Last Task

In this task of **Simulation** , you will design and build a bike that can balance and move in a virtual environment. Up until this task you have already installed software that are required and have a good understanding of the world of mathematical modeling along with Control Systems.

The task is divided into 3 sub-tasks:

Task 3A : Designing the Bike (Bike Banao)

Here you'll design and assemble a full bike in CoppeliaSim using the provided constraints. You will finally get to know how the rotary inverted pendulum system maps with the Lunar Scout !

Task 3B : Balancing & Yaw control (Balance Karwao)

Once the bike is designed, It's time to test the functionality & see if the design is really well thought - considering the dynamics of the system. The mathematical modeling that you have previously done for simple rotary inverted pendulums will have to be extended for your bike.

Task 3C : Path traversal (Chala ke dikhao)

Finally, if the self-balancing bike successfully stands on it's own, then it's time to make it move.

Last Task

Aim:

- In this Task you'll design your own Bike model in CoppeliaSim with the help of certain existing models that we will provide. The bike will use **an Omniwheel** and a normal drive wheel.
- We hope you have got a good introduction about CoppeliaSim now, because here it will be used extensively.
- Task 3A.zip file is to be used for this task, which has following files :
- Task_3A.ttt
- COM_tool.ttm

Steps:

1 . Create bike's body or chassis from primitive shapes - according to the BEST MASS DISTRIBUTION as per your intuition. **Decide what type of mass distribution will be helping bike balance easily**

2. Group all shapes you used for body & convert into single compound object - **"bike_respondable"**
3. Create Parent-Child Relationship in given hierarchy in Task_3A.ttt scene to complete the bike. RUN simulation & check if it works(**No error & parts stay joined, still movable**).
4. Create a **Visual design of bike chassis(without wheels) in CAD** software → then **IMPORT it in CoppeliaSim** & add as child object in hierarchy. **Change visual properties of bike**
5. Export as the bike hierarchy as re-usable model with the extension as ***".TTM"***

Details & RULES :

- There are few **pre-made objects provided in Task_3A.ttt. These should NOT BE REMOVED or manipulated**. They should be a part of hierarchy without a change.

Following is the list of elements provided and their uses:

▌ The elements higlighted in **yellow** are to be added by you.

NAME	*OBJECT TYPE*	*USAGE*	*TASK for User*
reference_frame	primitive shape	In future, for orientation feedback	DO NOT CHANGE
spherical_joint	JOINT(free)	connecting joint for required Degree of Freedom	DO NOT CHANGE
bike_respondable	primitive compound shape	GROUP of primitive shapes making bike body	To BUILD, GROUP and set Properties, etc.
front_motor	JOINT(velocity)	Motor for controlling Omniwheel/Yaw Angle	DO NOT CHANGE, Just Arrange in Hierarchy
omniwheel_sphere	primitive shape	Omniwheel respondable part	DO NOT CHANGE
roller	JOINT(free)	connecting joint for required Degree of Freedom	DO NOT CHANGE
roller_sphere	primitive shape	Omniwheel Roller respondable part	DO NOT CHANGE
omniwheel_visual	simple shape(imported mesh)	Omniwheel visual part	DO NOT CHANGE
drive_motor	JOINT(velocity)	Motor for driving forward/backward	DO NOT CHANGE, Just Arrange in Hierarchy
drivewheel_sphere	primitive shape	Drive-wheel respondable part	DO NOT CHANGE
drivewheel_visual	simple shape(imported mesh)	Drive-wheel visual part	DO NOT CHANGE
bike_visual	simple shape(imported mesh)	Massless, non-dynamic visual overlay of bike	To IMPORT, Re-size, Set properties, etc.

Last Task

Design Restrictions:

- **Do not edit or change any properties of the elements mentioned in the table above.**
- The Bike that you will build should **only use Omniwheel** based balancing.(algorithm will be completed in next task 3B)
- ***"Bounding box"*** is a virtual cuboid which specifies the maximum permissible dimensions of the Bike : length x height x width **(30cm x 20cm x 10cm)**

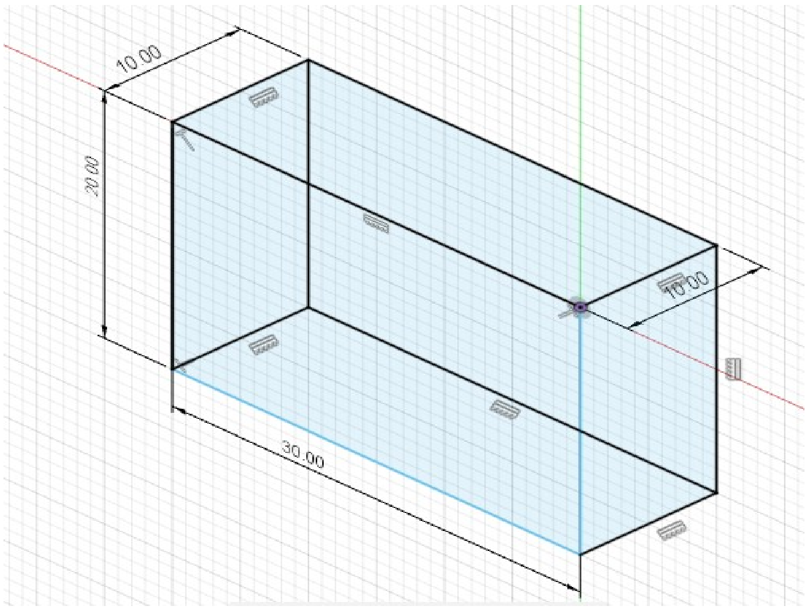
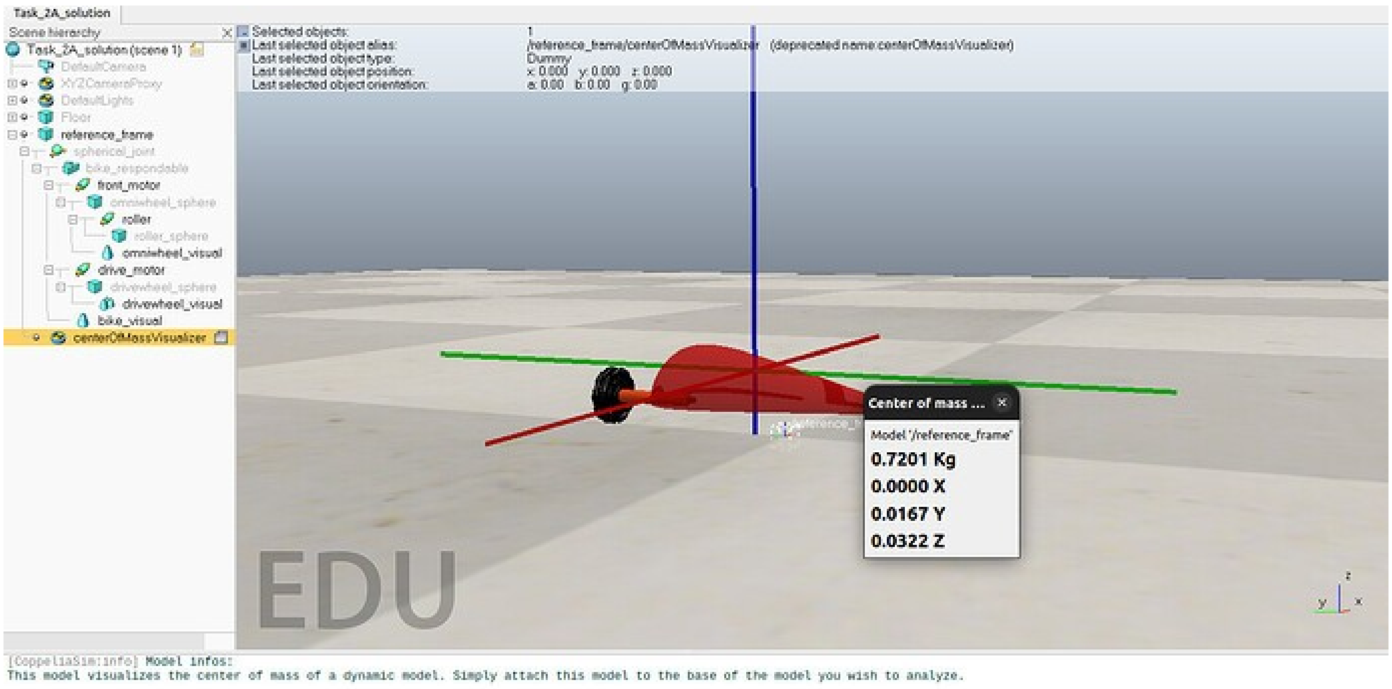


Figure 2: Bounding Box of Bike

- The maximum permissible total weight of whole model is **1.5kg**.
- **Points of contact with ground (2 max.)** : Bike should have only two elements touching the ground i.e. the front and rear(drive) wheel.
- For primitive shape like cuboid, sphere, cylinder etc. the center of mass is at it's geometric center. But for primitive compound shape(Grouped primitive shape) it is not same. You can see Geometric center where **local/object frame axes in red, blue, green** are visible inside object.
- Center of Mass of **"bike_respondable"** should be at **>= 2.5cm height from the floor**(world coordinate).
- To find the center of mass of the model, we have given a tool to visualize **"COM_tool.ttm"** in the downloaded folder. It will show the global position of center of mass and the value of total mass.

Last Task

- This tool works only with models, not objects. So for that first make sure that the Parent(Top-Most) Object of hierarchy is a "***model base***". To make it a model base, ***double-click*** on the parent object(HERE : **"reference_frame"**) to open the Scene object Properties windows, then click on Common, and check Object is model base.



```
Selected objects:                  1
Last selected object alias:        /reference_frame/centerOfMassVisualizer   (deprecated name:centerOfMassVisualizer)
Last selected object type:         Dummy
Last selected object position:     x: 0.000   y: 0.000   z: 0.000
Last selected object orientation:  a: 0.00    b: 0.00    g: 0.00
```

Center of mass ...

Model '/reference_frame'
0.7201 Kg
0.0000 X
0.0167 Y
0.0322 Z

```
[CoppeliaSim:info] Model infos:
This model visualizes the center of mass of a dynamic model. Simply attach this model to the base of the model you wish to analyze.
```

CoppeliaSim Hierarchy

- Inside CoppeliaSim on the left hand side you will find a block named Scene Hierarchy, on expanding the block you can look at the arrangement of components present, this arrangement defines the hierarchy or parent child relationship.

> **Important Note:**
> Make sure that this hierarchy is set up **as it is, including the naming convention.**

Hierarchy description:

Last Task

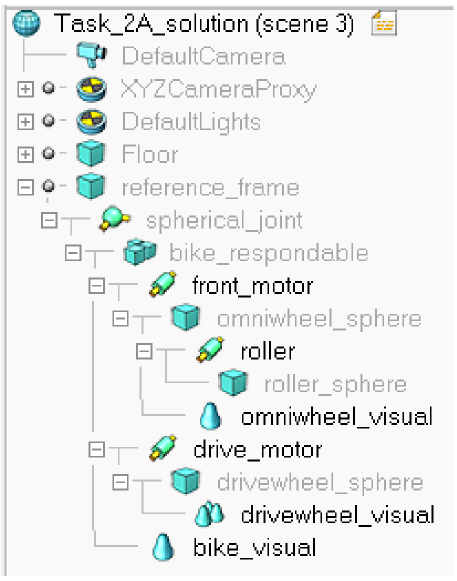


Figure 4: CoppeliaSim Hierarchy

Chassis Design

- The main objective here is to design the chassis using only **primitive shapes** in CoppeliaSim. (There are other types of shapes like compound shapes etc. available - but *they make the dynamics tough to calculate for physics engine* & hence *errors and unpredictability may happen*. Hence, it's suggested **NOT TO USE other shapes** .)

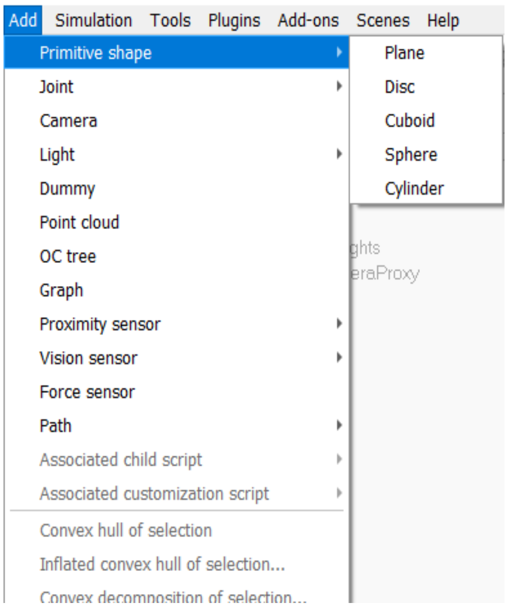


Figure 5: Primitive Shape menu in CoppeliaSim

Refer the following to learn how to use simple shapes : Click Here

Refer the following Video to learn how to **GROUP primitive shapes**: Click Here

Last Task

(But Remember : **DO NOT MERGE** as it is different operation & not recommended)

Designing Visual Skin of Bike using CAD

Before this step, we hope that you have created the bike's model in CoppeliaSim. Now your bike is functional!!

Going further, let's take this chance to give good looks to our bike so that it looks ***like a real bike*** and also learn a very important skill - **3D CAD design!**

> You are free to choose a simplistic design if you find CAD difficult. Do not take much load here. You can use Solidworks, Fusion 360 for it.

Autodesk Fusion 360 as an option:

- Watch this video to Install Fusion 360 (Education license):-

  Fusion 360 Installation | Education License 98

- Refer to the following video, to understand **how to design and import mesh into CoppeliaSim** via Autodesk Fusion360.

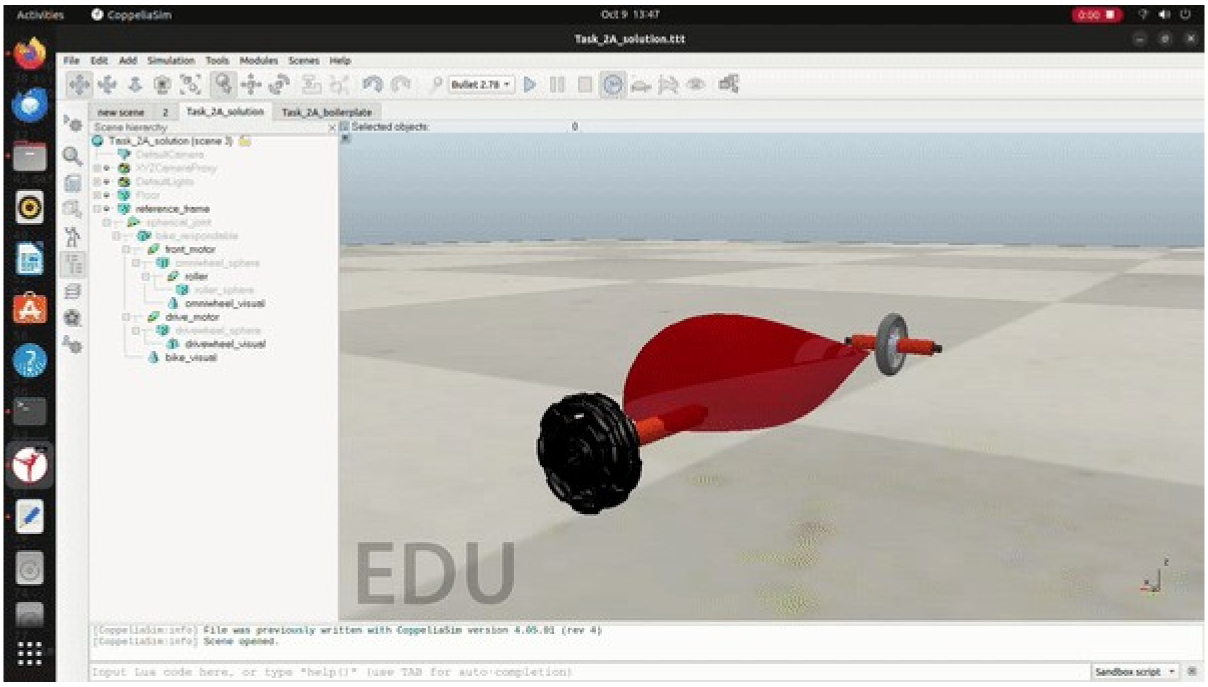
> **NOTE: This video talks in reference to an older theme called Dairy Bike. So please ignore any details mentioned related to any task in this video, as there are difference between the two themes.**

- However you can design the visual aspects in Fusion 360.
- Important point to understand here is that Visual aspects will be added as **non-dynamic** and **non-respondable** objects so they won't affect your RTF (Real Time Factor)

The making of complete Bike

- The chassis(***bike_respondable***) will act as a connecting element to the different components we have provided.
- The final bike & the hierarchy will look something like the following GIF. The RED visual part is just an EXAMPLE - you can be creative there !

Last Task



```
[CoppeliaSim:info] File was previously written with CoppeliaSim version 4.05.01 (rev 4)
[CoppeliaSim:info] Scene opened.
```

Figure 6: Bike Model Visual Properties in CoppeliaSim

Exporting as CoppeliaSim Model File (*.TTM)

- First make sure that the Parent(Top-Most) Object of hierarchy is a "***model base***". To make it a model base, ***double-click*** on the parent object(HERE : **"reference_frame"**) to open the Scene object Properties windows, then click on Common, and check Object is model base.
- Click on "model base" → You can now do : File → Save model as…
  follow the instructions until the end…
- To make it appear in the Model browser, just put your .ttm file under the model folder of CoppeliaSim
- It's ready to be re-used !!! Try dragging & dropping it in any blank scene !!

That's it !! Task 3A is complete!!

Congrats!!

Last Task

Task 3B

Aim

We hope that before starting this task, you have already completed Task 3A and built a CoppeliaSim model of the Bike !

In this task, you will have to import the bike model(***.ttm** file) you have created and **balance the bike at single point using PID or LQR control** algorithm. You have already derived the mathematical model and learnt how to apply and tune control logic in previous tasks.

Few things new in this task:

> **Any kind of additional script is not allowed inside our outside the coppelia scene**

•**Reference Frame :** You must have observed that there is a non-respondable object "**reference_frame"** in the hierarchy as parent of your model, as suggested in Task 3A. **You have to get the orientation of bike with respect to this frame.**

•You are allowed to correct your bike model if it was wrong, but it should still satisfy Task 3A criteria - before you use it for this task.

Downloadables

The boilerplate Task3B.zip contains a folder Task3B which will have the following files:

•Task_3B.ttt (scene without the bike)

Understanding RUN:
Open the scene **Task_3B.ttt** → then import the bike model by dragging & dropping it from model browser → Position the model base object(reference_frame) on top of the origin on the floor (x,y)=(0,0)

•Add your control logic similar to Task2B
•Modify your mathematical model for the bike model. You can find the mass, dimensions and moment of inertia in required units from properties menu → by double clicking objects.

•Balance , Balance , Balance , keep doing it, tune it more and more, it will balance.!!

> **NOTE:** You are allowed to use only **front_motor** JOINT **velocity** to complete this task.
> ALL OTHER WAYS OF COMPLETING TASK ARE INVALID.

Superb!! Way to go for next task → →

Last Task

Task 3C

Aim:

Now as your bike & control algorithm are both mature enough - in this task you have to actuate the drive motor and make the bike move around.

Few things new in this task:

•**Manual Control:** In this task, the balancing will be AUTOMATIC - by your control algorithm from child script. But the motion of the bike and actions like forward, backward, turning etc. will be MANUAL, via **keyboard input**.

Downloadables

The boilerplate Task3C.zip contains a folder Task3C which will have the following files:

•Task_3C.ttt (scene without the bike)

Understanding Boilerplate:

Open the scene **Task_3C.ttt** → then import the bike model by dragging & dropping it from model browser

•Add a **PYTHON** child script to **front_motor**.

•A python API stub is given below for the manual keypress signalling in CoppeliaSim Scene

For example, you can start with the following code in **sysCall_sensing():**

```
############### Keyboard Input ##############
    message,data,data2 = sim.getSimulatorMessage()

    if (message == sim.message_keypress):
        if (data[0]==2007): # forward up arrow
            drive_speed = #add drive wheel speed here

        if (data[0]==2008): # backward down arrow
            drive_speed = #add drive wheel speed here

        if (data[0]==2009): # left arrow key
            yaw_setpoint = #change yaw_setpoint for required turning over here

        if (data[0]==2010): # right arrow key
            yaw_setpoint = #change yaw_setpoint for required turning over here
    else:
        drive_speed = # This is an example, decide what's best
        yaw_setpoint = # # This is an example, decide what's best
    ########################################
```

Balancing still has to be automatic(from PID or LQR).

•Add your control logic similar to Task3B

→ Your solution code in Child script should help bike balance(using PID or LQR) and then user can give keyboard inputs to control forward, backward and turning motion of bike.

Last Task

THE END