Introduction

In this project, I implemented a Simultaneous Localization and Mapping (SLAM) algorithm for environment mapping using robot wheel odometry and Lidar data. Reproducible code is available as a git repository

Problem Statement

We are given time-series measurements of wheel odometry data and LIDAR scan data, as well as geometric measurements of the robot. The goal is to map the robot's trajectory while mapping the environment (empty space v.s. obstructions) on a 2D plane.

Data

The dataset consists of physical measurements of the robot, LIDAR scan data, and wheel odometry data of a 4-wheeled robot. - Physical measurements: refer to the spec sheet for details. - wheel diameter: 254 mm - wheelbase: 330.2 mm - distance between wheels (inner walls): 311.15 mm - distance between wheels (outer walls): 476.25 mm - Odometry measurements: each wheel contains an encoder which measures the wheel rotation. Each data entry is the sum of encoder ticks since the previous timepoint (positive value -> net forward motion). Each tick is 1/360 revolution. The encoder data has the following channels: - time: `int` type - FR: front right (ticks) - FL: front left (ticks) - RR: rear right (ticks) - RL: rear left (ticks) - Lidar measurements: A Lidar is mounted on the robot and continously scans a cone of -135 to 135 degrees in front of the robot. When the light beam hits an obstable, it bounces back to the detector and a distance is calculated. The Lidar data has the following channels: - time: `int` type - scan: `array` of distances of detected obstacles (meters) - angle: `array` of angles correspond to the scans (radians)

Approach

Here I implement two approaches: the naive (odometry only) approach and SLAM.

In the naive approach, I use dead reckoning to construct a trajectory of the robot. The limitation of this approach is its sensitivity of small errors, e.g. when the robot slips or small deviations in robot dimension parameters. Since the position update is relative, small initial errors can accumulate and produce large deviations in final trajectory.

SLAM samples from Gaussian noise to create a large number of particles with independently evolving trajectories. A consensus map is constructed taking a weighted sample of the particles based on the agreement of their mapping and the previous consensus map. The sampling procedure can be thought of as simulating a range of robot parameters, making SLAM robust to error.

1. Dead Reckoning

For each time point, we get the incremental distance traveled by a wheel by converting encoder ticks to distance d (mm) via $d = 2\pi(\frac{ticks}{360})$. To reduce the effect of slippage, we take the average of the front and back wheels to get the distance traveled by the left and right wheels

$$Left = \frac{FL + RL}{2} \tag{1}$$

$$Right = \frac{FR + RR}{2} \tag{2}$$

Refer to the visualization of discretized IMU signals in the appendix

2. Hidden Markov Mode

HMM is a probabilistic model over sequential data of arbitrary length. To efficiently handle arbitrary sequence lengths, the model uses the Markov assumption: the future state conditioned on the current state is independent of all past states. Simply put, whatever relevant information from the past for predicting the future is fully manifested in the present.

The HMM uses hidden states as internal representations for the data space. The model is parameterized by a transition matrix A, emission matrix B, and initialization probability π. $A \in R^{n_{hidden} \times n_{hidden}}$ parameterizes the transition probability between hidden states, where A_{ji} is the probability that state i transitions to state j. $B \in R^{n_{hidden} \times n_{clusters}}$ parameterizes the mapping between hidden states and observed states, where B_{jk} is the probability that hidden state $j \in n_{hidden}$ emits observed state $k \in n_{clusters}$.

My implementation makes several adjustments to the vanilla HMM: 1. Scaling to prevent underflow: Over arbitrarily long sequence, the probability of observing the sequence under the model $p(X|\Lambda)$ becomes small and runs into underflow. To address this, I use the scaling method introduced by Rabiner in Section V of the paper. Briefly, a scaling factor is calculated for every timestep to normalize $p(X|\Lambda, \pi)$ to 1, and the scaling factor is stored for calculating log-likelihood.

2. The initialization probability π is NOT learned. For a given class of motion, the corresponding HMM always initializes at the first hidden state. This is a valid simplifying assumption *as long as the experimental procedure always start the measurement in the same way within a type of motion.* Inconsistencies between classes of motion will not affect the results since the models are trained independently.

3. A left-to-right transition matrix A is enforced through the following initialization, which enforce a forward progression through the hidden states and helps convergence. Transition from the last to the first state is allowed to account for the motion repeating multiple times in one training instance:

$$\mathbf{A_0} = \begin{pmatrix} 0.9 & 0 & \cdots & 0.1 \\ 0.1 & 0.9 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0.9 \end{pmatrix}$$

To train the HMM, all training data for each category of motion is concatenated (training the same model n times and concatenating all n datapoints is exactly the same sequence of operations). The transition probability from the final to first hidden state takes care of the repeated motion introduced by concatenation. Each class of motion data trains it's own HMM, and each HMM does not see any other class of motion during training.

All models trained to convergence, refer to the loss curves in the appendix

3. Classification

Results

Test dataset: TODO: add images for odometry only TODO: add images for SLAM

Train split: TODO: add images

Validation split: TODO: ADD Images

Appendix