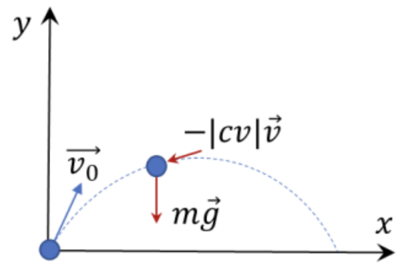
Robotics HW 2 Report

Praful Sigdel

February 16, 2024

Problem 1: Consider the problem of a throwing ball. The mass of the ball is m. Throw the ball with an initial velocity $\vec{v_0}$. While moving with a velocity $\vec{v}$, it is affected by a drag force of $\vec{F}_{drag} = -|cv|\vec{v}$, where c is the drag coefficient.

a. Derive the dynamics of the system.



Ans: Here, the drag force $\vec{F}_{drag} = -|cv|\vec{v}$ can further be written down as:

$$\vec{F}_{drag} = -c|v|\vec{v} = -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{x}.\hat{\imath} + -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{y}.\hat{\jmath}$$

where $\hat{\imath}$ and $\hat{\jmath}$ are unit vectors along x and y directions respectively.

Applying Euler-Lagrange's Formulation, we get,

$$L = T - V = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) - mgy$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_j}) - \frac{\partial L}{\partial q_j} = Q_j$$

if $q_j =$ x , y and $Q_j = F_{x_{drag}}, F_{y_{drag}}$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = F_{x_{drag}}$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{y}}) - \frac{\partial L}{\partial y} = F_{y_{drag}}$$

$$\frac{\partial L}{\partial \dot{x}} = m\dot{x}, \frac{\partial L}{\partial \dot{y}} = m\dot{y}, \frac{\partial L}{\partial x} = 0, \frac{\partial L}{\partial y} = -mg$$

Therefore,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = F_{x_{drag}}$$

$$\frac{d}{dt}(m\dot{x}) - 0 = -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{x}$$

$$m\ddot{x} = -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{x}$$

$$\boxed{\ddot{x} = -\frac{c}{m}\dot{x}\sqrt{\dot{x}^2 + \dot{y}^2}}$$

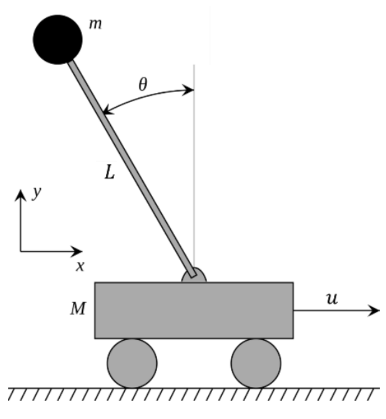
$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{y}}) - \frac{\partial L}{\partial y} = F_{y_{drag}}$$

$$\frac{d}{dt}(m\dot{y}) + mg = -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{y}$$

$$m\ddot{y} + mg = -c\sqrt{\dot{x}^2 + \dot{y}^2}.\dot{y}$$

$$\boxed{\ddot{y} = -g - \frac{c}{m}\dot{y}\sqrt{\dot{x}^2 + \dot{y}^2}}$$

2. Consider the following cart-pole system.



a. Derive the dynamics of the robot using Euler-Lagrange Equation. Assume that the total mass of the pole concentrated at the end of the pole.

Ans: Let us consider the system with x as the horizontal position of the cart, θ is the counter-clockwise rotation of the pole (zero is the hanging straight down). Let, $\mathbf{q} = [x, \theta]^T$, $\dot{\mathbf{q}} = [\dot{x}, \dot{\theta}]^T$, and $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$. Note: **L** on the L.H.S of most of the equations that follows are Lagrangian and **L** on the R.H.S of most of the equations are length of the pole. For Euler-Lagrangian Formulation, we have:

Kinetic Energy:

$$T = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m(L\cos(\theta)\dot{\theta}\dot{x} + L^2cos^2(\theta)\dot{\theta}^2 + L^2sin^2(\theta)\dot{\theta}^2)$$

$$T = \frac{1}{2}(M+m)\dot{x}^2 + mL\cos(\theta)\dot{\theta}\dot{x} + \frac{1}{2}mL^2\dot{\theta}^2$$

Potential Energy:

$$U = -mgLcos(\theta)$$

The Euler-Langrange Equation of motion yields:

$$L = T - V = \frac{1}{2}(M+m)\dot{x}^2 + mL\cos(\theta)\dot{\theta}\dot{x} + \frac{1}{2}mL^2\dot{\theta}^2 - (-mgLcos(\theta))$$

$$L = \frac{1}{2}(M+m)\dot{x}^2 + mL\cos(\theta)\dot{\theta}\dot{x} + \frac{1}{2}mL^2\dot{\theta}^2 + mgLcos(\theta)$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_j}) - \frac{\partial L}{\partial q_j} = Q_j$$

if q_j = x , θ and $Q_j = u, 0$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = u$$

$$\frac{\partial L}{\partial \dot{x}} = (M+m)\dot{x} + mLcos(\theta)\dot{\theta},$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{x}}) = (M+m)\ddot{x} - mLsin(\theta)\dot{\theta}^2 + mLcos(\theta)\ddot{\theta}$$

$$\frac{\partial L}{\partial x} = 0$$

Therefore,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = u$$

$$\boxed{(M+m)\ddot{x} - mLsin(\theta)\dot{\theta}^2 + mLcos(\theta)\ddot{\theta} = u}$$

And,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} = 0$$

$$\frac{\partial L}{\partial \dot{\theta}} = m\dot{x}Lcos(\theta) + mL^2\dot{\theta}$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) = m\ddot{x}Lcos(\theta) - m\dot{x}Lsin(\theta)\dot{\theta} + mL^2\ddot{\theta}$$

$$\frac{\partial L}{\partial \theta} = -m\dot{x}Lsin(\theta)\dot{\theta} - mgLsin(\theta)$$

Therefore,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} = 0$$

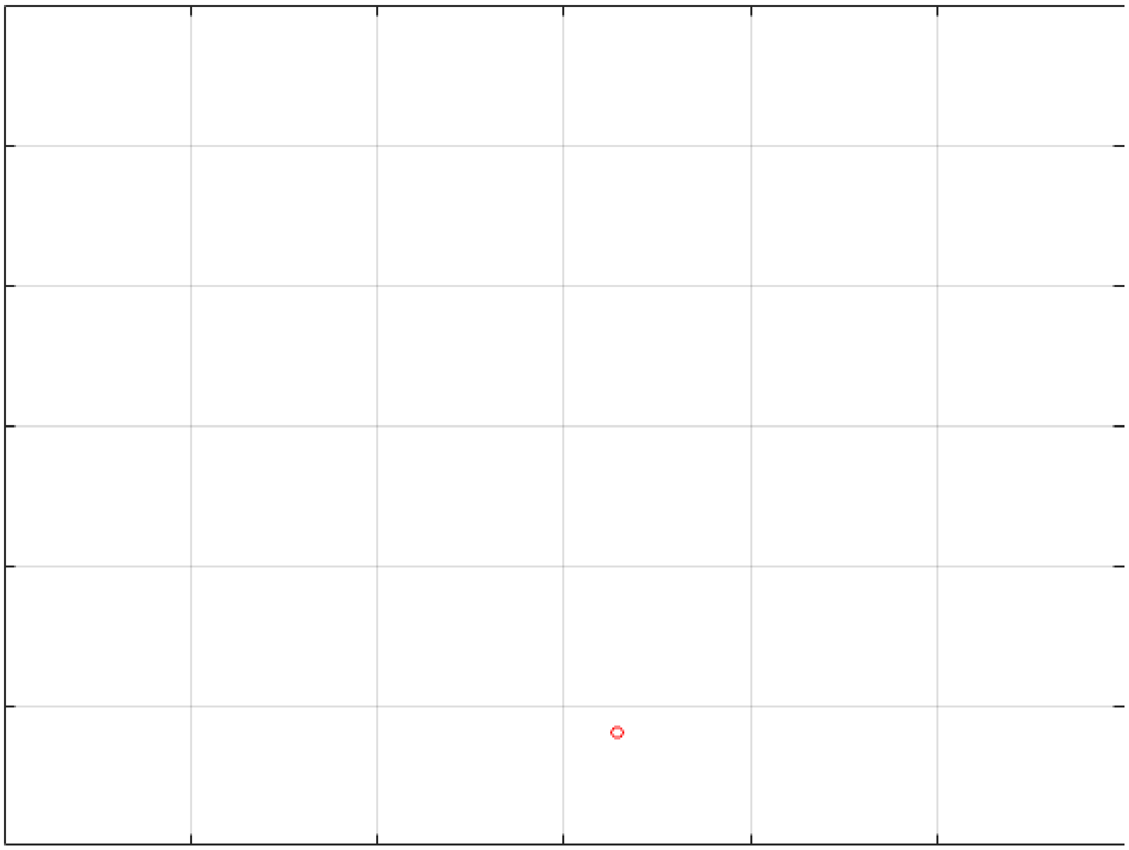
$$m\ddot{x}Lcos(\theta) - m\dot{x}Lsin(\theta)\dot{\theta} + mL^2\ddot{\theta} + m\dot{x}Lsin(\theta)\dot{\theta} + mgLsin(\theta) = 0$$

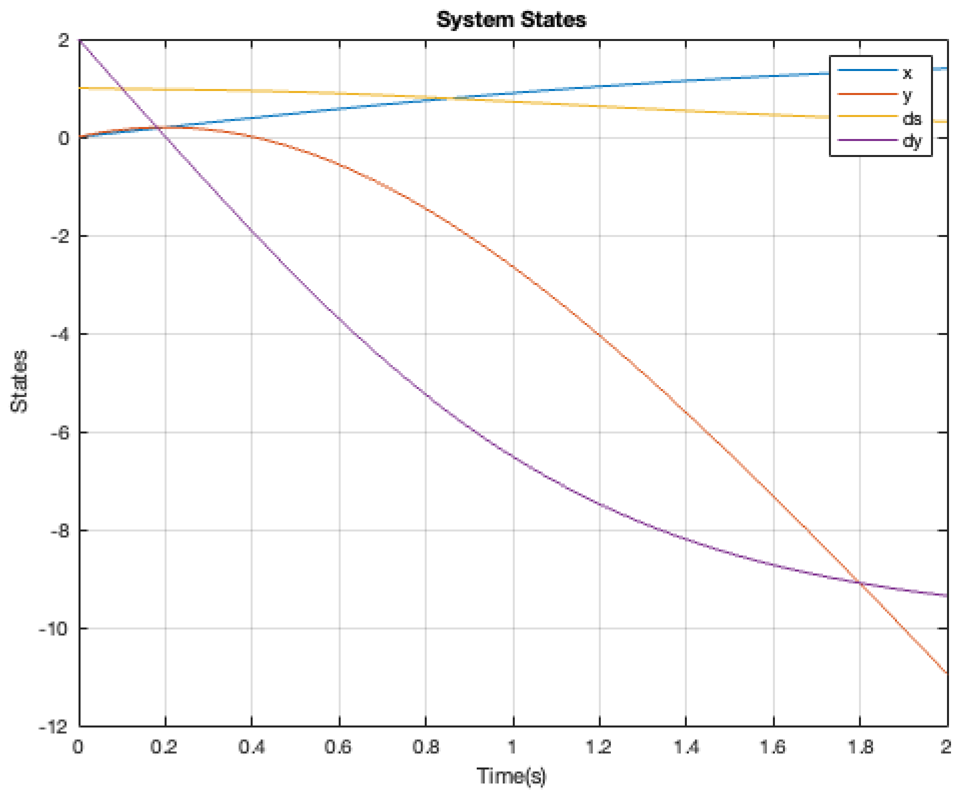
$$\boxed{m\ddot{x}Lcos(\theta) + mL^2\ddot{\theta} + mgLsin(\theta) = 0}$$

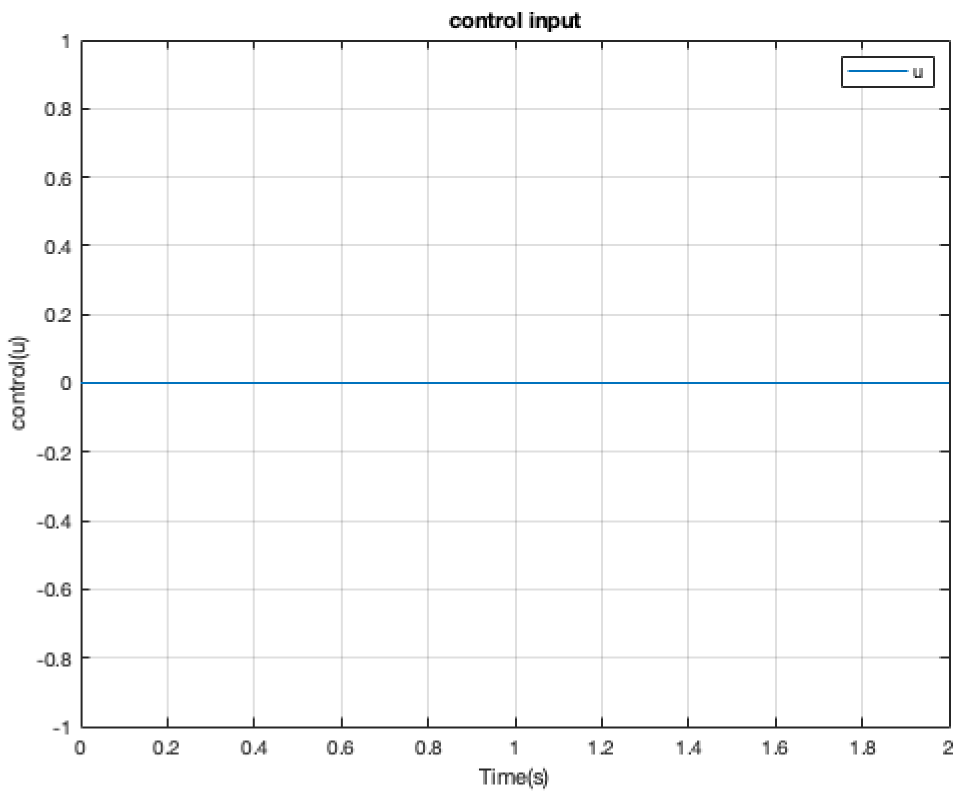
```
% Praful Sigdel
% AME 556: HW 2 Matlab Code for 2a
%

function HW2_projectile()
% simulation of a ball thrown in 2D space which is affected by drag force

% system states: X = [x;y;dx;dy];
% control input: u = F;

clc; clear all; close all;
global params;

m = 0.5 ; g = 9.81 ; c = 0.05;
params.m=m;
params.g =g;
params.c =c;

x0=[0;0;1;2]; % x0 is the intial state of the system

tspan=[0; 2]; % simulation time

[t,x]=ode45(@sys_dynamics,tspan,x0);

anim(t,x,1/24); % animate the system and save the simulation video

% recreate control inputs
 for i=1:length(t)
     u(:,i)=controller(t(i),x(i,:)');
 end

 % plot the simulation data
figure;
plot(t,x);
legend('x','y','ds','dy');
grid on;
xlabel('Time(s)');
ylabel('States');
title('System States');

figure;
plot(t,u);
grid on;
xlabel('Time(s)');
ylabel('control(u)');
legend('u'); title('control input');

end

function dx=sys_dynamics(t,x)
global params;
```

```
u=controller(t,x);

ds = x(3);

ddsx = -params.c/params.m * norm(x(3:4)) * x(3);

dy = x(4);

ddy = -params.g - params.c/params.m * norm(x(3:4)) * x(4);

dx = [ds;dy;ddsx;ddy];

end

function u=controller(t,x)

global params

u = 0; % you can put your controller here

end

function anim(t,x,ts)
[te,xe]=even_sample(t,x,1/ts);

figure(1);

axes1 = axes;

%save as a video
spwriter = VideoWriter('video_HW2_demo.mp4','MPEG-4');
set(spwriter, 'FrameRate', 1/ts,'Quality',100);
open(spwriter);

fig1 = figure(1);

figure_x_limits = [-15 15];
figure_y_limits = [-15 15];

axes1 = axes;
set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
set(axes1,'Position',[0 0 1 1]);
set(axes1,'Color','w');
grid on;

for k = 1:length(te)
    drawone(axes1, xe(k,:)');
    set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
    drawnow;
    pause(ts);
    frame = getframe(gcf);
    writeVideo(spwriter, frame);
end
```

```
end

function [Et, Ex] = even_sample(t, x, Fs)
%       CONVERTS A RANDOMLY SAMPLED SIGNAL SET INTO AN EVENLY SAMPLED
%       SIGNAL SET (by interpolation)
% Obtain the process related parameters
N = size(x, 2);    % number of signals to be interpolated
M = size(t, 1);    % Number of samples provided
t0 = t(1,1);       % Initial time
tf = t(M,1);       % Final time
EM = (tf-t0)*Fs;   % Number of samples in the evenly sampled case with
                   % the specified sampling frequency
Et = linspace(t0, tf, EM)';

% Using linear interpolation (used to be cubic spline interpolation)
% and re-sample each signal to obtain the evenly sampled forms
for s = 1:N
  Ex(:,s) = interp1(t(:,1), x(:,s), Et(:,1));
end
end

function drawone(parent, x)
% draw the robot at the current frame
global params
tem = get(parent,'Children');
delete(tem);
s = x(1);
y = x(2);
p = [s;
      y] ;

plot(p(1),p(2),'ro');
grid on;

end
```




System States
States
Time(s)
x
y
ds
dy
2
0
-2
-4
-6
-8
-10
-12
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

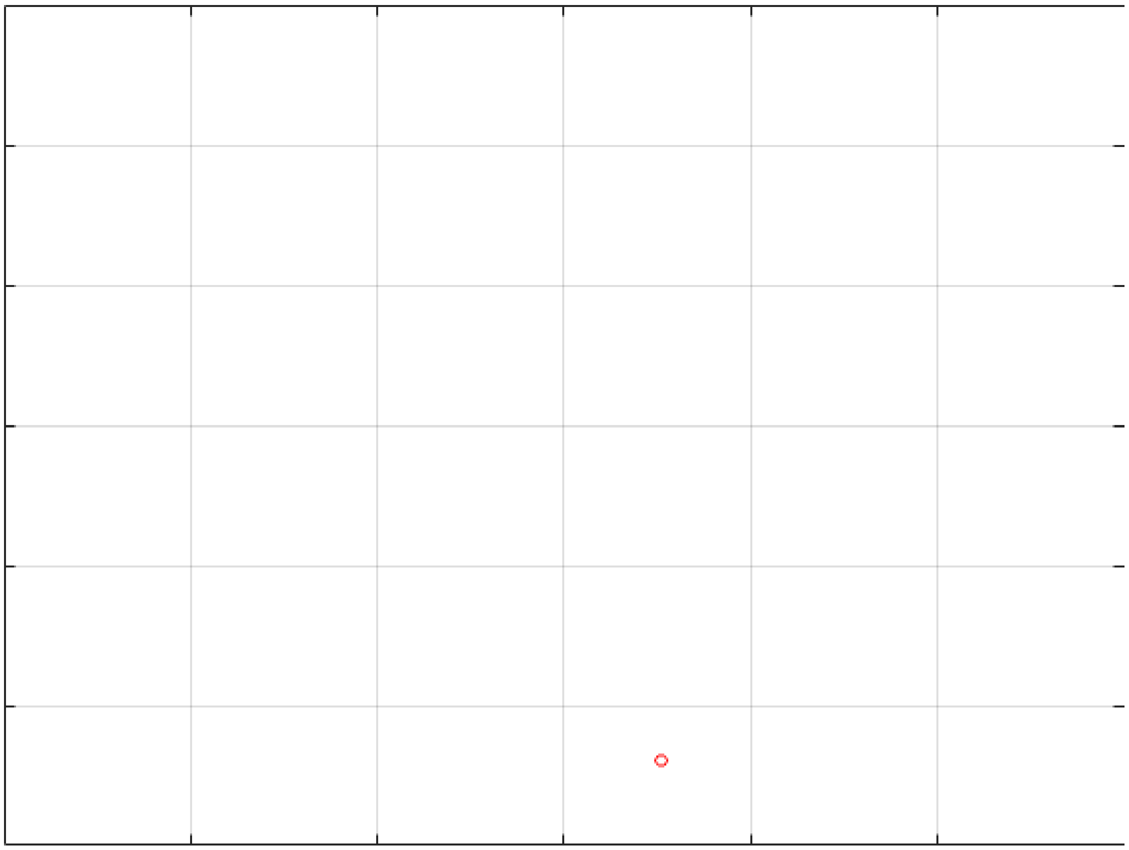


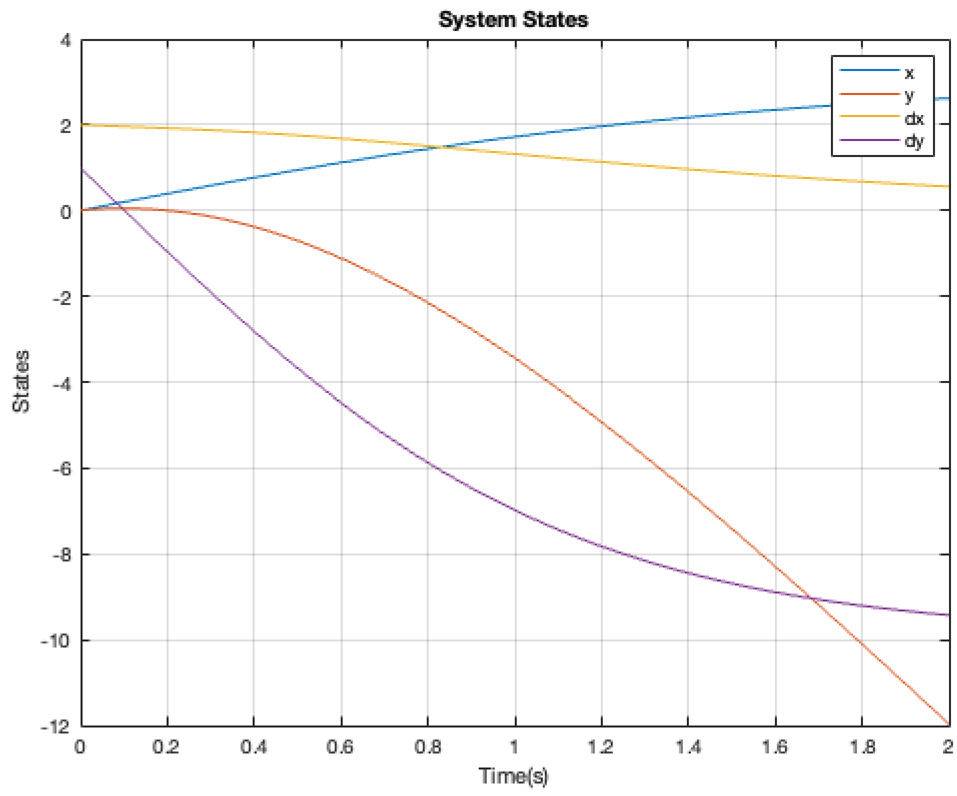
Published with MATLAB® R2023b

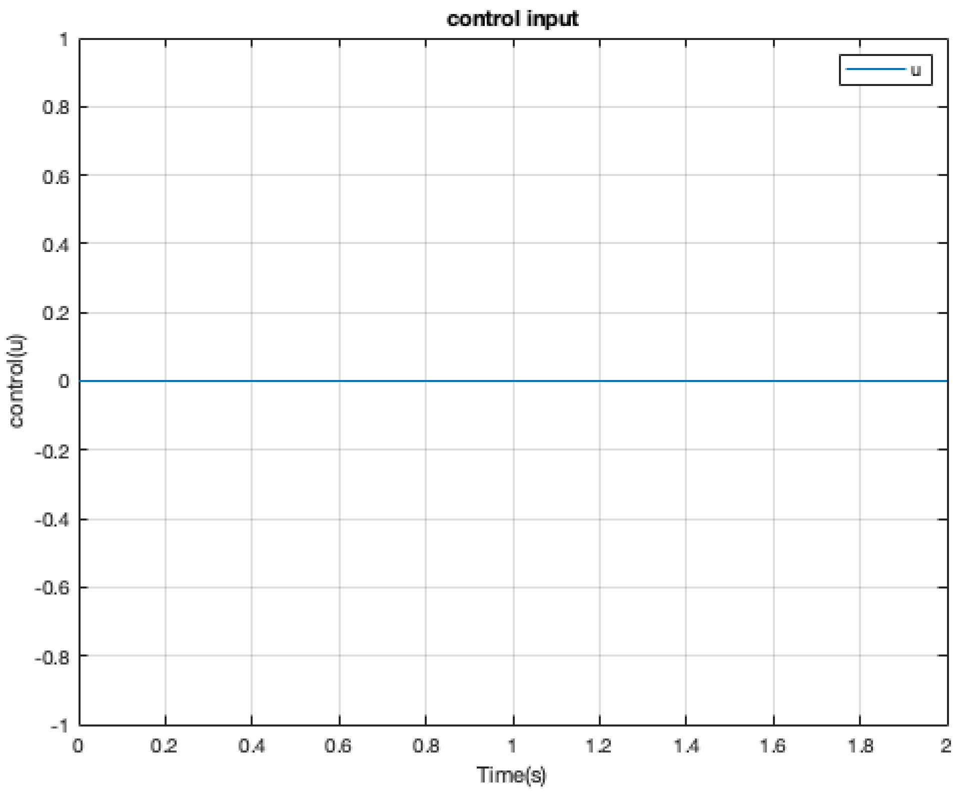
```matlab
% Praful Sigdel
% AME 556: HW 2 Matlab Code for 1ci.
%

function HW2_projectile1ci()
% simulation of a ball thrown in 2D space which is affected by drag force

% system states: X = [x;y;dx;dy];
% control input: u = F;
% v0 = [2;1] m/s
% c = 0.05

clc; clear all; close all;
global params;

m = 0.5 ; g = 9.81 ; c = 0.05;
params.m=m;
params.g =g;
params.c =c;

x0=[0;0;2;1]; % x0 is the intial state of the system

tspan=[0; 2]; % simulation time

[t,x]=ode45(@sys_dynamics,tspan,x0);

anim(t,x,1/24); % animate the system and save the simulation video

% recreate control inputs
 for i=1:length(t)
     u(:,i)=controller(t(i),x(i,:)');
 end

 % plot the simulation data
figure;
plot(t,x);
legend('x','y','dx','dy');
grid on;
xlabel('Time(s)');
ylabel('States');
title('System States');

figure;
plot(t,u);
grid on;
xlabel('Time(s)');
ylabel('control(u)');
legend('u'); title('control input');

end

function dx=sys_dynamics(t,x)
```

```
global params;

u=controller(t,x);

ds = x(3);

ddsx = -params.c/params.m * norm(x(3:4)) * x(3);

dy = x(4);

ddy = -params.g - params.c/params.m * norm(x(3:4)) * x(4);

dx = [ds;dy;ddsx;ddy];

end

function u=controller(t,x)

global params

u = 0; % you can put your controller here

end

function anim(t,x,ts)
[te,xe]=even_sample(t,x,1/ts);

figure(1);

axes1 = axes;

%save as a video
spwriter = VideoWriter('video_HW2_demo1ci.mp4','MPEG-4');
set(spwriter, 'FrameRate', 1/ts,'Quality',100);
open(spwriter);

fig1 = figure(1);

figure_x_limits = [-15 15];
figure_y_limits = [-15 15];

axes1 = axes;
set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
set(axes1,'Position',[0 0 1 1]);
set(axes1,'Color','w');
grid on;

for k = 1:length(te)
    drawone(axes1, xe(k,:)');
    set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
    drawnow;
    pause(ts);
    frame = getframe(gcf);
    writeVideo(spwriter, frame);
```

```
end

end

function [Et, Ex] = even_sample(t, x, Fs)
%       CONVERTS A RANDOMLY SAMPLED SIGNAL SET INTO AN EVENLY SAMPLED
%       SIGNAL SET (by interpolation)
% Obtain the process related parameters
N = size(x, 2);    % number of signals to be interpolated
M = size(t, 1);    % Number of samples provided
t0 = t(1,1);       % Initial time
tf = t(M,1);       % Final time
EM = (tf-t0)*Fs;   % Number of samples in the evenly sampled case with
                   % the specified sampling frequency
Et = linspace(t0, tf, EM)';

% Using linear interpolation (used to be cubic spline interpolation)
% and re-sample each signal to obtain the evenly sampled forms
for s = 1:N
  Ex(:,s) = interp1(t(:,1), x(:,s), Et(:,1));
end
end

function drawone(parent, x)
% draw the robot at the current frame
global params
tem = get(parent,'Children');
delete(tem);
s = x(1);
y = x(2);
p = [s;
      y] ;

plot(p(1),p(2),'ro');
grid on;

end
```




System States
States
Time(s)
x
y
dx
dy
4
2
0
-2
-4
-6
-8
-10
-12
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

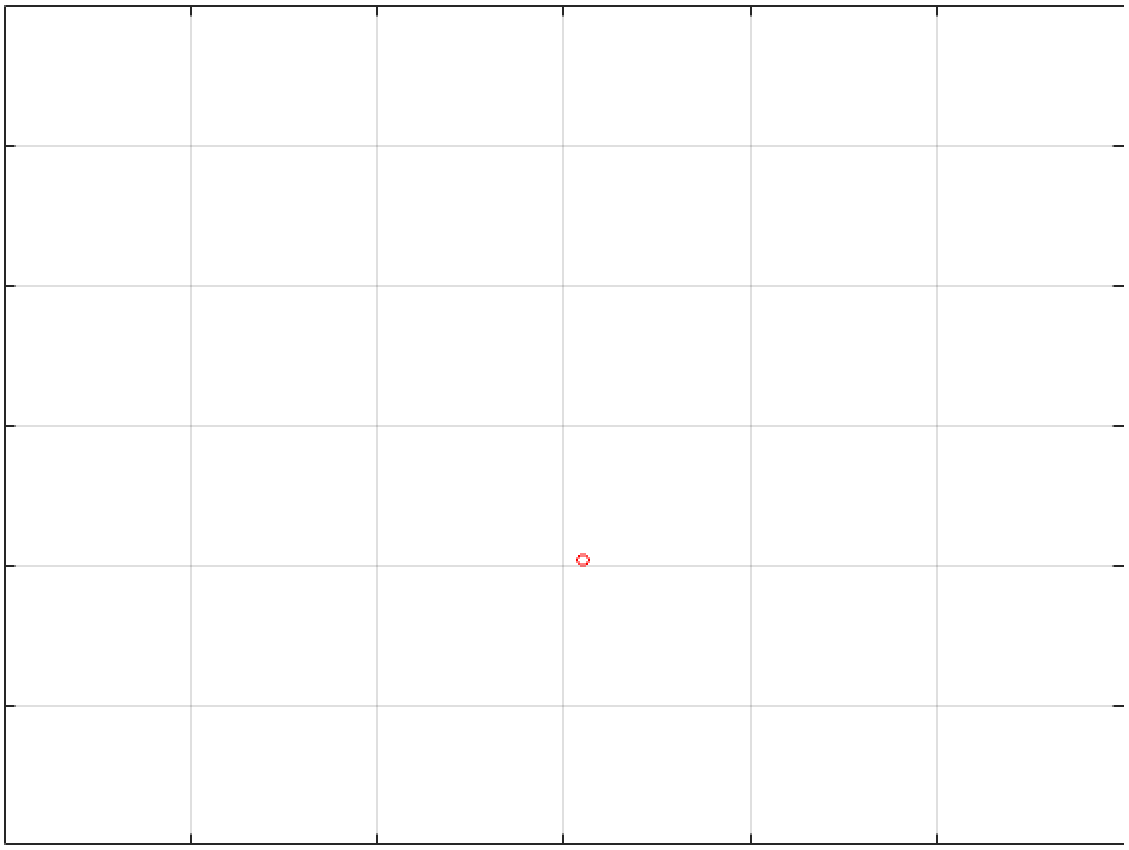


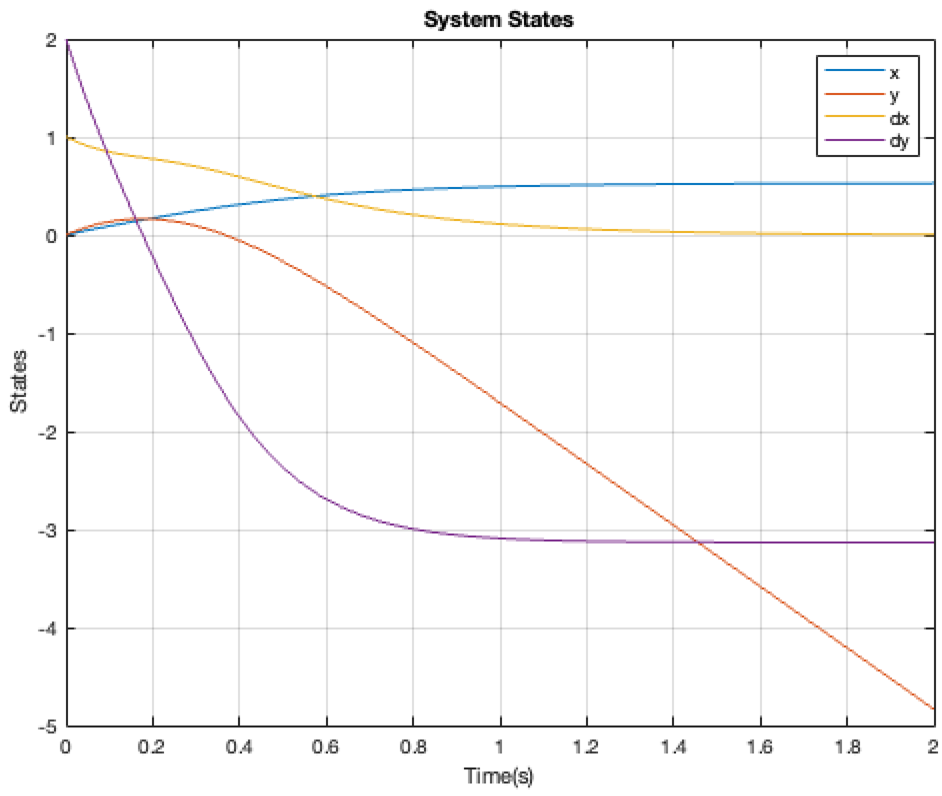
Published with MATLAB® R2023b

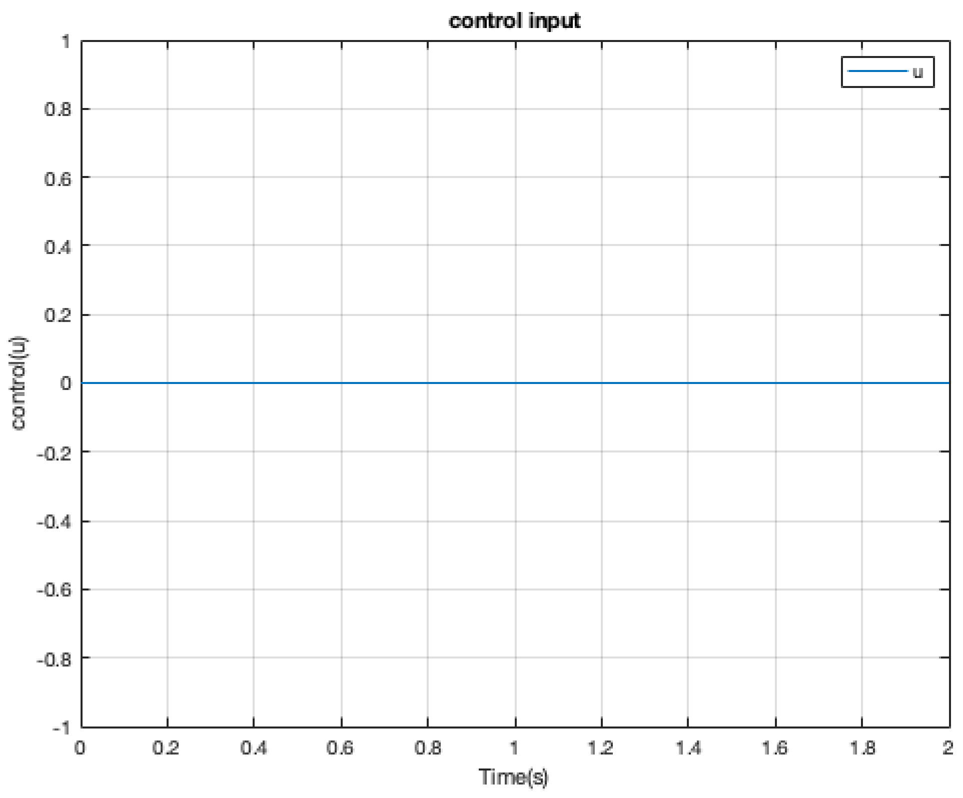
```
% Praful Sigdel
% AME 556: HW 2 Matlab Code for 1cii.
%

function HW2_projectile1cii()
% simulation of a ball thrown in 2D space which is affected by drag force

% system states: X = [x;y;dx;dy];
% control input: u = F;
% v0 = [1;2] m/s
% c = 0.5

clc; clear all; close all;
global params;

m = 0.5 ; g = 9.81 ; c = 0.5;
params.m=m;
params.g =g;
params.c =c;

x0=[0;0;1;2]; % x0 is the intial state of the system

tspan=[0; 2]; % simulation time

[t,x]=ode45(@sys_dynamics,tspan,x0);

anim(t,x,1/24); % animate the system and save the simulation video

% recreate control inputs
 for i=1:length(t)
     u(:,i)=controller(t(i),x(i,:)');
 end

 % plot the simulation data
figure;
plot(t,x);
legend('x','y','dx','dy');
grid on;
xlabel('Time(s)');
ylabel('States');
title('System States');

figure;
plot(t,u);
grid on;
xlabel('Time(s)');
ylabel('control(u)');
legend('u'); title('control input');

end

function dx=sys_dynamics(t,x)
```

```
global params;

u=controller(t,x);

ds = x(3);

ddsx = -params.c/params.m * norm(x(3:4)) * x(3);

dy = x(4);

ddy = -params.g - params.c/params.m * norm(x(3:4)) * x(4);

dx = [ds;dy;ddsx;ddy];

end

function u=controller(t,x)

global params

u = 0; % you can put your controller here

end

function anim(t,x,ts)
[te,xe]=even_sample(t,x,1/ts);

figure(1);

axes1 = axes;

%save as a video
spwriter = VideoWriter('video_HW2_demo1cii.mp4','MPEG-4');
set(spwriter, 'FrameRate', 1/ts,'Quality',100);
open(spwriter);

fig1 = figure(1);

figure_x_limits = [-15 15];
figure_y_limits = [-15 15];

axes1 = axes;
set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
set(axes1,'Position',[0 0 1 1]);
set(axes1,'Color','w');
grid on;

for k = 1:length(te)
    drawone(axes1, xe(k,:)');
    set(axes1,'XLim',figure_x_limits,'YLim',figure_y_limits);
    drawnow;
    pause(ts);
    frame = getframe(gcf);
    writeVideo(spwriter, frame);
```

```
end

end

function [Et, Ex] = even_sample(t, x, Fs)
%       CONVERTS A RANDOMLY SAMPLED SIGNAL SET INTO AN EVENLY SAMPLED
%       SIGNAL SET (by interpolation)
% Obtain the process related parameters
N = size(x, 2);    % number of signals to be interpolated
M = size(t, 1);    % Number of samples provided
t0 = t(1,1);       % Initial time
tf = t(M,1);       % Final time
EM = (tf-t0)*Fs;   % Number of samples in the evenly sampled case with
                   % the specified sampling frequency
Et = linspace(t0, tf, EM)';

% Using linear interpolation (used to be cubic spline interpolation)
% and re-sample each signal to obtain the evenly sampled forms
for s = 1:N
  Ex(:,s) = interp1(t(:,1), x(:,s), Et(:,1));
end
end

function drawone(parent, x)
% draw the robot at the current frame
global params
tem = get(parent,'Children');
delete(tem);
s = x(1);
y = x(2);
p = [s;
      y] ;

plot(p(1),p(2),'ro');
grid on;
end
```



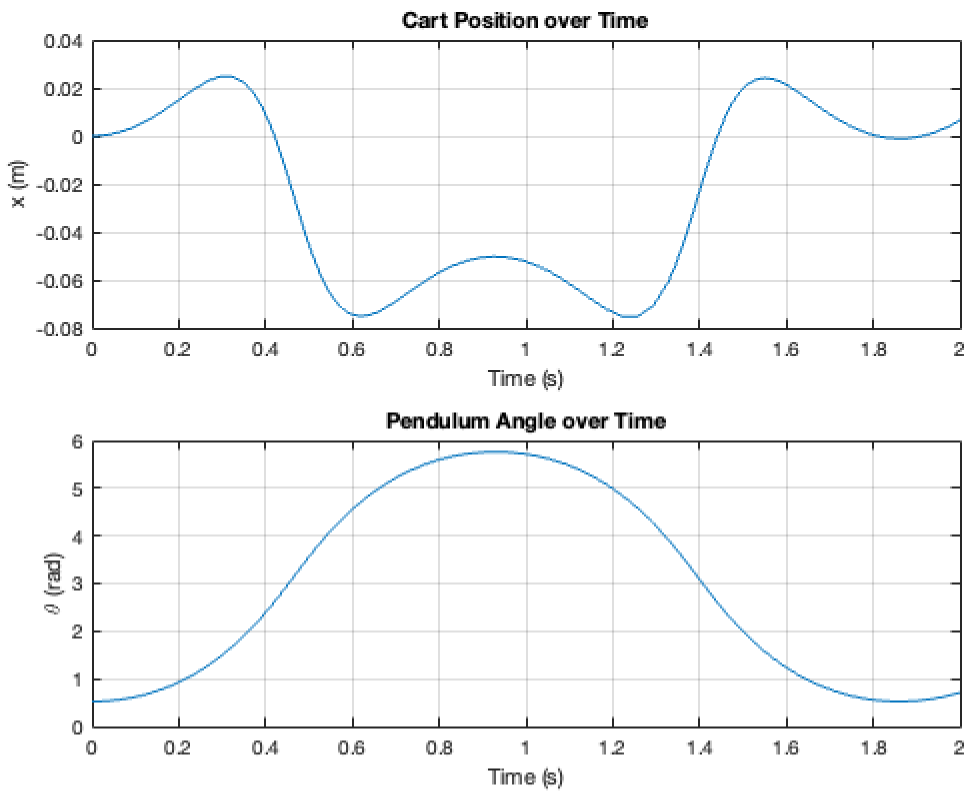

System States
States
Time(s)
x
y
dx
dy
2
1
0
-1
-2
-3
-4
-5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

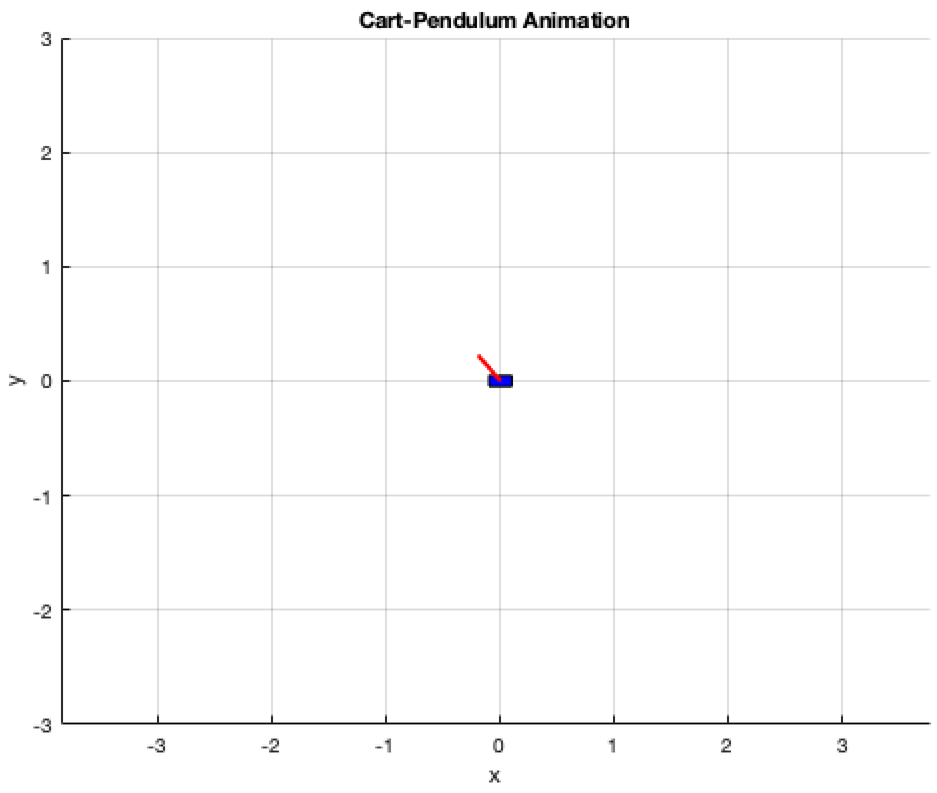


Published with MATLAB® R2023b

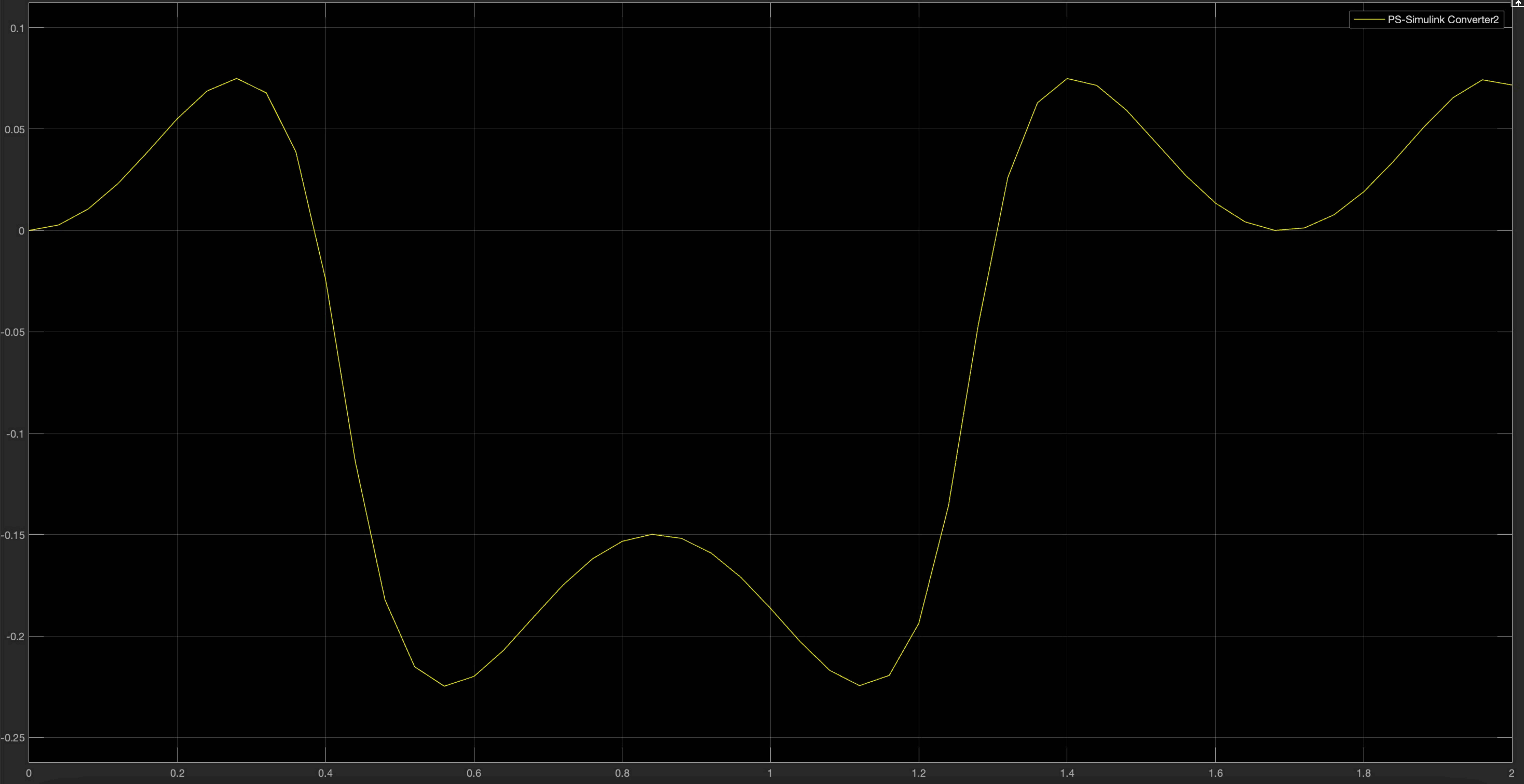
```
function cart_pendulum_simulation()
    % Define system parameters
    M = 1; % Mass of the cart (kg)
    m = 0.2; % Mass of the pendulum bob (kg)
    L = 0.3; % Length of the pendulum (m)
    g = 9.81; % Gravity (m/s^2)

    % Initial conditions
    x0 = 0; % Initial position of the cart (m)
    theta0 = pi + pi/6; % Initial angle of the pendulum (rad)
    dx0 = 0; % Initial velocity of the cart (m/s)
    dtheta0 = 0; % Initial angular velocity of the pendulum (rad/s)

    % State vector
    q0 = [x0; theta0];
    dq0 = [dx0; dtheta0];

    % Time span for simulation
    tspan = [0, 2]; % Simulate for 2 seconds

    % Solve for system dynamics using ode45
    [t, q] = ode45(@(t, q) cart_pendulum_dynamics(t, q, M, m, L, g), tspan,
[q0; dq0]);

     % Extract states
    x = q(:, 1);
    theta = q(:, 2);

    % Plot x(t) and theta(t)
    figure;
    subplot(2, 1, 1);
    plot(t, x);
    xlabel('Time (s)');
    ylabel('x (m)');
    title('Cart Position over Time');
    grid on;

    subplot(2, 1, 2);
    plot(t, (theta-pi));
    xlabel('Time (s)');
    ylabel('\theta (rad)');
    title('Pendulum Angle over Time');
    grid on;

    % Animate the motion
    animate_cart_pendulum(t, q(:, 1), q(:, 2), L);
end

function dqdt = cart_pendulum_dynamics(t, q, M, m, L, g)
    % Extract states
    x = q(1);
    theta = q(2);
    dx = q(3);
    dtheta = q(4);

    % Compute system dynamics: These are values taken from 2b.
    D = [(M + m), m*L*cos(theta);
         m*L*cos(theta), m*L^2];

    C = [-m*L*dtheta^2*sin(theta); 0];

    G = [0; m*g*L*sin(theta)];

    tau = [0; 0]; % No control input for now

    % Solve for accelerations
    ddq = D \ (tau - C - G);

    % Construct dqdt
    dqdt = [dx; dtheta; ddq];
end

function animate_cart_pendulum(t, x, theta, L)
    % Animation parameters
    fps = 30; % Frames per second
    dt = 1 / fps;

    % Create figure
    figure;
    axis([-10*L, 10*L, -10*L, 10*L]);
    xlabel('x');
    ylabel('y');
    title('Cart-Pendulum Animation');

    % Plot cart-pendulum
    cart_width = 0.2;
    cart_height = 0.1;
    pendulum_radius = 0.02;

    cart = rectangle('Position', [x(1) - cart_width/2, -cart_height/2,
cart_width, cart_height], 'Curvature', [0.1, 0.1], 'FaceColor', 'b');
    hold on;
    pendulum = plot([x(1), x(1) + L*sin(theta(1))], [0, -L*cos(theta(1))],
'r', 'LineWidth', 2);
    hold off;
    axis equal;
    grid on;
    pause(1);

    % Video creation
    v = VideoWriter("cart_pendulum_simulation.mp4", 'MPEG-4');
    v.FrameRate = 30; % Set frame rate
    open(v);

    % Animate motion
    for i = 1:length(t)
```

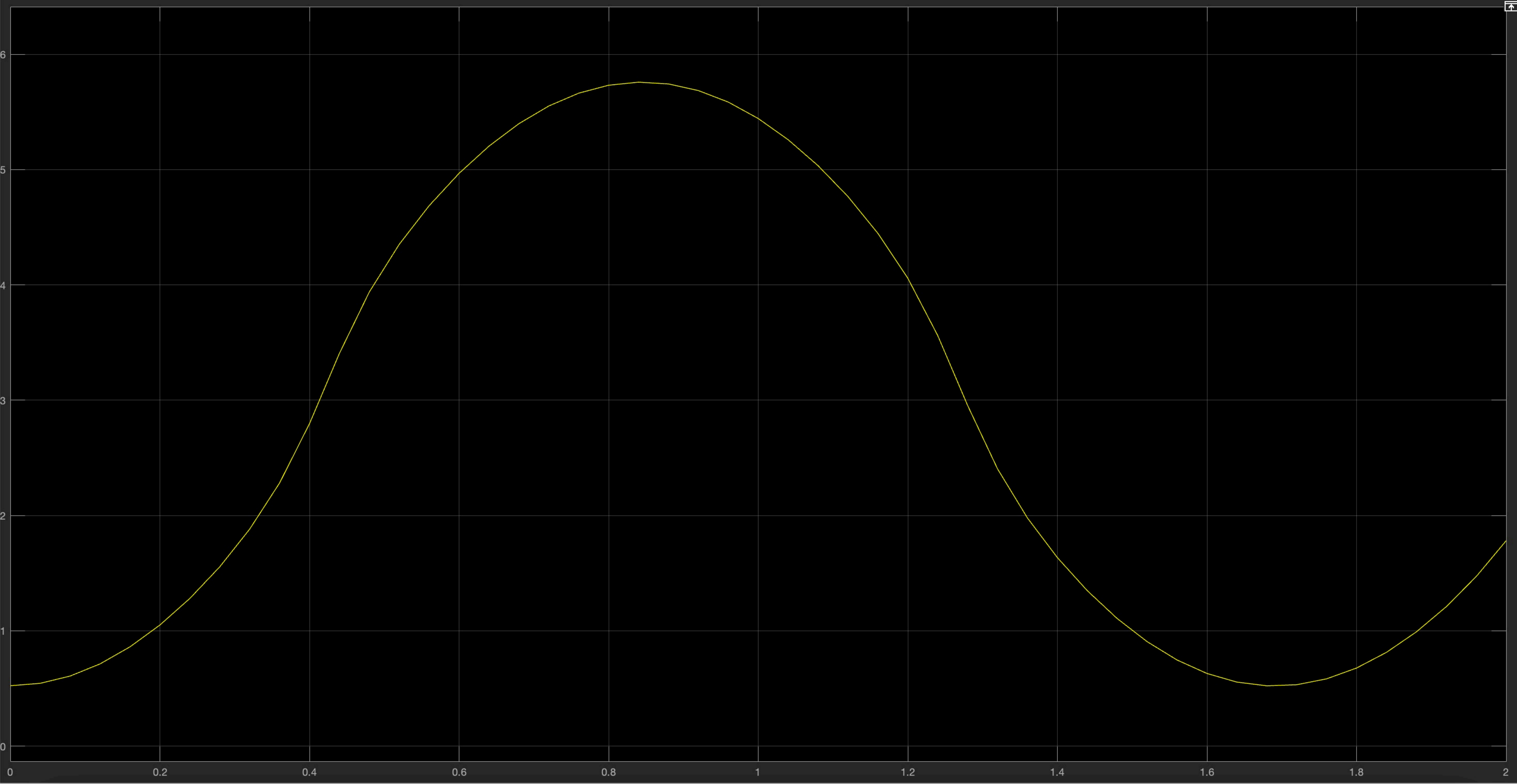
```
        % Update cart and pendulum position
        set(cart, 'Position', [x(i) - cart_width/2, -cart_height/2,
cart_width, cart_height]);
        set(pendulum, 'XData', [x(i), x(i) + L*sin(theta(i))], 'YData', [0,
-L*cos(theta(i))]);

        % Refresh plot
        drawnow;

        % Pause to achieve desired frame rate
        pause(dt);
        % Write frame to video
        frame = getframe(gcf);
        writeVideo(v, frame);
    end
    close(v)
end
```





Published with MATLAB® R2023b


PS-Simulink Converter2
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


6
5
4
3
2
1
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Praful Sigdel

AME 556: Robot Dynamics and Control Problem 2 b

```
function ddq = computeRobotDynamicsSymbolic()

% Symbolic variables
syms x theta dx dtheta q dq real;
syms M m L g real;

q = [x; theta];
dq = [dx; dtheta];

% Kinetic Energy
T = 0.5 * (M + m) * dx^2 + 0.5 * m * L^2 * dtheta^2 + m * L * dx * dtheta *
cos(theta);

% Potential Energy
U = -m*g*L*cos(theta);

% Lagrangian
L = simplify(T - U);

%control input
tau = [0;0];

fq_dq = (jacobian(L,dq))';

D_q = jacobian(fq_dq,dq);

Cq_dq = jacobian(fq_dq,q)*dq - (jacobian(T,q))';

g_q = (jacobian(U,q))';

Nq_dq = Cq_dq + g_q;

ddq = (D_q)\(tau - Nq_dq);

end
```

```
ans =

             (m*sin(theta)*(L*dtheta^2 + g*cos(theta)))/(- m*cos(theta)^2 + M
+ m)
-(sin(theta)*(L*m*cos(theta)*dtheta^2 + M*g + g*m))/(L*(- m*cos(theta)^2 + M
+ m))
```

Published with MATLAB® R2023b

```
function animate_quadruped(t, q, pF1, pF2, pF3, pF4)
    % Extract states
    x = q(:, 1);
    y = q(:, 2);
    z = q(:, 3);
    R_vec = q(:, 4:12);

    % Define body dimensions
    c = 0.3; % m
    b= 0.5; % m
    a = 0.15; % m

    vert1 = [b/2;c/2;a/2];
    vert2 = [b/2;-c/2;a/2];
    vert3 = [b/2;-c/2;-a/2];
    vert4 = [b/2;c/2;-a/2];
    vert5 = [-b/2;c/2;a/2];
    vert6 = [-b/2;-c/2;a/2];
    vert7 = [-b/2;-c/2;-a/2];
    vert8 = [-b/2;c/2;-a/2];

    % Define foot positions relative to inertial frame
    foot_positions = [pF1, pF2, pF3, pF4];

    % Create figure and axis
    figure;
    axis equal;
    axis([-0.5, 0.5, -0.5, 0.5, 0, 0.5]);
    xlabel('X (m)');
    ylabel('Y (m)');
    zlabel('Z (m)');
    title('Quadruped Robot Animation');
    hold on;
    view(3);

    % Animation parameters
    fps = 30; % Frames per second
    dt = 1 / fps;

    % Video creation
    v = VideoWriter("Quadruped_simulation.mp4", 'MPEG-4');
    v.FrameRate = 30; % Set frame rate
    open(v);



    % Animate robot motion
    for i = 1:length(t)
        % Calculate rotation matrix
        R = reshape(R_vec(i,:),[3,3]);

        % Pause to control animation speed
        pause(0.1);

        % Clear previous frame
        clf;
        hold on;

        vert1world = transformPointToWorldFrame(vert1,R,[x(i);y(i);z(i)]);
        vert2world = transformPointToWorldFrame(vert2,R,[x(i);y(i);z(i)]);
        vert3world = transformPointToWorldFrame(vert3,R,[x(i);y(i);z(i)]);
        vert4world = transformPointToWorldFrame(vert4,R,[x(i);y(i);z(i)]);
        vert5world = transformPointToWorldFrame(vert5,R,[x(i);y(i);z(i)]);
        vert6world = transformPointToWorldFrame(vert6,R,[x(i);y(i);z(i)]);
        vert7world = transformPointToWorldFrame(vert7,R,[x(i);y(i);z(i)]);
        vert8world = transformPointToWorldFrame(vert8,R,[x(i);y(i);z(i)]);

        body_vertices = [vert1world,vert2world,vert3world,vert4world,...
            vert5world, vert6world, vert7world, vert8world];

        body_faces = [1 2 3 4;1 5 8 4;1 2 6 5;2 3 7 6;5 6 7 8;3 4 8 7];

        %Plot feet
        for i = 1:size(foot_positions, 2)
            foot_position = foot_positions(:, i);
            plot3(foot_position(1), foot_position(2), foot_position(3), 'ro',
'MarkerSize', 10, 'MarkerFaceColor', 'r');
            quiver3(foot_position(1), foot_position(2),
foot_position(3),0,0,0.25525,0,'LineWidth',2,'Color','b');
        end

        axis equal;
        axis([-0.5, 0.5, -0.5, 0.5, -0.5, 0.5]);
        xlabel('X (m)');
        ylabel('Y (m)');
        zlabel('Z (m)');
        title('Quadruped Robot Animation');
        view(3);
        % Plot body
        patch('Vertices', body_vertices', 'Faces', body_faces, 'FaceColor',
'r', 'EdgeColor', 'k');
        pause(dt);
        % Write frame to video
        frame = getframe(gcf);
        writeVideo(v, frame);
    end
    close(v);
end

function p_world = transformPointToWorldFrame(p_body, R, d)
    % Construct homogeneous transformation matrix
    T = eye(4);
    T(1:3, 1:3) = R; % Set rotation part of the matrix
    T(1:3, 4) = d;   % Set translation part of the matrix

    % Apply homogeneous transformation
    p_homogeneous = [p_body; 1]; % Convert to homogeneous coordinates
    p_world_homogeneous = T * p_homogeneous;

    % Convert back to Cartesian coordinates
    p_world = p_world_homogeneous(1:3);
end
```

```
Not enough input arguments.

Error in animate_quadruped (line 3)
    x = q(:, 1);
```

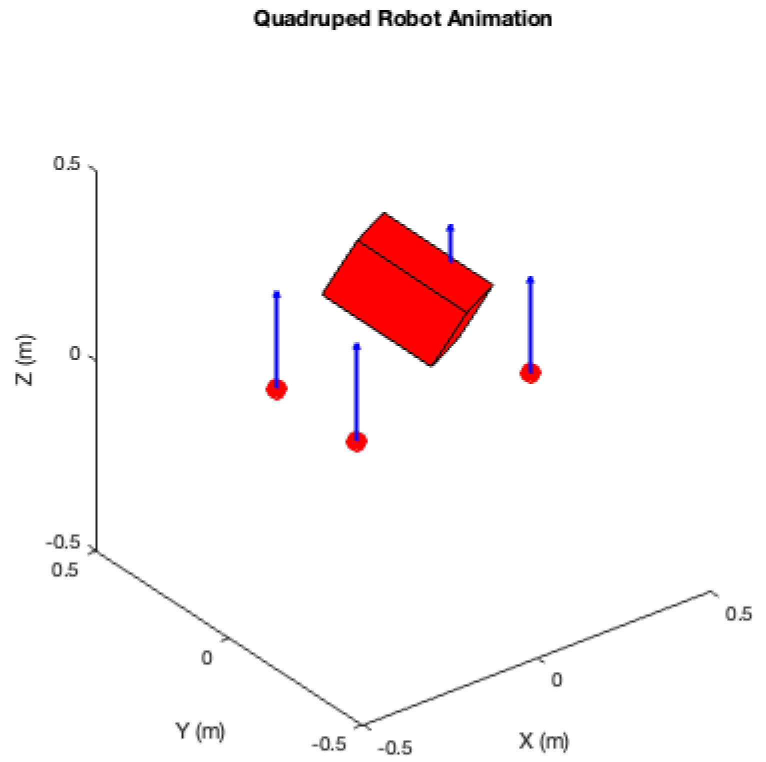
Published with MATLAB® R2023b

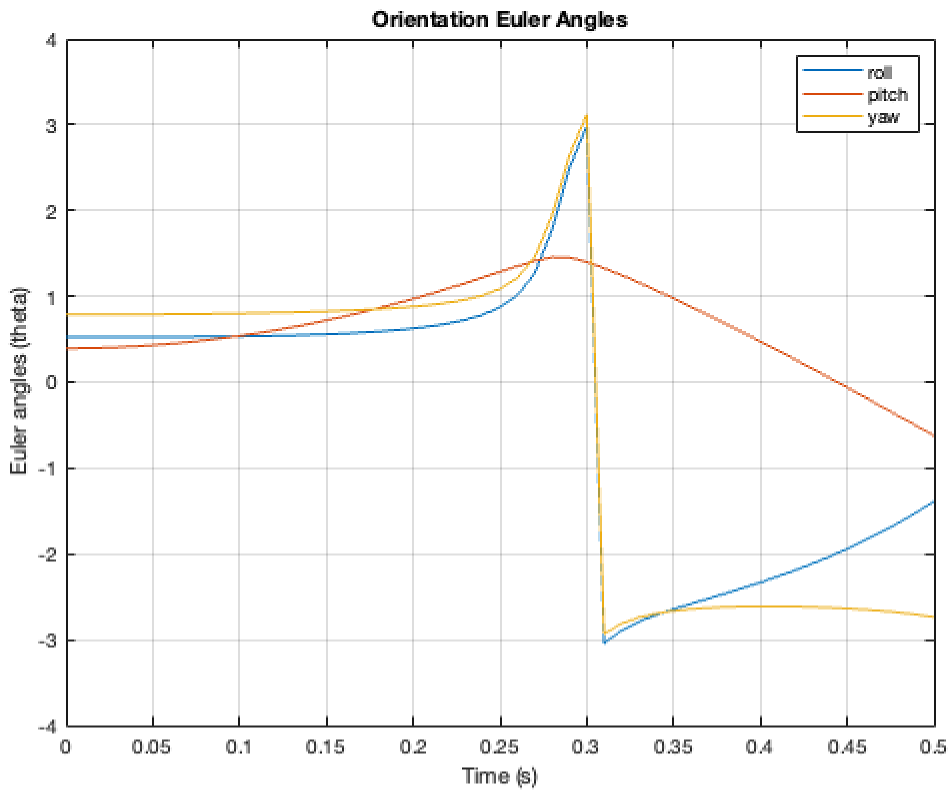
```
function dqdt = quadruped_dynamics(t, q, m, g, Ib, Fi, pF)
    % Unpack state variables
    x = q(1);
    y = q(2);
    z = q(3);
    dx = q(13);
    dy = q(14);
    dz = q(15);
    omega_xb = q(16);
    omega_yb = q(17);
    omega_zb = q(18);


    % Define rotation matrix from body-fixed frame to inertial frame
    R = q(4:12);

    R = reshape(R, [3, 3]);

    % Compute net force and torque
    F_total = sum(Fi, 2) - [0; 0; m * g]; % Net force
    tau_total = zeros(3, 1);

    for i = 1:size(Fi, 2)
        r = pF(:,i) - [x;y;z];
        tau_total = tau_total + cross(r, Fi(:, i));
    end

    tau_bf = R.'*tau_total;

    % Compute linear acceleration of center of mass
    ddq_linear = F_total / m;

    % Compute angular acceleration
    omega_dot_b = Ib \ (tau_bf - cross([omega_xb; omega_yb; omega_zb], Ib *
[omega_xb; omega_yb; omega_zb]));

    % Compute rotation matrix derivative
    R_dot = R * [0, -omega_zb, omega_yb; omega_zb, 0, -omega_xb; -omega_yb,
omega_xb, 0];
    R_dotshape = reshape(R_dot,[9,1]);
    % Combine linear and angular accelerations into dqdt vector
    dqdt = [dx; dy; dz;R_dotshape; ddq_linear; omega_dot_b];
end
```

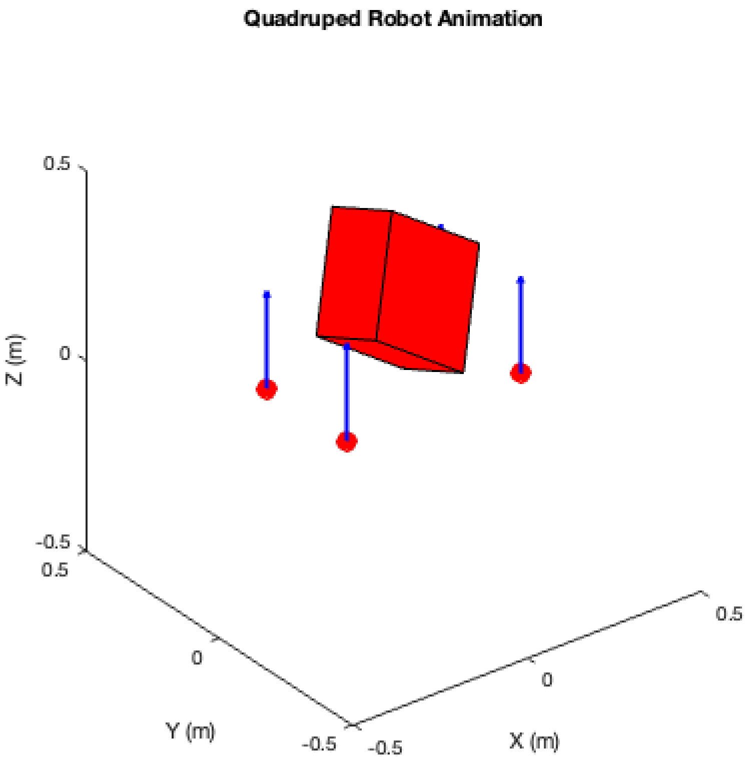
```
Not enough input arguments.

Error in quadruped_dynamics (line 3)
    x = q(1);
```

Published with MATLAB® R2023b

```matlab
% Define parameters
a = 0.15; % m
b = 0.5;  % m
c = 0.3;  % m
m = 10;   % kg
g = 9.81; % m/s^2

% Moment of inertia
Ib = [1/12*m*(a^2+c^2), 0, 0;
      0, 1/12*m*(a^2+b^2), 0;
      0, 0, 1/12*m*(b^2+c^2)];

% Control inputs
Fi = repmat([0; 0; m*g/4], 1, 4);

% Foot positions
pF1 = [0.25; 0.15; 0];
pF2 = [0.25; -0.15; 0];
pF3 = [-0.25; 0.15; 0];
pF4 = [-0.25; -0.15; 0];

pF = [pF1,pF2,pF3,pF4];
%
euler_angles = [pi/6; pi/8; pi/4]';
X = eul2rotm(flip(euler_angles));
R = reshape(X, [9,1]);
% Initial conditions
q0 = [0.05; 0.05; 0.2; R; 0; 0; 0; 0; 0; 0];

% Time vector
tspan = 0:0.01:0.5;

% Simulate dynamics
[t, q] = ode45(@(t, q) quadruped_dynamics(t, q, m, g, Ib, Fi, pF), tspan, q0);


eul_ang_ext = zeros([size(q,1),3]);
Rout = q(:,4:12);
for i = 1:size(q,1)
    ROut = reshape(Rout(i,:),[3,3]);
    eul_ang_ext(i,:) = flip(rotm2eul(ROut));
end
figure;
plot(t, eul_ang_ext);
legend('roll','pitch','yaw');
xlabel('Time (s)');
ylabel('Euler angles (theta)');
title('Orientation Euler Angles');
grid on;

% Animate motion
animate_quadruped(t, q, pF1, pF2, pF3, pF4);
```


Quadruped Robot Animation
Z (m)
Y (m)
X (m)
0.5
0
-0.5
0.5
0
-0.5
-0.5
0
0.5


Orientation Euler Angles
roll
pitch
yaw
Euler angles (theta)
Time (s)
4
3
2
1
0
-1
-2
-3
-4
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5



Published with MATLAB® R2023b