems are themselves subsystems, as shown in Figure 3 and Figure 4..

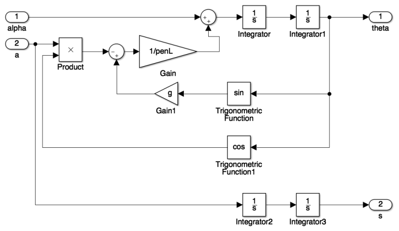


Fig. 3: Simulink diagram of inverted pendulum on cart system.

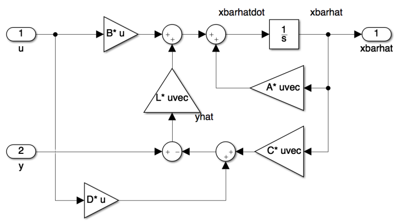


Fig. 4: Simulink diagram of observer system.

The simulation is run programmatically using the MATLAB script in Appendix A. All of the constants are provided in the workspace. In addition to this script, an animator was created. The animator is a visual tool that transforms the polar coordinates of the pendulum's angle to Cartesian coordinates. Below is an output plot from one of the simulations. Figure 5 shows the angle and cart position of the system as a function of time. As required, the cart reaches 1 meter under 10 seconds without any overshoot. Below that, in Figure 6, is a static animation of the output.

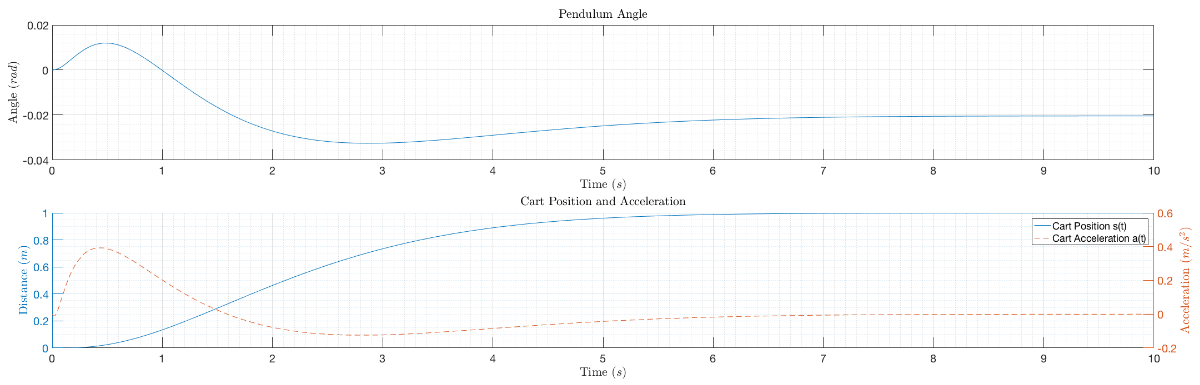


Fig. 5: The first plot illustrates the pendulum's angle as a function of time. As expected, the constant angular disturbance results in a non-zero steady state value. The plot on the bottom shows the position of the cart as a function of time. The cart gently approaches one meter, under ten seconds, and without any overshoot.

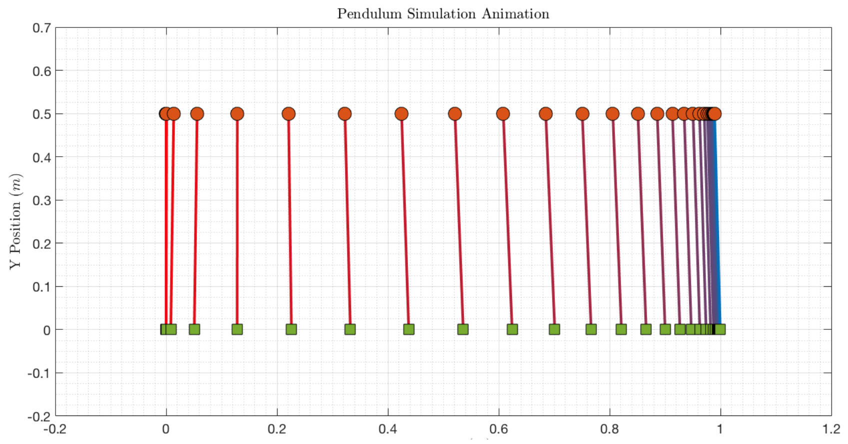


Fig. 6: The image above is a sampled animation of the controlled inverted pendulum. The pendulum's rod color starts red and linearly turns blue by the end of the animation to illustrate relative speeds. Also, note the slight angle of the pendulum at steady state. This is due to the constant angular disturbance, $\alpha = 0.4\ rad/s^2$.

IV. Discussion & Conclusion

Although the inverted pendulum is a highly unstable system, modern state-space control was effective at creating a robust controller that exceeded the constraints of the simulation study. Unlike classsical control, given a set of state equations, state-space control makes it simple to abstract both the cart and pendulum into one lumped system. Using a generalized 'state', we can treat the two components as one system: the pencartulum. The combined system has its own unique system matrix, $\boldsymbol{A}$. The system matrix showed that the pencartulum was unstable, yet controllable and observable. This means that it is possible to create an observer that can accurately estimate the state of the system based only on the measured output and control effort.
Using MATLAB's `lqr` and `place` functions, the control gains were designed to achieve the required system specifications. The cart moves one meter under ten seconds, and does not overshoot.

Now that the feedback control is designed, it is important to investigate the limits of operation. It was found that the maximum constant disturbance that the system can stably control without overshooting the final position is $\alpha = 0.6\ rad/s^2$. That specific simulation is shown in Figure 7.

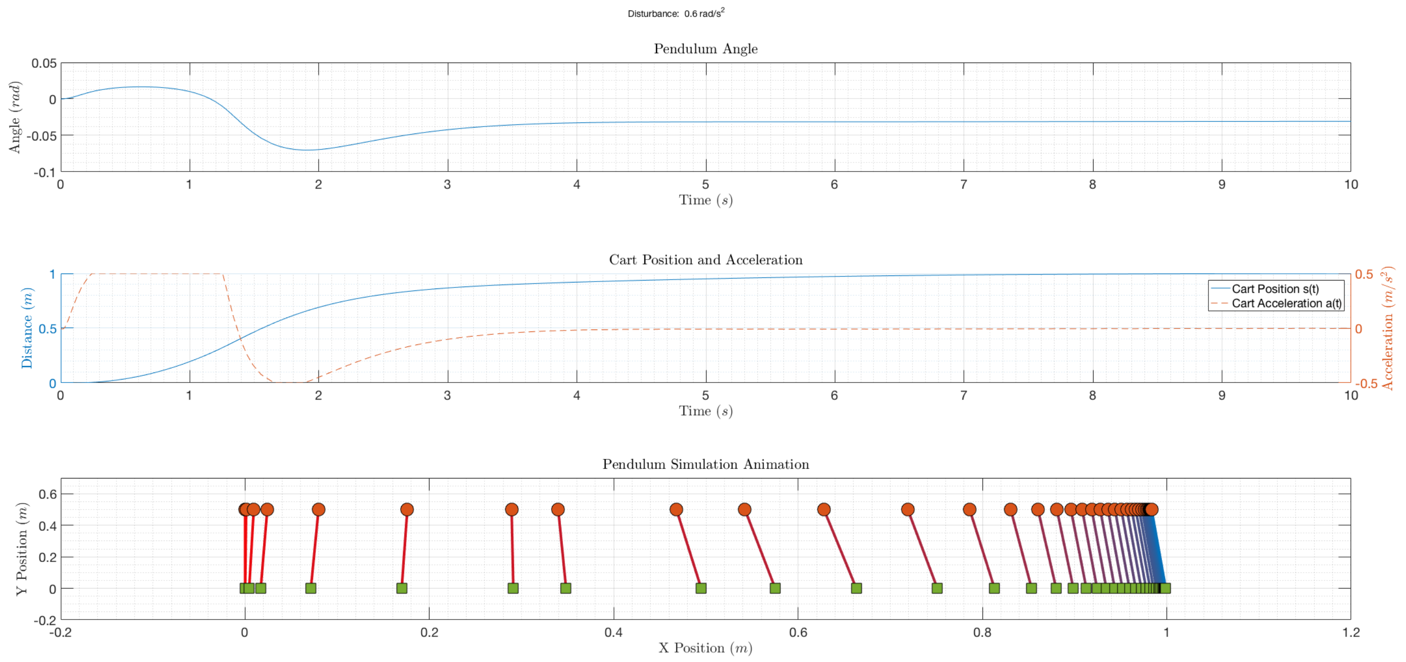


Fig. 7: Simulation output with maximum allowable disturbance of $\alpha = 0.6\ rad/s^2$, without instability. As shown in the second plot, the acceleration of the cart is saturated for approximately one second. The controller is operating at its limits

The largest, yet most realistic, limitation to the maximum allowable disturbance is the maximum acceleration of the cart. If the cart's maximum acceleration is unable to keep up with the falling pendulum, then it will be unable to control it, regardless of the control gains. One caveat of this simulation study is that it neglects the mass of the cart. The added inertia of the cart may make it more difficult to control the pendulum, especially if high accelerations are required.

Future work would focus on using an augmented system matrix to place both K_i and $\boldsymbol{K}$ using the Linear-Quadratic Regression function in MATLAB. Additionally, sensitivity analysis on different physical aspects of the pendulum may be insightful in better understanding which components of the pencarctulum greatly affect its stability. Lastly, it may be better to create a more realistic model of the pencartulum that includes frictional forces on the track and hinge, one that also accounts for the mass of the cart.

Appendix A
Main Matlab Script

```
%% Full state feedback test

penL = 0.5;        % pendulum length, meters
g = 9.807;      % gravitational acceleration, m/s^2
alpha = 0.4;    % pendulum disturbance, rad/s^2

% State matrices
A = [0   1  0  0;
     g/penL 0  0  0;
     0   0  0  1
     0   0  0  0];
B = [0; -1/penL; 0; 1];
C = [1 0 0 0;
     0 0 1 0];
D = [0; 0];

%% Stability, Controballity, and Observability of Linearized System

% stability
OLeig = eig(A);
if (any(OLeig > 0))
    disp('This system is unstable');
else
    disp('This system is stable');
end

% controlability
OLco = ctrb(OLSystem);
if (rank(A) == rank(OLco))
    disp('This system is controllable');
else
    disp('This system is not controllable');
end

% observability
OLob = obsv(OLSystem);
if (length(A) == rank(OLob))
    disp('This system is observable');
else
    disp('This system is not observable');
end

%%
R_Qratio = 10;
Q = R_Qratio*C'*C;
R = 1;
```

```
[K] = lqr(A,B,Q,R);

Ao = [[0 0; 0 0] -C; zeros(length(A), 2) A];
Bo = [-D; B];
Co = C;
Do = D;


% Place poles ten times further
p1 = -50;
offset = 1;
p2 = p1-offset;
p3 = p2-offset;
p4 = p3-offset;
p5 = p4-offset;

p = [p1 p2 p3 p4];
L = place(A',C', p)';

K1 = K;
Ki = 1;

tStep = 0;          % simulation step time, seconds
stepInitial = 0;    % initial reference value, meters
stepFinal = 1;      % final reference value, meters

minSaturationVal = -0.5;
maxSaturationVal = 0.5;

% Open Loop Simulation Test
tSpan = 10;
fs = 0.1;
time = linspace(0, tSpan, tSpan*fs);
% Run simulation
[t, x, y] = sim('Pencartulum', time);

% Plot output
time = t;
theta = y(:, 1);
basePos = y(:, 2);
acceleration = y(:, 3);

figure(2)
clf;
subplot(2, 1, 1)
plot(time, theta);
title('Pendulum Angle', 'interpreter', 'latex');
xlabel('Time ($s$)', 'interpreter', 'latex');
ylabel('Angle ($rad$)', 'interpreter', 'latex');
set(gca, 'fontsize', 20);
```

```
grid on; grid minor;

subplot(2, 1, 2); hold on;
% yyaxis left;
plot(time, basePos, '-');
title('Cart Postion', 'interpreter', 'latex');
xlabel('Time ($s$)', 'interpreter', 'latex');
ylabel('Distance ($m$)', 'interpreter','latex');
set(gca, 'fontsize', 20);

yyaxis right;
plot(time, acceleration, '--');
title('Cart Position and Acceleration', 'interpreter', 'latex');
ylabel('Acceleration ($m/s^2$)', 'interpreter','latex');
set(gca, 'fontsize', 20);

legend('Cart Position s(t)', 'Cart Acceleration a(t)')
grid on; grid minor; hold off;

%% Animate Output
figure(3);
plot(acceleration);
clearPlot = false;
shading = true;
animatePendulum(theta, penL, basePos, 5, shading, clearPlot);
```

APPENDIX B
PENDULUM ANIMATION SCRIPT

```
function animatePendulum( theta, L, basePos, animationStepSize, shading,...
clearPlot)
% animatePendulum.m Creates an animation for a pendulum, given angle and
% base position.

% Convert from polar to cartesian coordinates for xy plotting
[yPos, xPos] = pol2cart(theta, L);

% Add cart movement to pendulum movement in x position
xPos = xPos + basePos;

%// Plot starts here
cla; hold on; grid on; grid minor;

% Define and set xy limits
xMin = -0.2;
xMax = 1.2;
yMin = -0.2;
yMax = 0.7;
xlim([xMin xMax])
ylim([yMin yMax])

pbaspect([2 1 1])

title('Pendulum Simulation Animation', 'interpreter', 'latex');
xlabel('X Position ($m$)', 'interpreter', 'latex');
ylabel('Y Position ($m$)', 'interpreter', 'latex');
set(gca, 'fontsize', 20);

if (shading)
    for k = 1:animationStepSize:length(xPos)
        if(clearPlot)
            cla;
        end
        plot([xPos(k) basePos(k)], [yPos(k) 0], 'LineWidth', 4 , 'Color',...
        [ 1-k/length(xPos), k/length(xPos)*0.4470, (k/length(xPos))*0.7410])
        plot(xPos(k), yPos(k), 'o', 'markersize', 20, 'MarkerFaceColor',...
        [0.8500 0.3250 0.0980], 'MarkerEdgeColor','k')
        plot(basePos(k), 0, 's', 'markersize', 20, 'MarkerFaceColor',...
        [0.4660 0.6740 0.1880], 'MarkerEdgeColor','k');
        pause(0.1);
    end
else
    for k = 1:animationStepSize:length(xPos)
        if(clearPlot)
            cla;
        end
        plot([xPos(k) basePos(k)], [yPos(k) 0], 'LineWidth', 4 , 'Color',...
        [ 0, 0.4470, 0.7410])
        plot(xPos(k), yPos(k), 'o', 'markersize', 20, 'MarkerFaceColor',...
        [0.8500 0.3250 0.0980], 'MarkerEdgeColor','k')
        plot(basePos(k), 0, 's', 'markersize', 20, 'MarkerFaceColor',...
        [0.4660 0.6740 0.1880], 'MarkerEdgeColor','k');
        pause(0.1);
    end
end

end
```