Task 2

Task 2B

Problem Statement

In Task 2A, we determined the bot's position by analyzing feedback from our camera (UFO🛸:P) and localizing the bot within the arena. We used a force plugin to control wheel velocities (v1, v2, v3), like a real robot instead of directly controlling the velocities of the robot chassis (v_x, v_y, w_z).

Now all that is left to complete the simulation stage of this theme is to triple the number of bots in this task. Now, we need to refine our approach to accommodate three bots instead of one. We will be doing the same task as in 2A of go-to-goal(100 points times 3) But with **THREE BOTS** in this task.

Setup and Approach

Open the terminal and navigate to the `eyrc_hb` folder and run the below commands

```
cd hb_task_2_ws/src

git pull origin main

cd ..
```

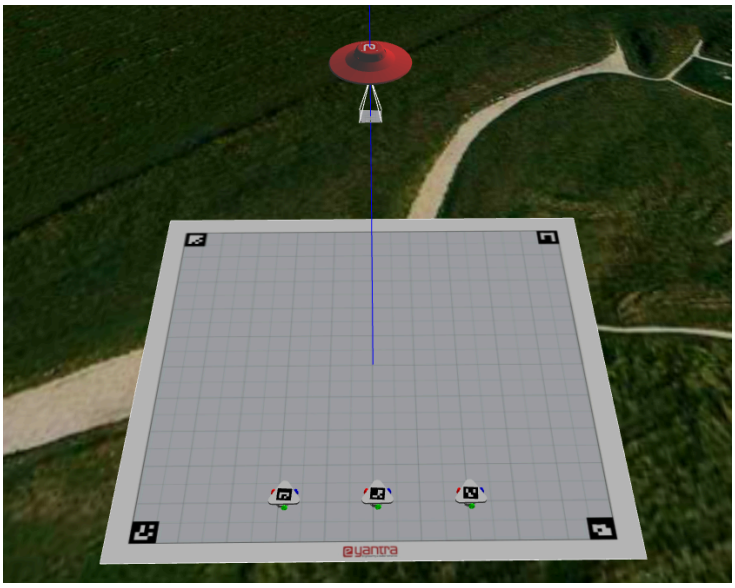
Note: Before pulling the repository, check once if the `hb_task2b` folder is present. If it is present, then skip the above step and build the workspace directly.

Then build and source your workspace using the below command

```
colcon build
source install/setup.bash
```

You can use the launch file provided to start the Gazebo environment with the hb_bot and the overhead camera (UFO 🛸:P) by using this command:

```
ros2 launch hb_task2b task2b.launch.py
```



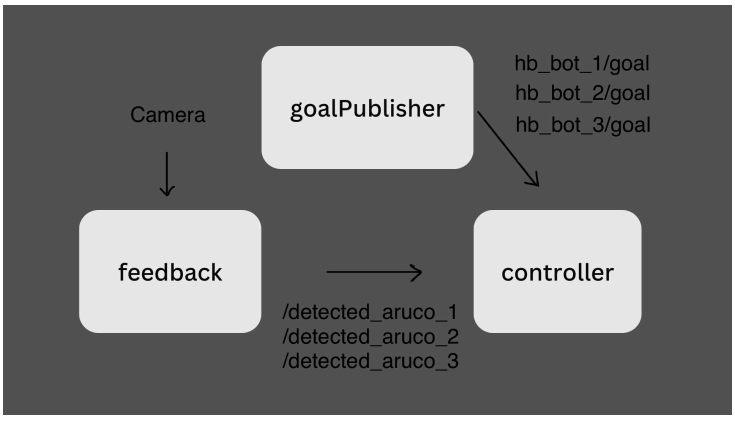
Once gazebo is launched, use the below command to run 2 nodes:

1. nextGoalPub.py
2. bot_controller.py (This file is just for reference, you can choose to have multiple or single node to control 3 bots) NOTE: feedback.py can also be added to this launch file or a new launch file (It's left to the teams to include it or not)

```
ros2 launch hb_task2b controller.launch.py
```

Watch the following video to understand the Task2b setup.

Now let's look at the different nodes in task 2B



To understand them better let's look at them one by one

Components Involved:

- **goalPublisher** (This will play the role that was of service_node):
    - This is of course already provided and need not be touched.
    - This node publishes goals for the bots.
    - It publishes goal poses on the topics hb_bot_1/goal, hb_bot_2/goal, and hb_bot_3/goal.
    - Each time it is launched, it publishes different dimensions and shapes for each bot.
    - Here we are using custom msg to publish the desired goal poses for the bots with their ID
    - The structure of the message :
        - int64 bot_id
        - float64[] x
        - float64[] y
        - float64 theta
    - **float64[] - array of 64-bit floating-point numbers**
    - In this approach, the array of poses for bots is published all at once. This strategy significantly reduces any delays that were previously encountered due to the service approach we used to obtain goal poses. On the controller side, the topics containing bot poses can be subscribed to. Sample subscriber for bot 1 is provided in the controller.py
    - **NOTE: It's crucial to note that with each launch of the controller, the dimensions of the shapes may change. Consequently, the poses generated for the bots will vary with every run.**
- **Feedback:**
    - Before we get to controlling the three robots, feedback.py from 2a will clearly need to be modified to estimate and publish the pose of all three robots.
    - create a `feedback2b.py` . You can use feedback.py from task2a to get a head start and extend it to detect **Aruco ID 1, 2, and 3**.
    - So now we shall have three topics `/detected_aruco_1` `/detected_aruco_2` `/detected_aruco_3` carrying the poses of the three robots published by `feedback2b.py`
- **Controller:**
    - This is the part where you need to work on.
    - There is freedom to implement the controller for the bots in any way you wish.
    - It can be approached by
        - having 3 separate controller nodes, where each is controlling a single bot. For ex: create `controller2b_bot1.py` for bot 1.
        - Alternatively, a single controller node can be used to manage all 3 bots. For ex: create `controller2b.py` that controls all three bots. There are more advanced techniques for creating controllers, consider multi-threading for efficient control.
    - Note: If you choose to have three different controller nodes, you'll also need to modify the launch file accordingly.
    - The controller will have to subscribe to the goals list topic `hb_bot_1/goal` . A boilerplate for doing this is provided in the `controller2b_bot1.py` file. extending this to Bot 2 and 3 is left to the participants.

Additional Resources for your curiosity:

- Thread-based parallelism
- An Intro to Threading in Python