Lab 2: BUG Algorithm (100 Points)

Due electronically Thursday 10/12/17, 11:30AM

In this lab you'll be implementing the BUG2 algorithm to move the GoPiGo from a designated starting point to a goal point within its environment. This lab will familiarize yourself with basic navigation control and utilizing sensors. You are required to submit all code samples and schedule a demo with the TAs to show that your code functions as advertised. **We will NOT be running your code**. If you do not participate in a demo you will not receive a grade for this assignment. There are no exceptions. Demos should last no longer than 5 minutes so please come prepared.

Assignment

Implement a BUG2 algorithm to move the GoPiGo from a designated starting point to a goal point in an environment. Your robot should start out along the straight line path from start to goal (the M Line). Using the ultrasound sensor, if it sees any obstacles along the way, it will invoke a perimeter following behavior until the M Line is reacquired and the robot again starts out for the goal point along the M Line. When the goal point is reached, the robot will stop.

Assume a trajectory from a start point to a goal point about 3 meters in front of the robot. Add 2-3 test obstacles (garbage cans and cardboard boxes work well here!) that impede the path, requiring the GoPiGo to invoke BUG2 behavior.

Your program should also map out your robot's progress graphically, showing robot position and robot orientation (robot pose) as it moves along (odometry). You should also display on the map locations where the ultrasound sensor has seen an obstacle. A quick tutorial on python plotting using matplotlib (installed on your RaspberryPi) is here: http://matplotlib.org/users/pyplot_tutorial.html

Video your robot as it does its movement from start to finish, avoiding obstacles. Post the video to youtube and include the link in your handed in README file.

Your BUG2 algorithm should also be aware when it is trapped inside of an obstacle and report this. Make sure you test this part of your algorithm as well (no video required of this, but we may test this case in our demos).

In the README, along with the items mentioned above, please also include a brief summary of any adjustments you made to make the BUG2 algorithm work for your GoPiGo and why they were necessary.

Tips

- Mapping may slow down your communication and affect performance of your robot. If you are experiencing this, then cache the pose and obstacle information and map it out after the robot stops.
- The BUG2 algorithm contains assumptions that don't apply to a real world robot. You may have to make small adjustments to satisfy real world constraints.

Instructions for submission

All assignments must be submitted with working code and a README that states that a section was completed and/or answers the questions listed in the section for the section of this assignment. Your README should also include All code must be written in Python, specifically python 2.7.

Your submitted code should be in a tarball (see here for how to tarball) and should be stored in a directory of the format 'hw2_<uni1>_<uni2>_<uni3>.tar.gz', where <uniX> is replaced with the uni of each teammate (i.e. hw2_djw2146_djw2156_djw2166.tar.gz). The directory of this folder should match the following:

hw2_djw2146_djw256_djw2166

-- README.md

-- bug2.py

Your README should follow the following format:

""

This is a Lab 2 submission for group **XXX**
Team Members: *member_1*(*UNI_1*), *member_2*(*UNI_2*), *member_3*(*UNI_3*)

Youtube link for bug2 demo: <link>

<Description of adjustments to Bug2 Algorithm>
<Descriptions of assumptions based on robot>
""

You should also include descriptions of any novel methodologies you used to implement the solutions for this assignment.

Any deviations from this will not be considered for grading.

Extra Credit: (10 points):

Robotic Mapping: Create a 2-D occupancy grid for your robot environment, and map it out with cells that are either occupied or empty. You might think of this as how a vacuum cleaner robot "learns" its environment so it can remember where to clean the floor and where obstacles are. Initially all cells in the grid are empty, and as the robot navigates the environment, using its ultrasound sensor, it fills in the grid spaces as free space or obstacles as they are sensed. The robot's behavior should be such that it covers the room area. You might try combinations of behaviors such as: random, in that it goes in a direction that is randomly chosen until it hits an obstacle, at which point it changes its behavior to continue exploring the environment; Perimeter Following of an obstacle, and Spiraling to cover an area. You can use any behavior you like, just output an occupancy grid map of the environment when the program finishes (see figures below). Some details:

1. Assume a grid cell size of the diameter of the create robot.
2. Make sure you create a bounded environment so the robot has a finite area to map (something like a rectangle 2-3 meters on a side). You can create your own obstacles or get an obstacle set of small plastic road cones at the lab to use as obstacles.
3. Stopping conditions: if your robot has not updated any new cells after a certain amount of time has passed, you should consider stopping.
4. Take a photo of the environment you are mapping and include it with your map so we can see how well your mapping went.