ECE 452 Robotics: Algorithm & Control

Project 3

Due Date: Apr 30 2021

In the previous project, you have learned how to control Turtlebot to move to a certain point. Furthermore, you have also studied how to rotate the robot around an object when the robot encounters an obstacle in the ROS environment, and display the results on the Gazebo simulator. Throughout this project, you will now apply the Bug algorithms with the implementations from Project 2.

1 Background Knowledge

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.

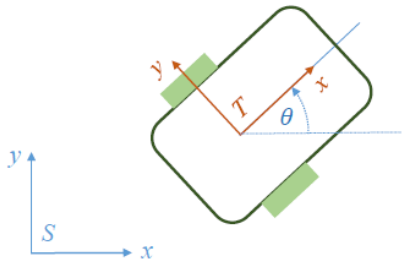

Figure 1: Robot model

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

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

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

According to this environment, the robot sensors will provide the current location of the robot **(odom.x, odom.y)**, and **odom.theta** which is the θ in Figure 1 and it is angle between the two frames S and T. Moreover, the global reference frame S is shown in Gazebo with RGB axis and each block's length is 1.

Now here is a figure showing the **LaserScan** sensor configuration of the robot. The numbers represent the angle from the x axis of the robot (the orange one) in degrees (not in radians!). For instance, if you want to bring the scanned distance from the left side of the robot, you need to bring **scan.region['left']**. For other directions, you just have to change the name of the direction inside the brackets.

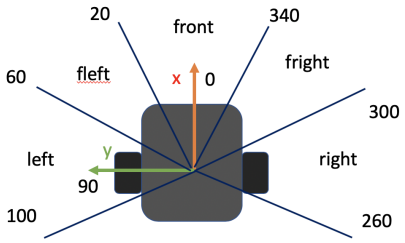

Figure 2: Robot Diagram

1.2 The Bug Algorithm

Let's assume that a robot is navigating a planar environment. The robot starts at an arbitrary configuration and its objective is to reach a designated goal point (the position of the goal point is given). However, the robot needs to avoid some obstacles where their positions and shapes are unknown. Therefore, the Bug algorithm is based on this idea and it provides the robot how to autonomously navigate through an unknown environment and to reach the predefined destination.

In this project, we will implement two Bug algorithms, the *Bug 1* and *Bug 2* algorithm, and here we provide modified algorithms with more similar structure to the codes you will be working on. If you want to look for more details about Bug algorithms, see Chapter 1 in Lectures on Robotic Planning and Kinematics by F. Bullo and S. L. Smith. Moreover, you can also check the slides by H.

Choset, the lead author of the *Principles of Robot Motion* textbook.

Algorithm 1: Bug 1 Algorithm

```
Initialization: Set initial state, read data from robot sensors;
while robot is operating do
    if state='go to goal' then
        Run Go_to_point (from Project 2);
        if reach goal point then
            Exit the program with "success";
        if robot encounters an object then
            if if the robot hit the object more than once then
                Exit the program with "failure"
            else
                record the current location as the hit point;
                state='circumnavigate obstacle';
            end
    else if state='circumnavigate obstacle' then
        Run Follow_wall (from Project 2);
        But you also need to keep updating the closest point from the
         obstacle boundary to the goal;
        if reach the hit point again then
            state='go back to closest point';
    else if state='go back to closest point' then
        Run Follow_wall (from Project 2);
        if reach the closest point then
            state='go back to goal';
end
```

Algorithm 2: Bug 2 Algorithm

```
Initialization: Set initial state, read data from robot sensors;
α=angle from the initial point to the goal point;
(α is saved as m-line in the code);
while robot is operating do
    β=angle from current location to the goal point;
    if state='go to goal' then
        Run Go_to_point (from Project 2);
        if reach goal point then
            Exit the program with "success";
        if robot encounters an object then
            record the current location as the hit point;
            state='circumnavigate obstacle';
    else if state='circumnavigate obstacle' then
        Run Follow_wall (from Project 2);
        Find the angle difference β − α;
        if the robot can move to β − α direction and it is closer to the
         goal compared to the hit point then
            state='go to goal';
        if the robot reaches the hit point again then
            Exit the program with "failure";
end
```

2 Environment Setup

Similar to the previous project, we need to perform the following.

- 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_3 rospy
$ cd ~/catkin_ws && catkin_make
$ mkdir -p ~/catkin_ws/src/project_3/scripts
```

  As usual, download the python codes and save them in the **scripts** directory.

- Download the **maze_world.zip** file. It should be on the **Downloads** directory, and use these commands:

```
$ cd ~/Downloads
$ unzip maze_world.zip
$ cd maze_world
$ ./setup.sh
```

3 The Task

In this project, you need to test the Bug algorithms for two different goal points. The initial position is at the origin of the maze. However, you have to set two different goal points, a point inside the maze and a point outside the maze, and check whether your program works for these different cases. Therefore, there should be 4 test sets: Bug 1 - goal inside the maze, Bug 1 - goal outside the maze, Bug 2 - goal inside the maze, Bug 2 - goal outside the maze.

After completing on writing the python files, open a terminal and follow these instructions to run the programs.

```
$ roslaunch turtlebot3_gazebo turtlebot3_maze.launch
```

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

```
$ rosrun project_3 bug_1.py (or bug_2.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 bug_1.py (or bug_2.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 like in the previous projects.

- In the report, it should contain the following
  - Explain the difficulties you have faced.
  - Compare Bug 1 and Bug 2 algorithm, and comment in terms of efficiency, completeness, etc.
  - Attach your well-commented code.
- The filename of the report and the video should be **ECE452_Spring2021_Proj03_netid** like in Project 1. You can add numbers in the end in case you have multiple videos.
- Upload **bug_1.py** and **bug_2.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.