# Task 2

## Why this task?

Any self-balancing vehicle is commonly modeled after some variant of the “Inverted Pendulum” dynamical system. This bike is also a self-balancing bike and thus it becomes essential to learn, observe & experiment with the dynamics of the physical system that the bike is based on, i.e. **" Rotary Inverted Pendulum "**

Task 2 is further divided into two parts.

## Task 2A - Mathematical Modeling

This is the first step, which will walk us through the steps to derive dynamics equations of the system, analyzing the behavior of the system from the math and applying appropriate control strategy on the system to balance the system in an inherently unstable position. The software that we learnt in the Task 1 should help us make this easier - **" Octave "**

## Task 2B - Intro to CoppeliaSim

After we have the math solved, we should simulate the system’s behavior and visually observe how the dynamics behaves under gravity. For this we’ll have to use a simulator equipped with physics engine, and a perfect fit for this would be **" CoppeliaSim "**

There are various control strategies that can be used to balance the system. We’ll learn a popular one called **" Linear Quadratic Regulator "** in the Task 2A. But, to be able to fine tune it, we should observe the simulated behavior, and this is where CoppeliaSim becomes necessary. This will make it easier to deal with the actual hardware system.

### Try googling :

•Kinematics vs Dynamics?
•What are degrees of freedom ? How many does a car or a drone have ? What about Rotary Pendulum then?
•Newtonian vs Lagrangian Mechanics?
•What are Rotation Matrices?
•What are open loop & closed loop systems?
•What is a state space model ? How is it different from transfer function ?
•What are linear & Non-linear systems?
•What is an Under-Actuated System?

**References :**

- Wikipedia Link - Inverted Pendulum
- Furata Wikipedia Link - Furata Pendulum
- LQR Basics - Youtube Playlist
- PID Basics - Youtube Playlist
- Control Bootcamp - Youtube Playlist

# Task 2

## Mathematical Modeling of a system

In the previous Task, we had discussed about the various steps involved in testing the stability of a system. In this document, we will be discussing the **Euler-Lagrange method** to derive the equations of motion of a given system.

Before you start, you might need to recapitulate a few topics if you want to fully understand what we are going to explain here.

You will need a good understanding of classical mechanics. “Concepts of Physics by Prof. H.C. Verma”

is a great place to brush up on those concepts.

You will also need to understand mathematical concepts like partial differentiation, jacobians, solving equations with two or more variables etc.

## Euler-Lagrange Method

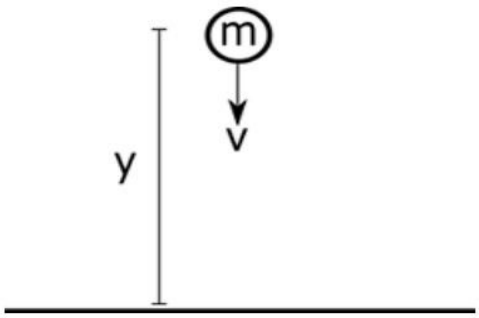
The Euler-Lagrange method states that the equations of motion of a system can be obtained by solving the following equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = 0$$

(Assuming that there are no non-conservative forces acting on the system)
Here,
L is the Lagrangian which is the difference between the Kinetic energy and Potential energy of the system.
Hence,

$$L = K.E - P.E$$

x and $\dot{x}$ are the state variables (In our case here, position and velocity respectively) in generalized coordinate system.
We will try to understand this using an example.
Consider the following system:




A point mass m is raised to height y. We need to calculate the equations of motion using the Euler-Lagrange method.

Firstly we calculate the KE and PE. Then use those values to calculate the Lagrangian L.

# Task 2

$$PE = mgy \qquad (1)$$

$$KE = \frac{1}{2}mv^2 = \frac{1}{2}m\dot{y}^2 \qquad (2)$$

$$L = KE - PE = \frac{1}{2}m\dot{y}^2 - mgy \qquad (3)$$

Equation (1) is self explanatory. The mass is raised to height y. So the potential energy stored in mass will be **mgy**.

Equation (2) is slightly tricky to understand. We know that kinetic energy of a point mass is (1/2) x mass x (velocity)^2. Now velocity v is nothing but rate of change of y with respect to t. Hence v can be written as **dy/dt** or **ẏ**

Equation (3) represents the Lagrangian (L) which is the difference between the KE and PE of the system. Now we calculate the Euler Lagrange equations of motion.

$$\frac{\partial L}{\partial y} = \frac{\partial}{\partial y}\left(\frac{1}{2}m\dot{y}^2 - mgy\right) = -mg \qquad (4)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{y}}\left(\frac{1}{2}m\dot{y}^2 - mgy\right)\right) = m\ddot{y} \qquad (5)$$

*Hence*

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) - \frac{\partial L}{\partial y} = 0$$

$$\Rightarrow m\ddot{y} - (-mg) = 0$$

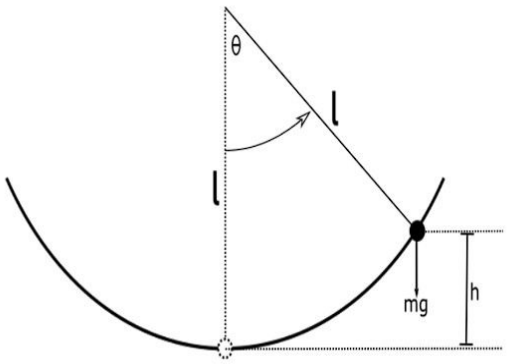
$$\Rightarrow \ddot{y} = -g \qquad (6)$$

Equation (6) gives the final answer. Here ÿ represents the acceleration.

Equation (6) makes sense as only gravitational force is acting on the system. Hence the acceleration of the system is the acceleration due to gravity.

If we had used the Newton's laws of motion, we would have arrived at the same result, albeit in a different way.

Let us now consider a somewhat more complex example.



# Task 2

We have our pendulum equation whose equations of motion we demonstrated in previous Task using Newtons Laws of motions. Now we will demonstrate the same using Euler-Lagrangian method.

In this system, we have a pendulum. The mass of the bob is given as m. The length of rod to which the bob is attached to is l. We have assumed the rod to be rigid and have no mass. So all the mass is concentrated to the bob.

While swinging, at any arbitrary point in the pendulum's trajectory, the pendulum can assumed to be at a height h from the bottom. h can be written as a function of θ where θ is the angle the pendulum bob makes with the vertical.

$$h = l - l\cos\theta$$

First, we need to calculate the Lagrangian L for this system. For that we need to compute the kinetic energy and potential energy of this system.

Calculating the potential energy is pretty straightforward.

$$PE = mgh = mg(l - l\cos\theta) = mgl(1 - \cos\theta) \qquad (7)$$

The kinetic energy will be defined by (1/2) x mass x velocity^2. Here the velocity is the tangential velocity of the bob. We can take x and y components of velocity v and solve for kinetic energy using those equations.

However, we can use rotational mechanics to make our calculations simpler. Since the pendulum bob is oscillating in a circular trajectory, the kinetic energy can be given by

$$KE = \frac{1}{2} I\omega^2$$

Where I is the moment of inertia and ω is the angular velocity.

But we know

$$I = ml^2 \quad and \quad \omega = \dot{\theta}$$

Angular velocity ω can be written as rate of change of θ with respect to time. Hence we can write ω= (dθ/dt).

Therefore we have the expression for kinetic energy

$$KE = \frac{1}{2} I\omega^2 = \frac{1}{2} ml^2\dot{\theta}^2$$

Now we have the expressions for PE and KE we will calculate the Lagrangian L and use it to calculate the equations of motion for this system.

$$L = KE - PE = \frac{1}{2} ml^2\dot{\theta}^2 - mgl(1 - \cos\theta) \qquad (9)$$

# Task 2

Since L is a function of θ, we need to select θ and θ as state variables of the system. Hence the equations of motion can be calculated as:

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

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{\theta}}\left(\frac{1}{2}ml^2\dot{\theta}^2-mgl(1-\cos\theta)\right)\right)-\frac{\partial}{\partial \theta}\left(\frac{1}{2}ml^2\dot{\theta}^2-mgl(1-\cos\theta)\right)=0$$

$$\Rightarrow \quad ml^2\ddot{\theta}+mgl\sin\theta=0$$

$$\ddot{\theta}=\frac{-g}{l}\sin\theta \qquad (10)$$

In previous Task, we had used the same pendulum example and calculated the equations of motion for pendulum using Newton's laws. We can confirm that the same equations have been derived using the Euler-Lagrange method.

Suppose we take

$$x_1=\theta$$

and,

$$x_2=\dot{\theta}$$

We can express the equations we formed in the following way

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

$$\dot{x}_2=\frac{-g}{l}\sin x_1$$

In this way we have a two equations that govern our system.

What happens if there is any external force acting on the system?

Consider the following system as given in Fig 3.

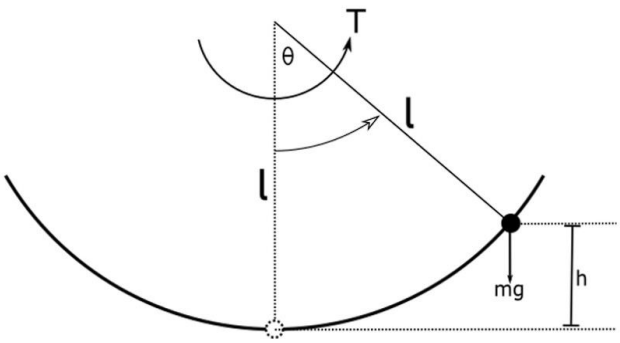



Figure 3: Pendulum with external torque

In the given simple pendulum system, we have applied an external torque to the system.

# Task 2

In this case, the Euler-Lagrange equation formed will be slightly different.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = T$$

Any non-conservative force acting on the system (Since states chosen are angular position and velocity that's why force should also be taken as angular force i.e. Torque. In case we use linear motions as in the first example then we'll use external linear force on the right hand side.) appears on the right side of the Euler-Lagrange equation. Consequently the equations of motion derived for this pendulum system will be as follows:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{-g}{l}\sin x_1 + \frac{1}{ml^2}T$$

You can see that there is an additional term in the second equation. We can check if this term is dimensionally correct.

We know $x_1$ is the angular position θ of the pendulum bob (with respect to vertical) and $x_2$ is the angular velocity theta.dot of the bob. Hence $\dot{x}_1 = x_2$ will correspond to the angular velocity theta.dot and $\dot{x}_2$ will be the angular acceleration theta.double.dot. Let the angular acceleration be denoted by α.

The units and dimensions of α are **rad/s^2** and [T^(-2) ] respectively.

We know T=Iα (where T= torque, I = moment of inertia, α=angular acceleration) and I = m*L^2.

(T/mL^2) equals angular acceleration α. Hence the last term is an angular acceleration term which is consistent with the equation. We can also calculate the dimensions of this term to verify. It will always come as [T^(-2) ]. This method is helpful to verify if or equations are valid.

---

Take the quiz on Project Whatsapp Group before moving forward:

# Task 2

## PART-02

## Introduction to State Space Analysis

We had briefly covered State Variables and State Equations in previous Task. In this section we will further elaborate on that topic and discuss the various control techniques that are associated with that.

In control engineering, a **state-space representation** is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations or difference equations. **State variables** are variables whose values evolve through time in a way that depends on the values they have at any given time and also depends on the externally imposed values of input variables. Output variables' values depend on the values of the state variables.

The state space equations for a linear time invariant system (LTI) system can be given as follows:

$$\text{State Equation} \Rightarrow \dot{x}(t) = Ax(t) + Bu(t)$$
$$\text{Output Equation} \Rightarrow y(t) = Cx(t) + Du(t)$$

Here

x(t)- State Vector (n x 1 matrix)

y(t)- Output Vector (p x 1 matrix)

u(t)- Input Vector (m x 1 matrix)

A - State (or system) matrix (n x n matrix)

B - Input matrix (n x m matrix)

C - Output Matrix (p x n matrix)

D - Feed-forward matrix (p x m matrix)

where p, m, n are:

We won't go into the theory of how these equations came into being. That's a lot of complicated math that cannot be covered here. You can refer to good Control Systems books.

Consider a set of equations:

# Task 2

$$\dot{x}_1 = x_1 x_2 - x_2$$
$$\dot{x}_2 = 2x_1 - x_2^2 \qquad (1)$$

We want to express this set of equations into the form

$$\dot{x} = Ax \qquad (2)$$

Notice, we have neglected the **Bu** term in this equation. That's because our system doesn't have any input. It only has state variables $x_1$ and $x_2$.

Can we express the set of equations (1) in terms of (2)??

The answer is no, we cannot. (1) is a set of non linear equations while (2) is a set of linear equations. However, if we linearize (1), it might be possible to express (1) in terms of (2).

How do we linearize (1)? That was basically the whole point of previous Task.

## 1. Find the equilibrium points

We want to find the point around which the system is stable. To find the equilibrium points we need to set $\dot{x}_1 = 0$ and $\dot{x}_2 = 0$. Then solve the equations for

$x_1$ and $x_2$.

$$0 = x_1 x_2 - x_2 \qquad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

If we solve (3) for $x_1$ and $x_2$ we will get the equilibrium points as (0,0), (1,√2) and (1,-√2).

## 2. Calculate the jacobian of the system of equations

The jacobian J for the system of equations (3) will be:

$$J = \begin{bmatrix} \frac{\partial(x_1 x_2 - x_2)}{\partial x_1} & \frac{\partial(x_1 x_2 - x_2)}{\partial x_2} \\ \frac{\partial(2x_1 - x_2^2)}{\partial x_1} & \frac{\partial(2x_1 - x_2^2)}{\partial x_2} \end{bmatrix}$$

$$J = \begin{bmatrix} x_2 & x_1 - 1 \\ 2 & -2x_2 \end{bmatrix}$$

## 3. For each equilibrium point, calculate the value of the jacobian.

The values of jacobian for each equilibrium point will be given as:

# Task 2

$$J_{(0,0)} = \begin{bmatrix} 0 & 0-1 \\ 2 & -2(0) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 2 & 0 \end{bmatrix}$$
$$J_{(1,\sqrt{2})} = \begin{bmatrix} \sqrt{2} & 1-1 \\ 2 & -2(\sqrt{2}) \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 \\ 2 & -2\sqrt{2} \end{bmatrix}$$
$$J_{(1,-\sqrt{2})} = \begin{bmatrix} -\sqrt{2} & 1-1 \\ 2 & -2(-\sqrt{2}) \end{bmatrix} = \begin{bmatrix} -\sqrt{2} & 0 \\ 2 & 2\sqrt{2} \end{bmatrix}$$

## 4. Construct the state equation for each equilibrium point.

The state equation for equilibrium point (0,0) will be:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Here,

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
$$A = \begin{bmatrix} 0 & -1 \\ 2 & 0 \end{bmatrix}$$

Therefore the set of equations has been expressed in the form:

$$\dot{x} = Ax$$

It is very important to note that this approximation of the set of non-linear equations given in (3) will only hold true for point close to the equilibrium point (0,0).

This means that around the vicinity of the equilibrium point (0,0), the non-linear system will behave like a linear system and the state equation given above will hold true around the vicinity of that point.

Likewise, the state equations for equilibrium points (1,√2) and (1,-√2) are:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 \\ 2 & -2\sqrt{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

and

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} & 0 \\ 2 & 2\sqrt{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

## Stability

---

We can find out whether the system is stable or unstable at each of the equilibrium points by finding out the eigenvalues of the A matrix. If any of the eigenvalues have a positive real part, the system will be unstable.

So, for equilibrium point (1, √2) of the system, the eigenvalues will be -2.824 and 1.414. Hence system will be unstable.

# Task 2

## Introducing Control Input

Let us consider the Pendulum with external applied torque system.

We derived the equations for this system as:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{-g}{l}\sin x_1 + \frac{1}{ml^2}T$$

We can apply the same linearization technique explained above.

## 1. Find the equilibrium points.

If we set $\dot{x}_1 = 0$ and $\dot{x}_2 = 0$, we will find the equilibrium points of this system as $(n\pi,0)$ where $n=0,\pm1,\pm2,\ldots$ .From the physical descriptions of the pendulum, it is clear that there are only two equilibrium positions (0,0) and $(\pi,0)$. The rest of equilibrium points are just repetitions based on number of full swings of the pendulum.

The equilibrium point (0,0) will be when the pendulum bob is vertically downwards.

The equilibrium point $(\pi,0)$ will be when the pendulum bob is vertically upwards.

Intuitively, we can guess that the system will be stable at equilibrium point (0,0) and unstable at equilibrium point $(\pi,0)$. Let us see if our intuition is correct.

## 2. Calculate the jacobian of the system of equations.

The jacobian $J_1$ for the A matrix of the state equation will be:

$$J_1 = \begin{bmatrix} \frac{\partial(x_2)}{\partial x_1} & \frac{\partial(x_2)}{\partial x_2} \\ \frac{\partial(\frac{-g}{l}\sin x_1 + \frac{1}{ml^2}T)}{\partial x_1} & \frac{\partial(\frac{-g}{l}\sin x_1 + \frac{1}{ml^2}T)}{\partial x_2} \end{bmatrix}$$
$$J_1 = \begin{bmatrix} 0 & 1 \\ \frac{-g}{l}\cos x_1 & 0 \end{bmatrix}$$

Since our system has input, we also need to calculate jacobian $J_2$ for the B matrix.

$J_2$ will be:

# Task 2

$$J_2 = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{bmatrix}, \quad f_1 = \dot{x}_1, f_2 = \dot{x}_2$$

Since T is input, we replace T with u

$$J_2 = \begin{bmatrix} \frac{\partial (x_2)}{\partial u} \\ \frac{\partial (\frac{-g}{l} \sin x_1 + \frac{1}{ml^2} u)}{\partial u} \end{bmatrix}$$

$$J_2 = \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}$$

## 3. For each equilibrium point, substitute value of ($x_1$,$x_2$) in the jacobian and calculate the A and B matrix.

The values of A matrix for each equilibrium point will be given as:

$$A_{(0,0)} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix}, \qquad A_{(\pi,0)} = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix}$$

The values of B matrix for all equilibrium points will be:

$$B = \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}$$

## 4. Construct the state equation for each equilibrium point.

The state equation for equilibrium point (0,0) will be:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u$$

The state equation for equilibrium point (π,0) will be:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u$$

## 5. Check the stability of the system at each equilibrium point.

# Task 2

At equilibrium point (0,0) the eigenvalues will be:

$$\lambda = \pm\sqrt{\frac{g}{l}}i$$

At equilibrium point (π,0) the eigenvalues will be:

$$\lambda = \pm\sqrt{\frac{g}{l}}$$

The eigenvalues for (0,0) will be purely imaginary. Hence the system will be marginally stable. Marginally stable means that system will be continue to oscillate about the equilibrium point indefinitely.

The eigenvalues for (π,0) will be purely real. One of the eigenvalues will have positive real part. Hence the system will be unstable.

Hence we proved that our earlier intuitions about the stability of the system are correct. The system will be stable for (0,0) and unstable for (π,0).

## Controllability and Observability

---

In control theory, controllability and observability are two very important properties of the system.

**Controllability** is the ability to drive a state from any initial value to a final value in finite amount of time by providing a suitable input. A matrix which determines if a system is fully controllable or not is called the controllability matrix.

**Observability** is the property of the system that for any possible sequnce of state and control inputs, the current state can be determined in finite time using only the outputs. A matrix which determines if a system is fully observable or not is called the observability matrix. A fully observable system means that it is possible to know all the state variables from the system outputs.

Controllability matrix (R) of a system is given by the following:

$$R = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}$$

Observability matrix (O) of a system is given by the following:

# Task 2

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

If a system is fully controllable, then

$$rank(R) = n$$

If a system is fully observable, then

$$rank(O) = n$$

Here, n is the number of state variables.

Rank of a matrix is defines as the maximum number of linearly independent rows or columns in a matrix.

In the pendulum example (with external torque), we have the A and B matrix available to us. Hence we can calculate the controllability of the system.

$$R = \begin{bmatrix} 0 & \frac{1}{ml^2} \\ \frac{1}{ml^2} & 0 \end{bmatrix}$$

The rank of R is 2 which is equal to the number of state variables. Hence the system is fully controllable.

---

Take the quiz on Project Whatsapp Group before moving forward:

# Task 2

## PART -03

## Controller Design

So far we have discussed the basics of state space analysis. In this section, we will discuss different types of controller design.

Consider the State Space Equations of a system:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

This system can be represented in form of a block diagram as follows:

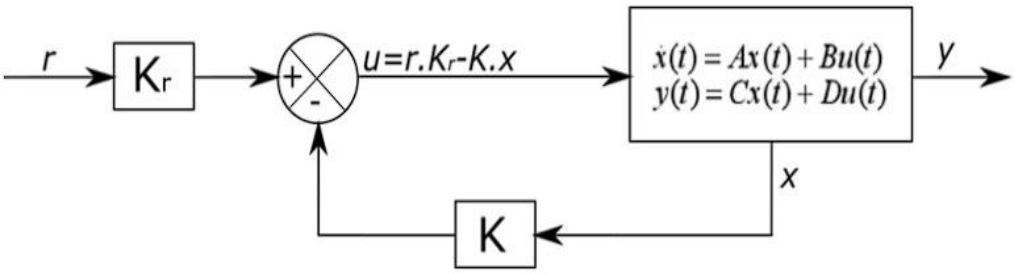



Figure 4: System block diagram

Here

r - Reference point

Kr - Gain by which reference is multiplied

x - State Vector

y -Output Vector

u - Input to system

K - Gain by which input is multiplied.

In this system we have taken the state vector x, multiplied that with some gain matrix K and fed that as feedback input to the system.

The State Equation for the system can be written as follows:

$$\dot{x} = Ax + B(rK_r - Kx)$$
$$\Rightarrow \dot{x} = Ax - BKx + BrK_r$$
$$\Rightarrow \dot{x} = \underbrace{(A - BK)}_{\text{New State Matrix}} x + BrK_r$$

# Task 2

The new state matrix (A-BK) defines the dynamics of the system where -Kx is fed as input. The system stability can be calculated by finding the eigenvalues of the (A-BK) matrix.

**Linear Quadratic Regulator (LQR)**
So far we have seen that if we have a system which is controllable, then we can place its eigenvalues anywhere in the left half plane by choosing appropriate gain matrix K. But the main question is where should we place our eigenvalues?

Till now we have only discussed about the stability of the system. But nowhere have we asked that what is our performance measure?
In this section, we'll see how to optimize the value of gain matrix K to get the desired performance measure from the system.

Linear Quadratic Regulator is a powerful tool which helps us choose the K matrix according to our desired response. Here we use a cost function.

$$J = \int_0^\infty (x^T Q x + u^T R u)$$

where, Q and R are positive semi-definite diagonal matrices (positive semi-definite matrices are those matrices whose all the eigenvalues are greater than or equal to zero). Also to remind you that for a diagonal matrix, the diagonal entries are its eigenvalues. x and u are the state vector and input vector respectively.

Let us say that you have your system with four states and one input.

Then $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ and input u. Let

$$Q = \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix}$$

Then $x^T Q x + u^T R u = Q_1 x_1^2 + Q_2 x_2^2 + Q_3 x_3^2 + Q_4 x_4^2 + R u^2$ . Thus, you may see that the system taken here is our usual system represented as

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

The controller is of the form u = -Kx which is a **Linear** controller and the underlying cost function is **Quadratic** in nature and hence the name **Linear Quadratic Regulator**.

# Task 2

With a careful look at the integrand of the cost function J, we may observe that each Qi are the weights for the respective states xi.

So, the trick is to choose weights Qi for each state xi so that the desired performance criteria is achieved. Greater the state objective is, greater will be the value of Q corresponding to the said state variable. We can choose R = 1 for single input system. In case we have multiple inputs, we could use similar arguments for weighing the inputs as well.

In case of inverted pendulums, we know that angle with the vertical and the angular velocity is very important and hence the weights corresponding to them should be more as compared to linear position and linear velocity .

LQR minimizes this cost function J based on the chosen matrices Q and R. Its a bit complicated to find out matrix K which minimizes this cost function. This is usually done by solving Algebraic Riccatti Equation (ARE). We'll not go into details of how to solve ARE, as it is not required in our tasks. There is inbuilt **lqr** command in octave to find K matrix. What is required to be done is to choose the Q and R matrix appropriately to get the desired performance.

---

**Now that you have understood the theory, let's move on to implementation of Task 2A**

## Task Instructions

In this task, we will be deriving the state space model of the Rotary Inverted Pendulum system and implementing the LQR control scheme on the model. You will be required to find out the equations of motion for the physical system and modify code as instructed to find the correct mathematical model & find optimal controller gains.
The Mathematical Modeling process is already demonstrated in the previous pages of this doc. Additionally a similar process for a two wheeled self balancing robot is shown below :

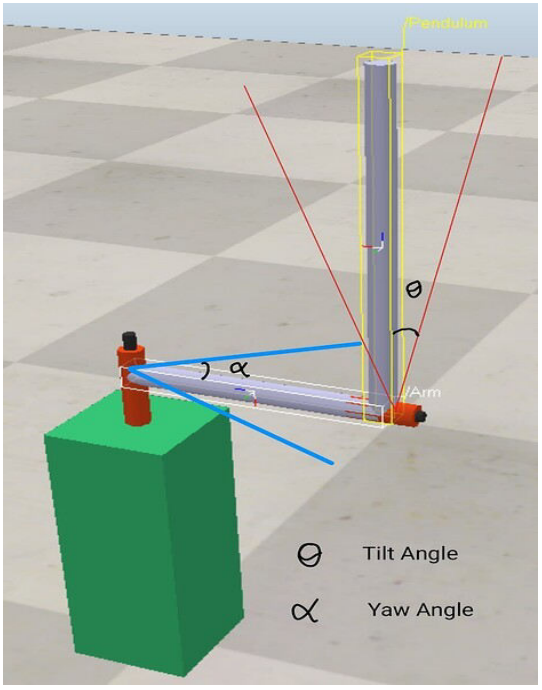
- Technical Session – 1 -> Youtube Link
- Technical Session – 2 -> Youtube Link

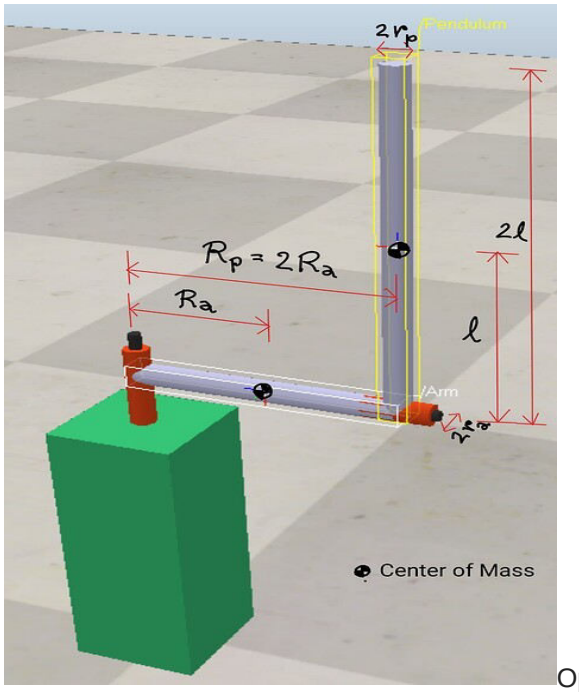
Open the **Task1A.zip ( Sent in group )** file and extract. For every Physical System that we will discuss here, you will find following file in the folder:

**1. Task_1A.m**

## Physical System - Rotary Inverted Pendulum

# Task 2




Open the file **Task_1A.m** using Octave.

In this file you will find the following functions defined:

- **Jacobian _A_B()**
- **lqr_Rotary_Inverted_Pendulum()**

# Task 2

•**Rotary_Inverted_Pendulum_main()**

## •1. Jacobian _A_B():

This function is used to define the dynamics of the system by using the equations of motion you have derived for this system, finally to get the State Space Model of the system.

```
Jacobian _A_B()

1. function [A,B] = Jacobian_A_B(Mp,l,g,Ma,Rp,Ra,I_arm,I_pendulum_theta,I_pendulum_alpha)

2.  alpha = sym('alpha');

3.  theta = sym('theta');

4.  theta_dot = sym('theta_dot');

5.  alpha_dot = sym('alpha_dot');

6.  u         = sym('u');

7.  cos_theta = cos(theta);

8.  sin_theta = sin(theta);

9.  cos_alpha = cos(alpha);

10. sin_alpha = sin(alpha);

  ########## ADD YOUR CODE HERE ################
  #{
  Steps :
    1. Define equations of motion (4-states so 4 equations). It is suggested to use Lagrangian method. You can try Newtonian methods too.
    2. Partial Differentiation of equations of motion to find the Jacobian matrices
    3. Linearization by substituting equillibrium point condition in Jacobians

  ### NOTE ### : Sequence of states should not be altered for evaluation

  SEQUENCE OF STATES : [alpha_dot; alpha; theta_dot; theta]
        Example:
                A = [x x x x;   # corresponds to ...alpha_dot
                     x x x x;   # ...alpha
                     x x x x;   # ...theta_dot
                     x x x x]   # ...theta
                B = [x;   # ...alpha_dot
                     x;   # ...alpha
                     x;   # ...theta_dot
                     x]   # ...theta
  #}
  ##############################################

11.  A = ; # A should be (double) datatype

12.  B = ; # B should be (double) datatype

13. endfunction
```

**Lets now understand this function:**

**Line 2 to Line 6** are the representation of states in symbols

**Line 7 to Line 10** are the symbolic representation of **cos** and **sin** angle required in forming dynamic equations of the system

There are HINTS given as comments in the function to help with the steps.

Use the same sequence of states in the model as given in comments. Use “**0”** degree angle as the reference equilibrium point for tilt angle.

Notice the Shape of **B matrix** given in comments. It says that you have to derive model considering it as **single input** system. (**Underactuated !!** )

# Task 2

•This function is intended to work as a helper to deal with long mathematical expressions in a simplified manner. **You can choose to do the math partially by-hand/on-paper and feed in the values here.**

## 2. lqr_Rotary_Inverted_Pendulum():

You do not have to modify anything in this function. This function demonstrates the use of Octave to apply LQR control and find optimal gains using performance specified by Q & R matrices.

It will be used in **Task 1B**.

lqr_Rotary_Inverted_Pendulum()

```
1. function K = lqr_Rotary_Inverted_Pendulum(A,B)

2. C = [1 0 0 0;
        0 1 0 0;
        0 0 1 0;
        0 0 0 1];            ## Initialise C matrix

3. D     = [0;0;0;0];          ## Initialise D matrix

4. Q     = eye(4);             ## Initialise Q matrix

5. R     = 1;                  ## Initialise R

6. sys   = ss(A,B,C,D);        ## State Space Model

7. K     = lqr(sys,Q,R);       ## Calculate K matrix from A,B,Q,R matrices using lqr()

8. endfunction
```

### 3. Rotary_Inverted_Pendulum_main():

This function defines the physical parameters of system. It uses state space model generated and finds optimal gain using respective functions.

Rotary_Inverted_Pendulum_main()

```
1. function Rotary_Inverted_Pendulum_main()

2. Mp = 0.5 ;                  # mass of the pendulum (Kg)

3. l = 0.15 ;                  # length from pendulum's center of mass to pendulum's base/pivot (meter)

4. g = 9.81 ;                  # Accelertion due to gravity (kgm/s^2)

5. Ma = 0.25 ;                 # mass of the arm (kg)

6. r_a = 0.01;                 # radius of arm cylinder (meter)
```

# Task 2

```
7. r_p = 0.01;                # radius of pendulum cylinder (meter)
8. Rp = 0.2 ;                 # length from pendulum's base to arm's pivot point (meter)
9. Ra = 0.1 ;                  # length from arm's center of mass to arm's pivot point (meter)
10. I_arm = ;                  # Moment of inertia of the arm in yaw angle i.e. alpha (kgm^2)
11. I_pendulum_theta = ;       # Moment of inertia of the pendulum in tilt angle i.e. theta (kgm^2)
12. I_pendulum_alpha = ;       # Moment of inertia of the pendulum in yaw angle (kgm^2)
13. [A,B] = Jacobian_A_B(Mp,l,g,Ma,Rp,Ra,I_arm,I_pendulum_theta,I_pendulum_alpha) ## find A , B matrix using  Jacobian_A_B() function
14. K = lqr_Rotary_Inverted_Pendulum(A,B)  ## find the gains using lqr_Rotary_Inverted_Pendulum() function
endfunction
```

You have to use the dimensions and other physical parameters of the model to compute respective **moments of inertia** for the system. You can feed it to use while deriving dynamics equations.

- **Do not change the arguments or return values of any function**

---------- **TASK 2A KHATAM** ----------

---------- **NEECHE ZAAO ABHI TASK 2B BHI HAI** ----------

# Task 2

## TASK - 2B

## Part – 1 : Intro To Coppeliasim

•This video introduces the basics of CoppeliaSim software.

•Teams will learn about Features, User Interface, Pages, Views, Scenes, Camera Navigation, Position/Orientation Manipulation, Simulation Settings, Scripts, User Settings, Collections, Layers, Video Recorder, Scene Object Properties etc

See this -> Coppeliasim Basics 1 ( Youtube Link )

**EXTRA:**

There's a popular youtube channel for CoppeliaSim tutorials, for people who prefer videos. You can explore what other great stuff this software is capable to do.

See this -> Coppeliasim Basics 2 ( Youtube Link )

Now that you have understood the theory, let's move on to implementation of Task 1B

### Task Instructions

- Open **task2B.zip( Sent In Project Group )** file and extract.
- In the extracted folder you will find following file
    - **Task_1B.ttt**

### 1. Problem Statement

In this task, teams have to write code to implement **LQR or PID** control strategy **to balance 3 rotary inverted pendulums at their unstable equilibrium point using python child scripts** in the given CoppeliaSim scene.

The 3 variants of rotary **pendulum should be balanced at upright position** and should **also parallelly maintain the angular position of arm in horizontal plane** - both together !!

This task should be accomplished using given scene environment without changing simulator or scene object properties. You can change only the child scripts, following the mentioned guideline.

Two of these systems are pure rotary inverted pendulum with two cylindrical rods with uniform mass distribution. Third one is having a slightly different system where actuator is not at the arm pivot but is at the other end of arm motorizing a wheel. But since the pendulum rod is still on a free joint in the third system, it acts just like a rotary inverted pendulum.

# Task 2

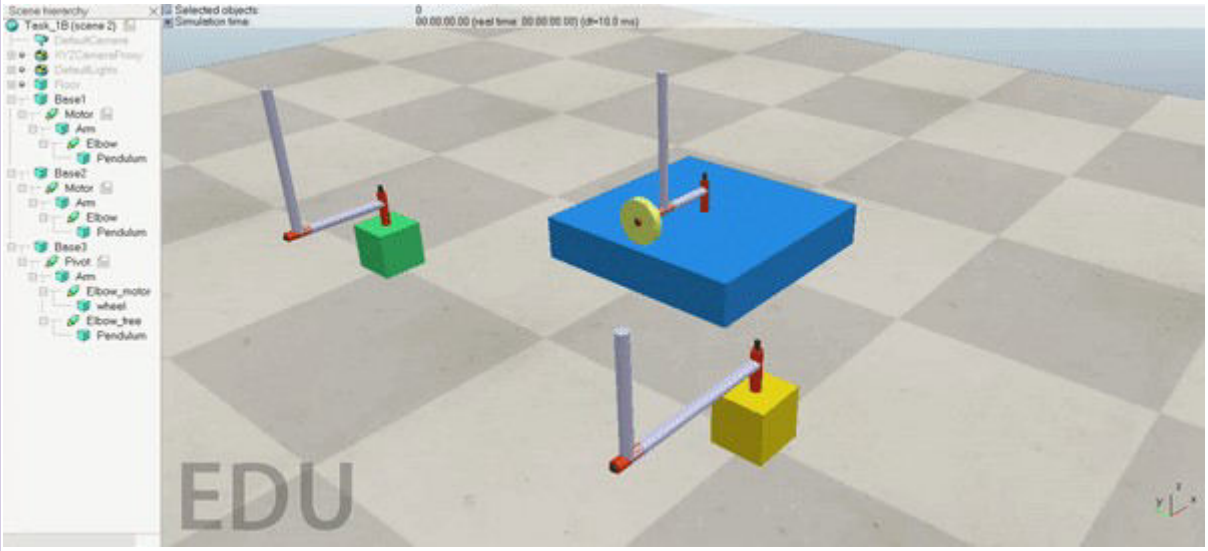



Figure 1: Rotary Pendulums Task Problem Statement

## 2. Given

### 1. CoppeliaSim's Scene File (Task_1B.ttt)

•As you open the scene file i.e. Task_1B.ttt in CoppeliaSim software as shown in Figure 1.
•You'll find a hierarchical tree structure similar to the one shown in Figure 2.

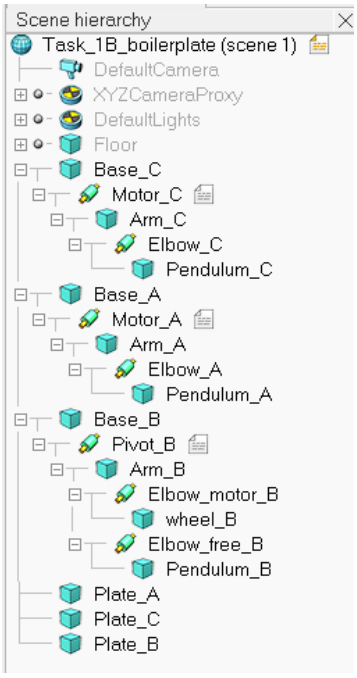



Figure 2: Example of Scene Hierarchy

•The **objects in the scene** along **with their names and uses** are given in Table Below.

| **Objects** | **Name in the scene** | **Use(s)** |
|---|---|---|
| Base | *Base* | Acts like an immovable base for Rotary Inverted Pendulum.<br>**The names will be suffixed with object name. For ex: Base_A, Base_B, Base_C** |

# Task 2

| Objects | Name in the scene | Use(s) |
|---|---|---|
| Arm | *Arm* | Acts as the rotary arm in horizontal plane which moves the pendulum |
| Free Joint | *Pivot* | As a pivot for the rotary arm around which the arm may rotate |
| Motor | *Motor* | As an actuating joint behaving as DC geared motor for the rotary arm around which the arm may rotate |
| Pendulum | *Pendulum* | Acts as the pendulum |
| Motor | *Elbow_motor* | The joint acting as DC geared motor actuating wheel on rotary arm. |
| Free Joint | *Elbow_free* | The joint on rotary arm acting as free pivot for pendulum rod's rotation in vertical plane |
| Free Joint | *Elbow* | The joint on rotary arm acting as free pivot for pendulum rod's rotation in vertical plane |

> **NOTE:**
>
> •In this task you are **NOT** allowed to **add or remove** objects in scene.
> •No change in the **names of the objects, their parent child relationships, their properties, position, orientation** etc should be made.

•**Pendulum** in each model is having a mass of 0.5kg.

•**Arm** in each model is having a mass of 0.25kg.

•**Pivot, Elbow, Elbow_free** are revolute joints in free mode.

•**Motor, Elbow_motor** are revolute joints in velocity control mode, with a max. torque rating.

•Similarly all other properties of all objects in scene can be known by double clicking the object name in scene hierarchy.

> If you've followed the introduction to coppeliasim tutorial, then try exploring other options in coppeliasim like the position and orientation of object in different frames, etc. You can set a good camera view in simulation for yourself .

## Understanding the Task:

•You'll find a script icon near the Motor & Pivot as shown in Figure 2.
•Click on the script icon. You'll have the script opened as shown in Figure 3. You have to edit this file by writing your code in appropriate function to execute the control loop to balance each rotary inverted pendulum.

> There are explainatory comments given in each function - read them through.

# Task 2

Child script "/Motor"

```python
#python

###### GLOBAL VARIABLES HERE ######
base = None
motor = None
arm = None
pendulum = None
U = None
# You can add variables here
# as required by your implementation.
###################################

def sysCall_init():
    # do some initialization here
    # This function will be executed once when the simulation starts

    ####### ADD YOUR CODE HERE ######
    # Hint: Initialize the scene objects which you will require
    #       Initialize algorithm related variables here

    ###################################
    pass

def sysCall_actuation():
    # put your actuation code here
    # This function will be executed at each simulation time step

    ####### ADD YOUR CODE HERE ######
    # Hint: Use the error feedback and apply control algorithm here
    #       Provide the resulting actuation as input to the actuator joint

    # Example psuedo code:
    #   x1 = error_state_1; # Error in states w.r.t desired setpoint
    #   x2 = error_state_2;
    #   x3 = error_state_3;
    #   x4 = error_state_4;
    #   k = [gain_1 , gain_1, gain_3, gain_4];      # These gains will be generated by control a
    #   U = -k[1]*x1 +k[2]*x2 -k[3]*x3 +k[4]*x4;    # +/- Sign convention may differ according t
```

Figure 3: Motor Child Script

Applying Control Technique (LQR/PID) :

- You are allowed to use either of LQR or PID control technique to balance the pendulums.

## LQR :

•Refer back to **Task 1A** where you have been provided with a function in octave named as lqr_Rotary_Inverted_Pendulum(A,B). If you refer to the comments of the function, you can know that is uses the state space model of system and using Q & R matrices it generates optimal gains for each of the 4 states as per desired performance

•This desired performance is specified by Q & R matrices. **Q is the state cost weight matrix** & **R is the input cost weight matrix**. Both are **diagonal matrices**.

•For Q - each diagonal value corresponds to a state(as per state sequence) and it is of nxn size for n states. Whereas R is of uxu size for n states and u inputs. The diagonal entries of R correspond to each input.

•Choosing Q & R is takes a thorough understanding and observation of system dynamics. Weights in Q matrix should be given in a sequence such that **higher priority is given to the most critical states** for the controller. Similary R matrix shares the energy cost among all the inputs. For single input system it's less significant as all energy is given to single input only.

•The relative values of weights matter for Q & R matrices. The absolute value doesn't make much change.( Learn more about using lqr() here.

# Task 2

> **WAIT HEY** … The Rotary Inverted Pendulum that you have modeled in Task 1A. Does it fit with all 3 of the pendulums given in this task ?
>
> - Prepare to make some changes in the state space model if you find it different here !!! 🙃

## PID :

•This technique can be applied without knowing state-space model. It requires lesser efforts in math as it's an observation based tuning method, but there are ways to mathematically tune it for our system for optimal performance.

•There are good resources on PID tuning like

•**PID-Wikipedia**

•**MATLAB Tech-Talks playlist on PID**

•Refer this to learn **more about PID controller design**

> **SUGGESTION** : Controlling 2 angles and total 4 states may require more than a straightforward PID. So you can try out variations of it. For example, cascade control or parallel PID etc.

---------- **KHATAM TATA BY BY** ----------