CS 458 Assignment 2
Motion Model & Landmark Detection

This is the first half of the 2D SLAM project. In this assignment, you will implement the motion model, the landmark detection algorithm from lidar scans, and the landmark matching method. All implementations must be in done Python3 language. Make sure you have NumPy, matplotlib, and tkinter installed in your environment. We provide a supplementary file called helper_functions.py to help you examine and interpret the results.

Environment Setup/Dataset

A turtlebot starting at $(1, 0)$ moves freely inside an arena of size 6×6 with six landmarks. Initially, its heading direction is parallel to the positive x-axis, i.e., the heading angle is 0. The left and right boundary of the arena locates at $x = 0$ and $x = 6$. While the top and bottom boundary sit at $y = 3$ and $y = -3$. For every 0.5 seconds, we gather robot ground truth location, motion reading, lidar scan reading and store them in the following files:

- location.txt - Each line contains two floating numbers, which are the ground truth (x, y) coordinate of the robot. Use this file as a reference to evaluate your result.
- robot_motion.txt - Two floating numbers in each line represent the velocity v in the robot's heading direction and its angular velocity v_θ in radius, respectively.
- lidar_scan.txt - Each line represent a lidar scan composed of multiple lasers at different angles. The first three numbers are the minimal laser angle α_{min}, maximum laser angle α_{max}, and angle difference between adjacent lasers $\Delta\alpha$ in radius. The remaining numbers are the detected distances $d(l_i)$ from the robot to the nearest landmarks/boundaries of all rays starting from α_{min} to α_{max}. Note that the laser angle corresponds to the robot's heading angle instead of the global coordinate.

Question 1: Motion Model

The first question is to reconstruct the robot moving trajectory from the motion model by implementing the following functions:

- location_reader(location file): read location.txt and store the data into a list of tuples
- robot_motion_reader(motion file): read motion.txt and store the data into a list of tuples

- motion_model_calculation(robot motion reading): calculate the robot's location (x^t, y^t) and its heading angle θ^t at each time step t from the robot motion readings. For each time step, store the calculated results into a robot_node class instance defined at the beginning of the provided code. This function should return a list of robot_nodes.

After implementing the motion model, you can visualize the reconstructed trajectory along with the ground truth using the draw_robot_trajectory function.

The report of this section should include the following components:

- math derivation of robot status at (x^t, y^t, θ^t) at any time step t in terms of motion reading (v^t, v_θ^t) and robot status at previous time step $(x^{t-1}, y^{t-1}, \theta^{t-1})$
- visualization of reconstructed robot trajectory from your motion model and the ground truth followed by your findings/interpretations

Question 2: Landmark detection

The next question is to detect landmarks that will further help you refine the robot trajectory. The following functions need to be implemented:

- lidar_scan_reader(robot node list, lidar scan file): read scan.txt and store them into the robot node list. If a laser distance reading is 10.0, it simply means it did not hit anything within its maximum range of 3.5. You will need to change all distance readings of 10.0 into 3.5 for better gradient calculation.
- landmark_detection(robot node list): Now each robot_node instance should contain the corresponding lidar scan data at that time step, this function will access lidar scan data from the robot_node instance. For each lidar scan, first, calculate the distance gradient of each laser $d'(l_i)$ using the following formula

$$d'(l_i) = \frac{d(l_{i+1}) - d(l_{i-1})}{2.0}$$

If a laser hits a landmark, there will be a sudden change in the measured distance, which can be detected by looking for large absolute gradient values. A large negative gradient indicates that the laser starts hitting a landmark while a large positive gradient indicates that the landmark no longer blocks the laser. The average angle of all lasers that hit the landmark, as well as their average distance, defines the location of the landmark corresponding to the robot's current pose. Another transformation needs to be performed to get the landmark location in the global coordinate. After the global coordinates of the detected landmarks are calculated, store them in the landmark attribute of robot_node instances. In this part, use $d'_{threshold} = 0.24$ as the threshold of a large gradient and add an offset of 0.15 to the average laser distance to compensate for the fact that the lasers only reach the surface of the landmarks while we want to calculate the center.

The report of this section should include the following components:

- Choose 3 different time steps, visualize the lidar scan, the gradient plot, and the detected landmarks(add lidar scan here as well). The visualization should be done in global coordinates and feel free to scale coordinates by a constant factor.

Question 3: Landmark matching

In this question, you need to implement an algorithm to match the detected landmarks with the ground truth(provided in help_functions.py) for each time step. The following functions need to be implemented:

- pair_landmarks(robot node list): At a certain time step, for each detected landmark l_i, find the closest ground truth landmark g_j and add pair (i, j) to the matching list. The final pairing list at each time step is a list of tuples where the tuples stores the indices of the match pairs. Store the final pairing list to the pairs attribute of robot node instances. Use a threshold of 1.0 during the matching process, i.e., if the closest ground truth is more than 1.0 away from the detected landmark, then this landmark does not have a matching pair.

The report of this section should include the following components:

- Choose 3 different time steps, visualize the matching pairs by plotting the detected landmarks to the ground truth landmarks using the same color(each pair should have distinct color). The visualization should be done in global coordinates and feel free to scale coordinates by a constant factor.

Submission instructions

Your submission should consist of two items:

- CS458_assignment2.py with all functions mentioned above implemented.
- A PDF report. This report should contain all results discussed above.

  Note: if you have any special explanations of how your code works, or instructions for running it, feel free to place them either in the report or in a separate README file.

Bundle all files into a zip or tarfile, and submit it via Brightspace. Name your submission "<your_username>.zip" or "<your_username>.tar.gz". For example, I would name my submission "zixing_wang.zip".