Project 2: Particle Filter 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. See the collaboration and academic integrity statement here:* https://natanaso.github.io/ece276a. *Books, websites, and papers may be consulted but not copied from. It is absolutely forbidden to copy or even look at anyone else's code. Please acknowledge **in writing** people you discuss the project with and provide references for any books or papers you use.*

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.

Problems

In square brackets are the points assigned to each part.

1. 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 your 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.
   - **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, 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 a distance of $(FR + RR)/2 * 0.0022$ m, while the left wheels travel a distance of $(FL + RL)/2 * 0.0022$ m.
     - **IMU**: Linear acceleration and angular velocity data is provided from an inertial measurement unit. The IMU data is noisy, since it is collected from a moving robot with high-frequency vibrations. You should consider applying a low-pass filter (e.g., with bandwidth around 10 Hz) to reduce the measurement noise. We will only use is 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.

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

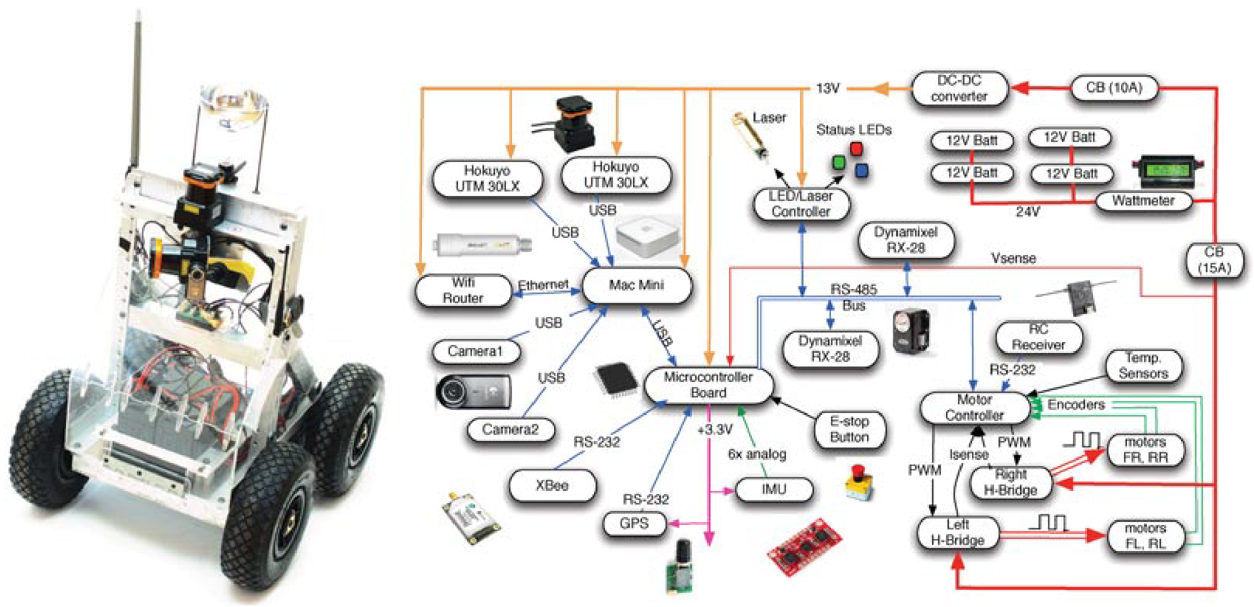

Figure 1: Differential-drive robot, equipped with encoders, IMU, 2-D LIDAR scanner, and an RGBD camera.

– **Hokuyo**: A horizontal LiDAR with 270° degree field of view and maximum range of 30 m provides distances to 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:

$$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05108211 & 0 & 242.94140713 \\ 0 & 585.05108211 & 315.83800193 \\ 0 & 0 & 1 \end{bmatrix}.$$

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 scan. 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:

$$\begin{aligned} dd &= (-0.00304d + 3.31) \\ \text{depth} &= \frac{1.03}{dd} \\ \text{rgbi} &= (526.37i + (-4.5 * 1750.46)dd + 19276.0)/585.051 \\ \text{rgbj} &= (526.37j + 16662)/585.051 \end{aligned}$$

The **Kinect data** is available at:
https://drive.google.com/drive/folders/1_SLyxhnkSIKVsb_cdXW8BEZlQEDXkuRx?usp=share_link

- **Particle-filter SLAM**: The first part of the project is to use a particle filter with a differential-drive motion model and scan-grid correlation observation model for simultaneous localization and occupancy-grid mapping. Here is an outline of the necessary operations:
  - [10 pts] **Mapping**: Assume the robot starts with identity pose and use the first LiDAR scan to create an occupancy grid map. Display the occupancy grid map to make sure that your transforms are correct before you start estimating the robot pose. 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.
  - [10 pts] **Prediction**: 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. To verify that your prediction step is working correctly, implement dead-reckoning first (prediction with no noise and only a single hypothesis for the robot pose) and plot the robot trajectory. You can also build a complete 2-D occupancy-grid map by combining dead-reckoning and mapping before implementing the particle filter. Once you convince yourself that the mapping and prediction operations work correctly, implement the particle-filter prediction step. Introduce N particles at the initial identity robot pose and add noise to the motion model to obtain a predicted pose $\mu^{(i)}_{t+1|t}$ for each particle $i = 1, \ldots, N$. Apply the motion model repeatedly to obtain N separate particle trajectories and visualize them.
  - [10 pts] **Update**: Once the prediction-only filter works, include an update step that uses scan-grid correlation to correct the robot pose. Remove scan points that are too close or too far. Try the update step with only $3-4$ particles at first to see if the weight updates make sense. Transform the LiDAR scan to the world frame using each particle's pose hypothesis. Compute the correlation between the world-frame LiDAR scan and the occupancy map using the mapCorrelation function. Call mapCorrelation with a grid of values (e.g., 9×9) around the current particle position to get a good correlation (see p2_utils.py). You should consider adding variation in the yaw angle of each particle to get good results. Once you obtain the correlation, update the particle weights according to the particle filter equations. If the number of effective particles N_{eff} falls below a threshold, resample the particles.
- [10 pts] **Texture map**: The second part of the project is to use the RGBD images and the estimated robot trajectory to produce a 2D color map of the floor surface. 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 grid map with the same resolution as the occupancy grid and color its cells with the RGB values according to the projected points.

2. 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 why the problem is important and present a brief overview of your approach.
   - [10 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.
   - [30 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. 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.

[2] https://docs.opencv.org/4.7.0/d4/d73/tutorial_py_contours_begin.html