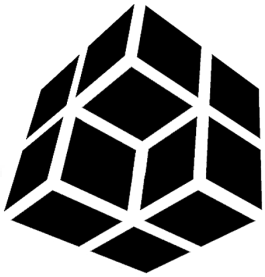


Robotics and
Machine Intelligence

RMI Inductions ‘23

ADVANCE TASKS

Weekly Tasks Track

Greetings from RMI!

Congratulations on successfully completing the intermediate tasks. We hope that you have got a good exposure to the various domains involved in robotics.

As mentioned before, the inductions will be conducted in two stages:

1. The Practical Implementation Stage: Here, you will be given a set of tasks to work on, that will facilitate your learning and understanding.
2. The Personal Interview Stage.

The Practical Implementation Stage is split into three sections: Basic, Intermediate, and Advanced. This document outlines all the Advance tasks to be completed. The advance tasks are designed in such a way that you can implement the concepts you might have learnt while attempting the intermediate tasks and introduce you to other new exciting concepts which is unique to the domain.

You might face issues or difficulties in completing the tasks, but do not give up. It is quite normal to hit roadblocks while progressing. In any case, feel free to contact any member of the club.

Rules:

1. Participants can attend any number of advance tasks from any domain. The number of questions attended, and the overall points will be taken into consideration for evaluation.
2. Topics associated with each task are also listed. Please go through them to understand the concepts behind the tasks.
3. It is good to work in teams, but the tasks are not meant to test your teamwork. The problems must be solved individually. Learning how everything works is key. Make sure you understand everything well.
4. Create a drive folder named “<Roll Number> - Advance Tasks” and sub-folders named “<Roll Number> - <Domain Name>” for each of the domains. Create folders for each task in a particular domain. Follow domain specific submission guidelines.

General Instructions:

1. Go through the problem statement and understand the task well before you start working on it. A task misunderstood and completed will not be considered. If you have trouble understanding it feel free to contact anyone of us.
2. We expect you to understand the concepts and write your own code. It is acceptable to refer to code snippets online but copying the code is not an option. You will be extensively questioned based on the code and will be asked to make modifications during PI's.
3. Divide a complex task into modules and work on the individual modules. Completing the modules will be appreciated even if you are unable to complete the entire task.
4. Partial submissions are accepted. What we look for is your approach and zeal, rather than the entire solution itself.
5. The internet is your biggest resource. All your doubts are a Google search away.
6. You can always reach out to us regarding any doubts, queries or issues through the WhatsApp group or PM one of us through the same. We will be glad to help.

HAPPY ROBOTICS!!!

Electronics and Embedded Systems

Instructions:

1. You should implement the solutions for all the tasks in **Proteus VSM software** only. Arduino IDE can be used for writing the Arduino code.
2. You can use any AVR microcontroller for implementing the tasks. We would recommend using the Arduino UNO development board, which has the ATmega328p microcontroller to get started.
3. All the code must be commented well enough, and the circuit must be labelled for easier understanding.

Task 1

Establish the communication between two microcontrollers, i.e., master and slave using I2C protocol (without the use of any library). The master is connected to the potentiometer, and the slave is connected to the serial monitor. Read the potentiometer value on the I2C master and display it in the serial monitor (in volts).

Bonus: Do the same using full embedded C.

Topics to learn: I2C, USART.

Task 2

Build a differential drive robot with the help of L293D IC. Based on the user's input through a Bluetooth device (preferably your own mobile phone) move the bot in the 'forward' and 'backward' direction and make a 'left' and 'right' turn. Operate the bot at 90% of maximum speed in the forward direction and 30% of maximum speed in the backward direction. While performing turning operations, the bot shouldn't exceed 40% of the maximum attainable speed.

NOTE: Define the maximum speed of the motor depending on the motor you use. The radius of curvature during turning operations should be zero.

Topics to learn: Controlling speed of DC Motor using PWM, Working of H-Bridge, Bluetooth Module.

Task 3

Design a circuit that is capable of reading passwords entered on the door. If the password is correct, the door will be unlocked for three seconds before getting locked again. The Default password is 10111001. Unlocked door is indicated by a turned ON LED and similarly the OFF state of LED indicates that the door is locked.

Bonus: Add a subcircuit to record a new password that can be used to unlock the door in the future.

Note: The circuit should be designed using basic elements of analog and digital electronics. No microcontroller or programming of any kind is allowed.

Topics to learn: Logic gates (AND, OR, NOT, NAND, NOR, XOR), and related ICs, combinational circuits (MUX, DEMUX, Decoder, Encoder, etc.), Sequential elements (D-flip flops, shift registers, etc.)

Task 4

Battery management system (BMS) is a technology dedicated to the oversight of a battery pack. The task of battery management systems is to ensure the optimal use of the residual energy present in a battery. In order to avoid loading the batteries, BMS systems protect the batteries from deep discharge and over-voltage, which are results of extreme fast charge and extreme high discharge current. Basic BMS generally consists of MOSFETs, resistors, and potentiometers and it can measure voltage and current. Design a BMS circuit using Arduino for measuring voltage and current from the battery.

Task 5

Generate a PWM signal using Arduino and control its frequency and duty cycle using 2 potentiometers. Adjust the potentiometers to set signal parameters (frequency and duty cycle), to values of your choice, and display this generated signal using an oscilloscope.

Also, get this generated signal as an input through a new pin and write a code to calculate the frequency, duty cycle of the PWM signal read from the new pin and display those values in the serial monitor.

Note: Only embedded C should be used.

Topics to learn: Embedded C programming, PWM signal manipulation.

Resources:

https://www.circuitbasics.com/introduction-to-dc-motors-2/

https://www.theengineeringprojects.com/2016/03/bluetooth-library-for-proteus.html

https://www.electronicwings.com/sensors-modules/bluetooth-module-hc-05-
https://circuitdigest.com/microcontroller-projects/arduino-timer-tutorial

https://www.synopsys.com/glossary/what-is-a-battery-management-system.html

https://youtu.be/GXYJ1xC10j4

https://youtube.com/playlist?list=PLA6BB228B08B03EDD

Submission Guidelines:

1. Make a folder titled "<Roll no.>_Tronix".
2. Make sub-folders titled with <Q_Question no.>, eg. **"Q1"** for question 1.
3. Upload all files related to the task into the respective folders.
4. For Proteus simulations, you can upload the circuit file and the Arduino program files.
5. Upload this folder in your drive and make the drive link shareable. Also fill the same link in the submission form.

Contacts:

Hemeshwaran : +91 77089 29656

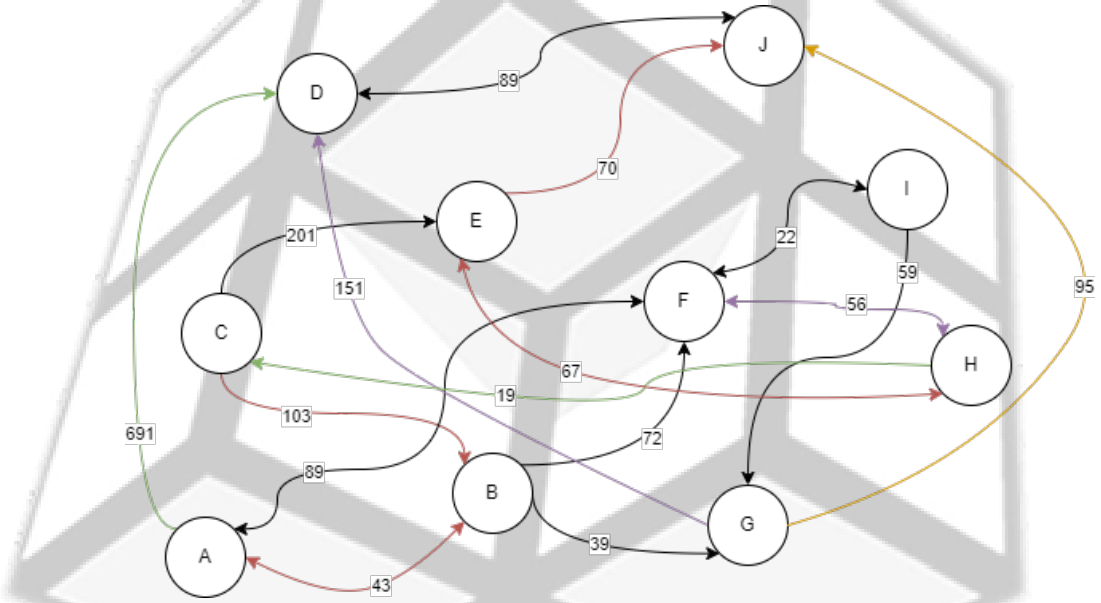
Rusheek : +91 99443 11855

Susmitha : +91 70940 84388

Vallimayl : +91 98431 77962

Task 1

You're going to construct a mobile robot that will travel through a series of checkpoints in order to reach its objective in the shortest amount of time. The time indicated in the below graph is all given in seconds and depends on the terrain and real distance between the places. It will collect containers that contain stacks of batteries with different capacities as it moves through the checkpoints. When it arrives at its destination, the robot will combine the containers and arrange them according to increasing battery capacity. Your robot will start from location *A* and travel to location *J*.



Note: The robot will collect the batteries from a stack and store it in a contiguous space inside the robot. It'll merge the containers only after reaching the destination. In addition to the time taken for navigation, each battery in the checkpoint will add 2 seconds of delay regardless of its capacity. Example: Since, there are 8 batteries in checkpoint *A*, Hence, it'll take $\mathbf{8 * 2 + 89 = 105s}$ to reach checkpoint *F* from *A*.

Checkpoint	**Stack of batteries** (lowest in stack at the left in *mAh*)
A	551, 431, 451, 981, 984, 401, 5109, 4210
B	923, 491, 490
C	4192, 491, 451, 987, 781, 901
D	9014, 10234, 1941
E	111, 9982, 231, 921
F	912, 4129, 9412, 412, 9012, 412, 519

G	9815, 9923
H	414, 4912,7891, 6893, 5891
I	914, 52, 29, 905
J	152, 123, 512, 598, 319

To determine the shortest route, you can either program it or visually demonstrate your findings. The original cost adjacency matrix and its evolution over time must be displayed.

You need to implement merge sort to merge the list of list of batteries collected from checkpoints in any programming language.

Input: 2D list

Output: sorted 1D list

Resources: Stack, merge sort, single source shortest path.

Task 2

A requirements chart for the self-balancing robot you are going to build is provided; it is depicted in the figure below. Sort the components in the chart according to their topology. You are allowed to choose an element based on its specifications, requirements, or your own preference if several elements have comparable topologies. You must explain why you gave a certain component (such as a DC Servo motor, DC Stepper motor, or DC motor) a higher priority.

Now that you've made the decision to design a controller for the robot. Initially, you're going to run simulations to determine which controller will be effective. Use of Matlab is required to create the control system for a two-wheeled self-balancing robot; either the online or desktop version will do.

Once Matlab is open, enter the following commands:

```
cd (userpath)
cd Examples/R2023a/control/InvertedPendulumExample
open_system('rct_pendulum.slx')
```

Note: `R2023a` may vary depending on the version of your matlab.

With the aforementioned commands, a Simulink system will be made available for you to use in order to simulate solid models made in Simscape Multibody. It has two predefined control systems: a state space kind of controller for its angular correction and a PD controller for position. You can look up the controls part of Intermediate Tasks for the equations of motion of the bot. Keep in mind that the State Space Model is not unique.

The simulation time must be changed to 15 seconds. Try to replace the default controllers with a variety of controllers and fine-tune them. You must display the bot's full screen footage while it attempts to balance. For implementing more control systems that stabilize the robot, more points will be given.

Bonus: You need to implement PID controllers for position and angular variables and their tuning parameters must be determined using machine learning methods. The confusion matrix for your ML model must be displayed. You can use 3-seconds intervals as class labels, such as 0-3, 3-6, 6-9, 9-12, and 12-15. The bot must be stabilized within 15 seconds; if not, modify the tuning parameters because they are poor. To build the ML model, you must generate your own training and test dataset from the results of simulation.

Pro-Bonus: You need to implement reinforcement learning to find the tuning parameters for these PID controllers.

Resources:

Topological sorting, cart-pendulum model, PID controller, LQR controller, Pole Placement, Fuzzy Logic Systems, Reinforcement Learning, Bang Bang Control.

Matlab student edition is free with our webmail. We didn't need *Simscape Multibody* for this task.

Task 3

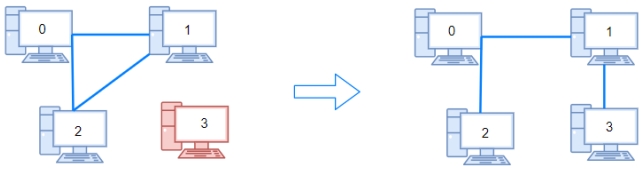
n computers numbered from $0\ to\ n-1$ are connected by ethernet cable connections forming a network where $connections[i] = [a_i, b_i]$ which represents a connection between computers a_i and b_i.

You're given an initial computer network connection. You can extract certain cables between two directly connected computers and place them

between any pair of disconnected computers to make them directly connected and thus make all of them connected in one single network.

Return the minimum number of times you need to do this in order to make all the computers connected. If it is not possible, return -1.

Example 1:



Input: $n = 4, connections = [[0,1],[0,2],[1,2]]$

Output: 1

Explanation: Remove the cable between computers 1 and 2 and place it between computers 1 and 3.

Example 2:

Input: $n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]]$

Output: 2

Example 3:

Input: $n = 6, connections = [[0,1],[0,2],[0,3],[1,2]]$

Output: -1

Explanation: There are not enough cables.

Constraints:

- $1 \leq n \leq 10^5$
- $1 \leq connections.length \leq \min\left(\frac{n(n-1)}{2}, 10^5\right)$
- $connections[j].length == 2$
- $a_i \neq b_i$

- There are no repeated connections.
- No two computers are connected by more than one cable.

Task 4

Ram wanted to participate in the Micromouse Challenge. He constructed a small micromouse robot inorder to solve the maze, now he wanted to know what are the different unique ways to reach the bottom-right corner from top-left corner.

You're given with an $m \times n$ integer array grid. There is a robot initially at the top-left corner (i.e., $grid[0][0]$). The robot tries to move to the bottom-right corner (i.e., $grid[m-1][n-1]$). The robot can only move either down or right at any point in time.

An obstacle where robot could not move and the free space are marked as *1* or *0* respectively in grid.

A path that the robot takes cannot include any square that contains an obstacle.

Return the number of possible unique paths that the robot can take to reach the bottom-right corner and help him.

Constraints:

- $m == obstacleGrid.length$
- $n == obstacleGrid[i].length$
- $1 \leq m, n \leq 100$
- $obstacleGrid[i][j]$ is *0* or *1*.

Input format:

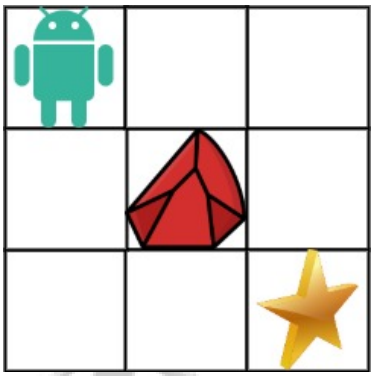
The first line contains two integers *'m'* and *'n'* denoting the number of rows and columns of the grid.

The next 'm' lines contain *'n'* single space-separated integers (0 *or* 1) denoting the elements of each row.

Output format:

Single integer denoting the no. of unique steps.

Example 1:



Input: $obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]$

Output: *2*

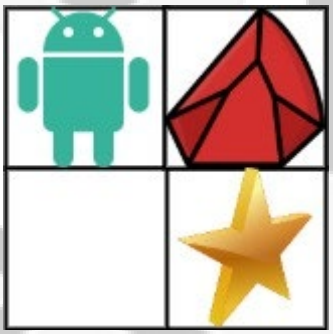
Explanation:

There is one obstacle in the middle of the 3 × 3 grid above.

There are two ways to reach the bottom-right corner:

1. Right → Right → Down → Down
2. Down → Down → Right → Right

Example 2



Input: $obstacleGrid = [[0,1],[0,0]]$

Output: *1*

Contacts:

Anas : +91 73790 83468

Ganishk : +91 95668: 53093

Computer Vision

Task 1

Recreate the popular game of Snake using OpenCV. Assign a region of the screen as a control pad (with up, down, left and right buttons). You must be able to place any object of your choice over one of these buttons, which must register as a move of the snake. The snake should move across the screen and be able to collect randomly placed insects (which can be represented using circles). Display the score at one corner of the screen.

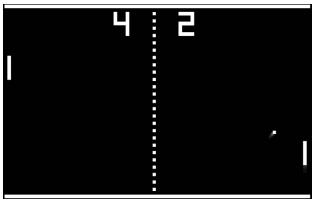
Bonus: Increase the length of the snake as the points increase.

Concepts to learn: Drawing using OpenCV, Object detection.

Task 2

This task involves creating a game of Pong using computer vision. The game consists of a screen divided into two halves, and two different coloured objects (chosen by the players) serve as the paddles. The objective is to use the coloured objects as paddles to bounce a ball off them. As the participants move their respective coloured objects (paddles) within their designated halves of the screen, the ball will interact with the paddles, bouncing off them accordingly. If the ball touches the backside of a player's half, a point is scored by the opposing player. The participants must utilize their coloured objects (i.e. paddles) strategically to prevent the ball from reaching the end of their own half, while attempting to score points by maneuvering the ball past their opponent's paddle. Also, the score of each player must be displayed on their half of the screen.

The final game should imitate the following layout:



Note that the background does not necessarily need to be blackened out. The aesthetics are completely upto you, we'd love to see what you can come up with 😊.

Concepts to learn: Contour detection, Object detection, drawing using OpenCV.

Task 3

Given is an image of a newspaper article. Your job is to read the sentence and convert it into a string. After converting it to a string, count the number of words, the number of digits and the number of special characters.

Concepts to learn: Character recognition, Image to text conversion.

Task 4

The user shows a number using their fingers. They show two numbers – one on each hand. Your task is to identify the number shown by the user by counting the number of fingers. Add the numbers shown on both hands and display the output on the screen. If the sum of the numbers shown on both hands is a multiple of two, display the number in green, else display it in red.

Note: No external ML libraries should be used.

Concepts to learn: Counting fingers

Contacts:

Aditya : +91 75185 36909

Gayathri : +91 81977 68603

Machine Learning

Task 1

Aim: Vehicle count detection using CNN model with transfer learning.

Task:

Develop a vehicle count detection system using a convolutional neural network (CNN) model. The goal is to train the model to accurately detect and count the number of vehicles in images. To enhance the model's performance, utilize transfer learning by initializing the model's weights with a pre-trained model.

Note: Collect your own dataset of traffic images containing vehicles and perform Image Augementation to train the CNN model. The dataset should include a diverse range of traffic scenarios and vehicle types to ensure the model's generalization ability.

Bonus:

Extend the code to detect and count the number of vehicles from your webcam.

Download and show the provided video in front of your webcam and check the results.

Concepts to learn: CNN, Transfer Learning, Data collection and preprocessing.

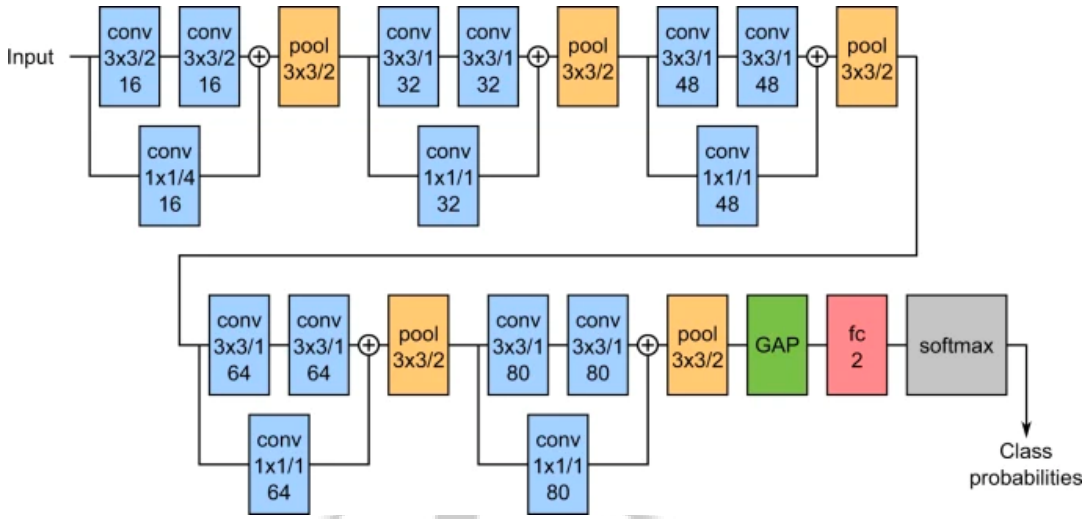
Submission Instructions: Submit a well-documented python code.

Task 2

Aim: To detect whether a person is infected with tuberculosis using lung X-rays.

Task:

Develop a Convolutional Neural Network that can identify whether a given person is diagnosed with tuberculosis or not using the x-ray images. The network architecture to be used is given below. It is further described in the link that is attached.



Links:

[Article]

[Dataset]

Concepts to learn: Convolutional Neural Networks, PyTorch.

Submission Instructions: Submit a well-documented python code.

Contact:

Gayathri : +91 81977 68603

Srivattsan : +91 75199 81998

Mechanics

Task 1

Aim: To design a differential drive robot.

Task:

Design a differential drive robot with two rear wheels and a castor wheel at the front. The robot should contain a bed which can be tilted like a dump truck tank. The whole assembly should be able to fit in a cuboidal box of dimensions $40 \times 20 \times 30\ cm^3$ (length $\times$ breadth $\times$ height) and there should be a minimum ground clearance of $5\ cm$. Clearly specify all the required parameters you have taken such as wheel base, ground clearance, length of the chassis, radius of the wheel chosen etc.

Bonus: Perform motion analysis for the wheels of the robot in any solid modelling software.

Task 2

Aim: To design a mechanism to tilt the dump bed.

Task:

Design an optimal mechanism which can be used to tilt the bed of the differential drive robot from **Task 1**. Design a lock to this mechanism so that you can hold the bed at any desired orientation. You are free to incorporate any electronic or mechanical component that is available. Mention the maximum elevation that can be achieved with your design.

Bonus: Do this task in a solid modelling software and try to assemble it with your differential drive robot.

Task 3

Aim: To move the differential drive robot in a desired path.

Task:

Assume that your differential drive robot is placed on the xy plane of the right-handed inertial coordinate system xyz. Another right-handed coordinate frame $x'y'z'$ is attached to the centre of the wheel axis of the

robot, whose projection is on the xy plane is denoted as "A". Initially "A" is at a distance of 3 units along the x axis and 0 units along the y axis in such a way that the y' axis is parallel to the y axis. Assume that all the required velocities can be achieved instantly. There is enough traction between the wheels and the plane such that there is no slippage of wheels.

Your robot is carrying a load in its dump bed which has to be transported to an unload point. The coordinate of the unload point is $B(-3,0,0)$ in the xyz coordinate system. If you have control over angular velocities of the left and right wheels of the robot, specify the angular velocities and directions of rotation of the wheels so that "A" aligns with "B" while the distance between "A" and the origin of the inertial reference frame is constant and equal to 3 units at every instant. Also, find the time at which it reaches the destination.

Topics to learn: Differential drive kinematics.

Task 5:

Aim: Finding the required orientation of the turret.

Task:

Assume that you have a mechanism similar to the turret of a tank or warship mounted on the bed in such a way that there is only relative rotational motion is between the turret and the bed about their common normal. Dimensions of the turret and the gun can be assumed by you in such a way it satisfies the constraints imposed in Task 1 and it should be able to launch a ball of $1\ cm$ radius and $200\ g$ mass. The velocity with which the turret launches the ball is $1\ m/s$. Assume the momentum transfer to be happening in $0.01\ sec$. Take the mass of the turret to be t kg.

Let's assume that the robot is at rest while shooting the ball. Write a code for calculating the required orientation of the turret to be able to hit the target at a given position (x, y, z) defined in $x'y'z'$ coordinate system mentioned in the **Task 3**. Assume that the given point is inside the workspace of the turret. Calculate the recoil force of the turret as well.

Task 5

Aim: To counter the recoil force in the differential drive robot.

Task:

As we know, while shooting a missile from the tank there will be a recoil force. In tanks or warships, it is countered by employing muzzle brakes. In **Task 4** the robot is assumed to be at rest which is only possible if we have some mechanism to counter the recoil. Devise a method, which will nullify the recoil force on the robot. You are free to make any design modifications to your robot and you are free to use any available electronic or mechanical components.

Submission Guidelines:

1. For design tasks you can also illustrate your design with a proper **hand drawing** subject to *ISO 8015* standard.
2. Make a drive a folder named *"Mechanics"* in the main folder.
3. Upload all files related to the task into their respective folders.
4. All the written content should be legible and understandable.
5. Submit the screenshot of different views in full screen if you use a design software.
6. Submit full screen recording of simulations.
7. Adhere to the general instructions in addition to the above for creating the folders as specified.

Contact:

Akhil : +91 73308 17529