CSCI 4972/6972: Algorithmic Robotics
Programming Assignment Three (PA3)
Deadline: 1pm GMT-05 March 28, 2019

This assignment will deepen your understanding of occupancy grid construction and of how noise can affect robot perception. Your robot will first explore an unknown planar environment with obstacles to construct an occupancy grid with perfect lidar data and motion control. Then it will redo this task while Gaussian noise corrupts the lidar data and motion controller.

You will submit a written report answering all the questions of all parts of this assignment and you will submit a ROS package that can perform the required demos. Click here to pa3_student.zip for all parts of this assignment. Read each part of the assignment carefully, so you know which parts of the code you are expected to write and which parts you must leave alone.

Part 1: Pose-to-Pose Motion Control

Goal:

For this part, you will need to answer everything in the questions section below and develop a ROS program to drive a turtle in turtlesim from an arbitrary initial pose (x_{init}, y_{init}, and θ_{init}) to a goal pose ($x_{goal}, y_{goal}, \theta_{goal}$) within a tolerance of 0.01, defined as by the following metric ρ:

$$\rho = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2 + c(\theta - \theta_{goal})^2} < 0.01, \tag{1}$$

where c is a constant to be chosen by you, x, y, θ is the final pose, and θ has units of radians.

Background:

You need to implement the turtlejectory node in *pa3_student/src/turtlejectory.py*. Do not modify any other files for this part. The turtlejectory node publishes Twist messages to *turtle1/cmd_vel*. The turtlejectory node subscribes to both *turtle1/pose* for the current pose and */goal* for the desired pose. The recommended pose-to-pose controller is described in Corke 4.1.1.4 for the bicycle model with important quantities shown in Figure 4.10. If you think a different controller makes more sense, then you are free to implement that one. The launch file *turtlejectory.launch* spins up every node needed and queries you for the goal pose (x, y, and θ) in an x-term window, where $\theta \in [-\pi, \pi)$.

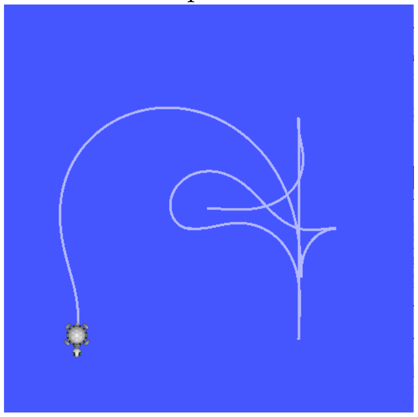
Hints:

1. Some trial and error will be needed to find "good" gains.
2. *turtlejectory.py* from project two is a good starting point.
3. Does velocity have to be positive?
4. How do you handle cases "behind" the robot?
5. Ideas from lecture 5 may help.

Demonstration:

1. Run the turtlejectory launchfile and with goal inputs in this order: $\{(8, 8, \pi/2), (8, 2, \pi/2), (5, 5, \pi), (9, 5, 0), (2, 2, -\pi/2)\}$.

Below is a screen shot generated by our implementation of Corke's move-to-pose controller with gains of 1, 7 and -4 for ρ, α and β respectively. Your implementation doesn't have to match this one so long as it is able to do all the poses



Questions:

1. Provide a screen shot of your robot executing the paths it found for the demonstration defined just above.
2. If you implemented Corke's move-to-pose controller, what values of c, k_α, k_ρ, and $k_\beta < 0$ did you choose? Explain why you chose them.
3. If you implemented Corke's move-to-pose controller, describe the robot's behavior when the controller had "bad" gains.
4. If you implemented a different controller, provide the equations defining the twist sent to the robot's wheel driver and the gains you used.
5. If you implemented a different controller, what value of c did you choose. Explain why you chose that value.
6. If you implemented a different controller, describe problems you encountered during its implementation.

Part 2: Occupancy Grid Construction

Goal:

Drive your robot under keyboard control to build an occupancy grid of the robot's environment.

Background:

You need to implement the occupancy grid algorithm in *pa3_student/src/mapping.py.* You have been provided with skeleton code that contains required constants and a high-level overview of the steps you will need to take. The mapping node publishes OccupancyGrid messages to */map.* It also subscribes to */r2d2/laser/scan* (LaserScan) and */groundtruth* (Odometry) which provide sensor data and the pose of the robot in the world frame respectively. Launch file: *pa3_student/launch/r2d2_mapping.launch.*

In order to move the robot in the environment you have been provided a simple teleoperation interface. Simply click on the X-term window and use the arrow keys to move the robot: the left arrow turns the robot left, the right arrow turns the robot right, the up arrow moves the robot forward, and the back arrow stops the robot. Note that you only need to press a movement key once and the robot will continue to perform that movement operation until a new movement key is pressed.

The recommended way to approach updating the occupancy grid is to use Bresenham's line algorithm to identify which grid cells are crossed by the rays of the lidar sensor and then update the occupancy probabilities (or log odds) using methods presented in lecture07. Assume the true positive and true negative rates of $\alpha = 0.9$ and $\beta = 0.7$ of obstacle detection, respectively.

Hints:

1. Your occupancy grid will use the ROS ***nav_msgs/OccupancyGrid*** message. Note that the rows represent the y-axis and the columns represent the x-axis. Entries in the grid must be between 0 and 100. The MapMetaData information will be provided for the OccupancyGrid messages within the code. Feel free to change the values, but make sure to revert them for submission.
2. The wikipedia page for Bresenham's line algorithm has a clear description of the algorithm.

Demonstration:

1. Demonstrate the correctness of your occupancy grid mapping code by driving the robot around in gazebo in order to map all four sides of the obstacle in the obstacle.world file. Submit a screen-shot of your map being displayed in RVIZ (you should not need to modify RVIZ to visualize the map).

Questions:

1. In a few short sentences, describe the intuition behind Bresenham's line algorithm.
2. Describe one reasonable approach for handling scans that partially go beyond the boundaries of the map.
3. How does the occupancy grid algorithm perform when $\beta = 1 - \alpha$?

Part 3: Sensor and Controller Noise

Goal:

Evaluate the performance of your algorithms from parts 1 and 2 under the influence of noise in the sensor readings and wheel commands.

Background:

Hardware in the real world is subject to random noise. Imperfect sensor readings and controller output have the potential to break algorithms that were developed without considering noise. Gaussian noise will be applied to your controller output and sensor readings for this section.

To run your code with noise include "*use_noise:=true*" as a parameter when launching the respective launch files provided in parts 1 and 2. For part 1, this will apply Gaussian noise to both the velocity and angular velocity published by the turtlejectory node. For part 2, Gaussian noise will be applied to the range readings from your sensor in mapping.py.

Demonstration:

There are no demonstrations for this section of the assignment.

Questions:

1. How does the twist noise impact the performance of your pose-to-pose motion controller?
2. How does the sensor noise impact the construction of your occupancy grid?

Submitting

You will submit a write-up and a ROS package, both in the same zip file and sent to trinkle@gmail.com by the deadline. There is no set length for the write-up. Use as much space as you need to answer all questions thoroughly. Make sure to include an acknowledgment statement describing any resources that you used or assistance that you received. The format of your submission is:

- Please name the zip file you submit as: pa3-*name*, where *name* is replaced by your first name.
- Your name, date, programming assignment title should appear on the write-up and in the comments of your code.
- List the questions asked above and give your answers.
- List and briefly describe useful things you discovered, that go beyond the questions of this assignment.
- Format the written part of your assignment as a PDF file

Grading

Submissions will be graded according to the following rubric:

Success of demonstrations	35%
Correctness of answers to questions	50%
Clarity of comments and PEP8 compliance of code	15%