Project 2: LiDAR-Based SLAM

*Collaboration in the sense of discussion is allowed, however, the assignment is **individual** and the work you turn in should be entirely your own. It is absolutely forbidden to copy or even look at anyone else's code or any code generated by AI. Please acknowledge **in writing** people or AI you discuss the project with. See the collaboration and academic integrity statement here:* https://natanaso.github.io/ece276a.

Submission

Please submit the following files on **Gradescope** by the deadline shown at the top right corner.

1. **Programming assignment**: upload all code you have written for the project (do not include the provided datasets) and a README file with a clear, concise description of the main file and how to run your code.
2. **Report**: upload your report in pdf format. You are encouraged but not required to use an IEEE conference template[1] for your report.

Project Description

In this project, we will implement simultaneous localization and mapping (SLAM) using encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements from a differential-drive robot. Use the odometry and LiDAR measurements to localize the robot and build a 2-D occupancy grid map of the environment. Use the RGBD images to assign colors to the 2-D map of the floor.

- **Robot**: A description of the differential-drive robot is provided in docs/RobotConfiguration.pdf. Fig. 1 shows a schematic of our robot. Even though the description file says that the origin of the robot body frame is at the back axle, it is arbitrary and you can choose your own origin. Using the geometric center of the robot body might yield better results because the differential-drive motion model assumes that the robot is rotating around its center.

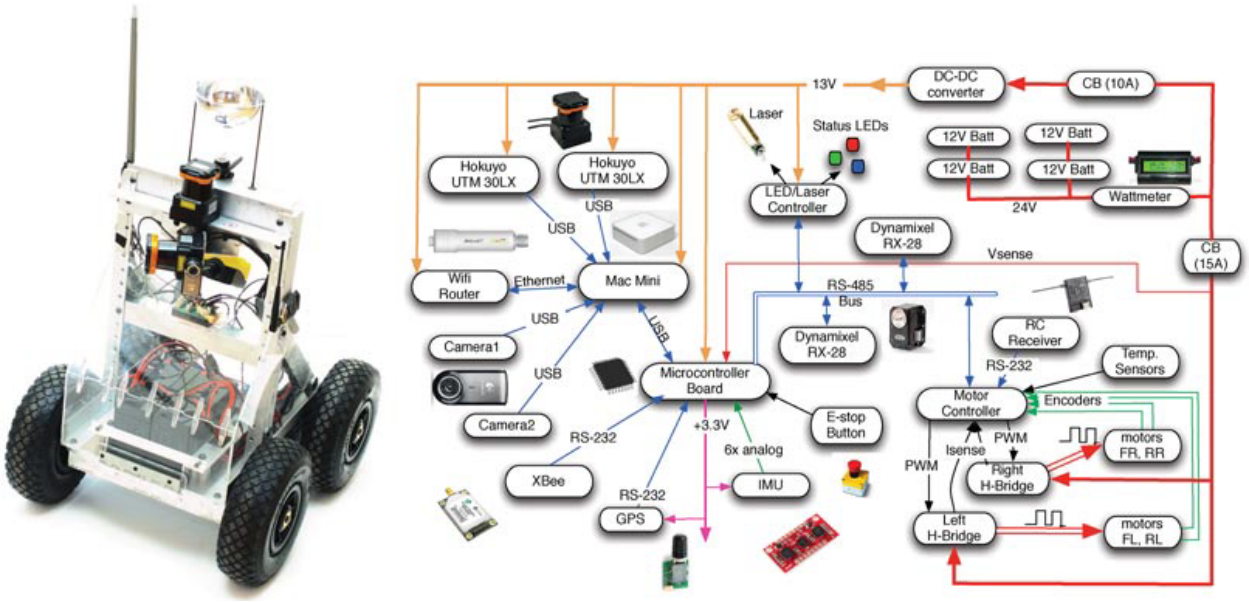

Figure 1: Differential-drive robot, equipped with encoders, IMU, 2-D LIDAR scanner, and RGBD camera.

[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html

- **Sensors**: The robot is equipped with a set of sensors described below. Each sensor reports measurements at its own frequency and, hence, **the sensor data is not synchronized**. The sensor measurements are associated with unix time stamps (seconds), which should be used to sort and associate the measurements.

  - **Encoders**: Encoders count the rotations of the four wheels at 40 Hz. The encoder counter is reset after each reading. For example, if one rotation corresponds to ℓ meters traveled, five consecutive encoder counts of $0, 1, 0, -2, 3$ correspond to $(0 + 1 + 0 - 2 + 3)\ell = 2\ell$ meters traveled for that wheel. The data sheet indicates that the wheel diameter is 0.254 m and since there are 360 ticks per revolution, the wheel travels 0.0022 meters per tic. Given encoder counts $[FR,\ FL,\ RR,\ RL]$ corresponding to the front-right, front-left, rear-right, and rear-left wheels, the right wheels travel distance $(FR + RR)/2 * 0.0022$ m, while the left wheels travel distance $(FL + RL)/2 * 0.0022$ m.
  - **IMU**: Linear acceleration and angular velocity data is provided from an inertial measurement unit. We will only use the **yaw rate** from the IMU as the angular velocity in the differential-drive model in order to predict the robot motion. It is not necessary to use the other IMU measurements.
  - **Hokuyo**: A horizontal LiDAR with 270° degree field of view and maximum range of 30 m provides distances to the obstacles in the environment. Each LiDAR scan contains 1081 measured range values. The sensor is called Hokuyo UTM-30LX and its specifications can be viewed online. The location of the sensor with respect to the robot body is shown in the provided robot description file. Make sure you know how to interpret the LiDAR data and how to convert from range measurements to (x, y) coordinates in the sensor frame, then to the body frame of the robot, and finally to the world frame.
  - **Kinect**: An RGBD camera provides RGB images and disparity images. The depth camera is located at (0.18, 0.005, 0.36) m with respect to the robot center and has orientation with roll 0 rad, pitch 0.36 rad, and yaw 0.021 rad. The intrinsic parameters of the depth camera are:

    $$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}.$$

    You will notice that the timing of the Kinect data is not regular (there are gaps of up to 0.2 sec). This is because the logging software was not able to keep up with all the data and some frames were dropped. You should try to find the closest SLAM pose that matches the time stamp of the current Kinect image. The depth camera and RGB camera are not in the same location. There is an x-axis offset between them and it is necessary to use a transformation to match the color to the depth. Given the values d at pixel (i, j) of the disparity image, you can use the following conversion to obtain the depth and the pixel location $(\text{rgbi}, \text{rgbj})$ of the associated RGB color:

    $$\begin{aligned} dd &= (-0.00304d + 3.31) \\ \text{depth} &= \frac{1.03}{dd} \\ \text{rgbi} &= (526.37i + 19276 - 7877.07dd)/585.051 \\ \text{rgbj} &= (526.37j + 16662)/585.051 \end{aligned}$$

    The **Kinect data** is available at:
    https://ucsdcloud-my.sharepoint.com/:f:/g/personal/natanasov_ucsd_edu/IgAc7jO0VlruT4_8drVPscWYARiI6i60bAeeUxMoIsogywM?e=sJTlFp

- **Software Setup:** We will use the Georgia Tech Smoothing and Mapping (GTSAM) library to perform factor graph optimization of the robot poses. Installation instructions are provided below.

  - **Linux and macOS users:** Open your terminal and run `pip install gtsam` to install GTSAM.

  - **Windows users:** Set up a Linux virtual environment using either Windows Subsystem for Linux (instructions: Install WSL) or a Linux Virtual Machine (consider using VirtualBox: VirtualBox or VMware: VMware). Then, in your Linux environment, run `pip install gtsam` to install GTSAM.

  A python script test_gtsam.py is provided in the **code** folder to test the installation of GTSAM.

- **LiDAR-Based SLAM**: The project consists of 4 parts. In the first part, we use the encoder and IMU measurements to estimate the robot odometry. In the second part, we improve the initial odometry estimates using LiDAR scan matching via the iterative closest point (ICP) algorithm. In the third part, we use the estimated robot trajectory to produce a 2-D occupancy map of the environment from the LiDAR scans as well as a 2-D texture map of the floor using the Kinect RGBD images. Finally, in the fourth part, we will optimize the robot trajectory estimates further by using the GTSAM library with loop-closure constraints. Details for each part follow.

  - [10 pts] **Encoder and IMU odometry**: Use linear velocity v_t obtained from the encoders and yaw rate ω_t obtained from the IMU to predict the robot motion via the differential-drive motion model. Assume the robot starts with identity pose. To verify that your motion model predictions are accurate, plot the robot trajectory. You can also build a complete 2-D occupancy-grid map by combining this trajectory with mapping before implementing scan matching.
  - [20 pts] **Point-cloud registration via iterative closest point (ICP)**: We will use the ICP algorithm to obtain the relative transformations between consecutive LiDAR scans.
    1. **Warm-up**: Implement your own ICP algorithm and test it by estimating object poses from depth images given canonical object models. The code folder provides data from two objects: a drill and a liquid container. Each object has 4 images with different viewing directions. You are provided with functions to read input point clouds and visualize the final results. Please include your final result visualization in your report. *Tips: ICP converges to a local optimum and needs a good initial pose. You may assume that the object only rotates around the z axis and initialize rotations by discretizing the yaw angles and using statistics from the results of different ICP runs (e.g., mean square errors of correspondences) to find the best results.*
    2. **Scan matching**: Once you have the robot odometry estimated using the motion model and the ICP implementation ready from the previous parts, you can implement LiDAR scan matching. Given two consecutive LIDAR scans, use ICP to estimate the 2-D relative pose change between them. Initialize ICP using the odometry estimated from the IMU and encoder measurements. Once the relative pose change between two consecutive robot frames ${}_tT_{t+1}$ is known, the robot pose T_{t+1} at time $t+1$ can be approximated as $T_{t+1} = T_t * {}_tT_{t+1}$. *Note that while in the warm-up you are implementing 3-D ICP, you only need 2-D ICP for scan matching.*
  - [15 pts] **Occupancy and texture mapping**: Use the first LiDAR scan to create and display an occupancy grid map in order to make sure that your transforms are correct. Remove scan points that are too close or too far from the robot. Transform the LiDAR points from the LiDAR frame to the world frame. Use bresenham2D or OpenCV's drawContours[2] to obtain the occupied cells and free cells that correspond to the LiDAR scan. Increase the log-odds of the occupied cells and decrease the log-odds of the free cells in the occupancy-grid map. Once you verify the occupancy grid map for the first LiDAR scan, apply the same procedure to the whole robot trajectory estimated by the ICP algorithm in order to obtain a complete occupancy grid map of the environment.

    Use the RGBD images and the estimated robot trajectory to produce a 2-D color map of the floor texture in addition to the occupancy map. Obtain the depth of each RGB pixel as described in the Kinect description above. Project the colored points from the depth camera frame to the world frame. Find the plane that corresponds to the occupancy grid in the transformed data via thresholding on the height (i.e., 0 m height corresponds to the floor plane). Crate a second

[2] https://docs.opencv.org/4.13.0/d4/d73/tutorial_py_contours_begin.html

grid map with the same resolution as the occupancy grid and color its cells with the RGB values according to the projected points.

- [10 pts] **Pose graph optimization and loop closure**: Enhance the accuracy of the robot trajectory estimation through pose graph optimization with loop closure constraints using GTSAM. Implement odometry factors based on the ICP scan matching and explore methods for loop closure detection to refine the overall trajectory estimation. The refined trajectory can be used to construct more accurate occupancy and texture grid maps.
  1. **Factor graph in GTSAM:** Begin by constructing a factor graph consisting of nodes representing robot poses at different time steps and edges representing relative pose measurements between these poses obtained from ICP scan matching.
  2. **Loop closure detection:** Use the suggested methods below to introduce loop closure factors into the factor graph. Use GTSAM to optimize the factor graph. This optimization process seeks to minimize the discrepancy between the measured relative poses and estimated absolute poses, thus refining the robot trajectory.
     (a) **Fixed-interval loop closure:** After every fixed number of robot poses (e.g., every 10 poses), introduce a loop closure constraint by adding an edge in the factor graph. Use the ICP scan matching algorithm to estimate the relative pose between the two (10-poses-apart) LiDAR scans, ensuring the loop closure is physically plausible.
     (b) **Proximity-based loop closure:** Detect potential loop closures by examining poses that are in close proximity. Compare LiDAR scans at these nearby poses – if the scans match well, this indicates a loop closure. Add an edge in the factor graph to represent this loop-closure constraint. Optionally, you may also consider matching between LiDAR scans and the occupancy map obtained from the prior robot trajectory.
     (c) **Consistency Check:** To ensure the graph optimization is robust, implement a consistency check before adding a loop closure factor. Only add a loop closure factor if the ICP fitness score (mean square error of correspondences) is below a specific threshold.
  3. **Applying the optimized trajectory:** Given the optimized trajectory from GTSAM, reconstruct the occupancy grid map and the texture map. The refined poses allow for more accurate mapping, as they correct cumulative errors in the robot's path estimation.

Project Report

Write a project report describing your approach to the SLAM and texture mapping problems. Your report should include the following sections:

- [5 pts] **Introduction**: Discuss what problem we are solving and why it is important. Present a brief overview of your approach. This section should be short.
- [5 pts] **Problem Formulation**: State the problem you are trying to solve in mathematical terms. This section should be short and clear and should define the quantities you are interested in precisely.
- [20 pts] **Technical Approach**: Describe your technical approach to SLAM and texture mapping.
- [15 pts] **Results**: Present your results, and discuss them – what worked, what did not, and why. Analyze the impact of loop closure detection and pose graph optimization on the accuracy of the robot trajectory estimate and the resulting map quality. Make sure your results include (a) images of the trajectory and occupancy grid map over time constructed by your SLAM algorithm and (b) textured maps over time. If you have videos, include them in your submission zip file and refer to them in your report.