Task 2A

Problem Statement

(as mentioned in motivation page) **Task 1B, in short:** *Implement a simple controller to make the simulated robot (simplified) go to a series of desired pose given localisation in simulator.*

Task 2A, in short:

Repeat Task 1B! But...

- **with more realistic localisation.**
- **with more realistic model of controlling the holonomic drive.**

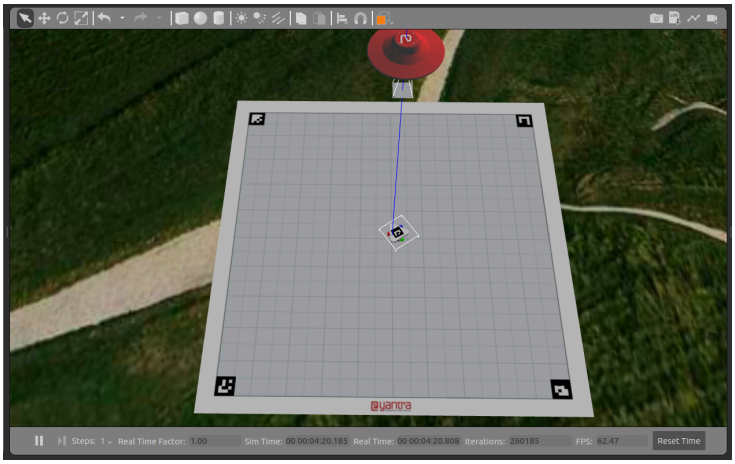
What exactly do we mean by "more realistic"?

We have made some changes to the model (that we had been working with in task 1B) and the world to make interfacing with it more realistic.

Specifically,

- we have modified the **URDF** and used a different plugin to control wheel velocities **(v1, v2, v3)** (well... not exactly... but more elaboration on this later) like a real robot instead of directly controlling velocities of the robot chassis **(v_x, v_y, w_z)**.
- And we have also added an **Aruco marker** and a camera to do localisation like we would in **Stage 2**.

In task 2B: we will be multiplying the number of bots.



The Setup

First step is to source the `setup.bash` file of gazebo in your `bashrc` file.

```
source /usr/share/gazebo-11/setup.bash
```

Now, open the terminal and navigate to the `eyrc_hb` folder and run the below commands

```
mkdir -p hb_task_2_ws/src
cd hb_task_2_ws/src
git clone https://github.com/eYantra-Robotics-Competition/eYRC-2023_Hologlyph_Bots.git

cd ..
```

Then build and source your workspace using 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_task2a task2a.launch.py
```

Note: Before launching the gazebo environment you have to source your workspace in bashrc file.

In this task we'll need to "Localise" the robot using the stream of images from a virtual overhead camera in Gazebo.

When it comes to Image Processing its quite an obvious choice to use **OpenCV**!

To install library

Install "pip3" (which we have already installed in task 0 before, so no need to do it again)

```
sudo apt install python3-pip
```

Install "OpenCV2 Python and Numpy"

```
pip3 install opencv-contrib-python==4.7.0.72
pip3 install numpy==1.21.5
```

Note: We suggest that you should use above mentioned versions for opencv-contrib-python and numpy as they are tested to be stable for this theme, but you may choose other version only if you are confident to do so. Doing so, will not guarantee support from e-Yantra team.

Localisation with OpenCV, Aruco Markers

Write your code in the **feedback.py** file.

- **Subscribes** for Image of the virtual camera from `/camera/image_raw` topic.
- **Publishes** (x,y,theta) on topic `/detected_aruco` which is of the msg type Pose2D from geometry_msgs.msg
  (this will replace `/odom` from Task 1A)

The Localisation problem boils down to find (x,y,theta) i.e. pose of the robot from an image with ArUco marker in it.

Which has the following steps:

1. Convertion of ROS Image to OpenCV image
   For converting the ROS Image to OpenCV image you have to use following commands (You can

explore more on offical CvBridge package repository) :

```
cv_image = self.cv_bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
```

2. Find ArUco!
   OpenCV makes this super easy, we just use `cv2.aruco.detectMarkers` to find all the ArUco Markers!
   This will return `(corners, ids, rejected)`

3. Take the corners of the quadralateral (i.e. list of (x,y) of the four corners) of id of interest... and... extract (x,y,theta) from it 😛
   Well that is just high school math/geometry, some logical thinking and some basic programming, so we leave it for you to figure out. (There can be more than one way to do it.)

Also here its important to take care of sign and other conventions! What is zero, what is positive, what is negative for all the variables involved above?

This ofcourse also effects the controller.py sign conventions.

There are way too many resources explaining step 1, online and therefore shall not be repeated here.

Note:

- The goals provided are in reference to the camera frame with dimensions 500 x 500 pixels.
- 0,0 with reference to the camera is in the top left side, The goal provided uses 0,0 in the center, perform the required transformation, and then use the goal poses.

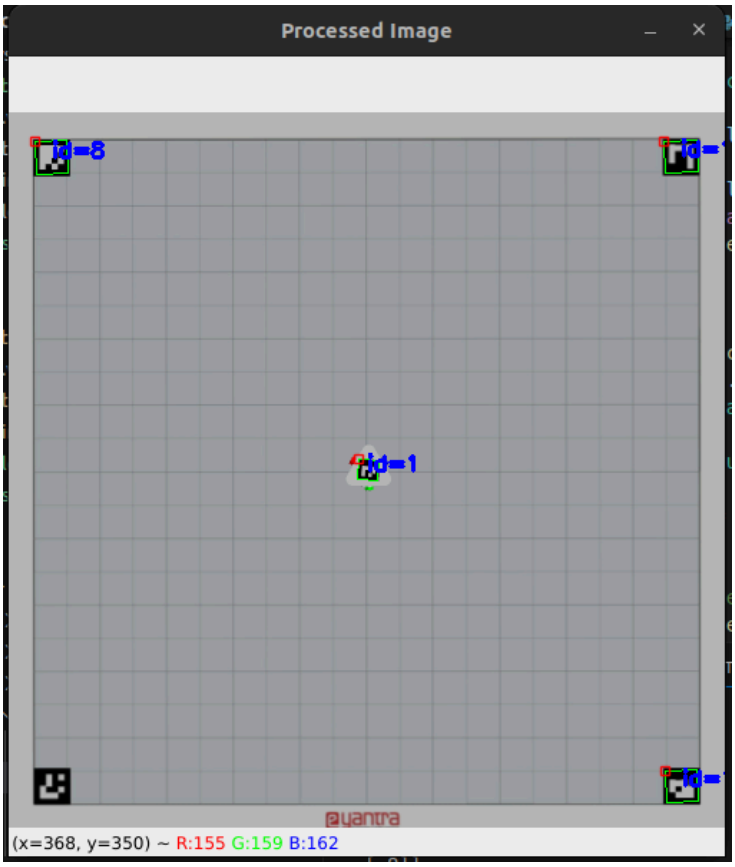


Figure: A window that shows the overhead camera feed. (here AruCo detection is visualised as well)

Inverse Kinematics

- Write your code in the **controller.py** file.
- **subscribes** to the `/detected_aruco` topic (published by feedback described above)
- **publishes** the three wheel velocities in the three publishers (already declared in the boiler plate code given).
  i.e. on topics `/hb_bot_1/left_wheel_force`, `/hb_bot_1/right_wheel_force`, `/hb_bot_1/rear_wheel_force`
  Teams are ofcourse NOT allowed to publish to `/hb_bot_1/cmd_vel` (chassis velocities) that was used in Task 1B

So to recap from Task 1B:

In controller we:

- calculate error in x,y,theta
- use P controller to calculate (v_x, v_y, w_z)

Now, the Inverse Kinematics boils down to find three omni-wheel velocities (v_1, v_2, v_3) given velocity of the chassis (v_x, v_y, w_z)

- this is done by a matrix multiplication (Linear Algebra Concept: Transformation of Vectors)
- so the question is what is the 3x3 matrix that will convert (v_x, v_y, w_z) to (v_1, v_2, v_3)?
- and publish it in ----------.force.y variable of the three wheel-forces topics given.

We will leave the exercise of finding this matrix as a topic for you to research on for now.

This will be explained further in the live session that will be announced in the discussion forum.

An important note on the topic provided: The three topic for three wheel respectively are of geometry_msgs/Wrench message type. Check link Out of the six variable in each, we will use the force.y variable to mimic the force applied by the motor on the ground. Not we said we are controlling wheel velocity earlier but are saying force here. We are not bothering to think about the relationship between force and wheel velocity. So we shall adjust for this using the kP used in P controller.

This is frankly not a good idea to ignore this topic of relation between force, torque, velocity, angular velocity etc... but is good enough for now. Because eventually on hardware we shall be using a continuous servo that will have another control system to achieve the desired (v_1, v_2, v_3) directly!

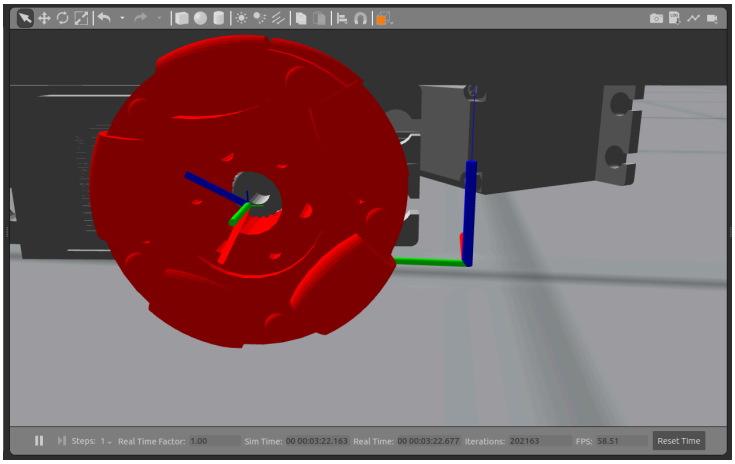


Figure: See the tiny little green axis hiding inside the wheel? That is the y axis of the dummy link on which the "wrench" force will act.

Therefore the main job in this task is to complete the two python files: **feedback.py** & **controller.py**

Note: the goal positions are provided through a service just as in Task 1B. but the length of the list will be much bigger than last time.

Note: You'll need to explore OpenCV to figure out how to detect Aruco marker, and you'll need to explore some simple papers or other resources to learn about how to get wheel velocities (v1, v2, v3) from chassis velocities (v_x, v_y, w). We shall make an announcement on discuss forum regarding further hints and live sessions to help you with the same, but until then happy exploring and we hope you find the answers before we give the hints!

If everything mentioned above is done then **congratulations on completing Task 2A**!

Only one sub-task left to complete Stage 1 of Hologlyph Theme eYRC2023-24!
