Task 1B

Note: Deadline of Task 1a and Task 1b is 10th October 2023

The objective of this task is:

- to explore and understand **Gazebo-ROS, URDF**,
- to understand **manual control** of **"high-level model"** of a generic holonomic robot (using Planar Move Plugin),
- and implement some simple controller on a holonomic drive robot (for ex: 3 P controllers).

Some Motivation

Before we get to controlling a team of three holonomic robots, a prior major milestone will be to control one holonomic drive robot to do the desired task. This will require us to understand and implement the following essential building blocks:

1. Designing position control of Holonomic Drive Robot (ground vehicle). (this will be done on a "high-level" model of the robot as mentioned above in Task 1B objective).
2. Understanding forward and inverse kinematics of three-omni-wheeled bot. (the holonomic drive robot that we shall use in this theme)
   - We'll only need to implement inverse kinematics, for the purpose of taking the output of the above mentioned controller and controlling the three wheel velocities to achieve the desired (v_x, v_y, w)
3. Localisation of the robot. In task 1b we'll build the position control with localisation given (for free) by the simulator. But in the real world (outside simulator) we'll need to worry about how to localise the robot (i.e. answering the question: where is the robot?), using some sensors (in our case it will be an overhead camera).
4. At an even higher level than the controller (discussed above) is the question of where to even go? We could call this "Planning".
5. If we wish to draw smooth shapes simple position control may not suffice. Therefore we may need to explore more fancy/advanced ideas in control.
6. All of the above can be implemented directly in hardware, but it is a good idea to simulate and find problems without actually breaking some hardware :stuck_out_tongue:. Here comes Gazebo Simulator and ROS.

In task 1b we shall explore the first and last (6th) points listed above. In task 2a we shall look at the 2nd and 3rd points. In tasks after 2a we shall look at point 4, 5 and also all the points (not mentioned here) related to multi-robot systems

Problem Statement

Implement a simple controller to make the **simulated robot** (simplified) go to a series of desired pose given ideal localisation.

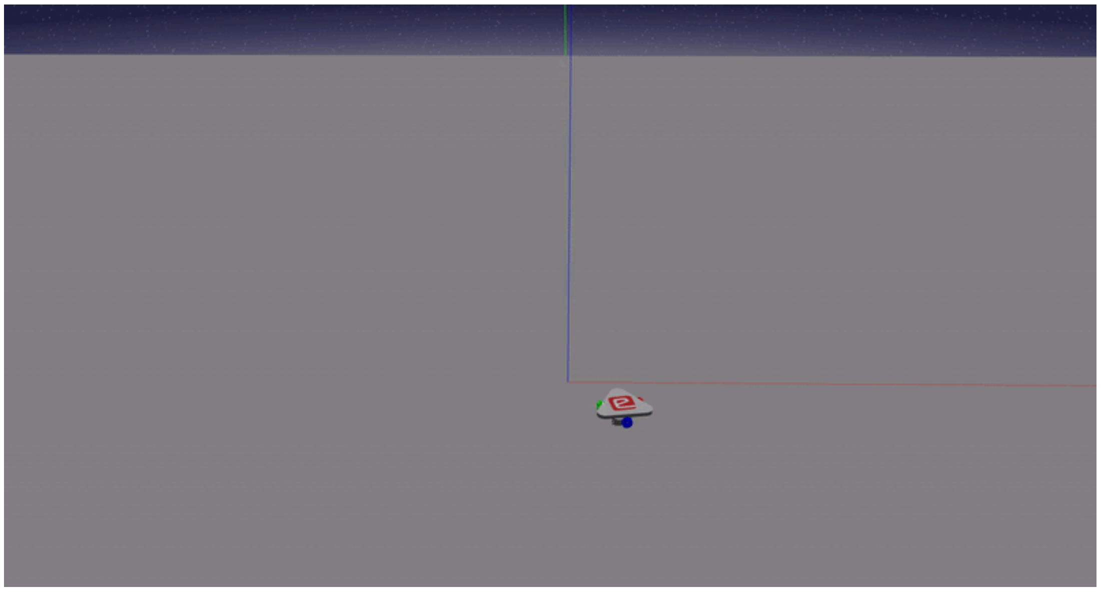
There are quite a few things that are happening in the highly condense statement above. (So let's read between the lines :😛

- **simulated** robot (**simplified**)
  - first we shall manual actuate the simulated robot
  - (simplified): by using planar move gazebo model plugin.
    Meaning we shall directly control the **chassis velocities** (v_x, v_y, w) - linear velocities v_x, v_y and angular velicity w, instead of controlling wheel velocities (v1, v2, v3). How these two are related (i.e. kinematics) will be explained in task 2.
- go-to-pose controller - given ideal localisation
- extend the above code to handle a series of desired poses

The gif below is a demo/example of what one should be able to do at the end of Task 1. *Note that there are many shapes that the bot can make, so don't worry if your output isn't exactly same as the example provided. (more about it later...)*

An important insightful **NOTE** (again)
We are using ideal localisation using transforms in gazebo and direct control of the chassis velocities (we have not studying the kinematics yet).

Basically there are a lot "cheats" available to us :stuck_out_tongue: because we are working in a simulator. To do the same task in real life we will have to add quite a few more layers, which we shall in future tasks.



Expected output

Approach

So how shall we tackle Task 1B?

Let's look at the check list above and work on it one by one.

Step 1: Gazebo and URDF

- simulated robot
  - manual actuation of the simulated robot
  - also get ideal localisation

Step 2: controller.py

- **Step 2A:** go-to-goal-pose controller
  - given ideal localisation and simplified actuation from step 1.
- **Step 2B:** go-to sequence of goal-poses
  - Just extend (add some logic) to the above code to handle a series of desired poses.