COMP313P Lab 02: Simple Robot Control

January 16, 2018

1 Introduction

In this lab exercise, we will introduce you to the Simple Two Dimensional Robot (STDR) Simulator and ask you to write a simple controller to drive a robot from its current state $\mathbf{x}_I$ to a goal state $\mathbf{X}_G$. We will continue to use STDR through this module. Furthermore, the code you develop here is one that you can use in the courseworks later.

2 STDR

2.1 Description

To quote from the website, STDR

> ...is not to be the most realistic simulator, or the one with the most functionalities. Our intention is to make a single robot's, or a swarm's simulation as simple as possible, by minimizing the needed actions the researcher has to perform to start his/hers experiment. In addition, STDR can function with or without a graphical environment, which allows for experiments to take place even using ssh connections.

Despite its simplicity, it supports a wide range of capabilities, including simulating sensors. Figure 2.1, for example, shows, an output from the simulator. It shows a map, consisting of black walls, together with the location of two robots (blue circles). Each robot is equipped with a set of sensors which produce detection fans. Laser scan data, which has high angular resolution, is shown in red. Sonar, on the other hand has long angular resolution and exibits a phenomena called "range constant depth" and is shown in green.

2.2 Installing STDR

If you are using the Docker image, stdr is already included and you do not have to do anything.

If you are using your own installation (Linux / native macOS), see the installation instructions from the website.

```
sudo apt-get install ros-kinetic-stdr-simulator
```

If you are using Luna or running on macOS, you will need to build from source. I have only tried with macOS, and there are a few issues with a dependent package of a dependent package not currently being available in homebrew. Talk to me (Simon) for details!

2.3 Familiarisation Tasking

You should try some of the STDR Tutorials. In particular:

- Running STDR Simulator to see how to run the simulator and to run a basic example.
- The tutorials here and here on how the GUI works.

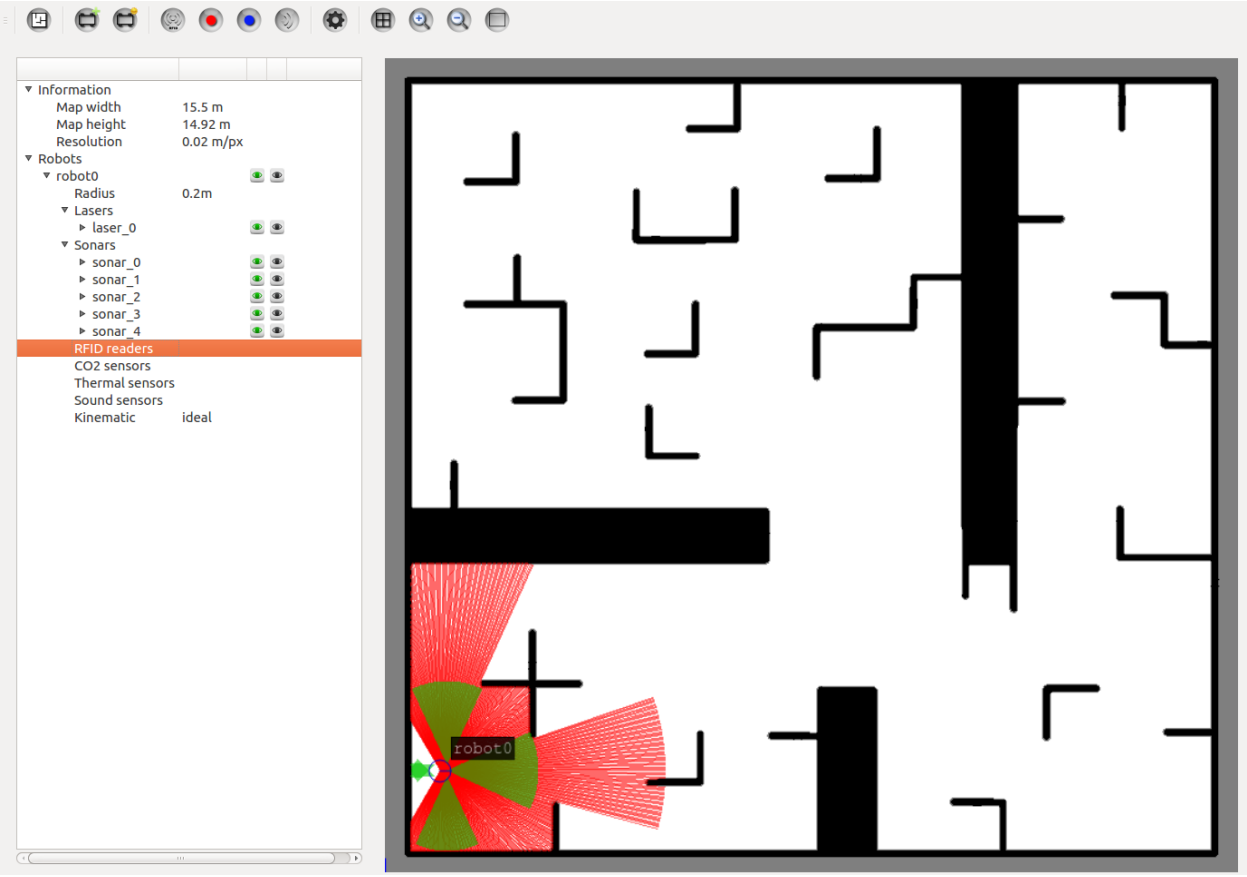
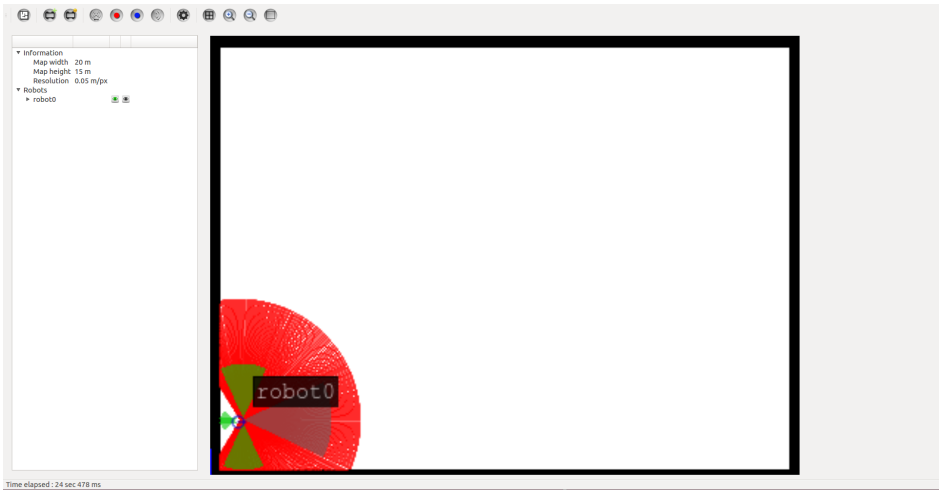


Figure 1: STDR Simulator GUI using the default example configuration. The single robot is in the bottom left hand corner. The red and green fans are caused by laser and sonar data.



3 Install ROS Nodes for the Lab Exercise

The code for the lab exercise is available on github. Go to your catkin workspace (if you don't have one, create it using the instructions here). Run the commands:

```
cd <your_catkin_ws>/src
git clone https://github.com/grdwyer/comp313p.git
cd ..
catkin_make
source ./devel/setup.bash
```

To start up an example version of stdr, run

```
roslaunch comp313p_launch lab_position_control.launch
```

This should open the GUI shown in Figure **??**. The scenario here is a simple one in which there is a single robot in an empty room. You will also be presented on a command line with:

```
[INFO] [1516099758.766810]: Current position, x: 1.0, y:2.0, theta:0.0
Enter desired linear velocity:
```

The robot itself is a turtlebot. Its state is given by

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}. \tag{1}$$

The robot action, $\mathbf{u}$ is the linear speed v and angular velocity of the robot ω which can be applied for a time step ΔT. Given this, the new robot state is given by:

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\Delta T \cos\theta \\ y + v\Delta T \sin\theta \\ \theta + \Delta T\omega \end{bmatrix}. \tag{2}$$

(Note this is an approximation — the nasty-looking equations on the slides are the precise solution.)

You can now enter values for linear speed, angular velocity (in degrees per second) and the time for which these values will be held. Internally, stdr breaks the time you specify into smaller time steps, and moves the robot position fowards one step each time. Once the robot has driven for the specified time, it will stop and tell you its position again. For example:

```
[INFO] [1516100752.083056]: Current position, x: 1.0, y:2.0, theta:0.0
Enter desired linear velocity: 1
Enter desired rotational velocity: 0
Enter desired duration: 5
[INFO] [1516100765.095951]: Current position, x: 6.000497378, y:2.0, theta:0.0
Enter desired linear velocity: 0
Enter desired rotational velocity: 10
Enter desired duration: 9
[INFO] [1516100785.967360]: Current position, x: 6.000497378, y:2.0, theta:90.00789596
Enter desired linear velocity: 1
Enter desired rotational velocity: 10
Enter desired duration: 9
[INFO] [1516100805.893849]: Current position, x: 0.570088542527, y:7.72097234551, theta:177.00694759
```

Note that, as the robot moves, its sensor detection pattern will change. Also, when the robot hits the wall. This explains why, in the final manoeuvre, the robot does not have a theta value of 180 degress. When the robot is against a wall, it can rotate and it can drive away from the wall.

4 Labwork Assignment

4.1 Overview

Write a ROS node, in Python, which will drive the stdr simulated through a set of waypoints and sends a messages to confirm each time the robot reaches a waypoint and sends a final message when it has reached the end.

For each set of waypoints, you should consider this as writing a controller which moves the robot from an initial state $\mathbf{x}_I$ to a final state $\mathbf{x}_F$ which lies in the goal set $\mathbf{X}_G$.

The intial state is the current position and orientation of the robot. The goal state is defined as follows. We define a nominal goal $\mathbf{x}_G = [x_G, y_G, \theta_G]$ is the desired position and orientation of the robot. The goal set is defined as

$$\mathbf{X}_G = \left\{ \mathbf{x} : \sqrt{(x - x_G)^2 + (y - y_G)^2} \leq d \wedge |\theta - \theta_G| \leq t \right\}. \tag{3}$$

In other words, the robot needs to get within a distance d and an angle t of the goal. For this coursework $d = 0.1$ and $t = 5$deg.

4.2 Waypoints

The waypoints will be specified in a file of the form:

```
WAYPOINT1_X WAYPOINT1_Y WAYPOINT1_THETA
WAYPOINT2_X WAYPOINT2_Y WAYPOINT2_THETA
```

For the submitted code, the script should accept the name of the waypoint file as a single argument. For example, if your script is called `myscript.py`, running:

```
python myscript.py theseWaypoints.txt
```

should use `theseWaypoints.txt`. If no waypoint file is specified, you can use any suitable default strategy. Instructions on how to identify the number of command line arguments and use them can be found here. The number of waypoints, the values of the waypoints, and the initial position and orientation of the turtle is not known in advance. (You might find the discussion here useful for writing the file reader.)

A set of waypoints are provided in a zip file which can be downloaded from moodle.

4.3 Messages

Your controller should publish messages to the new topic `robot_status`. It should send the following:

- The message `STARTED <<robot_x>> <<robot_y>> <<robot_theta>>` when the program starts and first gets information about the robot position.
- The message `WAYPOINT <<waypoint_number>> <<robot_x>> <<robot_y>> <<robot_theta>>` when the robot has reached a waypoint. This includes the last waypoint in the file.
- The message `FINISHED` when the robot has reached the last waypoint.

4.4 Assessment

As noted, this is a lab exercise and will not be marked. However, it exercises many important skills you will need to have to complete the other courseworks. Therefore, we will be ticking off your group on a tick off sheet to confirm that you were able to complete the exercise successfully.

4.5 Resources

You are welcome to consult whatever sources you feel you can use to help to achieve this task.

Ones which might be of use are:

- The file `catkin_ws/src/comp313p_example/scripts/moving_the_robot.py` is the command line based one you used to move the robot around. It shows you how to set up a ROS node, how to query the position and orientation of the robot, and how to print the results.
- The TurtleSim Rotating Left and right and Moving in a Straight Line tutorials show you how to orient towards a location and drive a given distance.

The turtlesim also has a Go To Goal tutorial which attempts to move the robot to a given position. Although it's worth a look, it does not provide a complete solution for the tasking here. There are two reasons. First, we have found that the controller is very unstable and often the robot can just flip out. Second, this controller does not specify the final rotation of the robot.