ECE 452 Robotics: Algorithm & Control

Project 2

Due Date: April 16 2021

As mentioned in Project 1, you will now be working on a mobile robot called Turtlebot3 in a simulated environment. Throughout the project you will get familiar with Robot Operating Systems (ROS) (see Section 6.1) and learn how to control and operate the robot under this system. You will also have a look on how the robot is visualized using Gazebo (see Section 6.3) which is an advanced virtual robot simulator that will allow us to replace a physical robot with its realistic model. Using these tools, your given two main tasks where you have to implement a python program to move the robot to certain multiple goal points, and another program which the robot encounters a wall and tries to follow around it.

1 Introduction

1.1 Robot Model

Choose the body frame T so that its origin is at the midpoint between the two wheels on the common axis of the two wheels. The x-axis should point in the direction of the robot motion while the y-axis is perpendicular to the x-axis, consistent with the axes of the wheels. The following figure shows the spatial frame S and the body frame T.

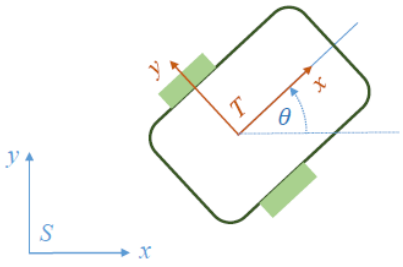


Figure 1: Robot model

Note that the robot "lives" in $SE(2)$ space (on a plane), not $SE(3)$. That is, using x, y and θ one can describe the robot's configuration. However, you

can utilize the standard SE(3) representation and set:

$$z = 0 \quad \omega = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

1.2 Navigation Method 1 (First Task)

We are going to apply the dead-reckoning navigation for the robot to reach a set of goal points. So first, we have to find the distance (both in x and y direction) between the robot's current location and our goal point. Now we can determine the angle that the robot should rotate before it moves forward and this has to match with θ shown in Figure 1. Afterwards, the robot will start driving and when it arrives at the goal point, it should follow the same steps to go to the next destination or stop if it was the last one.

Algorithm 1: Go to point

```
Initialization: Set start and goal point, read data from robot sensors;
while robot is operating do
    find x, y distance from current position to goal;
    find angle to the goal point wrt global frame;
    find the angle difference;
    calculate the Euclidean distance;
    if angles doesn't match then
        apply angular velocity;
    else if didn't reach goal then
        apply linear velocity;
    else
        stop;
    end
end
```

The algorithm above is a pseudo code that runs for a single goal point. Remember that this is not for multiple goal points, so you have to figure out how to use this algorithm recursively. Remember when you reach a certain goal point, you have to set the current location as the new starting point.

1.3 Navigation Method 2 (Second Task)

Now we will try to control the robot to circulate around the walls infinitely using the **LaserScan** sensor information. In this task, there are three main states: 'go to point', 'wall detected', and 'follow the wall'. The initial state is the same one described in the previous section. However, you need an additional condition for the robot when it encounters a wall during the process and whenever it does, the robot should move on to the next state.

In the 'wall detected' state, you have to design whether the robot should rotate to the left or right until there's some distance between the wall and the

robot's front so that it can move forward. Remember to move on to the next state when the conditions above are satisfied.

During the final state, you will keep reading the **LaserScan** sensor values and determine the angular and linear velocities for each possible cases. For instance, when the robot was moving along the wall on its left and suddenly there was nothing detected in the left sensor, the robot should realize that it has to stop and start to rotate. Moreover, the robot has to remain on this state and permanently follow the wall until the program is terminated.

Algorithm 2: Follow Wall

```
Initialization: Set initial state, read data from robot sensors;
while robot is operating do
    if state='go to point' then
        Run Go_to_point;
        if robot encounters a wall then
            state='wall detected';
    else if state='wall detected' then
        Rotate to the left or right until the robot can move forward.
        Afterwards, state='follow the wall'
    else if state='follow the wall' then
        If-else statements deciding the velocities applied to the robot for
        each circumstances
end
```

Here is a brief algorithm summarizing the overall process and below is a figure showing the **LaserScan** sensor regions in the robot.

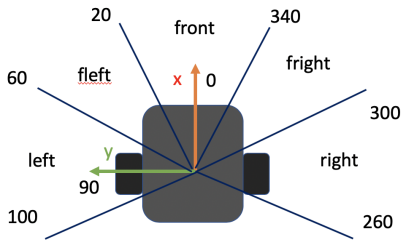


Figure 2: Robot Diagram

2 Software Setup

2.1 For Task 1

Here are some steps to create a new ROS package and set the Turtlebot model permanently.

- Create catkin package **project_2** where you will keep the codes for this project. Type the following commands in the terminal:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg project_2 rospy
$ cd ~/catkin_ws && catkin_make
$ mkdir -p ~/catkin_ws/src/project_2/scripts
```

  Python codes must be saved in the **scripts** directory.

- Following commands are used in the terminal to export the turtlebot model permanently. First, let's open **bashrc** file:

```
$ sudo gedit ~/.bashrc
```

  Then paste this line at the end of the file:

```
export TURTLEBOT3_MODEL=burger
```

  Save and close the file, and source **bashrc**:

```
$ source ~/.bashrc
```

2.2 For Task 2

- As usual, download the Python code and save under the **scripts** directory.
- Download the **wall_following.zip** file. It should be on the **Downloads** directory, and follow these commands:

```
$ cd ~/Downloads
$ sudo apt-get install unzip
$ unzip wall_following.zip
$ cd wall_following
$ ./setup.sh
```

3 Run the programs

3.1 Task 1

Complete writing the ROS program (python) that drives the Turtlebot from the initial configuration to specified points (the points are given in the python file). Download the **go_to_point.py** and save it under the **scripts** directory. You should run the robot in the empty world first:

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

Open a new terminal and let's run the python program:

```
$ rosrun project_2 go_to_point.py
```

Now the robot should move to the desired destinations. In case if the python code is not identified, run the following command:

```
$ sudo chmod +x go_to_point.py
```

Please read the comments in the python file carefully to easily accomplish the project.

3.2 Task 2

Similar to the previous section, download the **follow_wall.py** and save it under the **scripts** directory. After you finish writing the code, let's start running the empty world first:

```
$ roslaunch turtlebot3_gazebo turtlebot3_wall_following.launch
```

Open a new terminal and let's run the python program:

```
$ rosrun project_2 follow_wall.py
```

In case if the python code is not identified, run the following command:

```
$ sudo chmod +x follow_wall.py
```

Please read the comments in the python file carefully to easily accomplish the project.

4 Report

- Record screen videos where the robot performs each tasks. Save the video file and upload on Blackboard as in Project 1.
- Please submit a report summarizing the difficulties you have faced and your well-commented code in the end.
- The filename of the report and the video should be **ECE452_Spring2021_Proj02_netid** like in Project 1. You can add numbers in the end in case you have multiple videos.
- Upload **go_to_point.py** on Blackboard. You can also upload additional files if necessary.
- Please check whether the video plays properly after submission and files with wrong names will not be graded.

5 On Plagiarism

It is important that you thoroughly understand how to work with the robot. Copying the code from another student or other source is therefore not allowed. It will be considered plagiarism, resulting in a failing grade and further possible disciplinary action.

6 Appendix

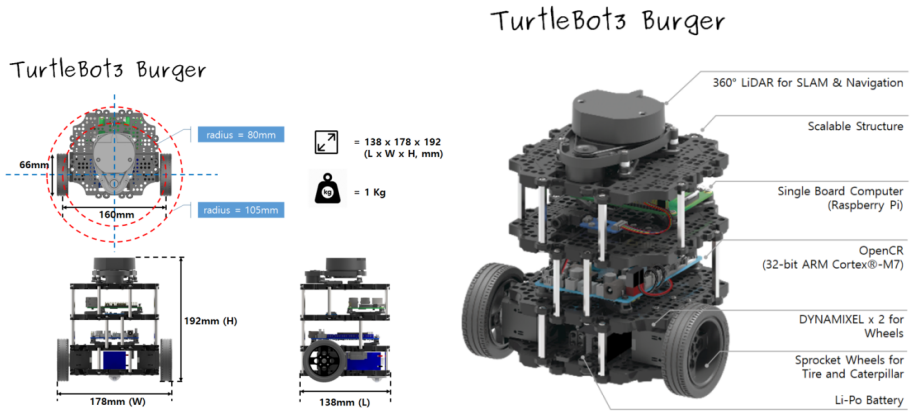
6.1 ROS

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware ab-

straction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. Visit http://wiki.ros.org for more information. The basic concepts of ROS are described at http://wiki.ros.org/ROS/Concepts. And you can find a more in-depth discussion of ROS in this freely available book.

6.2 TurtleBot3

TurtleBot is a mobile robot platform that was developed to exploit what ROS has to offer. TurtleBot3 is the latest generation of TurtleBot robots released in 2017. It is made up of a modular chassis, available in two sizes: small (Burger) and medium (Waffle). It is driven by Dynamixel motors from ROBOTIS, and equipped with laser scan sensors that provide the information about the environment in 2D. Turtlebot3 can be customized and can run multiple tasks such as Navigation, SLAM (Simultaneous Localization and Mapping) and Manipulation. Figure 2 depicts the physical characteristics of the Turtlebot3 Burger. It has maximum linear velocity of 0.22 m/s and angular velocity of 2.84 rad/s (162.72 deg/s). Also, it is equipped with a LaserScan sensor for obstacle detection in 2D. For detailed specifications see http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/#hardware-specifications.



(a) Dimensions (b) Components

Figure 3: TurtleBot3 Specifications

6.3 Gazebo

Gazebo is a simulation tool that provides the ability to create a realistic simulated environment for a robot, create physically exact and realistic simulation

of the robot as it interacts with the environment, and finally provide a simulation of sensor readings in response to the robot moving in the environment for many common robot sensors. In this project, you will be working with a model of the TurtleBot3 Burger in Gazebo simulated environment. Gazebo is fully integrated with ROS. This means that you can control the simulated robot in Gazebo in the same way you would control the physical robot.