MTRX5700: Experimental Robotics

Assignment 3

Note: This assignment contributes 15% towards your final mark. This assignment is due on Sunday May 8th, 2022 by 11:59pm. Submit your report via the eLearning TurnItIn submission on the course's Canvas site. Late assignments will be subjected to the University's late submission policy unless accompanied by a valid special consideration form. Plagiarism will be dealt with in accordance with the University of Sydney plagiarism policy. The objective of this assignment is to explore **localisation and mapping** with a focus on laser data. You may use Matlab or Python(help code provided). This assignment should take an average student **15-17 hours** to complete with a passing grade.

Total Marks: 100

Submission: The front page of your report should include the SIDs of your group (do not include your name to comply with the University's anonymous marking policy).
1. First upload a **.zip file** with your videos, code, images, text or data files. The code should be ready to run and have the correct file extensions (.py, .m, .cpp etc.). Please include all the files needed to run your code. Forgetting to upload the code in the .zip file will result in a penalty.
2. Then upload a **PDF of your report.** Please include your code once again in the PDF appendix. We will provide feedback by marking on the PDF directly. Adding your code to the PDF will help us in providing feedback for the code as well.

Data and tips for Q1:

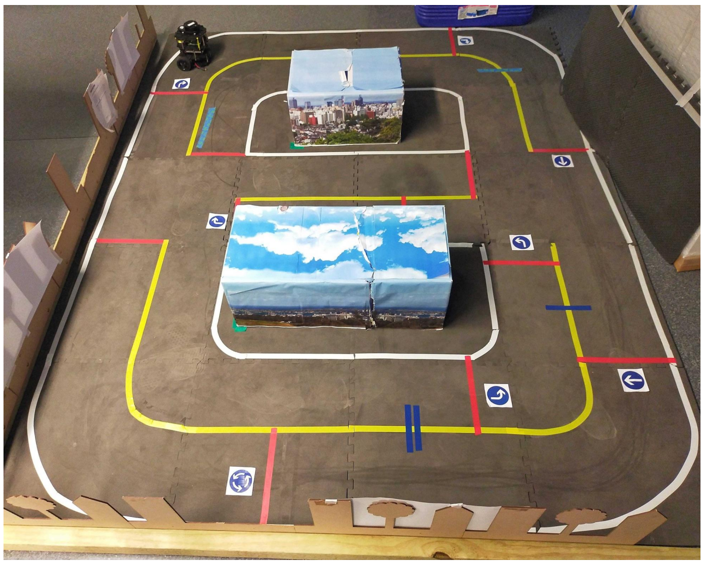

Figure 1. A Picture of Duckietown.

Download the "output.bag" rosbag from Canvas, this rosbag contains a number of messages/topics with the measurements from different sensors mounted on the Turtlebot Burger.

Setup: Install the dependencies

> ➢ sudo apt-get install ros-melodic-csm

For this assignment you will be using the following ROS-messages:

● */scan*	type: sensor_msgs/LaserScan	frame: scan_base
● */tf*	type: tf/tfMessage	frame: odom - base_footprint - scan_base
Optional:		
● */odom*	type: nav_msgs/Odometry	frame: base_footprint
● */imu*	type: sensor_msgs/Imu	frame: base_footprint

Feel free to explore these topics by playing the bag and visualizing them on RVIZ or checking the raw data using `rostopic echo topic_name` or `rqt_bag path_to_bag/output.bag`

Q1 Iterative Closest Point [40 Marks]

The ICP algorithm can be used to register two point clouds which have small offsets.
The ROS bag file we provide output.bag contains 2D LIDAR data acquired using the TurtleBot Burger moving around in the environment shown in Figure 1.
You are tasked to apply the Iterative Closest Point (ICP) algorithm to calculate the pose change transformation between all consecutive locations of the LIDAR. You can implement the ICP registration algorithm yourself or you can use already existing implementations. Concatenate all the transformations and obtain the trajectory of the sensor.

a) [15 Marks] Plot both, the trajectory of the LIDAR obtained using ICP, and the trajectory of the robot obtained using odometry readings. Compare both trajectories, LIDAR and odometry using Absolute Trajectory Error (ATE) and Relative Error (RE). You can learn how to compute these errors in here: http://rpg.ifi.uzh.ch/docs/IROS18_Zhang.pdf (eq. 24 and 27)
b) [10 Marks] Apply a simple maximum distance filter to your scans **and compare the ICP and odometry trajectory again**. Describe any other filtering methods you may use.
c) [5 Marks] Describe very briefly a physical experimental setup which would allow you to estimate the accuracy and precision of the LIDAR sensor (see Figure 2). Note that the LIDAR readings could be expressed as the [x,y] position or the [range,angle] of a return.
d) [10 Marks] Demonstrate the ICP algorithm running while teleoperating the turtlebot in the lab environment. Plot both trajectories (odometry and ICP) in real time. ***Note.*** *The remote students should send the code and ask the tutors to run it on the robot.*

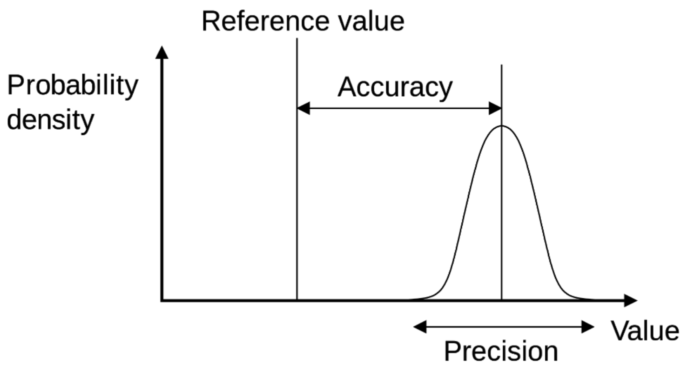

Figure 2: A reminder of accuracy and precision (Wikimedia Commons)

Q2 Simultaneous Localisation and Mapping SLAM [60 Marks]

In this question, you are required to implement a SLAM algorithm on the turtlebot. We are providing a simulated environment for this assignment (see Figure 5.) containing colour coded landmarks. The robot is equipped with Intel real sense, odometer and IMU sensors.

The SLAM algorithm to be implemented is a smoothing algorithm that uses Non-Linear Least Squares to minimize the residual(error) in the prediction of robotic state.

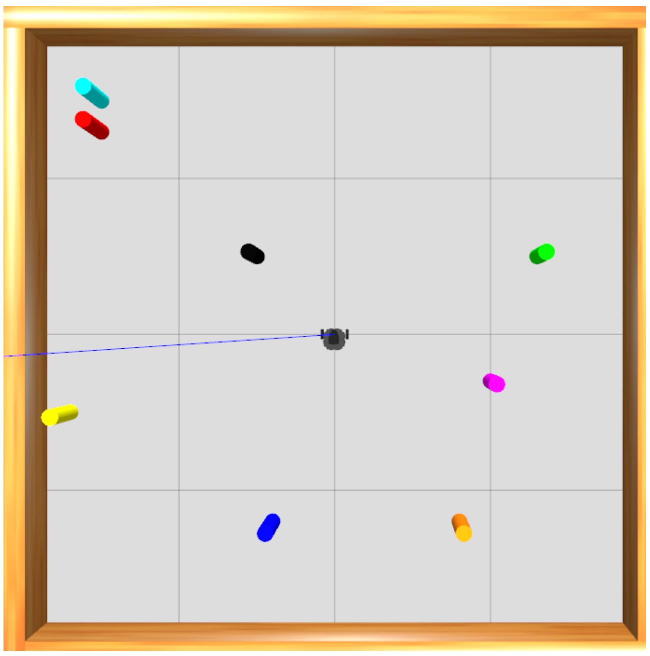
Figure 5. Simulated world with colour coded landmarks.

a) [15 Marks] Get familiar with the help code we provide and with the structure of the skeleton of the Landmark SLAM algorithm implemented in the Python file "**SLAM_students.py**". Here we have provided a unit test of measurements to help you implement and test your slam algorithm. Use this test to validate your system has been constructed correctly, show your final solution for the trajectory of the robot and the map, given this unit test.
b) [15 Marks] Extend your implementation to run on the Turtlebot in the simulated environment (see readme file for how to run simulated environment). Drive the robot around the simulated environment and collect odometry and landmark measurements. At this stage, you are asked to implement a **batch solution** to the SLAM problem. **Report the number of iterations** your solver required to converge and **plot** the estimated trajectory and the landmarks and the corresponding uncertainty for each landmark.
c) [10 Marks] Next run the algorithm in **incremental mode**, where the state of the system increments and updates every step. Drive the robot around the simulated environment and display the trajectory of the robot and the landmarks RVIZ. Record a video of your execution. ***Note:*** *use the video guidelines provided in the Assignment 2.*
d) [10 Marks] Change the values for the standard deviations using the corresponding constants: std_dev_landmark_x, std_dev_landmark_y, std_dev_linear_vel, std_dev_angular_velocity, run the SLAM algorithm and save the solutions (at least three variants). The solutions of your SLAM algorithms should be saved to a

landmark solution file and then compared with a ground truth location of the landmarks.

In the solution file, you will have lines starting with **POINT2D**, then the **landmark ID**, and the **x, y** coordinates of the cylinder in world coordinates. Ground truth landmark positions are provided in the **gt.txt** file but you will have to collect the groundtruth poses form the simulator.

Use **ErrorFunction.py** to evaluate the solution of the SLAM algorithm using different standard deviation values and report and comment the results. Note that here we evaluate the map rather than the robot trajectory.

e) [10 Marks] Write a ROS node capable of processing the Intel real sense RGB image and the corresponding point cloud and extract the landmark measurement and ID required by the SLAM algorithm. Intrinsic camera parameters can be read on the .../camera_info topic. Use the colours of cylinders to identify them. Run your SLAM algorithm using your own cylinder detector. ***Note:*** *that the existing implementation directly obtains this information from the simulator and adds noise to the measurement.*