CIS 3415, Project-4 (**8-pt**)

1 Description

This project is all about localization. First you'll learn how to make the robot move around when it is localized without having to do anything. Then you'll move on to using the Player implementation of the particle filter.

2 Starting up

1. Login to your computer (user name is `student`, password is `student`) open terminal windows, get a copy of the project files (`project4.tgz`) from the instructor, put this on the Desktop and extract the files.
2. This time your starting point is the file `local-roomba.cc`.
3. Start by running it. To do that start up player:
   ```
   player world41.cfg
   ```
   Then you can compile and run `local-roomba` in your second window:
   ```
   ./build local-roomba
   ./local-roomba
   ```
   and as before the simulated robot will start moving
4. At first it may look exactly the same as before, but this time you'll see the robot start to output information like:
   ```
   We are at
   X: -4.18556
   Y: -7.81444
   A: -0.785398
   ```
   These are the coordinates of the robot — it knows where it is.
5. Note that the robot isn't calculating where it is, it is being told by the simulator Stage (fake localization).
6. The challenge is to take `local-roomba.cc` and edit it so that the robot will drive from its starting point to the location $X = 5$, $Y = -3.5$.
7. You should do this **without** using odometry.

   Hint: The way for the robot to move directly to its destination is to calculate the difference between the robot's current orientation and its desired direction toward the destination (using trigonometry), make the robot turn so that this angle difference becomes close to zero and only then drive straight toward the destination. You can stop anywhere within half a meter of the destination.

   Hint: Proportional control, which is effective for this kind of task, should be considered. It has the robot move more quickly when it is far from the position/orientation it is aiming for, and more slowly as it gets closer, following a linear profile. That way the robot doesn't overshoot its target when run in the real world, but also doesn't take too long to get into the neighborhood of the target. It would even recover if the robot is temporarily out of its desired position/track. Proportional control applies to both rotational and translational movement.
8. When you are done, save your program as ***(*****your-names*****)*****-proj4-part1.cc** and make sure you put your name in the comments.
9. You'll need to submit this to me after you are done with the project.

3 Now real localization

1. The second part of this project is to use a laser with the simulated robot to really localize (where the robot computes its position).
2. This time you need to run:

   ```
   player world42.cfg
   ```

   You'll see the same world as before, but this time the robot has a laser.
3. Compile and run the controller:

   ```
   ./build real-local
   ./real-local
   ```

   and watch the robot run.
4. The robot, as it has at the start of previous projects, will trundle off to the south-east.
5. However, the display shows some new things.
6. The display is now dominated by a series of these:

   ```
   X: 4.05511   Y: 4.68217   A: -0.116743   W: 0.005
   X: -2.45264  Y: 2.15322   A: 0.076456    W: 0.0035
   X: 2.55298   Y: 0.762078  A: -0.0410866  W: 0.0031
   X: -5.30263  Y: 4.97457   A: -0.0395662  W: 0.0029
   X: 6.02264   Y: 0.729982  A: 0.00262671  W: 0.0024
   X: 4.01215   Y: -3.67916  A: -0.0180902  W: 0.0022
   X: -5.04173  Y: 0.98516   A: -0.225971   W: 0.0019
   X: -0.675089 Y: 5.43747   A: -0.0741516  W: 0.0018
   ```

7. Each of these is a *hypothesis* about where the robot is, with its x coordinate, y coordinate, angle and *weight*.
8. The weight is an estimate of how likely the robot is to be at that location.

4 It's all about configuration

1. Take a look at the configuration file `world42.cfg`.
2. The difference between this configuration file and the others we used before is in the last element.
3. The AMCL driver is a version of the particle filter that we talked about in the lecture on localization.
4. All those possible positions you see on the screen are (more or less) the particles.

5 Know where you are

1. You should notice the number of hypotheses reduces as the robot runs.
2. Your first task is to modify the controller so that the robot localizes properly.
3. That means so that the robot ends up with just one or two hypotheses.
4. If there is one, then clearly localization has figured out where it thinks the robot is.
5. If there are two, then as long as one has a probability of very close to 1, localization thinks it has figured out where the robot is.

6. Your job is to get the robot so that localization not only has one or two hypotheses, but that at least one of them is reasonable. As in it is close to where the robot really is (which you can see from the simulator).
7. Don't expect localization to be correct, but it should be approximately correct.
8. It's recommended that you add this line `update_thresh [0.2 0.02]` to the **amcl** driver in the file world42.cfg. This forces the amcl to update with changes of 0.2-m in position or 0.02-rad in angle; the default setting is [0.2 pi/6] which causes no updates when angle is changed < 30 degrees.
9. You will probably want to make the robot moves in different ways to do this — moving robots localize quicker than stationary ones, and having the robot rotate, so that its sensors "see" different bits of the world, usually helps as well.
10. Since AMCL localization doesn't always converge to the correct position, you could save time by checking if the convergence is close to (-6, -6), the actual initial position (which is supposed to be unknown). If it's not close, the AMCL localization has already failed and you could simply allow the program to quit by itself so you can immediately rerun it.
11. Since particle filtering is a non-deterministic process, I will not be able to run your program and obtain the exact same result. Therefore it is important that you save a successful run by redirecting all the output display into a text log file: `$ ./real-local > log.txt`
12. Please include in the log file both the output generated during the localization phase and the navigation phase (described in the next section). Please print "Success Localization!" and "Success Navigation!" in the log file where they were achieved so I can focus on the key moments of your run more easily. You only need one successful run.

6 Know that you know where you are

1. When localization is generating just a couple of hypotheses, it has figured out where it thinks the robot is.
2. But does the robot control program, the bit in main, know that the position reported by the localization module is the right position?
3. Modify the controller so that when localization is down to a few hypotheses, the controller stops the robot moving.
4. Have the controller print out a message from main that says:

   ```
   I am at:
   ```

   and then the location and then:

   ```
   I am N percent sure of my position
   ```

   where `N` is the weight of the most likely hypothesis.
5. To do this you will have to modify the function `readPosition` since right now this just returns the location in the last hypothesis returned from localization (which is typically **not** the most likely one).
6. Once your controller stops the robot when the robot is localized, you can add the code from your solution to part 1, so that the robot can navigate to the destination point.
7. When you are done, save your program as ***(your-names)*-proj4-part2.cc** and make sure you put your names in the comments. You'll need to submit this to me after you are done with the project.

7 Handling Proxies

All the relevant commands for the localization and laser proxies are demonstrated in the code.