Project 3: Maze navigation skills

OVERVIEW

In this lab, you will implement several skills to help your Turtlebot navigate through a maze. Building on last project’s work with odometry data, you will also consider data from the Turtlebot’s Kinect sensor.

TASKS

1. Tool setup. A couple of things to think about now that you are gaining experience operating your robot. You can get more useful console output with git if you enable color mode:

   ```
   git config --global color.ui true
   ```

   This will do nice highlighting, for instance when you run `git status`.

   Additionally, you might consider using the `tmux` tool to multiplex your ssh sessions. You can read an introduction to `tmux` at this page: [http://danielmiessler.com/study/tmux/](http://danielmiessler.com/study/tmux/) (or just google “tmux tutorial”).

2. Grab the starter code. Update your git repository to get the starter code:

   ```
   cd ~/catkin_ws/src/e28-labs
   git clone git@github.swarthmore.edu:e28-fall2016/project3-TEAM.git project3
   ```

   Once again, replace TEAM with your team name that you used for project 2. This command will populate your `project3` directory.

3. Run the starter code. Before you get started, you will probably need to make the starter code script executable. To do so, run the commands

   ```
   roscd project3/scripts
   chmod +x starter.py
   ```

   Make sure the robot is running by launching both `minimal.launch` and `3dsensor.launch` from the `turtlebot_bringup` package. Place the robot on the ground roughly facing a wall (maybe 5-15° away from perpendicular, and about 2-4 feet away), and run the command

   ```
   rosrun project3 starter.py
   ```
The console should print two messages: **waiting for TF and laser scan...**, and **waiting for commands...**

In a separate terminal or `tmux` pane, run the command

```
rostopic pub -1 /maze_command std_msgs/String straighten
```

The robot should straighten out with respect to the wall.

**4. Read through the starter code.** Read through the starter code to get an idea of what it’s doing. You might find these resources are also helpful to see how the code works:

- Definition of the laser scan message (with comments):
  [http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html](http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html)
- Specification document concerning “special values” in distance measurements:
- ROS package that converts the Kinect output to a laser scan:
  [http://wiki.ros.org/depthimage_to_laserscan](http://wiki.ros.org/depthimage_to_laserscan)
- Try visualizing the robot’s laser scan in RViz to see what the program is “seeing”:

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

Before you go on, make sure you can answer the following questions:

1. What happens when the robot receives a laser scan message? Does the robot take any immediate action?
2. What code was responsible for setting the robot’s state when you typed the `rostopic pub` command above?
3. At each control cycle during the `straighten` behavior, the controller examines the 2D locations of the endpoints of the “laser beams” ±5° off center. How is the displacement vector between these two points (expressed in the robot’s own body frame) used to estimate an angular error?
4. Why does the `lookup_angles` method need to check for NaN values? How does it try to compensate if the desired angle doesn’t have a valid range associated with it?
5. What happens if you try running the `straighten` behavior when the robot is facing directly into a corner? What happens if you try running the it when the robot is too close (just a few inches or so) to the wall? Why does the robot do what it does in these cases?
5. **Implement additional skills.** Now that you know what’s going on in the starter code, please implement the additional skills demonstrated in the lab kickoff:

- **turnleft** and **turnright** – each of these should turn the robot by $\pm 90^\circ$ from its initial orientation, but end up “snapped” to a wall. A good way to implement this would be to use a similar control scheme to the one from project 2 to begin making a right-angle turn, but smoothly\(^1\) switch to the **straighten** behavior when the angular error (remaining angle to turn) is less than 5-10° or so.

- **nudge** – this should examine the $x$-coordinate of the point at 0° (straight in front of the robot), and drive forward or backwards until that distance is equal to 1.5 feet plus an integer number of 3-foot maze cells. Here is a code snippet you might find useful to round the measured distance to the nearest cell center:

  \[
  \text{rounded_dist} = \text{numpy.floor}(\text{measured\_dist}/(3\times\text{FT}))\times3\times\text{FT} + 1.5\times\text{FT}
  \]

- **forward** – this should drive 2-2.5 feet forward before smoothly transitioning to the **nudge** behavior to “snap” to the nearest cell center.

6. **Going further.** Go beyond the functionality described in the section above. Enhance your project by addressing some of the following points:

- Think about how to perform some of these behaviors if the robot can’t see any walls. You may assume a **NaN** range reading always implies “distance too close”.

- Can you add extra “safety” behaviors, like refusing to drive into walls or unexpected obstacles found by the laser scan?

- How much can you increase linear and angular velocity while still keeping the behaviors smooth?

- Can you pay attention to additional distances (i.e. off to the sides of the robot) to generate steering commands when the robot is driving **forward**, in order to keep it centered in the corridor?

- Try developing an extra skill/behavior. Easy: **drivetowall** drives forward until the robot as 1.5 feet from a wall (i.e. to zoom down a long corridor). **Difficult:** **recover** positions the robot in the center of the cell it starts in, facing square to some wall – regardless of its initial position and orientation. This should work even if the robot starts out wedged in a corner, or glimpsing the edge of a wall.

\(^1\)i.e., without resetting the state timer – see the second argument to **reset_state**
WHAT TO TURN IN
Please use git to add, commit, and turn in your project 3 code. Also, please submit a plain text file or PDF addressing these points:

- Who did what in your group?
- Write out answers to the questions about the starter code above – you might want to provide a diagram for question 3 if it makes your answer clearer.
- Provide a short (2-3 paragraph) description of the functionality you added beyond the basic skills required.
- Include a link to a YouTube video (accessible by me – this means “unlisted” rather than “private”) which demonstrates all of the behaviors that you implemented as well as your extra functionality. Make sure that each distinct behavior is demonstrated at least twice. If you want, you can demonstrate these behaviors in the context of a larger task, like traversing the mini-maze set up in the lab, once in each direction.