Project 2: Simultaneous Localization and Mapping

Collaboration in the sense of discussion is allowed, however, the work you turn in should be your own - you should not split parts of the assignments with other students and you should certainly not copy other students' solutions or code. See the collaboration and academic integrity statement here: https://natanaso.github.io/ece276a. *Books may be consulted but not copied from.*

Submission

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

1. Theoretical problems: upload your solutions to Problem 1-3. You may use latex, scanned handwritten notes (write legibly!), or any other method to prepare a pdf file. Do not just write the final result. Present your work in detail, explaining your approach at every step.
2. Programming assignment: upload all code you have written for the project (do not include the training and test datasets) and a README file with clear description of each file.
3. Report: upload your report. 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 problem.

1. [26 pts] You lost your green turtle and are not very happy about it. Luckily, you installed the turtle-communication app on your Android and can get some messages from your turtle. Your turtle is not very good at sensing its environment but can tell you (with some noise) how far away it is from the nearest puddle. You have a couple of other handy tools at your disposal – a particle filter from Stanford's Intro to AI course[2] and a Google Maps app showing the city blocks (black squares) and the puddle locations (light blue dots). See Fig. 1 below. How can you use the particle filter to localize your turtle?
   (a) Write the equations for the prior probability density function $p_{0|0}(x_0)$, the turtle motion model $p_f(x_{t+1} \mid x_t, u_t)$, in terms of the turtle position (x, y), orientation θ, and velocity v, and the turtle observation model $p_h(z_t \mid x_t)$, in terms of the puddle positions $b_1, \dots, b_{24} \in \mathbb{R}^2$.
   (b) Write the equations for the particle filter prediction and update steps in terms of the models you specified above. Explain the meaning of these equations. What does each step do, and how are the particle positions and weights updated? How is the gray dot estimate generated based on the particles?
2. [26 pts] You are using a particle filter to track the position of a robot moving along the x-axis. The robot is moving according to the following motion model:

$$p_f(x_{t+1} \mid x_t, u_t) := \begin{cases} x_t + u_t & \text{with prob. } 1/2 \\ x_t + u_t + \mathbf{sgn}(u_t) & \text{with prob. } 1/3 \\ x_t + u_t - \mathbf{sgn}(u_t) & \text{with prob. } 1/6 \end{cases} \qquad \mathbf{sgn}(u) := \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{if } u = 0 \\ -1 & \text{if } u < 0 \end{cases}$$

   Your sensor is described by the following observation model:

$$p_h(z_t \mid x_t) = \begin{cases} x_t & \text{with prob. } 1/2 \\ x_t + 1 & \text{with prob. } 1/4 \\ x_t - 1 & \text{with prob. } 1/4 \end{cases}$$

   You model the prior distribution over the robot position x_0 using 5 equally weighted particles with positions $\{1, 2, 3, 4, 5\}$. You receive the following observations $z_0 = 2$, $z_1 = 3$, $z_2 = 3$ and also know that

[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html
[2] https://github.com/mjl/particle_filter_demo

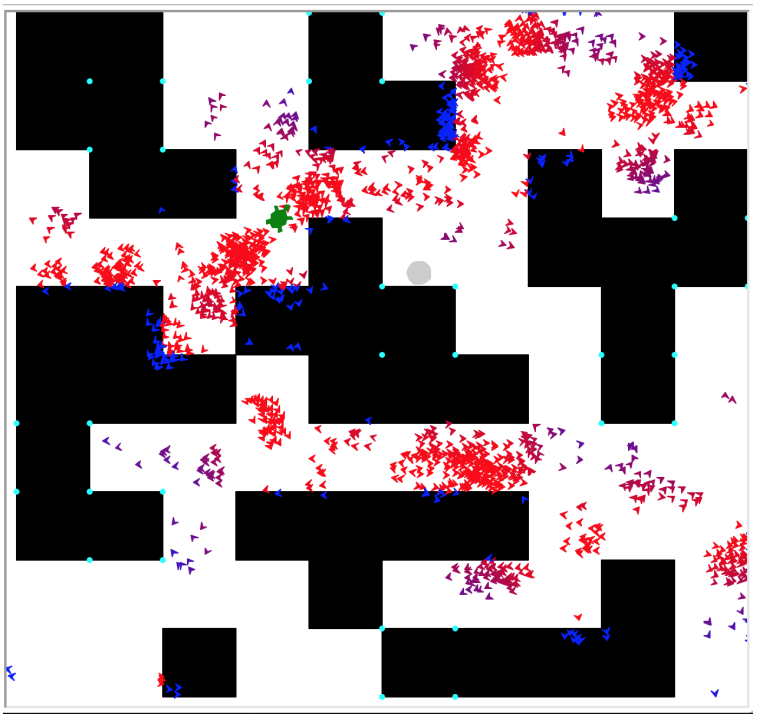

Figure 1: Google Maps view of the city and your particle filter. The black squares are buildings, the light blue dots are puddles. The position of your green turtle is unknown. The gray dot is your particle filter's estimate.

the robot applies the controls $u_0 = +1$, $u_1 = -1$, $u_2 = +1$. Evaluate the posterior probability density function $p(\cdot \mid z_{0:2}, u_{0:2})$ at $x_3 = 4$ when (a) no resampling is used and (b) stratified resampling is used after every update step with $u = \frac{1}{2N}$ in line 5 of slide 28 of Lecture 6. Is it better to use or not use resampling in this example?

3. [26 pts] Consider a picture frame, represented by four line segments with vertices $\mathbf{0}, \mathbf{e}_1, \mathbf{e}_3, (\mathbf{e}_1 + \mathbf{e}_3) \in \mathbb{R}^3$, where $\mathbf{e}_i$ is the ith standard basis vector. Suppose that the picture frame is moving with constant velocity $\mathbf{e}_1$ along the x-axis. Your robot is observing the picture frame with a camera located at position $p \in \mathbb{R}^3$ and orientation, specified by a quaternion q (Hamilton convention):

$$p = \begin{pmatrix} -\frac{5\sqrt{2}}{2} \\ \frac{5\sqrt{2}}{2} \\ 3 \end{pmatrix} \qquad q = \begin{pmatrix} \cos(\frac{45^\circ}{2})\cos(-\frac{30^\circ}{2}) \\ -\sin(\frac{45^\circ}{2})\sin(-\frac{30^\circ}{2}) \\ \cos(\frac{45^\circ}{2})\sin(-\frac{30^\circ}{2}) \\ \sin(\frac{45^\circ}{2})\cos(-\frac{30^\circ}{2}) \end{pmatrix}$$

   (a) Compute the projections of the 4 picture frame vertices in the image plane.

   (b) Compute the velocities of the 4 projected points in the image plane. Which point appears to move fastest?

   (c) Compute the projected lengths of the 4 line segments.

   (d) Compute the rate of change of these projected lengths. Are they growing or shrinking? Which projected line segment is changing length the fastest?

4. [140 pts] Implement simultaneous localization and mapping (SLAM) using odometry, inertial, 2-D laser range, and RGBD measurements from a differential-drive robot. Use the IMU, odometry, and laser measurements to localize the robot and build a 2-D occupancy grid map of the environment. Use the RGBD information to color the floor of your 2-D map.

- The robot configuration is provided in RobotConfiguration.pdf. Fig. 2 shows a schematic of our robot. Even though the configuration says that the origin of the robot is at the back axle, it is arbitrary and you can choose your own origin. Using the geometrical center of the robot would yield better results since the differential drive motion model assumes that the robot is rotating around its center.

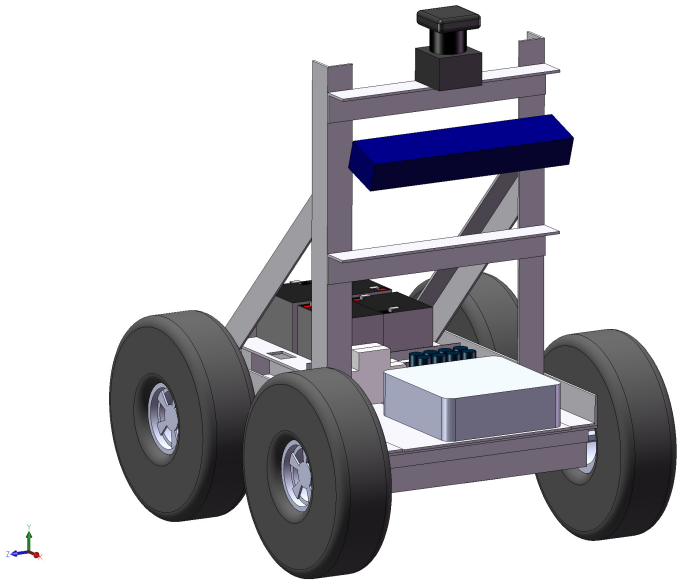

Figure 2: Rendering of a differential drive robot, equipped with encoders, IMU, 2-D LIDAR scanner, and an RGBD camera.

- The robot is equipped with the following sensors:
  - **Encoders**: The ground robot is equipped with encoders which 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 odometer counts $[0, 1, 0, -2, 3]$ correspond to $2 \times l$ meters traveled for that wheel. The data sheet indicates that the wheel diameter is 0.254m and since there are 360 tics per revolution, the wheel travels 0.0022 meters per tic. Given encoder counts $[FR,\ FL,\ RR,\ RL]$ corresponding to the four wheels, the right wheels travel a distance of $(FR + RR)/2 * 0.0022$, while the left wheels travel a distance of $(FL + RL)/2 * 0.0022$.
  - **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 a lot of high frequency vibrations. You should consider applying a low-pass filter with the highest bandwidth that still gives you reasonable performance in terms of rejecting noise (e.g., a first order filter with bandwidth around 10Hz). The only imu information that we will use is the **yaw rate** in order to predict the robot motion using the particle filter. It is not necessary to use the other imu measurements.
  - **Hokuyo**: a horizontal LIDAR with 270° degree field of view and maximum range of 30m provides distances to obstacles in the environment. Each LIDAR scan contains 1081 measured ranges. The sensor is called Hokuyo UTM-30LX and its specifications can be viewed online. Location of the sensor with respect to the robot body is shown in the provided configuration pdf. Make sure you know how to interpret the lidar data and how to convert from ranges to (x, y) coordinates in the sensor frame, and then to the body frame of the robot.
  - **Kinect**: a RGBD camera provides both RGB images and disparity images. The camera is located at $(0.18,\ 0.005,\ 0.36)$m with respect to the robot center and has orientation $roll = 0$rad, $pitch = 0.36$ rad, and yaw 0.021 rad. 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 hence it is necessary to use a transformation to match the color to the depth. Given the values d at location (i, j) of the disparity image, you can use the following mapping to obtain the depth and associated color:

$$dd = (-0.00304 * d + 3.31)$$
$$\textbf{depth} = \frac{1.03}{dd}$$
$$rgbi = (i * 526.37 + dd * (-4.5 * 1750.46) + 19276.0)/585.051$$
$$rgbj = (j * 526.37 + 16662.0)/585.051$$

The **Kinect data** is available at:
https://drive.google.com/open?id=1_SLyxhnkSIKVsb_cdXW8BEZlQEDXkuRx

- The goal of the projects is to use an occupancy grid for mapping and a particle filter with a laser-grid correlation model for localization. Here is an outline of every necessary operation:
  - **Mapping**: Try mapping from the first scan and plot the map to make sure your transforms are correct before you start estimating the robot pose.
  - **Prediction**: Implement a prediction-only particle filter at first. In other words, use the wheel odometry and the yaw gyro measurements to estimate the robot trajectory and build a 2-D map based on this estimate before correcting it with the laser readings.
  - **Update**: Once the prediction-only filter works, include an update step that uses scan matching to correct the robot pose. Remove scan points that are too close, too far, or hit the ground.
  - **Texture map**: Project colored points from the RGBD sensor onto your occupancy grid in order to color the floor. Find the ground plane in the transformed data via thresholding on the height.
- Below you can also find detailed tips for every step.
  - Test each capability individually first:
    * Try mapping from the first laser scan and plot the map
    * Try dead-reckoning (using only prediction steps with no noise and a single particle) and plot the robot trajectory
    * Try prediction only and plot the robot trajectories (e.g., 100 for $N = 100$ particles)
    * Try the update step with only 3-4 particles and see if the weight update makes sense
  - **Mapping**:
    * Input:
    (a) Current log-odds map m_{t-1}
    (b) Laser scan z_t
    (c) Transform from lidar frame to body: ${}_bT_l$
    (d) Robot pose x_t (determines a body-to-world transform: ${}_wT_b$) (use the current best particle)
    * Output:
    (a) Updated log-odds map m_t
    * Pseudo-code:
    (a) Remove scan points that are too close or too far
    (b) Transform z_t via ${}_wT_{bb}T_l$ to the world frame
    (c) Use bresenham2D or cv2.drawContours to obtain the cell locations y_t^o that are occupied according to the laser and y_t^f that are free according to the laser
    (d) Increase the log-odds in m_{t-1} of the occupied cells y_t^o and decrease the odds of the free cells y_t^f to obtain m_t
  - **Localization Prediction**: use the motion model with input from the encoder measurements and the IMU yaw rate and additive noise to compute an updated pose $\mu_{t+1|t}^{(i)}$ for each particles $i = 1, \ldots, N$

- **Localization Update**:
  - Input:
    (a) Current particle positions and weights: $(\mu_{t+1|t}^{(i)}, \alpha_{t+1|t}^{(i)}), i = 1, \ldots, N$
    (b) Laser scan z_{t+1}
    (c) Current map m_t
    (d) Transform from lidar frame to body: ${}_bT_l$
  - Output:
    (a) Updated particle positions and weights: $(\mu_{t+1|t+1}^{(i)}, \alpha_{t+1|t+1}^{(i)}), i = 1, \ldots, N$
  - Pseudo-code:
    (a) For each particle $i = 1, \ldots, N$:
      · Remove scan points that are too close or too far
      · Transform z_t via ${}_wT_b{}_bT_l$ to the world frame, where ${}_wT_b$ is determined from $\mu_{t+1|t}^{(i)}$ to obtain z_t^{world}
      · Compute $corr(m_t, z_t^{world})$ using the mapCorrelation. Call mapCorrelation with a grid of values (e.g., 9x9) around the current particle position to get a good correlation. See map_utils.py for an example variation in x and y. You should also consider adding variation in yaw for the particle.
    (b) Update the particle weights according to the particle filter equations
    (c) If $N_{eff} < N_{threshold}$, resample the particles
- **Texture Mapping**:
  - Input:
    (a) RGB image I_t and depth image D_t
    (b) Transform from ir to body: ${}_bT_i$ and rgb to body ${}_bT_r$
    (c) Robot pose x_t determining transform from body to world: ${}_wT_b$ (current best particle)
    (d) Current texture map: tm_t
  - Output:
    (a) Updated texture map: tm_{t+1}
  - Pseudo-code:
    (a) Transform I_t and D_t via x_t and ${}_bT_i$ and ${}_bT_r$
    (b) Find the ground plane in the transformed data via thresholding on the height
    (c) Color the cells in tm_t using the points from I_t and D_t that belong to the ground plane

5. [32 pts] Write a project report describing your approach to the SLAM and texture mapping problems. Your report should include the following sections:
   - **Introduction**: discuss why the problem is important and present a brief overview of your approach
   - **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.
   - **Technical Approach**: describe your approach to SLAM and texture mapping
   - **Results**: present your training results, test results, and discuss them – what worked, what did not, and why. Make sure your results include (a) images of your SLAM system over time, (b) textured maps of the training and test datasets. If you have videos do include them in the zip file and refer to them in your report!