Lab 2
Characterizing odometry

OVERVIEW
In this lab, you will learn about odometry and characterize the accuracy of your turtlebot.

TASKS
1. **Reading.** Please read the “Introduction to tf” at the ROS tf tutorials page: [http://wiki.ros.org/tf/Tutorials](http://wiki.ros.org/tf/Tutorials). Then, read through (and optionally complete) tutorials 1 and 2 (Python versions). You may also want to watch some of the video tutorials as well.

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

```bash
cd ~/catkin_ws/src/e28-labs
git checkout master
git pull
git checkout turnin
git merge master
```

At this point, you should also be starting to become more familiar with git. My favorite resource is the “Git Cheatsheet” at [http://ndpsoftware.com/git-cheatsheet.html](http://ndpsoftware.com/git-cheatsheet.html).

3. **Drive in a square.** Modify the `starter.py` program in the `lab2` directory to create a new file, `drive_square.py`, which replaces the “turn left” behavior with a “drive straight” behavior. This behavior should instruct the robot to drive 1.2192 m forward (this is equal to four 12” floor tiles’ distance) before turning right.

Your program should exit (`sys.exit()` in Python) after two full square “laps”. After it ends, carefully measure the robot’s position and orientation, and compare it to the odometry estimate of the overall change in position and orientation since the program was started.

Assume the robot starts at \((x, y, \theta) = (0, 0, 0)\). Let \((x_e, y_e, \theta_e)\) be the expected final pose of the robot – in this case, also \((0, 0, 0)\). Next, let \((x_m, y_m, \theta_m)\) be the measured final pose when the robot halts (yes, it’s hard to measure the robot’s heading in the lab – do your best). Finally, let \((x_o, y_o, \theta_o)\) be the final pose of the robot reported by odometry (again, assuming you started at the zero pose).

We can now compute two different types of error: *execution error*, which compares where the robot actually ended up with where we expected it to end up, and *dead reckoning error*, which compares

\[1\]

Incidentally, the way you should compute this is by looking at the relative transformation between the initial and final poses of the robot, as reported by the odometry node.
where the robot actually ended up to where it thinks it ended up. Mathematically,

\[
\begin{align*}
\varepsilon_{e,xy} &= \sqrt{(x_m - x_e)^2 + (y_m - y_e)^2} \\
\varepsilon_{d,xy} &= \sqrt{(x_m - x_d)^2 + (y_m - y_d)^2}
\end{align*}
\]

\[
\varepsilon_{e,\theta} = \|\theta_m - \theta_e\| \\
\varepsilon_{d,\theta} = \|\theta_m - \theta_d\|
\]

Run three trials of the square driving procedure, and report all four of these errors for each trial, along with averages. Also, record a video showing your robot driving a square, along with whatever ad-hoc procedures you devised for measuring the robot’s final position and orientation.

4. **Follow a specific sequence of actions.** Create a new node, `odom_test.py`, which causes the robot to perform the following sequence of actions:

   1. Drive four tiles forward.
   2. Turn 90° right.
   3. Drive two tiles forward.
   4. Turn 90° left.
   5. Drive six tiles forward.

The robot should pause briefly between each action. Once again, measure all four errors for each of three trials and report them along with their averages.

Also, record video of your robot executing this sequence of actions once.

**WHAT TO TURN IN**

Please use git to add, commit, and turn in your lab2 code (both nodes you created).

Finally, write a short informal report describing how you developed your program, including:

- who did what
- what problems you ran into, and how you solved them
- measurements and averages of errors mentioned above
- link to a YouTube video (it can be private, as long as I can access it), showing your robot running the programs associated with both nodes once, along with video documentation of your calibration and measurement procedures

Submit your report (in PDF format) using Git, too.