Task Instructions

In this task, we will be implementing Pole Placement Controller and the LQR controller on different types of physical systems. You will be required to find out the equations of motion for each of the physical systems and modify code as instructed in order to simulate each of the physical systems in a proper manner. You are also required to answer a set of questions relating to each physical system so as to ensure us that you have thoroughly understood the task given.

Physical System 1 - Simple Pendulum

We will revisit the Simple Pendulum(with external torque) once again.

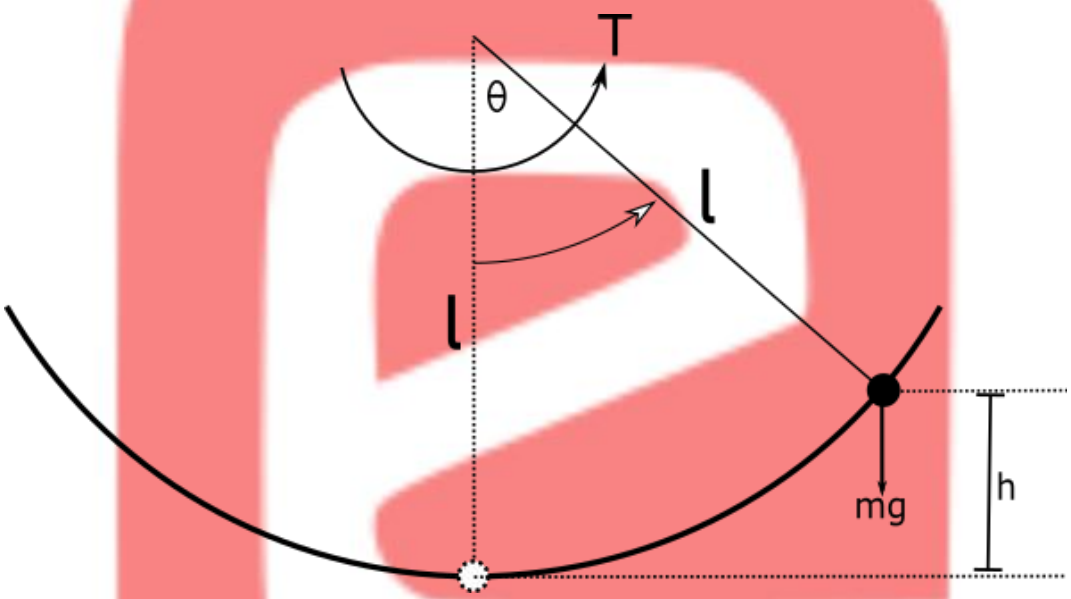

Figure 1: Simple Pendulum (with external torque)

We also derived the equations of motion of this system as:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{-g}{l}\sin x_1 + \frac{1}{ml^2}T$$

Open the file ***Simple_Pendulum.m*** using Octave.
In this file you will find the following functions defined:

- *draw_pendulum()*
- *pendulum_dynamics()*
- *sim_pendulum()*
- *pendulum_AB_matrix()*
- *pole_place_pendulum()*
- *lqr_pendulum()*
- *simple_pendulum_main()*

draw_pendulum():
This function is used to draw the pendulum on a 2D plot. **You are not allowed to make any changes to this function.**

pendulum_dynamics():
This function is used to define the dynamics of the system by using the equations of motion you have derived for this system.

```
1 function dy = pendulum_dynamics(y, m, L, g, u)
2   sin_theta = sin(y(1));
3   cos_theta = cos(y(1));

4   dy(1,1) = y(2);
5   dy(2,1) = ;
6 endfunction
```

In this function,
y - denotes the state vector. The state vector consists of 2 state variables y(1) and y(2).
y(1) corresponds to x_1 (or theta) in the system of equations.
y(2) corresponds to x_2 (or theta_dot) in the system of equations.
dy(1,1) corresponds to $\dot{x}_1$ (or theta_dot) in the system of equations.
dy(2,1) corresponds to $\dot{x}_2$ (or theta_dot_dot) in the system of equations.
m is the mass of pendulum bob.
g is the acceleration due to gravity.
L is the length of pendulum bob.
u is the input to the system.

Line 4 denotes the first equation of motion for the system of equations
You are required to complete Line 5 by filling in the second equation of motion in the space provided.

sim_pendulum():
This function simulates the Simple Pendulum with no input.

```
1 function [t,y] = sim_pendulum(m, g, L, y0)
2   tspan = 0:0.1:10;          ## Initialize time step
3   u = 0;                     ## No Input
4   [t,y] = ode45(@(t,y)pendulum_dynamics(y, m, L, g, u),tspan,y0);
5 endfunction
```

y0 is the initial condition of the system.
Line 4 integrates the differential equation defined in the pendulum_dynamics() function across the length of tspan with initial condition y0. Line 4 returns an output [t, y].
t is the time step array.

y is the solution array where each element in y is the solution of the first order differential equation and corresponds to an element in t.
[t,y] is returned as output of the sim_pendulum() function.

You are not allowed to make any changes to this function.

pendulum_AB_matrix():
This function returns the A and B matrices for the system.

```
1 function [A,B] = pendulum_AB_matrix(m, g, L)
2   A = [0 1; g/L 0];
3   B = ;
4 endfunction
```

Declare the A and B matrices for the Simple Pendulum system.

pole_place_pendulum()
This function simulates the Simple Pendulum with **u = -Kx** input. K matrix is calculated using pole placement.

```
1 function [t,y] = pole_place_pendulum(m, g, L, y_setpoint, y0)
2   [A,B] = ;
3   eigs =  ;
4   K = ;

5   tspan = 0:0.1:10;
6   [t,y] = ode45(@(t,y)pendulum_dynamics(y, m, L, g,
-K*(y-y_setpoint)),tspan,y0);
7 endfunction
```

You are required to make the following changes in this function:
Line 2 - Use pendulum_AB_matrix() to return A and B matrix
Line 3 - Initialize a 2x1 matrix with two eigenvalues. The eigenvalues should be selected so that the system is stable.
Line 4 - Calculate the K matrix. Use the ***place()*** function in octave to calculate K.

y0 is the **initial state** or **initial condition** of the system. Which means the system will start from this state.
y_setpoint is the final required state of the system. The pole placement controller will try to drive the system to this state. Here the y_setpoint is (π,0) which is an equilibrium point
Line 6 integrates the differential equation defined in the pendulum_dynamics() function across the length of tspan with initial condition y0 and u = -Kx

lqr_pendulum()

This function simulates the Simple Pendulum with **u = -Kx** input. K matrix is calculated using LQR.

```
1 function [t,y] = lqr_pendulum(m, g, L, y_setpoint, y0)
2   [A, B] = ;                  ## Initialize A and B matrix
3
4   Q = ;                       ## Initialize Q matrix
5   R = ;                       ## Initialize R

6   K = ;                   ## Calculate K matrix from A,B,Q,R matrices

7   tspan = 0:0.1:10;     ## Initialise time step
8  [t,y] = ode45(@(t,y)pendulum_dynamics(y, m, L, g,
-K*(y-y_setpoint)),tspan,y0);
11 endfunction
```

You are required to make the following changes in this function:
Line 2 - Use pendulum_AB_matrix() to return A and B matrix
Line 4 - Choose an appropriate Q matrix
Line 5 - Choose an appropriate value of R.
Line 6 - Calculate the K matrix using the *lqr()* function in octave.

y0 is the **initial state** or **initial condition** of the system. Which means the system will start from this state.
y_setpoint is the final required state of the system. The LQR controller will try to drive the system to this state. Here the y_setpoint is (π, 0) which is an equilibrium point

The last two lines are same as pole_place_pendulum(). Line 8 integrates the differential equation defined in the pendulum_dynamics() function across the length of tspan with initial condition y0 and u = -Kx

simple_pendulum_main()

This function is used for testing our code by calling the various functions.

```
1 function simple_pendulum_main()
2    m = 1;
3    g = 9.8;
4    L = 1;
5    y_setpoint = [pi;0];
6    y0 = [pi/6; 0];

7    [t,y] = sim_pendulum(m,g,L, y0);       ## Test Simple Pendulum
8  ##  [t,y] = pole_place_pendulum(m,g,L, y_setpoint, y0); ## Test
   Simple Pendulum with Pole Placement Controller
```

```
9  ##  [t,y] = lqr_pendulum(m,g,L, y_setpoint, y0);        ## Test
   Simple Pendulum with LQR Controller

10 for k = 1:length(t)
     draw_pendulum(y(k, :), L);
   endfor
  endfunction
```

This function can be used to test out the 3 functions sim_pendulum(), lqr_pendulum() and pole_place_pendulum() individually by uncommenting one function and commenting out the rest.
You can modify the system parameters like mass, gravity, length of pendulum etc.
Line 10 - runs a for-loop to animate the pendulum behavior on a 2D plot.

The required behavior of the pendulum due to the three functions can be seen here at this link.

After you have completed the code, open **Think_and_Answer.docx** and answer all the questions in section 1.

Note: In order to execute ***simple_pendulum_main()*** from the Octave command window, first run the ***Simple_Pendulum*** script in the command window. This will enable Octave to recognize the functions defined in Simple_Pendulum. These functions can then be directly called from the command window.

Let's move on to the second physical system.

Physical System 2 - Mass Spring System

We will examine a new system now. In the given figure we have a Mass Spring system.

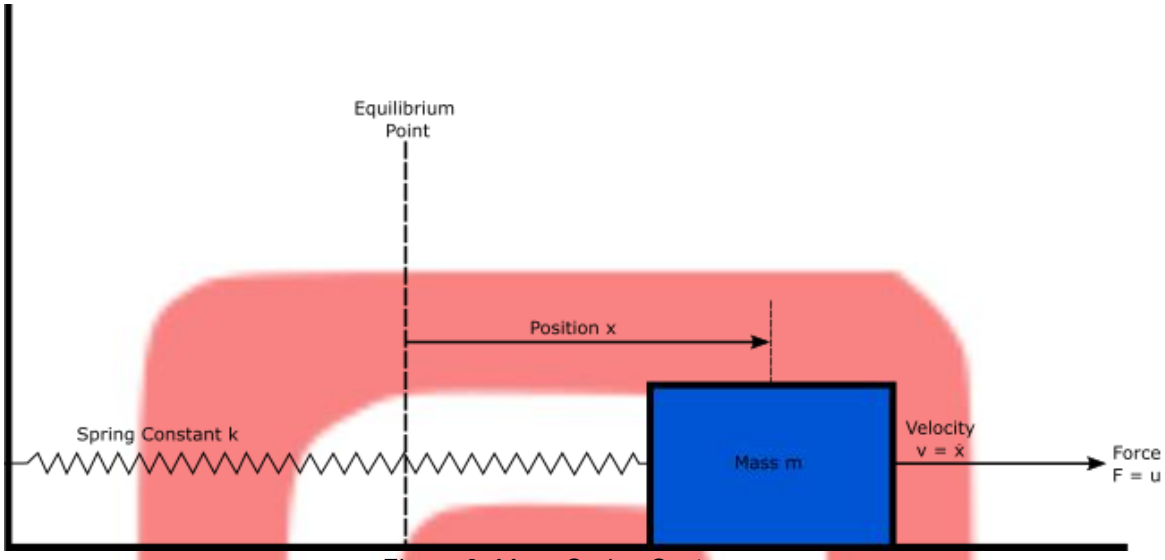

Figure 2: Mass Spring System

The mass-spring system has block of ***mass m*** which is connected to wall with a spring of ***spring constant k***. The equilibrium point is the position where the spring is neither expanded nor contracted. The ***position*** of mass with respect to equilibrium point is denoted by x. A ***force u*** is acting on the the mass in horizontal direction. The velocity of the mass is denoted by $\dot{x}$.

You are required to do the following:

1. Derive the equations of motion for this system using the Euler-Lagrangian method we discussed earlier. The state variables for this system will be the position of the mass $x_1 = x$ and velocity of mass $x_2 = \dot{x}$.
2. Derive the A and B matrices for the system.
3. Open ***Mass_Spring_System.m***. In this file you will find the following functions:
    - *draw_mass_spring()*
    - *mass_spring_dynamics()*
    - *sim_mass_spring()*
    - *mass_spring_AB_matrix()*
    - *pole_place_mass_spring()*
    - *lqr_mass_spring()*
    - *mass_spring_main()*

    a) Do no edit *draw_mass_spring()* function.

    b) *mass_spring_dynamics()* determines the dynamics of the system. You will need to fill in the equations of motion you have derived in this function (similar to as you did in *pendulum_dynamics()* function).

    c) *sim_mass_spring()* is similar to *sim_pendulum()*. It will simulate the behaviour of the system under condition of no input.

d) *mass_spring_AB_matrix()* returns the A and B matrix of the system. You need to fill in the A and B matrices you have derived.
e) *pole_place_mass_spring()* and *lqr_mass_spring()* are similar to *pole_place_pendulum()* and *lqr_pendulum()*. You need to complete the code in a similar way.
f) *mass_spring_main()* can be called from command window to test the behaviour of the system. All the constants are defined in this function. **y0** is the **initial state** or **initial condition** of the system. Which means the system will start from this state. Here y0 is [-0.3;0] which means the initial position of block is -0.3 and initial velocity is 0.
g) **y_setpoint** is the final required state of the system. The pole placement or LQR controller will try to drive the system to this state. Here final required state is [0.7; 0]. That means the final position of block should be 0.7 and final velocity should be 0.

You can view the expected behaviour of the system due to the three functions here.

Now, open **Think_and_Answer.docx** and answer all the questions in section 2.

Physical System 3 - Simple Pulley System

The third system is given as follows:

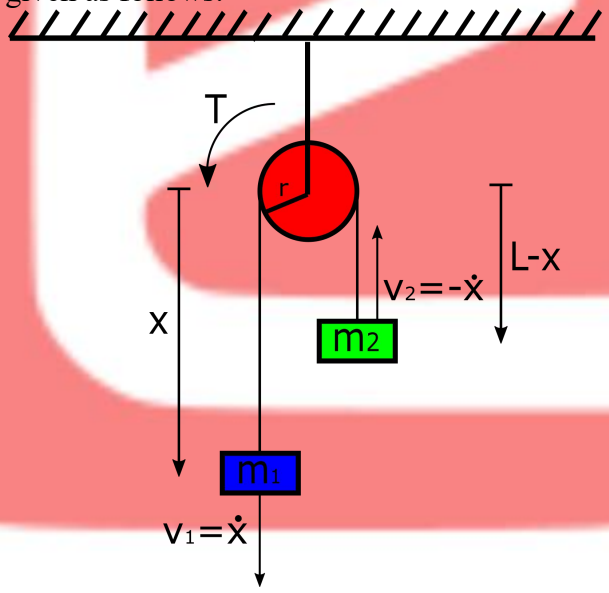

Figure 3: Simple Pulley System

A Simple Pulley system is shown. A pulley of ***radius r*** hangs from the ceiling.
The pulley is considered mass-less. There are two blocks of ***mass*** m_1 and m_2 which hang on either side of the pulley connected by a in-extensible string of ***length L***. The ***position*** of mass m_1 with respect to pulley is ***x*** and ***position*** of m_2 with respect to pulley is ***L-x***. The ***velocity*** of m_1 is considered to be $\dot{x}$ (downward direction is taken as positive). The ***velocity*** of m_2 is considered to be $-\dot{x}$. A torque T is applied as input to the system.

You are required to do the following:

1. Derive the equations of motion for this system. Make sure your system of equations are dimensionally consistent. The state variables for this system will be the position of the mass (with respect to pulley) $x_1 = x$ and velocity of mass $x_2 = \dot{x}$.
2. Derive the A and B matrices of the system.
3. Open the file ***Simple_Pulley.m.*** In this file you will find the following functions:
   - *draw_pulley()*
   - *pulley_dynamics()*
   - *sim_pulley()*
   - *pulley_AB_matrix()*
   - *pole_place_pulley()*
   - *lqr_pulley()*
   - *simple_pulley_main()*

   a) Do not edit the *draw_pulley()* function.

   b) The rest of the functions are similar to their corresponding functions in the last two cases. That is, *pulley_dynamics()* functions similar to *mass_spring_dynamics()* and *pendulum_dynamics()* and so on. Hence these functions need to completed according to the earlier instructions.

   c) *Simple_pulley_main()* can be called from command window to test the behaviour of the system. **y_setpoint** and **y0** are defined here.

You can view the expected behaviour of the system due to the three functions here.

Now, open **Think_and_Answer.docx** and answer all the questions in section 3.

Physical System 4 - Complex Pulley System

The fourth system is given as follows:

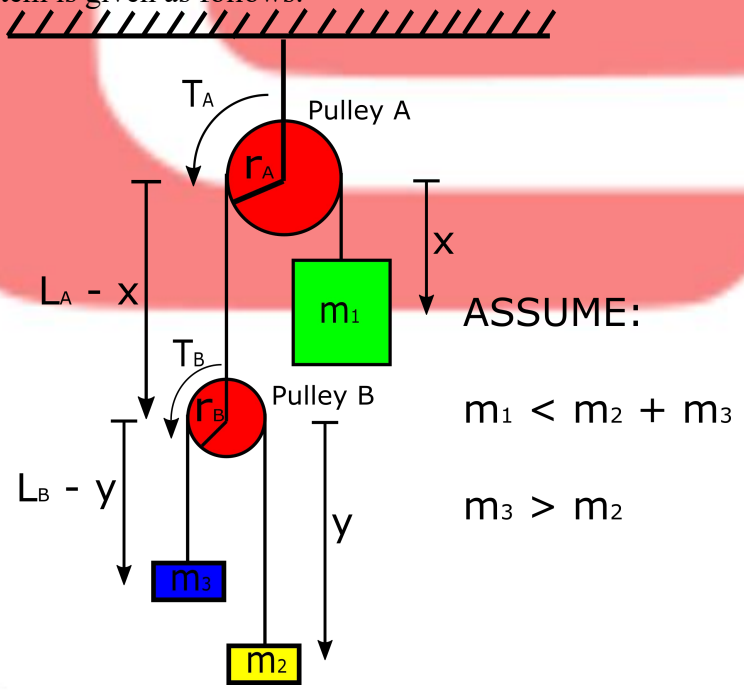

Figure 4: Complex Pulley System

A complex pulley system is shown.

Pulley A with **radius** r_a hangs from a ceiling. A block of **mass** m_1 hangs from one side of the pulley A. On the other side of the pulley A, another Pulley B is hanging.

Pulley B has radius r_b **.** Pulley B and mass m_2 are connected by an in-extensible string of length L_A. Mass m_2 and m_3 are hanging on opposite sides of pulley B and connected by another in-extensible string of length L_B.

The position of m_1 with respect to pulley A is ***x***. The position of m2 with respect to pulley B is ***y***.

We have made the following assumptions related to the masses m_1, m_2 and m_3.

$$m_1 < m_2 + m_3$$
$$m_3 > m_2$$

There are two torques T_A and T_B applied to pulley A and pulley B respectively as input.

You are required to do the following:

1. Derive the equations of motion for this system. Make sure your system of equations are dimensionally consistent. The state variables for this system will be:
   - Position of mass m_1 with respect to pulley A, $x_1 = x$
   - Velocity of mass m_1 with respect to pulley A, $x_2 = \dot{x}$
   - Position of mass m_2 with respect to pulley B, $x_3 = y$
   - Velocity of mass m_2 with respect to pulley B, $x_4 = \dot{y}$
2. Derive the A and B matrices of the system. Keep in mind that the number of state variables and inputs in this system is 4 and 2 respectively (as opposed to 2 state variables and 1 input in earlier systems). Hence the dimensions of A and B system will be different in this case.
3. Open the file **Complex_Pulley.m**.In this file, you will find the following functions:
   - *draw_complex_pulley()*
   - *complex_pulley_dynamics()*
   - *sim_complex_pulley()*
   - *complex_pulley_AB_matrix()*
   - *pole_place_complex_pulley()*
   - *lqr_complex_pulley()*
   - *complex_pulley_main()*

   a) Do not edit the *draw_complex pulley()* function.

   b) The rest of the functions need to be completed in similar way as earlier systems.

   c) *Complex_pulley_main()* can be called from command window to test the behaviour of the system. **y_setpoint** and **y0** are defined here.

You can view the expected behaviour of the system due to the three functions here.

Open **Think_and_Answer.docx** and answer all the questions in section 4.

Physical System 5 - Inverted Pendulum System

The fifth system is given as follows:

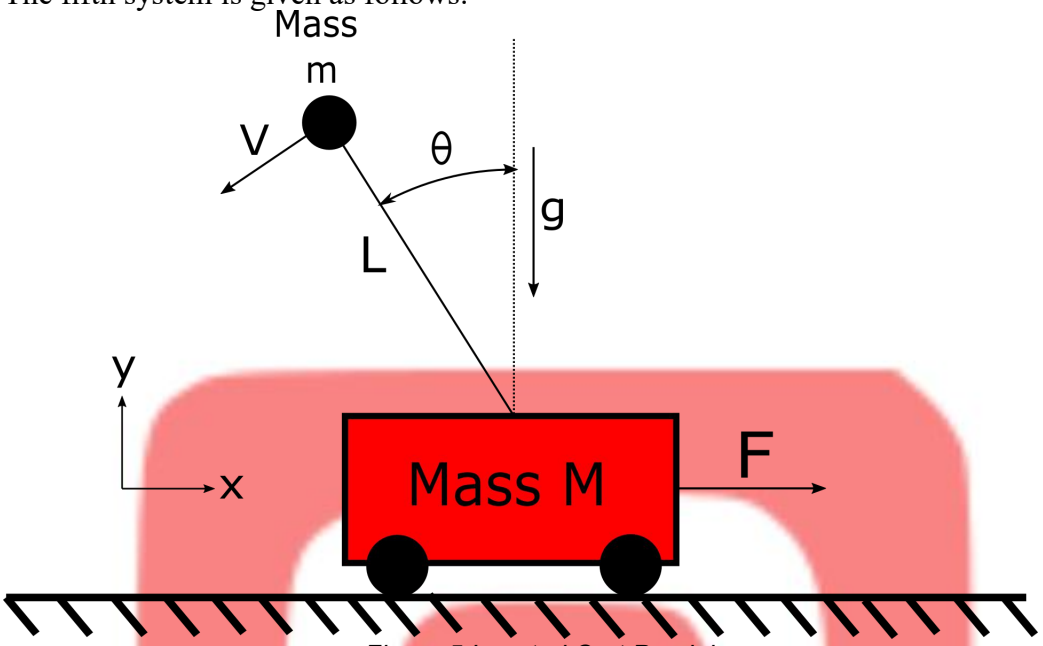

Figure 5:Inverted Cart Pendulum

This is an **Inverted Cart Pendulum** system (on which a balance bot is based on).

A cart with **mass *M*** rests on a friction-less surface. A pendulum bob of **mass *m*** is suspended vertically upwards and connected to the cart with a rigid rod of **length *L***. The **angle** with respect to vertical is ***θ***. ***v*** is the **tangential velocity** of the pendulum bob. A horizontal force F is applied on the cart.

You are required to do the following:

1. Derive the equations of motion for this system. Make sure your system of equations are dimensionally consistent. The state variables for this system will be:
   - Horizontal position of cart, $x_1 = x$
   - Horizontal velocity of cart, $x_2 = \dot{x}$
   - Angle of inverted pendulum with respect to vertical axis, $x_3 = \theta$
   - Angular velocity of pendulum bob, $x_4 = \dot{\theta}$

2. Derive the A and B matrices of the system. Keep in mind that the number of state variables and inputs in this system is 4 and 1 respectively (as opposed to 2 state variables and 1 input in earlier systems). Hence the dimensions of A and B system will be different in this case.

3. Open the Cart_Pendulum.m. In this function, you will find the following functions:
   - *draw_cart_pendulum()*
   - *cart_pendulum_dynamics()*
   - *sim_cart_pendulum()*
   - *cart_pendulum_AB_matrix()*
   - *pole_place_cart_pendulum()*
   - *lqr_cart_pendulum()*
   - *cart_pendulum_main()*

   a) Do not edit the *draw_cart_pendulum()* function.
   b) The rest of the functions need to be completed in similar way as earlier systems.
   c) *cart_pendulum_main()* can be called from command window to test the behaviour of the system. **y_setpoint** and **y0** are defined here.

You can view the expected behaviour of the system due to the three functions here.

Now, open **Think_and_Answer.docx** and answer all the questions in section 5.

That's it !! Task 1.2 is complete!! You can move on to the Submission Instructions given in Read Me pdf to complete submission of task.