CSci 5551: Introduction to Intelligent Robotic Systems
Spring 2022

Programming Assignment [20 points]

Due: 11:59 PM (CST), Monday May 2, 2022

1 The Bug Algorithms

The insect-inspired Bug algorithms attempt to mimic the motion of bugs to create robot navigation algorithms. There are variants of this algorithm, and our focus for this project would be the *Bug*-0 version. Specifically, the Bug-0 algorithm attempts to navigate to a goal location by following a straight-line path from its current position to the goal. The robot is assumed to have limited local sensing only, so it is not aware of a (a) global map, or (b) obstacles along the path unless it is very close and directly in its path. When a robot encounters an obstacle, it attempts to *go around* it, until there is free space along the straight line from its position to the goal; at that time, the robot starts to navigate towards the goal along this line. Figure 1 demonstrates this idea.

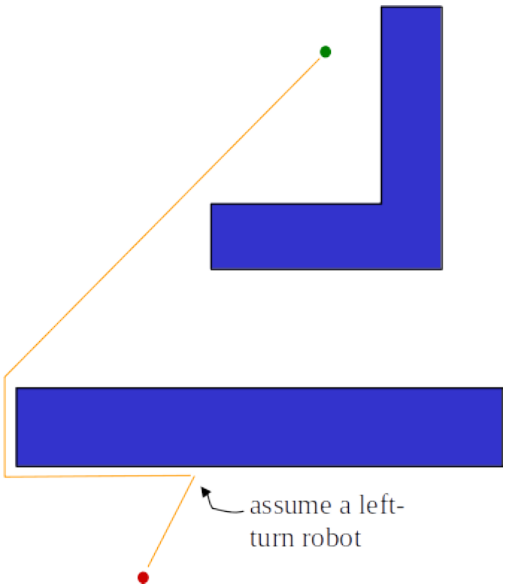

Figure 1: Robot navigation using the Bug-0 algorithm.

The Bug-0 algorithm thus attempts to:

1. head towards goal,
2. follow obstacles until you can head towards the goal again, and
3. continue until goal is reached.

2 The Task

The task in this assignment is to implement the Bug-0 algorithm. The platform of implementation and demonstration will be ROS and the Stage 2D simulator. ROS-Stage is a ROS package for two-dimensional robot simulation. Stage is, for all purposes and needs, a visualization platform as well (shown in the image below), where an designer can test and validate an algorithm for motion planning (and other robotic behaviors).

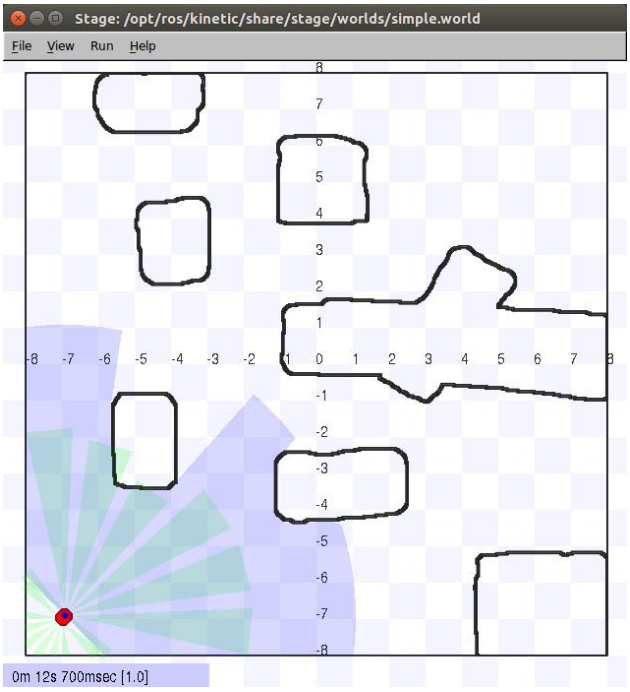

Figure 2: The Stage Simulator Screen.

Sample code to run Stage and programmatically control a robot in Stage is provided in a zip archive. The code demonstrates how to set up a robot, a world map, and other simulation parameters to control a robot in a simulated, two-dimensional map. Running Stage as a ROS node is also demonstrated. For this task, there is no need to reconfigure the robot. It has been set up with a LIDAR and a GPS-like localizer, which gives us its absolute position in the world (*i.e.*, Stage) coordinates.

You are free to choose your own language (Python or C/C++), as long as it is supported by ROS (*i.e.*, has ROS bindings). The task is to build on the sample code to implement the Bug-0 algorithm to be visualized in Stage. The program (or ROS nodes to be precise) will take the 2D coordinate of the destination and the final robot pose as arguments. These arguments maybe be passed as command-line arguments, GUI input or in any arbitrary fashion that you choose. Once the goal location is set, the robot will attempt to navigate from the current position to the goal, and the visualization shall be seen on Stage. For simplicity, you can assumed that the destination shall always be reachable (will not be inside an obstacle or outside the perimeter).

A successful navigation should:

1. find a path when one exists (this includes consideration for robot's footprint),
2. not collide with obstacles
3. demonstrate the whole path traversal in Stage.

There will be only one robot in the environment at any given time, so collisions with other robots will not be a concern.

3 Running Stage

ROS Stage documentation for simulating one robot is available here: `http://wiki.ros.org/stage/Tutorials/SimulatingOneRobot`. The tutorial is written for ROS Groovy, and a bit outdated, but most of the information is still valid. Please skip step 1 in the tutorial, and any step that involves running rosmake must be avoided as well.

However, here's a quick summary of the process:

1. Start `roscore` in a terminal.
2. In another terminal, navigate to the directory containing the *world* file. In the sample, the file is called *bug-test.world*.
3. Type the command `rosrun stage_ros stageros bug-test.world`. This will bring up stage with a map with polygonal/non-polygonal obstacles.
4. You can try the `teleop_twist_keyboard` package to teleoperate the robot by typing `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`. Follow the on-screen instructions to move the robot.
5. Stage will publish the topics needed, and you can seem them all using the `rostopic list` command.
6. See the attached code `stage_mover`. It is a C++ ROS package with a node to move the robot using a `Twist` message type, which is the message type used to control the robot in the given Stage world.
7. We are also providing Python example code.

4 What to submit

Submit the following in **one archive (zip, tar.bz2, etc) file**:

1. Source code (cpp/py files, and launch files and all necessary supplemental files) in the form of a ROS project. Please do not submit your entire catkin workspace, **just the project**.
2. A brief "readme" outlining instructions in running your code.