Task 2A

Develop a Ball Tracking Algorithm

[*Last Updated on:* ***12th November 2020, 09:29 Hrs***]

The aim of this task is as follows:

- Process the image captured by Vision Sensor in CoppeliaSim using Remote API
- Apply concepts of Task 1A to find the correct **Shape, Color, Centroid X and Centroid Y** of the ball(s) in dynamic CoppeliaSim scene.

NOTE: Before proceeding further, make sure you have thoroughly understood the concepts of **Task 1A** and **1B**.

---

1. Problem Statement

The aim of this task is to **develop an algorithm to track ball(s)** in the given CoppeliaSim scene.

---

2. Given

1. CoppeliaSim's Scene File (`task_2a_scene.ttt`)

- A scene file i.e. `task_2a_scene.ttt` of CoppeliaSim software as shown in Figure 1.

Figure 1: CoppeliaSim scene file (task_2a_scene.ttt).

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

Objects	Name in the scene	Use(s)
Top Plate Respondable	*top_plate_respondable_1*	Representing **respondable part** of the **top plate** in Ball Balancing Platform Bot
Top Plate Visual	*top_plate_visual_1*	Representing **visual part** of the **top plate** of Ball Balancing Platform
Ball 1	*ball_1*	**Red color ball** that is supposed to be tracked
Ball 2	*ball_2*	**Blue color ball** that is supposed to be tracked
Vision Sensor Dummy	*vision_sensor_dummy_1*	**Dummy** used to position and orient Vision Sensor

Objects	Name in the scene	Use(s)
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**.

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

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

NOTE:

- In this task you are **NOT** required to **change anything in the scene file**.
- Any change in the **names of the objects, their parent child relationships, their properties, position, orientation** etc. will result in **poor evaluation** and hence low marks.

---

2. Main Script (`task_2a.py`)

- The python script i.e. `task_2a.py` **NEEDS** to be edited by the respective teams.
- It **contains functions** which are going to be called by the `test_task_2a.exe` or `test_task_2a`
- The **details of each function** is mentioned in the file and in this documentation. Read **both the files carefully** before attempting the task.

3. Test executable (`test_task_2a.exe` or `test_task_2a`)

- The team **should run this executable ONLY after they have completed** writing the `task_2a.py` script.
- It **will evaluate** the `task_2a.py` , `task_1a_part1.py` and `task_1b.py` scripts and display the result scored in the task.

---

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**:
    - **task_2a_develop_ball_track_algo_windows.zip**

- **Ubuntu OS Users**:
  - **task_2a_develop_ball_track_algo_ubuntu.zip**
- **Macintosh OS Users**:
  - **task_2a_develop_ball_track_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_1a_part1.py` and `task_1b.py` files made by your team in **Task 1A** and **1B** and paste in the extracted folder.

NOTE:

- The teams are supposed to edit `task_1a_part1.py` and `task_1b.py` made by them in **Task 1A** and **1B**.
- These Python scripts are **not** included in the **zip file of Task 2A**.
- These Python scripts are already imported or called in `test_task_2a.exe` / `test_task_2a` .
- Do **NOT** change the names of these files.

- Make sure you add following files in the same directory where all the Python scripts of this **Task 2A** 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_2a_scene.ttt`
  - `test_task_2a.exe` / `test_task_2a`
  - `task_2a.py`
  - `task_1a_part1.py`
  - `task_1b.py`
  - `sim.py`

- `simConst.py`
- `remoteApi.dll` (***for Windows***) OR `remoteApi.so` (***for Linux***) OR `remoteApi.dylib` (***for Macintosh***)

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

- In this task you will have to edit **3 Python scripts**.
- These are `task_2a.py`, `task_1a_part1.py` and `task_1b.py`.
- The given script i.e. `task_2a.py` has **6 functions** that are supposed to be completed by the team.
- The remaining **2 functions** are supposed to be edited in `task_1a_part1.py` and `task_1b.py`
- These functions along with their uses are shown in **Table 2**.

Sr. No.	Name	Use	Python script to be used
1	`init_remote_api_server()`	Start a **communication thread** with the server i.e. **CoppeliaSim**	`task_2a.py`
2	`start_simulation()`	**Start** the CoppeliaSim **simulation**	`task_2a.py`
3	`stop_simulation()`	**Stop** the CoppeliaSim **simulation**	`task_2a.py`
4	`exit_remote_api_server()`	End the **communication thread** with the server i.e. **CoppeliaSim**	`task_2a.py`

Sr. No.	Name	Use	Python script to be used
5	`get_vision_sensor_image()`	Get the **image array** from **vision sensor** in CoppeliaSim scene	`task_2a.py`
6	`transform_vision_sensor_image(vision_sensor_image, resolution)`	Convert the **image array** obtained from vision sensor to **NumPy array**	`task_2a.py`
7	`applyPerspectiveTransform(transformed_image)`	Apply **Perspective Transform** to the image to isolate the ball and the table	`task_1b.py`
8	`scan_image(warped_image)`	Detect **shape, color, centroid X and centroid Y** from the warped image	`task_1a_part1.py`

Table 2: Functions to be edited by teams in **task_2a.py, task_1b.py and task_1a_part1.py**.

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

1. `init_remote_api_server()`

Parameters	Details
Purpose	This function should first **close any opened connection** and then **start a communication thread with server** i.e. Coppeliasim. The ***client_id*** **should be stored** in a **global variable**.
Input Arguments	None
Return	`client_id` : [***integer***] generated from the **start connection remote API**
Example Call	`client_id = init_remote_api_server()`

Table 3: ***init_remote_api_server()*** function to be edited by teams in **task_2a.py**.

NOTE: In this task, do not call the `stop_simulation` function **in case of failed connection** to the server. The `test_task_2a.exe` / `test_task_2a` file will handle that.

2. `start_simulation()`

Parameters	Details
Purpose	1. This function should first **start the simulation** if the connection to server i.e. CoppeliaSim was successful. 2. It should then **retrieve** the **time needed for a command** to be sent to the server, executed, and sent back to this function. This will help to **ensure synchronization** between your python file and the server.
Input Arguments	None
Return	`return_code` : [***integer***] generated from the **start running simulation remote API**
Example Call	`return_code = start_simulation()`

Table 4: ***start_simulation()*** function to be edited by teams in **task_2a.py**.

3. `stop_simulation()`

Parameters	Details
Purpose	This function should **stop the running simulation** in CoppeliaSim server.
Input Arguments	None
Return	`return_code` : [***integer***] generated from the **stop running simulation remote API**
Example Call	`return_code = stop_simulation()`

Table 5: ***stop_simulation()*** function to be edited by teams in **task_2a.py**.

4. `exit_remote_api_server()`

Parameters	Details

Parameters	Details
Purpose	1. This function should **wait for the last command** sent to arrive at the CoppeliaSim server **before closing the connection** 2. It should then **end the communication thread** with server i.e. CoppeliaSim **using simxFinish Remote API**.
Input Arguments	None
Return	None
Example Call	`exit_remote_api_server()`

Table 6: ***exit_remote_api_server()*** function to be edited by teams in **task_2a.py**.

5. `get_vision_sensor_image()`

Parameters	Details
Purpose	This function **should first get the handle of the vision sensor** and then using that, **get the vision sensor's image** array from the CoppeliaSim's scene.
Input Arguments	None
Return	`vision_sensor_image` : [***list***] **1 dimensional** array from get vision sensor image remote API `image_resolution` : [***list***] **resolution** of imagefrom get vision sensor image remote API `return_code` : [***integer***] return code generated from calling remote API
Example Call	`vision_sensor_image, image_resolution, return_code = get_vision_sensor_image()`

Table 7: ***get_vision_sensor_image*** function to be edited by teams in **task_2a.py**.

6. `transform_vision_sensor_image(vision_sensor_image, image_resolution)`

Parameters	Details

Parameters	Details
Purpose	This function should do the following: 1. Convert the **vision sensor's array to a NumPy array** with **data-type as *uint8***. 2. Resize new **1 dimensional** numpy array to a **3 dimensional NumPy** array. 3. Change the **color of the image** from **BGR to RGB** 4. **Flip** the image **about the 'x' axis**.
Input Arguments	`vision_sensor_image` : [***list***] **1 dimensional** array from get vision sensor image remote API `image_resolution` : [***list***] **resolution** of imagefrom get vision sensor image remote API
Return	`transformed_image` : [***numpy array***] resultant transformed image array after completing the above mentioned steps
Example Call	`transformed_image = transform_vision_sensor_image(vision_sensor_image, image_resolution)`

Table 8: ***transform_vision_sensor_image*** function to be edited by teams in **task_2a.py**.

NOTE: The following 2 functions should **not** be added/edited in the `task_2a.py` script:

- `applyPerspectiveTransform()`
- `scan_image()`

7. `applyPerspectiveTransform(transformed_image)`

Parameters	Details
Purpose	This function takes the new **transformed image** as input and applies **Perspective Transform** on it **to isolate the top plate and the ball**. The **resolution** of the **warped image** should strictly be **1280x1280**.
Input Arguments	`transformed_image` : [***numpy array***] resultant transformed image array after completing the above mentioned steps
Return	`warped_image` : [***numpy array***] after applying the Perspective Transform
Example Call	`warped_image = task_1b.applyPerspectiveTransform(transformed_image)`

Table 9: ***applyPerspectiveTransform*** function to be edited by teams in **task_1b.py**.

NOTE: The **resolution** of the warped image should strictly be **1280x1280**. Any deviation in this resolution will result in incorrect data being generated by the `scan_image` function.

- Make sure that **appropriate** value of the **threshold** is chosen.
- If the thresholding is **incorrect**, the output of `applyPerspectiveTransform` function will also be incorrect.
- Figure 2 compares the **appropriate and inappropriate** method of thresholding.

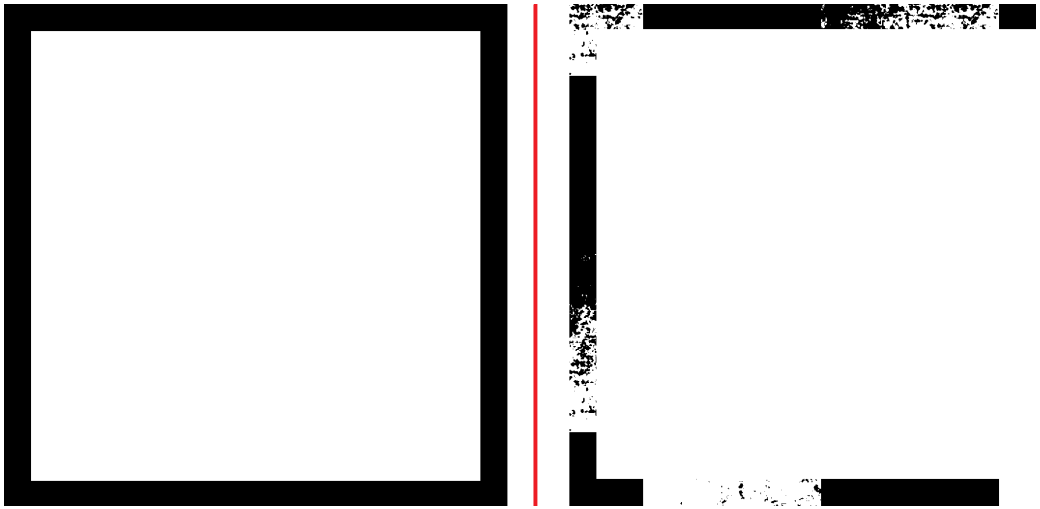

Figure 2: Output of **appropriate** (on left) and **inappropriate** (on right) thresholding.

8. `scan_image(warped_image)`

Parameters	Details
Purpose	This function detects **shape**, **color**, **centroid x** and **centroid y** from the warped image. In case of **two balls being detected**, it should return the said parameters **in the alphabetical order** of their **color** detected.

Parameters	Details
Input Arguments	`warped_image` : [***numpy array***] after applying the Perspective Transform **NOTE:** In **Task 1A Part 1**, the input to the function was `img_file_path` , but here it is image in **NumPy array** format. Hence make changes accordingly.

Return	**Dictionary** with **Shape as the key** and **Python list** of **Color**, **Centroid X** and **Centroid Y** as **the value**.
	Example Usage:
	In case of a **single ball being detected** it should return the **dictionary** as:
	`{'Circle': ['red', 100, 99]}`
	In case of **multiple balls being detected** it should return the **dictionary** in **alphabetical order** of ball **Color** as:
	`{'Circle': [['blue', 100, 99], ['red', 99, 100]]}`
	
	Name of **Shape** and **Color** are **case sensitive**.

Example Call	`shapes = task_1a_part1.scan_image(warped_img)`

Table 10: ***scan_image*** function to be edited by teams in **task_1a_part1.py**.

NOTE:

- In `task_1a_part1.py` , the **input to the scan_image** function was ***img_file_path***. However now it is changed to **warped_image**. Hence make changes accordingly.
- You are allowed to use helper functions in the allocated space of `task_2a.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 Catalina (10.15)***

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

1. Test using `task_2a.py`

- To test your solution,
  - **Activate** your **Conda environment**, **navigate to the directory** where you have downloaded all the files and run `python task_2a.py`
  - Open the provided scene file `task_2a_scene.ttt` in CoppeliaSim.
- If the code runs without any errors and you are **satisfied with the output** obtained, you can proceed ahead with the **next part**.

2. Test using `test_task_2a.exe` or `test_task_2a`

- This executable file will call the above mentioned **8 functions** from `task_2a.py`, `task_1a_part1.py` and `task_1b.py` regularly.
- Make sure that your **Conda environment is activated**.
- Open the provided scene file `task_2a_scene.ttt` in CoppeliaSim.
- When the `test_task_2a.exe` or `test_task_2a` 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.

  - **For Windows**:
    - Type the following command:
      - `test_task_2a.exe`
    - If there are no errors in `task_2a.py`, `task_1a_part1.py` and `task_1b.py`, you will see the output resemble Figure 3.

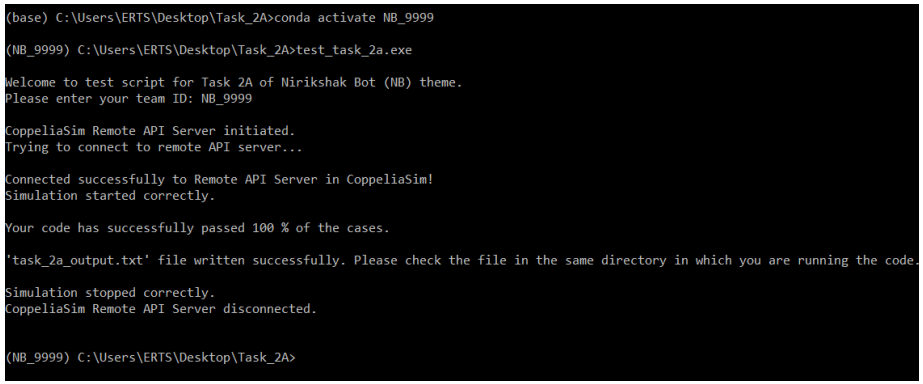

Figure 3: Output of **test_task_2a.exe** on Windows.

- **For Ubuntu:**
  - Type the following command:
    - `./test_task_2a`
  - If there are no errors in `task_2a.py`, `task_1a_part1.py` and `task_1b.py`, you will see the output resemble Figure 4.

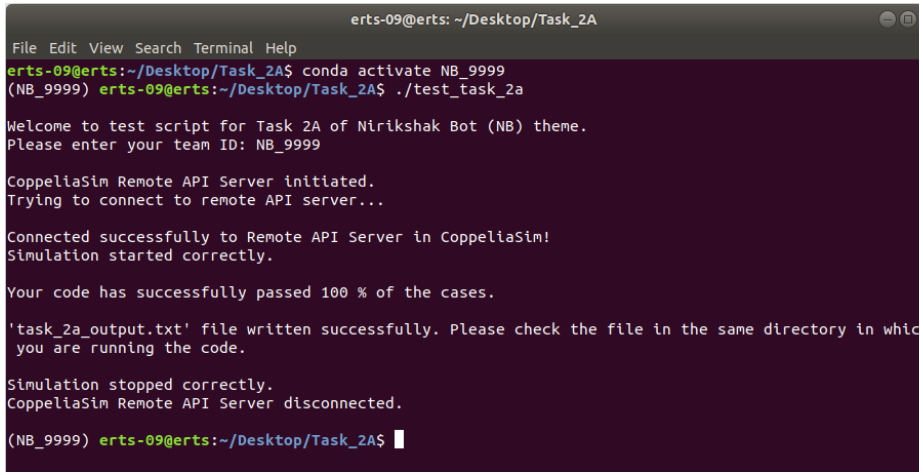

Figure 4: Output of **test_task_2a** on Ubuntu.

- **For Macintosh:**
  - Type the following command:
    - `./test_task_2a`

- If there are no errors in `task_2a.py`, `task_1a_part1.py` and `task_1b.py`, you will see the output resemble Figure 5.

Figure 5: Output of **test_task_2a** on Macintosh.

- During **execution** of the file **in Ubuntu**, you will find the output as shown in Figure 6. **Similar output** will be observed in **Windows** and **Macintosh OS**.

Figure 6: Execution of **test_task_2a** on Ubuntu.

- After execution of the file, `task_2a_output.txt` will be **created in the same directory** in which the `test_task_2a.py` was running.

> **NOTE:** During the time when `test_task_2a.exe` or `test_task_2a` is running, your CoppeliaSim's simulation or ball's movement may lag because of the the time it takes for `task_2a.py`, `task_1a_part1.py` and `task_1b.py` to return the **output**. **Do not** worry about the lag.

Submission Instructions

For **Task 2A 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_2A`.
    - For example: if your team ID is **9999** then you need to create a folder named `NB_9999_Task_2A`.
2. Now copy and paste following files into this folder:
    - `task_2a.py` (with **modified functions**)
    - `task_1a_part1.py` (with **modified scan_image** function)
    - `task_1b.py` (with **modified applyPerspectiveTransform** function)
    - `task_2a_output.txt` (generated after running `test_task_2a.exe` OR `test_task_2a`)
3. Compress this folder into a zip file and name it as `NB_9999_Task_2A.zip`.

4. Now go to the eYRC Portal and follow the instructions to upload this **.zip** file for **Task_2A** as shown in Figure 7.

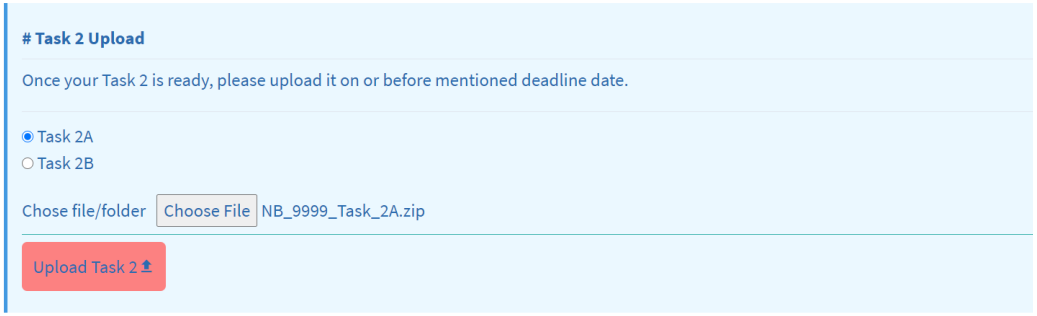

Figure 7: Submission of **NB_9999_Task_2A.zip** file on eYRC portal.

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

---

5. After you have **successfully submitted Task 2A and Task 2B files**, you can verify the zip file uploaded from the **'Verify Task 2 Upload'** section on the eYRC portal. The same is shown is Figure 8.

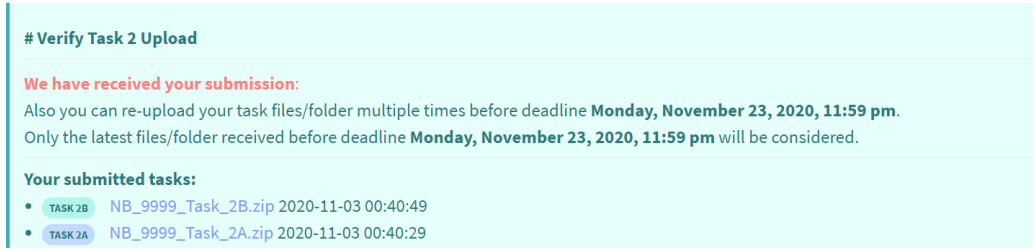

Figure 8: Successful submission of **Task 2A and Task 2B** on eYRC portal.

ALL THE BEST!!