CSCI 4972/6972: Algorithmic Robotics
Programming Assignment Four (PA4)
Deadline: 11:59pm GMT-05 April 25, 2019

This assignment will emulate a real-life working situation; there is too much to do in the available time, so you need figure out what parts are worth doing and what parts are not. The theme of the assignment is autonomous search and rescue. There are three parts: (Part 1) you will use SLAM techniques to build a map ensuring that your robot has "seen" everything, (Part 2) you will use motion planning algorithms with a given map to generate paths between poses, (Part 3) you will combine components of Parts 1 and 2 to perform search-and-rescue tasks in a known map.

The provided code pa4_student.zip solves some of the problems, but not all of them. Problems not already solved are stubbed out. Points are assigned based on your answers to questions (50%) and your code's performance (50%).

In this assignment, the light-weight simulator STDR (Simple Two-Dimensional Robot) will be used instead of Gazebo to reduce the simulation load on your computer. STDR is fully ROS compliant; each robot and sensor emits a ROS transformation, all measurements are published as ROS topics, it is compatible with RVIZ, and it can run across multiple computers.

In the following sections, the assigned work in each part has been broken into tasks and demonstrations. Tasks are to be implemented on your own time. Your points for the tasks will be assigned based on your descriptions of them in your report. Demonstrations will be performed in class on the date that the assignment is due, April 25. Each demonstration will have its own launch file in the pa4_student/launch directory. These files have been provided (you will need to add your code to make them work appropriately) and will be launched on demonstration day as your submission. The final report is to be emailed to trinkle@gmail.com before midnight on demo day, April 25.

In order to install the software necessary for this assignment you will need to run the following commands in your Linux terminal:

$ sudo apt-get install ros-kinetic-stdr-simulator

$ sudo apt-get install ros-kinetic-teleop-twist-keyboard

Part 1: Localization and Mapping

Goal:

The robot's task is to build a map as completely and accurately as possible, while being controlled by a human via tele-operation.

Background:

The provided code allows you to explore the map via tele-operation from the keyboard. It also launches the ROS package GMapping that will perform localization and mapping using a "highly efficient Rao-Blackwellized particle filter to learn grid maps from laser range data" [openslam.org]. You are to use GMapping as a performance reference for your tasks and demonstrations. It is unlikely that your code will run faster than GMapping, but it will not necessarily be orders of magnitude slower. Neither the sensors nor the controls are perfect; both are corrupted by additive Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 0.1$. Otherwise the robot's controller has an accurate model of the robot's motion, i.e., if the noise were absent, the robot's odometry model would be perfect. Landmarks exist within the map that publish their positions in the world frame when the robot's true position is within a threshold distance from the landmark. The robot will spawn in the world at position (1, 2, 0) with orientation (0, 0, 0, 1) in the world frame, i.e., the robot's body-fixed axes begin parallel to those of the world frame.

You may not use the output from the GMapping package topics */gmapping/map* and */robot0/odom* (note that when GMapping is not running the true position of the robot is published to this topic) in your code

In order to launch the appropriate files for this portion of the assignment you will need to open two terminals.
In one terminal launch pa4_student/slam.launch:
$ roslaunch pa4_student slam.launch
In the second terminal you will run the tele-operation node:
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=robot0/input_vel

Launching the p1_score.py file will evaluate your code's performance on the demonstrations. Note that you will have to publish your answers to the appropriate ROS topics for scoring *p1_demo1/answer* and *p1_demo2/answer*.

Hints:

1. The true pose of the robot is published on the */robot0/odom* topic.
2. The landmarks will publish their locations (with noise) to the */landmarks* topic when the true pose of the robot is within a threshold distance from the landmark. The w component (the fourth component) of the orientation will hold the landmark's ID.
3. Information about landmark ids, positions, and thresholds can be found in pa4_student/src/landmarks.py. Running set_landmarks.sh will show the landmarks in the STDR GUI.
4. To run the code with noise added to the robot controls or LIDAR sensor add **control_noise:=true** or **lidar_noise:=true**, respectively, to the end of the launch command.
5. Dead reckoning makes use of the initial pose, the commanded twist, and motion model (unicycle) of the robot in order to determine its new pose.

Tasks:

Points for the tasks below will be awarded based on your PA4 write-up. You may not use the output from the GMapping package topics */gmapping/map* and */robot0/odom* (note that when GMapping is not running the true position of the robot is published to this topic) in your code that performs the following tasks:

1. **6 points:** Localize your robot using the dead reckoning technique of calculating the robot's current pose based on the control inputs. Publish the pose in the world frame to the */my_dead_reckoning* topic using the visualization_msgs Marker message. At a minimum, your write-up of this task should include two plots: one showing the exact (x,y)-path of the robot and the other showing the path computed by dead reckoning and the other plot showing the exact orientation and the dead reckoned orientation versus time.

2. **7 points:** Localize your robot using the extended Kalman filter algorithm. Publish the location of the robot in the world frame to the */my_kalman_filter* topic using the visualization_msgs Marker message. At a minimum, your write-up of this task should include two plots: one showing the exact (x,y)-path of the robot and the other showing the path computed by dead reckoning and the other plot showing the exact orientation and the dead reckoned orientation versus time.

3. **5 points:** Using your code from PA3 as a starting point; create your own occupancy grid algorithm where the map size stays constant. Publish your map to the */my_static_map* topic using the nav_msgs OccupancyGrid message. At a minimum, your write-up of this task should include two figures: one showing the map produced by your algorithm and the other showing the one produced by the GMapping package.

4. **7 points:** Modify your static occupancy grid code so that it is dynamic. The starting grid should not encompass the entire area you are attempting to map and should change size (and maybe origin) when the laser scans go beyond the current map boundary. Publish your map to the */my_dynamic_map* topic using the nav_msgs OccupancyGrid message. At a minimum, your write-up of this task should include two figures: one showing the map produced by your algorithm over time as it changes size/shape and the other showing the one produced by the GMapping package.

Demonstrations:

Points for the tasks below will be awarded based on your in class demonstration performance. You may not use the output from the GMapping package topics */gmapping/map* and */robot0/odom* (note that when GMapping is not running the true position of the robot is published to this topic) in your code that performs the following tasks:

1. **12.5 points:** On the day the assignment is due you will receive a new map *p1_demo1.yaml* a new *landmarks.py* file that will be used in this demonstration. Build a map of the environment using tele-op given perfect controls and lidar data.

2. **12.5 points:** On the day the assignment is due you will receive a new map *p1_demo2.yaml* and a new *landmarks.py* file that will be used in this demonstration. Build a map of the environment using tele-op given noisy controls and lidar.

Questions:

1. **12 points:** Explain a few of the advantages and disadvantages of using only dead reckoning.

2. **12 points:** What are the landmarks used for in the extended Kalman filter algorithm?

3. **13 points:** How would you localize your robot if you had no noise in the controls (i.e. your robot instantaneously achieves the control velocity)?

4. **13 points:** Why would we want to use a dynamic occupancy grid when working within a SLAM scenario.

Scoring:

Launching the pa1_score.py file [NOT YET PROVIDED] will evaluate your code's performance on the demonstrations. Note that you will have to publish your answers to the appropriate ROS topics for scoring *p1_demo1/answer* and *p1_demo2/answer*. The final map output by your algorithm needs to be processed by you so that it conforms to the following specifications for scoring:

- resolution: 0.2
- origin: position = (0.0, 0.0, 0.0); orientation = (0, 0, 0, 1)
- width = 775; height = 746
- thresholds: occupied = 60; free = 30

Each map from the Demonstrations section will be scored based on its accuracy. Any grid cell with a value less than or equal to 30 will be changed to 0 and all other grid cells will be changed to 100 (unseen cells should remain at the default value of -1). Your score will then be:

$$score = P \cdot [1 - (n_{errors}/N_{cells})] \tag{1}$$

Where P is the number of points for that demonstration, n_{errors} is the number of classified cells, and N_{cells} is the number of total cells in the grid.

Each algorithm from the Tasks section will be scored on an all or nothing basis. If your algorithm seems to be executing the appropriate task you will get full marks. Otherwise, you will receive no marks for that task.

Part 2: Motion Planning

Goal:

For this part, you will need to answer all the questions section below and develop a ROS program to plan paths and execute them as live demos.

Background:

There is no code provided other than navigation.launch. The launch file can be edited past the comment "custom additions," but do not change anything before that line. The odometry is completely accurate, unlike Part 1. The launch file takes in a starting pose (x, y, θ) and an argument for a map file name (.yaml files in maps directory). You cannot use ROS packages to assist in path planning and navigation.

The launch file spins up a goal input node that queries the user for a goal pose. Upon reaching the goal pose, the goal input node publishes that the goal has been met and requeries the user for a goal pose. After a 30 second timeout, the goal input node ends the goal pose query and requeries the user for a goal pose.

Preprocessing of a map is only allowed for certain demos and tasks. Preprocessing takes in a map and creates a graph that helps speed up path-finding calculations during the demonstration or task.

The launch file spins up a scoreboard node that keeps track of the goal poses achieved and the distance travelled to reach those goal poses. This information is used to calculate an overall score for that navigation task.

Hints:

1. Probabalistic Roadmap Method (PRM) is a "multi-shot" planning algorithm useful when a single map will be used for many planning queries. It is especially useful if you can preprocess offline prior to receiving the planning queries.
2. Rapidly-exploring Random Tree (RRT) is a "single-shot" planning algorithm, useful when a map will be used for only one planning query.
3. Launch file arguments can be helpful for making a distinction between tasks.
4. The ROS move_base package provides a good framework to mimick as you design and develop your code.

Tasks:

(x, y, θ) is used to define poses. To get credit for these tasks, show a screen shot of the results from the scoreboard node for each task and discuss the results in your write-up.

1. **10 points**: You are given the map *hospital_section.yaml* to preprocess. Your initial pose is $(1.5, 20, 0)$. You must get to the poses:
   $(4, 15.4, 0), (50, 15, 0), (38, 10.5, \pi), (31.75, 1, \frac{\pi}{2}), (30.8, 20, \frac{-\pi}{2})$
2. **5 points**: You are given the map *simple_rooms.yaml*, but cannot perform any preprocessing. Your initial pose is $(3, 7.5, 0)$. You must get to the pose:
   $(18, 7.5, \pi)$
3. **5 points**: You are given the map *sparse_obstacles.yaml*, but cannot perform any preprocessing. Your initial pose is $(8.5, 5.5, 0)$. You must get to the pose:
   $(14.4, 12.5, \frac{\pi}{2})$

4. **5 points**: You are given the map *hospital_section.yaml*, but cannot perform any preprocessing. Your initial pose is $(30.8, 20, \frac{-\pi}{2})$. You must get to the pose:
   $(1.5, 20, 0)$

Demonstrations:

1. **15 points**: The day the demonstration, you will be given the map *doras_map.yaml* to preprocess. During the demonstration, you will be given an initial pose and a set of goal poses. The robot must get to every goal pose.
2. **10 points**: The day the assignment is due, you will be given the map *amazing.yaml*, but cannot perform any preprocessing. During the demonstration, you will be given an initial pose and a goal pose. The robot must get to the goal pose.

Questions:

1. **10 points:** Which planning algorithm(s) did you implement? Why did you choose it (them)?
2. **10 points:** What sort of preprocessing did you do on the maps (if any)? How did it help?
3. **10 points:** What sort of postprocessing did you do on the paths (if any)? How did it help?
4. **10 points:** What algorithms did you use for navigation along the paths your algorithms planned?
5. **10 points:** What algorithms did you consider using for navigation but did not implement? Why not?

Scoring

The scoring for this Part is based on whether or not your robot reaches the goals. For each demo and task, the points awarded is $P\frac{n}{N}$, where P is the number of points for the task or demo in question, N is the number of goals to be reached, and n is the number of goals your robot reached. Your robot is considered to have reached a goal pose if its final pose satisfies the following inequality:

$$(x_{goal} - x)^2 + (y_{goal} - y)^2 + (\theta_{goal} - \theta)^2 < 0.1 \tag{2}$$

where the variables without subscripts are the final pose of the robot.

Part 3: Search and Rescue

Goal:

Your robot is a first-responder entering a disaster site. It's job is to autonomously navigate its environment while searching for victims and reporting their locations as quickly and accurately as possible.

Background:

Since the environment is the scene of a disaster, the robot does not know what is navigable and what is not. Its main job is not to map the environment, but rather to find victims and report their locations accurately. Some partial map construction may be beneficial as it needs to leave an explored area for unexplored territory. In order to not penalize any errors in your localization and mapping code from Part 1 twice you may use the GMapping package to handle your map construction and maintenance for the tasks and demos of Part 3.

The provided code launches the STDR simulator, RVIZ, and a node that will publish the locations of the victims. Victim locations are published to the */victims* topic as a geometry_msgs PoseArray once the true position of the robot is within a threshold distance. However, the victim positions are corrupted by additive Gaussian noise with mean $\mu = 0$ and standard deviation σ, the latter being a piece-wise, monotonically decreasing function of the robot's distance from the victim.

In order to launch the template for this assignment you will need to input the following into a terminal:
$ roslaunch pa4_student search_rescue.launch

Running the p3_score.py file will evaluate your code's performance on the demonstrations. Note that you will have to have already written the victim_report.csv file before launching the scoring code.

Hints:

1. The average reported position of the victim may be more useful than the reported position of the victim.
2. Victim noise can be turned on by including the **victim_noise:=true** argument when you launch the file.
3. Make sure you do not erase other victims from your csv file when adding a new victim.
4. When the robot gets close to a victim, more local planning algorithms such as Bug or gradient descent may be better than PRM or RRT algorithms.

Tasks:

1. **12.5 points:** Your robot will need to autonomously navigate the environment (no tele-op) with the goal of covering the entire map. Think of this as motion planning for sensor coverage of the whole map, so it will find all the victims. Write an algorithm that allows the robot to search the entire the environment given a LIDAR sensor and occupancy grid map.
2. **12.5 points:** Write an algorithm that takes your robot from its current pose to its best estimate of the victim's location. It will be helpful to re-calculate the victim's location as the robot moves toward the most recent estimate because the noise in the victim's location will reduce the closer the robot is to the victim.

Demonstration:

This demonstration is intended for you to bring together your code from tasks 1 and 2.

1. **25 points:** On the day the assignment is due you will receive a new map *p3_demo1.yaml* a new *landmarks.py* file and a new *victims.py* file that will be used in this demonstration.Your robot will be randomly placed within a known map. It must then drive autonomously through the map attempting to find as many victims as possible. When a victim is found its location [x,y,0] should be output to a csv file called victim_report.csv. Each line of this file should contain one victim location.

Questions:

1. **10 points:** Choose a sensor not currently on the robot (No LIDAR) that would help in a real life search and rescue mission. How would this sensory help the real world robot?
2. **10 points:** Create and explain a heuristic for not reporting the same victim twice.
3. **10 points:** How would you approach this problem if you were not given a map of the environment?
4. **10 points:** Describe a few of the advantages and disadvantages of using a bug algorithm to navigate to a victim.
5. **10 points:** Describe a few of the advantages and disadvantages of using a gradient descent algorithm to navigate to a victim.

Scoring

The score for your demonstration will be determined by the number of victims reported and the accuracy of their reported positions victims. Assuming that there are N victims in the environment, each victim will be worth $\frac{25}{N}$ points. Your score for a reported victim position will be determined by its distance to the closest true victim position such that you lose 0.5 points per 0.1m from the true position. You will lose 2 points for each victim position that shares the same closest true victim position.

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: pa4-*name*, where *name* is replaced by your 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 tasks and demonstrations	50%
Correctness and thoroughness of answers to questions	50%

To get points for thoroughness, you need to explain your answers in a way that demonstrates your knowledge of the technical concepts used to solve the problems.