COMP 417/765 Assignment 1

Due: 23:59, Feb. 21st, 2016

1 Pedagogical objectives

Experience with practical implementations of sensor-based path planning algorithms. Simple feed-back controllers. Empirical analysis of runtimes and robustness.

2 Problems

Each of these problems requires starting a Gazebo simulation of a robot, called the Turtlebot, inside an office building. The command for starting the simulator will always be:

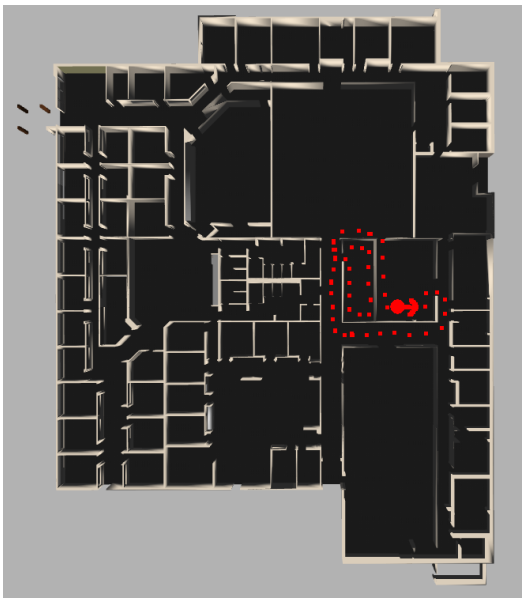
roslaunch turtlebot_world.launch world_file:=worlds/willowgarage.world

Your code will implement a ROS node which accesses the simulated sensors and actuators and compute motions that guide the robot as specified by each problem. You will submit the code you produce for each problem and will be marked on its robust performance, so ensure to make your solutions robust to a range of starting and ending positions. The starting position can be modified by manually dragging the robot to a desired position before you start your code, or you can set it programmatically. You should accept goal positions as ROS parameters when appropriate.

Your solution may be implemented in either Python or C++. You are responsible for testing that it performs properly on the ROS/Gazebo setup on the SOCS lab machines at Trottier. The crucial components are Gazebo version 2.2.3 and ROS Indigo. If you wish to work on your own machine, we recommend you still do a final test at Trottier. It will often be the case that using a matching version at home gives matcing performance, but this is at your own risk.

2.1 Problem One - Wall Follower

A pre-requisite of all Bug algorithms is a wall-following controller that allows the robot to stay next to an obstacle until the condition in the algorithm is met. Begin by implementing a wall follower that is able to follow paths such as the one shown here, utilizing only the laser scans on the **/scan** topic as sensory input.



Your solution may use any logic that you like for this component as long as you manage to follow walls successfully. It is often useful to consider a specify a fixed desired distance from the wall, to estimate the actual robot's distance by performing some computation of sensor data, and by turning a bit left or a bit right accordingly. Also, you are likely to need least two beams from the laser so that you can also estimate the robot's angle with respect to the wall and prevent crashing head first into concavities.

2.2 Problem Two - Bug 2 Algorithm

Now you are ready to implement the Bug 2 algorithm, which we described in lecture. Your solution should make use of the wall following code from Problem One. In order to keep track of the distance to the goal and detect

intersections with the s-line, you may access Gazebo's helpful ground truth localization, which can be found in the topic **/gazebo/link_states**

Ensure that you can successfully navigate on goals similar to the ones shown in this image before submitting. However, you should also attempt several other start and goal configurations. Goals should be specified as two command-line parameters passed when you **rosrun** your node, with the form $goal\_x := 10.0$ $goal\_y := 15.0$.

3 Submission

Submit two files: **wall_follow** and **bug2**. In each case, use the extension **.py** for Python implementations and **.cpp** if you used C++. Submissions are through My Courses.

Our assignment deadline policy for this course is that if you attempt to submit code which is minimally functional by the deadline, you are allowed to consult with other students, the TA or the instructors in order to gain an understanding of what you missed at first. You can submit an improved solution within one week of receiving the initial mark. For this assignment,

minimal functionality means your code must be a proper ROS node that compiles and runs, it must at least move the robot roughly in the right directions initially, but it does not have to reach the goal.