eYRC 2020-21: Nirikshak Bot (NB)

Task 4B

Develop Ball Navigation Algorithm

[*Last Updated on:* ***05th January 2021, 20:00 Hrs***]

NOTE: Before proceeding further, make sure you have **thoroughly understood** the **concepts of previous tasks**.

---

1. Problem Statement

Develop an algorithm **to navigate the ball through the maze** on top of the ball balancing platforr in the given CoppeliaSim scene.

---

2. Given

1. CoppeliaSim's Scene File (`task_4b_scene.ttt`)

- A scene file i.e. `task_4b_scene.ttt` of CoppeliaSim software as shown in Figure 1.

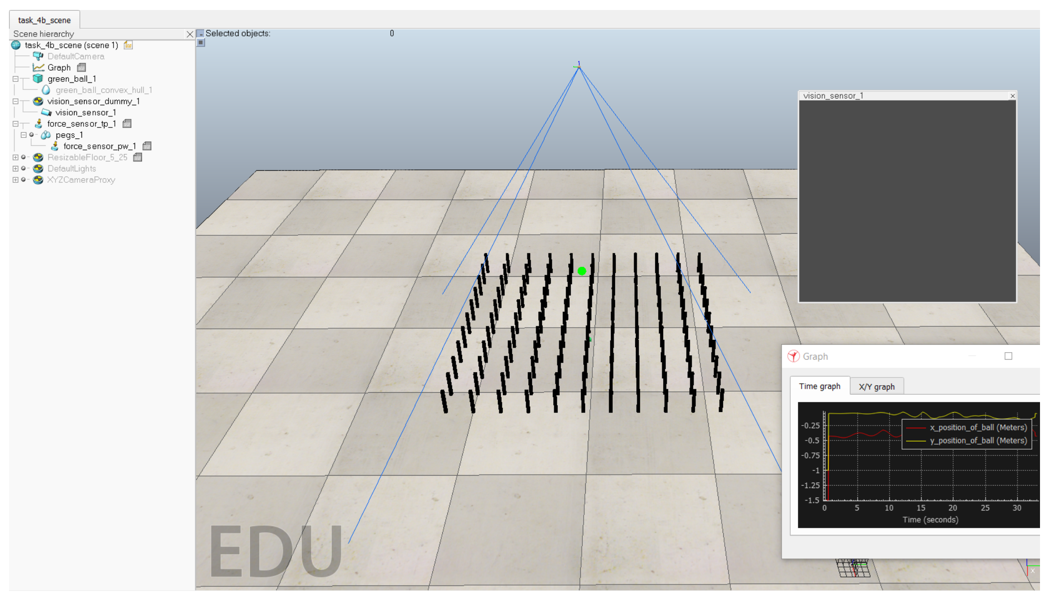

Figure 1: CoppeliaSim scene file for Task 4B.

- The **objects in the scene** along **with their names and uses** are given in Table 1.

Objects	Name in the scene	Use(s)
Green Ball	*green_ball_1*	**Green color ball** that is supposed to be steered from one cell of the maze to the other.
Green Ball Convex Hull	*green_ball_convex_hull_1*	This will be used for **detecting collision** with the wall.
Vision Sensor Dummy	*vision_sensor_dummy_1*	**Dummy** used to position and orient Vision Sensor.
Vision Sensor	*vision_sensor_1*	Works as the **camera** in the scene. The **output of vision sensor** is supposed to be used for **Image Processing**. It has resolution of 1024x1024 pixels.
Force Sensor TP	*force_sensor_tp_1*	This will serve as the **rigid link** between top_plate_respondable_1 and pegs_1.
Pegs	*pegs_1*	Pegs will be used as a **support for the walls** of the maze.
Force Sensor PW	*force_sensor_pw_1*	This will serve as the **rigid link** between *pegs_1* and *walls_1*.
Graph	*Graph*	**Plots** the **x and y CoppeliaSim co-ordinates of the ball** when the simulation is running.

Table 1: Objects along with their names and uses in the given scene file.

NOTE: To make it easier for teams to balance the ball in this task, we have increased the moment of inertia of the ball by **100 times**.

- A floating view of the output of ***vision_sensor_1*** is also shown.

NOTE: In this task:

- You are **NOT** allowed to **remove** the above mentioned objects.

- You are **ALLOWED** to **change the resolution of vision sensor** according to your logic

---

- Teams will have to **import their ball balancing platform** from Task 3.

2. Main Script (`task_4b.py`)

- The python script i.e. `task_4b.py` **NEEDS** to be edited by the respective teams.
- It **contains functions** which are going to be called by `task_4b_script.lua` and `test_task_4b.exe` or `test_task_4b` .
- The **details of each function** are mentioned in the file and in this documentation. Read **both the files carefully** before attempting the task.

3. Lua Script (`task_4b_script.lua`)

- This script is **similar** to the **customization script** made by teams **in Task 2B**.
- Teams will have to **complete the new functions added** in this script and **paste the function already made by your team** in Task 2B.
- The **details of each function** is mentioned in the file and in this documentation. Read **both the files carefully** before attempting the task.

4. Test Cases Folder (`test_cases`)

- This folder contains maze images as were provided with **Task 4A**.
- The main script `task_4b.py` considers only one image from this folder as input.
- You can test your solution against other images provided in this folder to make it more robust and generic.

5. Test executable (`test_task_4b.exe or test_task_4b`)

- The team **should run this executable ONLY after they have completed** writing the `task_4b.py` script.

---

3. Getting Started

- Download the following zip file containing the above mentioned files. **Right-click** on the hyperlink and select **Save Link As...** option to download.

- Windows OS Users: [**Updated on 05-01-2021, 20:00 Hrs**]
  - **task_4b_develop_ball_navigation_algo_windows.zip**
- Ubuntu OS Users: [**Updated on 05-01-2021, 20:00 Hrs**]
  - **task_4b_develop_ball_navigation_algo_ubuntu.zip**
- Macintosh OS Users: [**Updated on 05-01-2021, 20:00 Hrs**]
  - **task_4b_develop_ball_navigation_algo_macintosh.zip**

NOTE: The browser might warn that the file can harm your PC, but it will not and you can safely download it.

- **Extract** the downloaded zip file.
- Copy `task_4a.py`, `task_3.py`, `task_2b.py`, `task_2a.py`, `task_1b.py` and `task_1a_part1.py` files made by your team in **Task 4A**, **Task 3**, **Task 2B**, **Task 2A**, ,**Task 1B** and **Task 1A Part1** respectively.
- Paste them in the extracted folder.
- Make sure you add following files in the same directory where all the Python scripts of this **Task 4B** are present.
  - `sim.py`
  - `simConst.py`
  - `remoteApi.dll` (***for Windows***) OR `remoteApi.so` (***for Linux***) OR `remoteApi.dylib` (***for Macintosh***)
- Refer **Task 0 Test Setup** for further details.
- **After completing the above steps**, your folder should have the following files:
  - `task_4b_scene.ttt`
  - `test_task_4b.exe` / `test_task_4b`
  - `task_4b.py`
  - `task_4b_script.lua`
  - `task_4a.py`
  - `task_3.py`
  - `task_2b.py`
  - `task_2a.py`

- `task_1b.py`
- `task_1a_part1.py`
- `sim.py`
- `simConst.py`
- `remoteApi.dll` (***for Windows***) OR `remoteApi.so` (***for Linux***) OR `remoteApi.dylib` (***for Macintosh***)
- `test_cases` folder

Now, read the following instructions carefully. **Any deviation from the listed instructions will result in poor evaluation and hence low marks**.

---

4. Understanding the Task

A. CoppeliaSim Scene (`task_4b_scene.ttt`)

- Export the model made by your team in Task 3 as a **CoppeliaSim Model File**. You can refer Figure 10 of Task 1C- Design Ball Balance Platform documentation in order to do so.
- Paste the **.ttm** file in:
  - Ubuntu OS - `CoppeliaSim_Edu_V4_0_0_Ubuntu18_04/models/`
  - Windows OS - `C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu\models\`
- Refer Figure 2 of Task 1C - Design Ball Balance Platform to understand better.
- After you open `task_4b_scene.ttt`, you have to **add this model file**.
- Position the parent of your design at **[0, 0, 0.0055]** and orient at **[0, -90, 0]** with respect to th world frame.
- **To check** that you have **positioned and oriented your design correctly**, make sure that whe you hit the play button of simulation, you are able to **match** the **position of the ball** in the **floating view of *vision_sensor_1*** and your **design**.

NOTE:

- Make sure that the **height** of your design or the **z co-ordinate** of the ***top_plate_respondable*** is **less than or equal to 0.35** in CoppeliaSim co-ordinates.
- Make sure that there is no gap between ***top_plate_respondable*** and ***pegs_1***.

Welcome to NB theme!
Rulebook
Task 0
Task 1
Task 2
Task 3
Task 4
4A - Path Planning
4B - Develop Ball Navigation Algorithm
4C - Theme Analysis and Implementation
Task 5
Practice Task
Instructions for Task 6
Task 6 Scene Details
Coding Standard
Git and GitHub
Live Session 1 - 24th October 2020
Live Session 2 - 21st November 2020
Live Session 3 - 12th December 2020
Live Session 4 - 10th January 2021
Changelog

- Now assign the hierarchy of ***force_sensor_tp_1*** to ***top_plate_respondable_1*** as shown in the Figure 2.

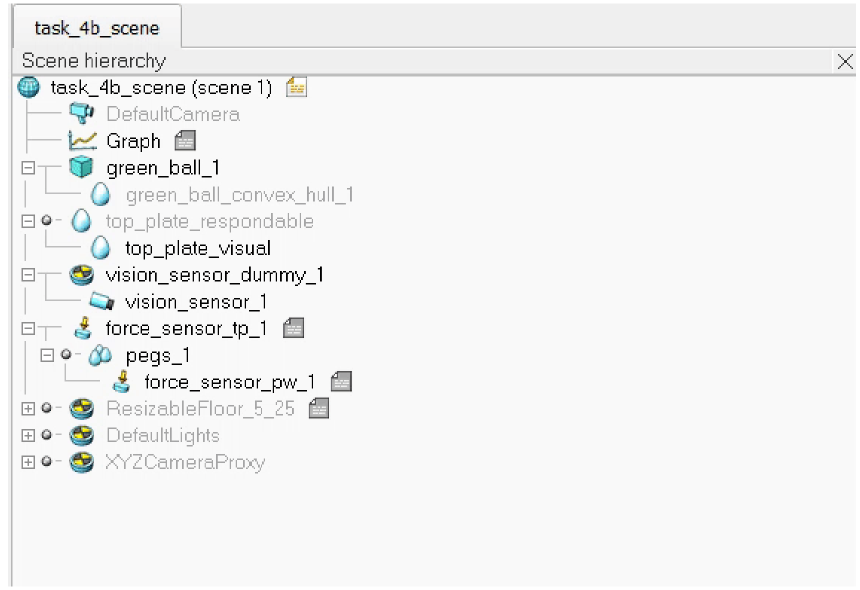

Figure 2: Assigning the hierarchy of force_sensor_tp_1 to top_plate_respondable_1.

- Click on the **Collections button**  from the **left toolbar pane** of CoppeliaSim scene and verify that a collection by the name of ***colliding_objects*** is added. This collection should have ***pegs_1*** already added in it. Refer Figure 3.

Welcome to NB theme!

Rulebook

Task 0

Task 1

Task 2

Task 3

Task 4

4A - Path Planning

4B - Develop Ball Navigation Algorithm

4C - Theme Analysis and Implementation

Task 5

Practice Task

Instructions for Task 6

Task 6 Scene Details

Coding Standard

Git and GitHub

Live Session 1 - 24th October 2020

Live Session 2 - 21st November 2020

Live Session 3 - 12th December 2020

Live Session 4 - 10th January 2021

Changelog

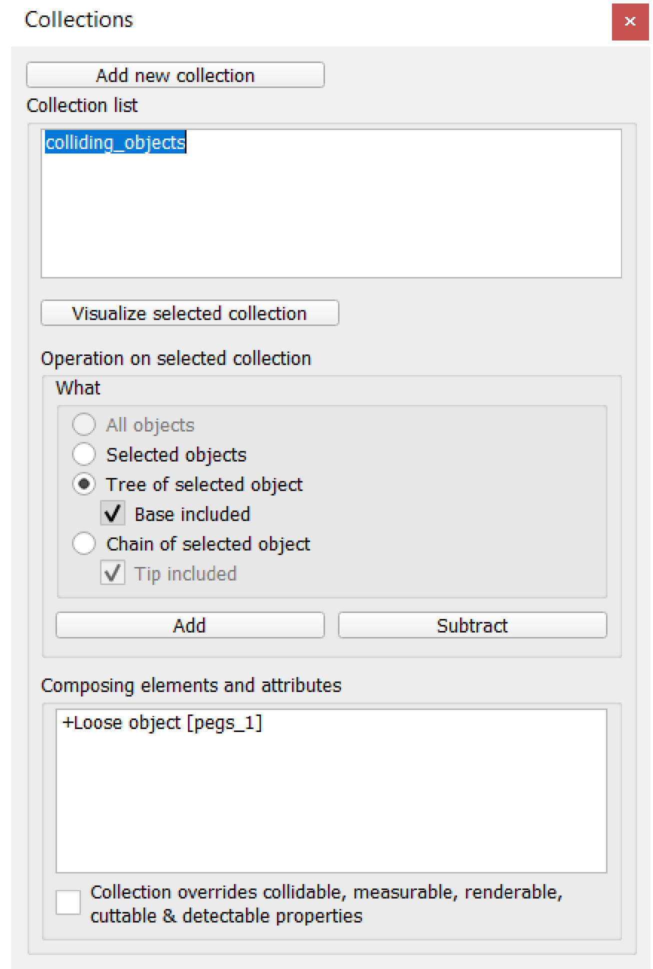

Figure 3: Verify *colliding_objects* collection in the scene.

- Click on **Calculation Module Properties**  button from the **left toolbar pane** of CoppeliaSim scene and verify that a **collision object** by the name of ***ball_collision*** is added. This **collision object** should have ***green_ball_convex_hull_1*** and ***colliding_objects*** collection already added in it. Refer Figure 4.

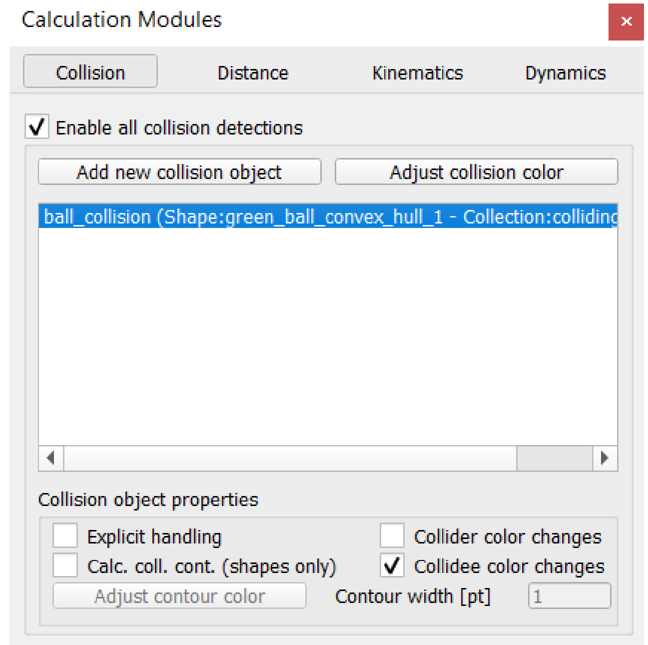

Figure 4: Verify *ball_collision* object in the collision calculation properties.

- Now add a customization script to ***top_plate_respondable_1***. Refer the gif shown in Figure 5 to add a customization script.

Welcome to NB theme!

Rulebook

Task 0

Task 1

Task 2

Task 3

Task 4

4A - Path Planning

4B - Develop Ball Navigation Algorithm

4C - Theme Analysis and Implementation

Task 5

Practice Task

Instructions for Task 6

Task 6 Scene Details

Coding Standard

Git and GitHub

Live Session 1 - 24th October 2020

Live Session 2 - 21st November 2020

Live Session 3 - 12th December 2020

Live Session 4 - 10th January 2021

Changelog

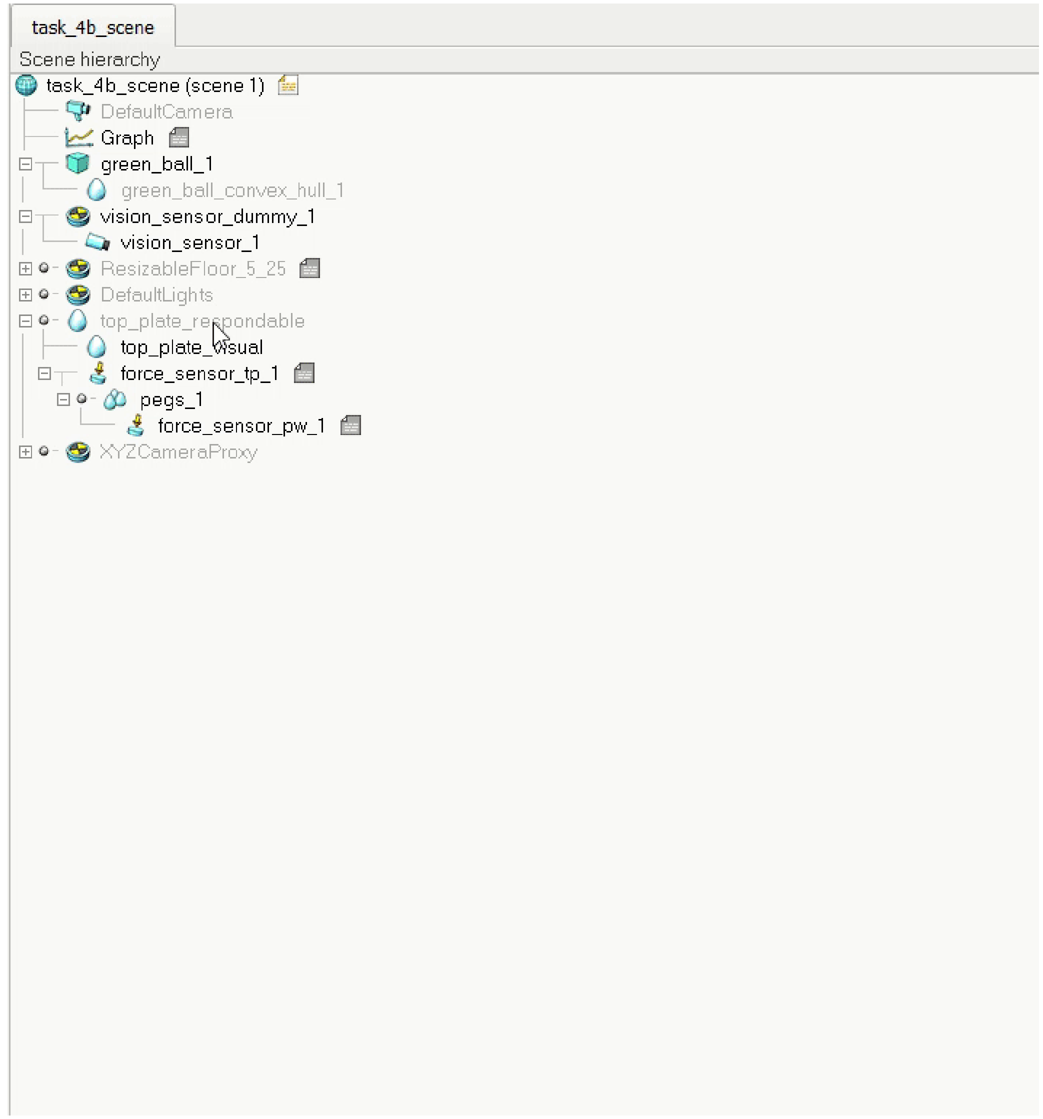

Figure 5: Add a customization script to *top_plate_respondable_1*.

- Copy the contents of `task_4b_script.lua` in this customization script.

NOTE: To reduce the difficulty in this task, **walls of the maze** and **pegs/support** are **set as non-renderable**.

B. Lua Script (`task_4b_script.lua`)

- The given script i.e. `task_4b_script.lua` has **3 new functions**. The other functions can be completed by **pasting the functions already made by your team** in Task 2B.
- **Two of these** functions i.e. `groupWalls()` and `addToCollection()` are **supposed to be completed by the teams**.
- **Remaining one** function i.e. `drawPath()` is given as an example for the teams. Teams **may/may not modify this function** depending on their logic.
- These functions along with their uses are shown in **Table 2**.

Sr. No.	Name	Use
1	`groupWalls()`	**Group** the various walls created.
2	`addToCollection()`	**Add the grouped walls** to ***colliding_objects*** collection.
3	`drawPath()`	**Draw the path** generated in CoppeliaSim.

Table 2: Functions to be edited by teams in customization script of *task_4b_scene.ttt*.

- The **details** for each of the mentioned **functions in Table 2** are shown in the underlying sections.

1. `groupWalls()`

Parameters	Details
Purpose	This function should **group the various walls** in the scene to one object. The **name of the new grouped object** should be set as ***walls_1***.
Input Arguments	None
Return	None
Example Call	`groupWalls()`

Table 3: ***groupWalls()*** function to be edited by teams in **customization script**.

2. `addToCollection()`

Parameters	Details

Parameters	Details
Purpose	This function should **add** the ***walls_1*** grouped object to the collection ***colliding_objects***.
Input Arguments	None
Return	None
Example Call	`addToCollection()`

Table 4: ***addToCollection()*** function to be edited by teams in **customization script**.

3. **`drawPath()` (May / May not be edited by teams)**

Parameters	Details
Purpose	This function should **build blue colored lines** in the **CoppeliaSim** scene according to the path generated from `task_4a.py` script.
Input Arguments	`inInts` : Table of Ints `inFloats` : Table of Floats (containing the path generated) `inStrings` : Table of Strings `inBuffer` : string
Return	`inInts` : Table of Ints `inFloats` : Table of Floats `inStrings` : Table of Strings `inBuffer` : string
Example Call	Shall be called from the python script.

Table 5: ***drawPath()*** function **may / may NOT** to be edited by teams in **customization script**.

NOTE: In `deleteWalls()` function, teams should now delete the grouped walls object i.e. ***walls_1***.

---

4. **`createWall()` (Should NOT be edited by teams)**

Parameters	Details
Purpose	Creates a **black-colored wall** of dimensions **90cm x 10cm x 10cm**. The mass of each wall is set to **0.1grams**. The walls should be **respondable only with the ball** and NOT with each other. Hence the **global respondable masks** should be set. The respondable mask is denoted by a **16 bit number**. Hence to set only the global respondable mask, we will have to pass **65280** (2^{16} - 2^{8}). Refer here and here.
Input Arguments	None
Return	`wallObjectHandle` : [**number**] the object handle of the wall created
Example Call	`wallObjectHandle = createWall()`

Table 6: ***createWall()*** function should **NOT** be edited by teams in **customization script**

C. Python Script (`task_4b.py`)

- **Two of the** functions in `task_4b.py` i.e. `convert_path_to_pixels()` and `traverse_path()` are **supposed to be completed by the teams**.
- **Remaining one** function i.e. `send_data_to_draw_path()` is given as an example for the teams. Teams **may/may not modify this function** depending on their logic.

Sr. No.	Name	Use
1	`convert_path_to_pixels(path)`	**Convert the obtained path** given by `task_4a.py` from cell numbers to pixels.
2	`traverse_path(pixel_path)`	Implements the **ball traversal algorithm**.

Sr. No.	Name	Use
3	`send_data_to_draw_path(rec_client_id,path)`	**Send data to customization script function** i.e. `drawPath()` to draw the path calculated.

Table 7: Functions to be edited by teams in **task_4b.py**.

- The **details** for each of the mentioned **functions in Table 6** are shown in the underlying sections.

1. `convert_path_to_pixels(path)`

Parameters	Details
Purpose	This function should **convert** the **path of** `task_4a.py` **from list of tuples to pixels**. Teams are free to choose the number of points and logic for this conversion. **NOTE:** Teams are free to choose the **appropriate resolution** of **images and vision sensor** in this task.
Input Arguments	`path` : [**list**] Path **returned** from `task_4a.py` ***find_path()*** function.
Return	`pixel_path` : **[type can be decided by teams]** Path to be given as setpoint(s) to the control logic designed by teams.
Example Call	`pixel_path=convert_path_to_pixels(path))`

Table 8: ***convert_path_to_pixels()*** function to be edited by teams in **task_4b.py**.

2. `traverse_path(pixel_path)`

Parameters	Details

Parameters	Details
Purpose	This function should **make the ball traverse the calculated path**. Teams are **free to choose logic for this function**. **Important** : Refer the code of main function in `task_3.py` . **NOTE**: This function is **ONLY** called **once** by main or the executable. Hence teams should write the code accordingly.
Input Arguments	`pixel_path` : **[type can be decided by teams]** Path to be given as setpoint(s) to the control logic designed by teams.
Return	None
Example Call	`traverse_path(pixel_path)`

Table 9: ***traverse_path()*** function to be edited by teams in **task_4b.py**.

3. **`send_data_to_draw_path(rec_client_id, path)` (May / May not be edited by teams)**

Parameters	Details
Purpose	This function should: 1. **Convert** and 2. **Send** a flattened path to `drawPath()` function of `task_4b_script.lua` .
Input Arguments	`rec_client_id` : [**integer**] ***client_id*** generated from start connection remote API, should be **stored in a global variable.** `path` : [**list**] Path **returned** from `task_4a.py` ***find_path()*** function.
Return	None
Example Call	`send_data_to_draw_path(rec_client_id,path)`

Table 10: ***send_data_to_draw_path()*** function to may OR may **NOT** be edited by teams i **task_4b.py**.

5. Testing the Solution

- Before testing the solution make sure you have **plugged in your laptop to power source** and **closed unnecessary applications** open in your PC.

NOTE: The installation of all software/libraries has been tested **only** on the following **64 bit OS**:

- ***Windows 7, 8 and 10***
- ***Ubuntu 16.04 and 18.04***
- ***macOS Big Sur v11.0.1***

- The files i.e. `task_4b.py` and `test_task_4b.exe` will call the following functions in the given order:

NOTE: The function `calculate_path_from_maze_image()` in `task_4b.py` will **call the first 3 functions** from Table 11.

Sr. No.	Function Name	Python script
1	`applyPerspectiveTransform(input_img)`	`task_1b.py`
2	`detectMaze(warped_img)`	`task_1b.py`
3	`find_path(maze_array, start_coord, end_coord)`	`task_4a.py`
4	`init_remote_api_server()`	`task_2a.py`
5	`send_data(client_id, maze_array)`	`task_2b.py`
6	`start_simulation()`	`task_2a.py`
7	`init_setup(client_id)`	`task_3.py`
8	`send_data_to_draw_path(client_id,path)`	`task_4b.py`
9	`convert_path_to_pixels(path)`	`task_4b.py`
10	`traverse_path(pixel_path)`	`task_4b.py`
11	`stop_simulation()`	`task_2a.py`

Table 11: List of functions that will be called by **task_4b.py and test_task_4b.exe** in the given orde

---

NOTE:

- In `task_2b.py`, the **input** to the `send_data()` function was only ***maze_array***. However, in this you need to add one more input parameter to this function, i.e. ***client_id***.
- We will be calling the `task_2b.send_data()` function and pass the ***maze_array*** returned from `task_1b.detectMaze()` along with ***client_id*** to the CoppeliaSim server to generate the maze in scene.

- The testing of solution is **divided into two** sub parts:

---

1. Using `task_4b.py`

- To test your solution:
  - Open the provided scene file `task_4b_scene.ttt` in CoppeliaSim.
  - Enable the **real time mode option** (if not enabled).
  - **Activate** your **Conda environment**, **navigate to the directory** where you have downloaded all the files and run:

    ```
    python task_4b.py
    ```

- The ***green_ball_1*** should be positioned at `[-0.429, -0.042, z]` where, the z-coordinate will depend on the height of the design, at the beginning of the run.
- The ball should **traverse the path** as **fast** as possible making sure to **avoid collision with th walls**.
- At the **beginning of the script**, the output should be as shown in Figure 6.

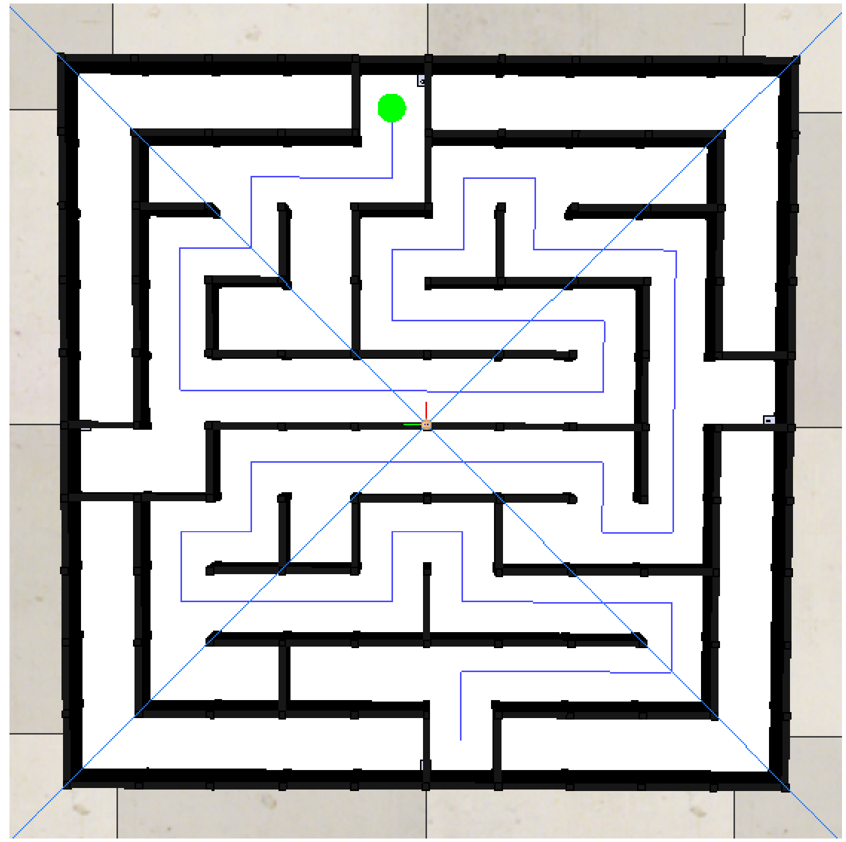

Figure 6: Expected output at the start of **task_4b.py** script.

---

2. Using `test_task_4b.exe` or `test_task_4b`

- This executable file will call the functions listed in Table 10.
- At the beginning of the run, **an image of the maze will be stored**. For this, the **maze and pegs** will be **set to renderable for a moment** and then **set back to non-renderable** by the `test_task_4b.exe`.
- It will also **store the path traversed** by the ball, **the number of collisions, deviations in the path traversed by ball w.r.t. path calculated by** `task_4a.find_path()` and the **simulation time elapsed**.
- An **evaluation model** will be loaded to **assess your files**. Do **NOT edit** this model.

- The **maximum time allotted** for this run is **600 seconds of simulation time**.

- To record the video of simulation, click on the  button of CoppeliaSim and set the parameters as shown in Figure 7.

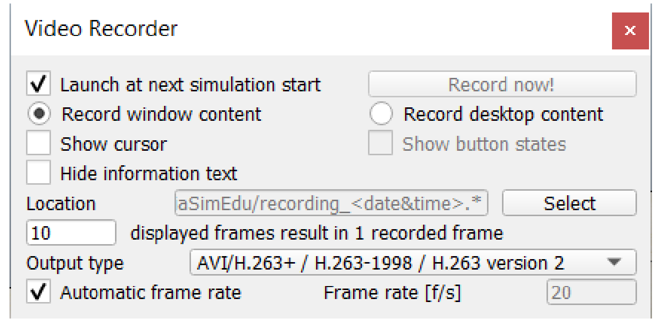

Figure 7: Video Recorder dialog box of CoppeliaSim.

NOTE:

- Do **NOT** change any of the above parameters. It may otherwise lead to **slow simulation** and **increase in lag**.
- Do **NOT** forget to **uncheck** the ***Hide information text*** option.

- Now, go to the **page selector**  button available in the **top toolbar of CoppeliaSim** and select page 8.

- The **view of the scene** should be as shown in Figure 8.

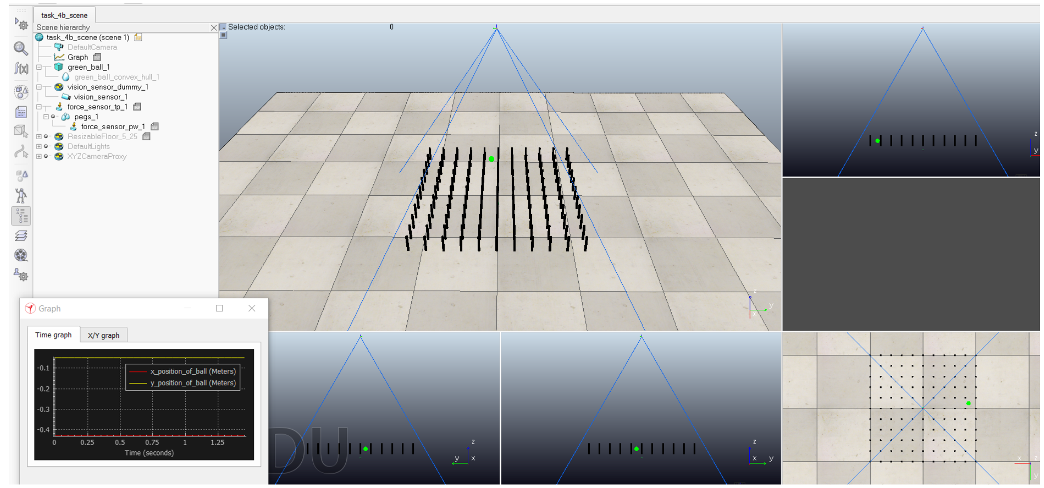

Figure 8: View of the CoppeliaSim scene before beginning recording of the video.

NOTE: In the above scene, the design is not visible. You are **NOT** allowed to **hide** any object/joint/force sensor/dummy.

- Make sure that your **Conda environment is activated**.
- When the `test_task_4b.exe` or `test_task_4b` is running, make sure you **do not disturb the code or the CoppeliaSim scene**.
- Navigate to the directory **where all the files mentioned were downloaded and extracted**.

NOTE: After you run the following command(s) in your respective OS, it may take **up to 1 minute** for the file to initialize.

- Type the following command:
- `test_task_4b.exe` (on **Windows**) OR
- `./test_task_4b` (on **Ubuntu** and **Mac**)
- If **there are no errors** you will see the output on your terminal as shown in Figure 9:

Select Anaconda Prompt (miniconda3) - "F:\Miniconda_3\Installation\condabin\conda.bat" activate NB_9999

```
(NB_9999) C:\Users\ERTS\Desktop\Task_4B>test_task_4b.exe

Welcome to test script for Task 4B of Nirikshak Bot (NB) theme.
Please enter your team ID: NB_9999

Connection to CoppeliaSim Remote API Server initiated.
Trying to connect to Remote API Server...

Connected successfully to Remote API Server in CoppeliaSim!
Evaluation script loaded successfully.

============================================

For maze00.jpg

Encoded Maze Array = [

============================================

Path calculated between (0, 4) and (9, 5) is = [

============================================

Simulation started correctly in CoppeliaSim.

============================================

Path sent to drawPath function of Lua script is


Path calculated between (0, 4) and (9, 5) in pixels is = [
```

```
============================================

Path successfully traversed.

============================================

Number of times the ball deviated from the path calculated by your task_4a.py is/are  0

============================================

'task_4b_output.txt' file written successfully. Please check the file in the same directory in which you are running the code

Simulation stopped correctly.

Disconnected successfully from Remote API Server in CoppeliaSim!

(NB_9999) C:\Users\ERTS\Desktop\Task 4B>
```

Figure 9: Correct output of executing **test_task_4b**.

- If there are **few deviations in the path traversed by ball** OR **path calculated is not traversed entirely**, you will see the output on your terminal as shown in Figure 10:

```
============================================

The path calculated was not traversed. Please try again.

============================================

Number of times the ball deviated from the path calculated by your task_4a.py is/are  10

============================================

'task_4b_output.txt' file written successfully. Please check the file in the same directory in which you are running the code.

Simulation stopped correctly.

Disconnected successfully from Remote API Server in CoppeliaSim!
```

- During **execution** of the file, you will find the output as shown in Figure 11.

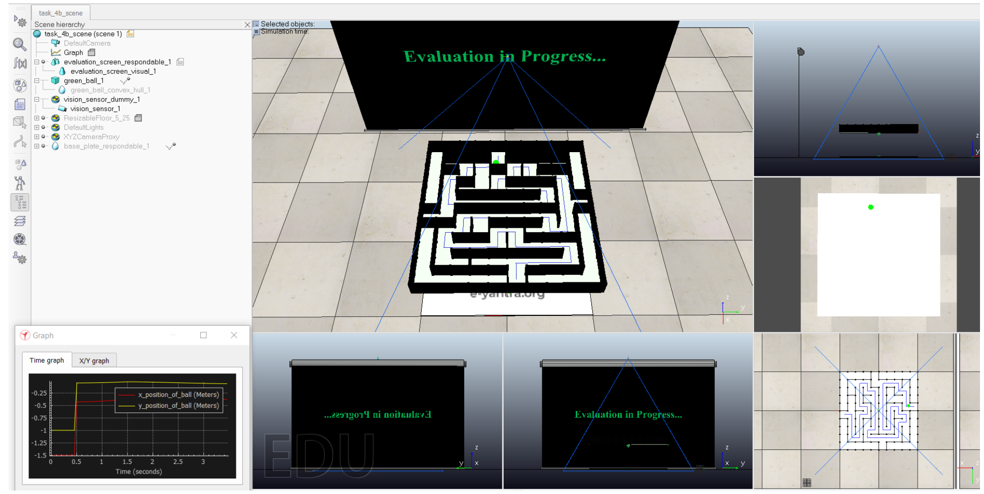

Figure 11: Expected output during execution of **test_task_4b**.

- After execution of the file, `task_4b_output.txt` will be **created in the same directory** i which the `test_task_4b.exe` OR `test_task_4b` was running.

Submission Instructions

For **Task 4B submission** you have to upload a **.zip** file. To create the appropriate file please follow instructions given below:

1. Create a new folder named `NB_<Team-ID>_Task_4B`.
   - For example: if your team ID is **9999** then you need to create a folder named `NB_9999_Task_4B`.
2. Now copy and paste following files into this folder:
   - `task_4b_scene.ttt` (with the **modified model**)
   - `task_4b_output.xml` (Refer Figure 8 of Task 1C - Design Ball Balance Platform documen to learn how to export CoppeliaSim scene as an XML file)
   - `task_4b_output.txt` (generated after running `test_task_4b.exe` OR `test_task_4b`)
   - `task_4b_simulation.avi` (generated by recording video of simulation from CoppeliaSin
   - `task_4b.py`

- `task_4b_script.lua` (**copy the CoppeliaSim customization script in a new file and save it as .lua extension**)
- `task_4a.py`
- `task_3.py`
- `task_2b.py`
- `task_2a.py`
- `task_1b.py`
- `task_1a_part1.py`

3. **Compress** this folder into **a zip file** and name it as `NB_9999_Task_4B.zip`.

4. Now go to the eYRC Portal and follow the instructions to upload this **.zip** file for **Task_4B** as shown in Figure 12.

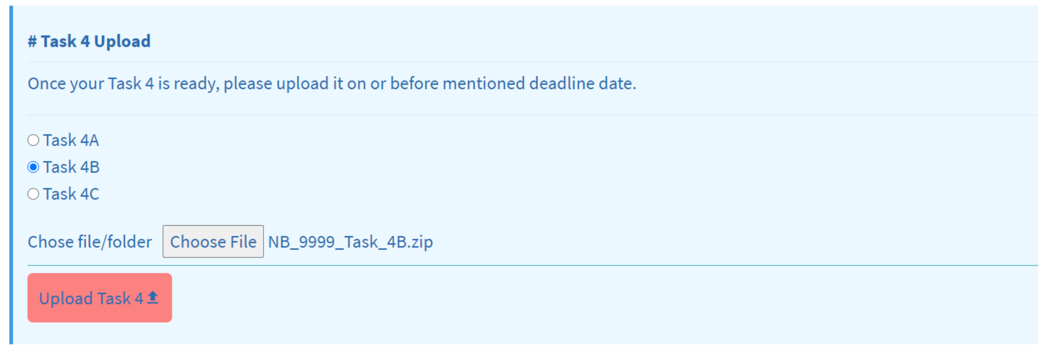

Figure 12: Submission of **NB_9999_Task_4B.zip** file on eYRC portal.

NOTE: File names mentioned are case sensitive. Verify all the file names before creating the zip file.

---

5. After you have **successfully submitted Task 4 files**, you can verify the zip file uploaded from the **'Verify Task 4 Upload'** section on the eYRC portal. The same is shown is Figure 13.

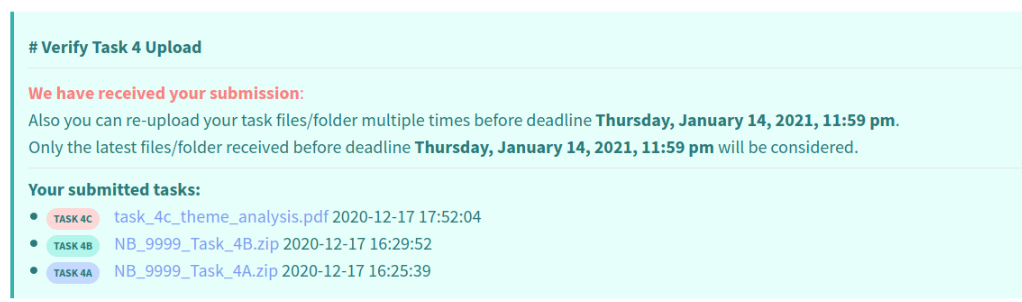

Figure 13: Successful submission of **Task 4** on eYRC portal.

ALL THE BEST!!