LiDAR-Based SLAM with Factor Graph Optimization

Jay Paek
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, California
jpaek@ucsd.edu

***Abstract*—This project addresses the challenges of Simultaneous Localization and Mapping (SLAM) and texture mapping in robotics. Using a differential-drive robot equipped with encoders, an Inertial Measurement Unit (IMU), a 2-D LiDAR scanner, and an RGBD camera, we aim to accurately estimate the robot's pose and construct detailed maps of the environment while also generating texture maps of the floor. Our approach involves integrating sensor measurements and control inputs to predict the robot's motion, refining pose estimates through LiDAR scan matching, and enhancing trajectory estimation via pose graph optimization with loop closure constraints. Additionally, we utilize RGBD images to generate textured representations of the floor, providing richer environmental understanding. Through these methods, we enable autonomous navigation and enhance perception capabilities in unknown environments, paving the way for applications such as autonomous exploration, surveillance, and navigation in indoor settings.**

***Index Terms*—Optimization, robotics, sensor fusion, SLAM, factor graph optimization, occupancy mapping, pose estimation.**

I. Introduction

Autonomous navigation in unknown environments is a fundamental challenge in robotics, with applications ranging from autonomous vehicles to mobile robots in indoor environments. Simultaneous Localization and Mapping (SLAM) is a key technique that addresses this challenge by enabling robots to simultaneously localize themselves while building a map of their surroundings. Additionally, generating texture maps of the environment enhances the robot's perception capabilities, facilitating tasks such as object recognition and navigation.

In this project, we focus on implementing SLAM and texture mapping using a differential-drive robot equipped with various sensors, including encoders, an Inertial Measurement Unit (IMU), a 2-D LiDAR scanner, and an RGBD camera. Our objective is to leverage data from these sensors to accurately estimate the robot's pose and construct a detailed map of the environment. Furthermore, we aim to generate a texture map of the floor using RGBD images captured by the robot's camera.

Our approach involves:

- Integrating sensor measurements and control inputs to estimate the robot's trajectory via Euler's method
- Perform LiDAR scan matching to refine pose estimates via ICP matching with the Kabsch algorithm
- Employ pose graph optimization with loop closure constraints to enhance trajectory estimation using George Tech Smoothing and Mapping (GTSAM) library
- Create an occupancy map using optimized trajectories and LiDAR points using Bresenham ray 2D ray tracing algorithm incorporating log-odd probabilities of occupancy in the grid cells.
- Project RGBD images to the world frame with camera projection matrix and depth values to generate a texture map of the floor, providing a richer representation of the environment

II. Problem Formulation

Our goal is to estimate the robot's trajectory and the map of the environment given sensor measurements and control inputs. We will use the odometry and LiDAR measurements to localize the robot and build a 2-D occupancy grid map of the environment. Then we will use the RGBD images to assign colors to the 2-D map of the floor.

Let $\mathbf{x}_{1:T} \in \mathbb{R}_3$ be the positions of the robot at discrete time steps, where the first two entries are the 2D positions of the robot while the last entry is the direction it is facing in radians i.e. $\begin{bmatrix} x_t & y_t & \theta_t \end{bmatrix}$. $\mathbf{x}_0$ will be $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. Let $\mathbf{z}_{1:T} \in \mathbb{R}^{1081}$ be the LiDAR readings at the discrete time steps.

Before any usage of probabilistic models, the initial predictions of the robot states need to be optimized. We are given the following data:

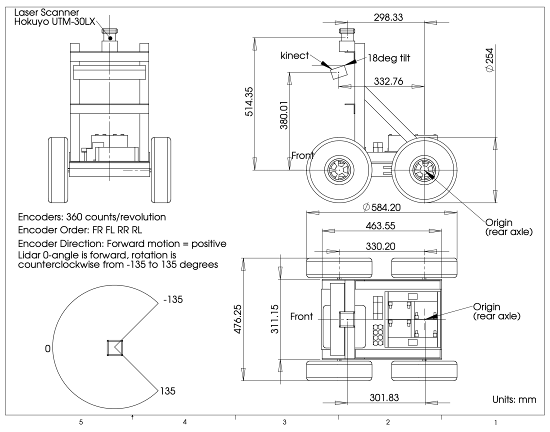
- Motor encoder data that measures how much rotation the front-left, front-right, back-left, back-right wheels underwent, along with the time stamps of their readings. Positive ticks represent forward motion, and negative ticks represent backwards motion, where there are 360 ticks per revolution.
- Accelerometer and gyroscope data data given by the IMU. No calibration is needed since data is given in gravity units and radians per second repsectivelty. Since the robot is moving in two dimensions, only the yaw rate is needed. Due to the differential drive kinematis of the robots, the acceleration will not be used.
- Horizontal LiDAR readings from a Hokuyo UTM-30LX with 270 degree field of view: from -135 degrees to 135 degrees with a valid range of 0.1 to 20 meters.

- RGBD images from a Kinect. The projection matrix of the depth camera are:

$$K = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}$$

All of the data are associated with individual UNIX time stamps in seconds. It is crucial to note that none of the data are synchronized, so it is important to formulate an algorithm to make sure the data are associated with their respective measurements.

The physical configurations are given in Fig. 1



Laser Scanner
Hokuyo UTM-30LX
kinect
18deg tilt
298.33
332.76
514.35
380.01
∅254
Front
∅584.20
463.55
330.20
Origin
(rear axle)
Encoders: 360 counts/revolution
Encoder Order: FR FL RR RL
Encoder Direction: Forward motion = positive
Lidar 0-angle is forward, rotation is counterclockwise from -135 to 135 degrees
-135
0
135
476.25
311.15
Front
Origin
(rear axle)
301.83
Units: mm
5
4
3
2
1

Fig. 1: Robot's physical configurations

Using the noisy IMU and encoder data, we will create an initial estimate for $\mathbf{x}_{1:T}$. Then with the given $\mathbf{x}_{1:T}$, we will use $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ to transform the closest time LiDAR readings to create two point clouds in $\mathbb{R}^2$. Then we will perform iterative closest point (ICP) matching between the two point clouds to improve the estimations for $\mathbf{x}_{1:T}$. Using the optimized values, we will use the LiDAR scans once more in order to create an occupancy grid map. As a final touch, we will use the Georgia Tech Smooth and Library in order to perform factor graph optimization $\mathbf{x}_{1:T}$u using the improved $\mathbf{x}_{1:T}$ and LiDAR observations $\mathbf{z}_{1:T}$.

For texture mapping, the goal is to generate a 2-D color map of the floor texture using RGBD images captured by the robot's camera. This involves associating RGB values with depth information to create a textured representation of the environment. Using the projection matrix of the depth camera, we will identify which points of each photo at time step t represent the floor. Then these selected pixels will be used to color the grid map accordingly.

III. Technical Approach

A. Odometry Estimation

In order to synchronize between two data sets, we first determine which dataset has more points. Without loss of generality, let dataset 1 and dataset 2 have timestamps t_i, t'_j respectively. Additionally let dataset 1 have T_1, T_2 total points respectively.

Algorithm 1 Data synchronization

```
$i \leftarrow 1, j \leftarrow 1$
**for** $i = 1, \ldots, T_1$ **do**
    **if** $t_i > t'_j$ **then**
        Associate data $t_i$ and $t'_j$, $j$++
    **end if**
**end for**=0
```

We begin by utilizing encoder and IMU measurements to predict the robot's motion. The differential-drive motion model describes the robot's motion based on linear and angular velocities. Given the encoder counts and IMU readings, we employ this model to predict the robot's trajectory, assuming a constant linear velocity and angular velocity between consecutive time steps. Given FR, FL, BR, BL, the front-right, front-left, back-right, back-left motor encoder values, at any time step t, the we can define the approximate linear translation of the robot

$$d_t := 0.022\frac{FL + FR + BL + BR}{4}$$

This equation is derived from the wheel diameter and the fact that each tick of the encoder corresponds to a $\frac{\pi}{180}$ radian turn of the wheel. Then the encoder values are averaged for the right wheels and the left wheels, then with the averaged wheel turns on either side, they are multiplied by the distance traveled by one tick.

With v_t and $\boldsymbol{\omega}_t$, which is already given by the IMU, we find an iterative method to approximate the poses of the robot over time. Initially, the differential drive model admits the following equation

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \tau_t \begin{bmatrix} v_t \cos(\theta_t) & v_t \sin(\theta_t) & \omega_t \end{bmatrix}^T$$

where τ_t is the time stamp increment between t and $t-1$. However, realizing that $\tau_t v_t$ is just linear displacement, we the following expression.

$$= \mathbf{x}_t + \begin{bmatrix} d_t \cos(\theta_t) & d_t \sin(\theta_t) & \tau_t \omega_t \end{bmatrix}^T$$

B. LiDAR Scan Matching

Next, we refine the initial odometry estimates using LiDAR scan matching. The LiDAR sensor provides distance measurements to obstacles in the environment, allowing us to generate 2-D scans of the surroundings. We employ the Iterative Closest Point (ICP) algorithm to align consecutive LiDAR scans and estimate the relative pose change between them.

ICP iteratively aligns two point clouds by minimizing the distance between corresponding points. By comparing consecutive LiDAR scans, we can estimate the transformation that best aligns them, thereby improving the accuracy of our pose estimates. Prior to any rotation adjustments, we will translate the the source points by $\mu_t - \mu_s$, where μ_t, μ_s denote the centroid of the point clouds respectively. Let s_i, t_i be the source points and closest target point to s_i, respectively.

There will be two steps for each iteration. First, we will use scipy's KDTree to find the nearest neighbor to each of the source points to create a point association between the source points and the target points. Then, to numerically compute such transformations for each iteration, we will use the Kabsch Algorithm. This algorithm is suitable for such situation because the current optimization problem is as follows:

$$\min_{R\in SO(3)} \sum_{n=1}^{T} \|(t_i - \mu_t) - R(s_i - \mu_s)\|_2^2$$

However, with some rearrangement, the optimization problem admits the following form

$$\min_{R\in SO(3)} Q^T R$$

Where $Q = \sum_{n=1}^{T}(t_n - \mu_t)(s_i - \mu_s)^T$. The Kabsch Algorithm offers a solution for R given the the svd of Q. After finding $Q = U\Sigma V^T$, the optimal R is the following

$$U \begin{bmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \det(UV^T) \end{bmatrix} V^T$$

Then the source points will be rotated appropriately, and the transformed source points will undergo another iteration of closest point matching and the Kabsch algorithm. However, due to numerical inaccuracies of a computer, too many iterations could deform R out of $SO(3)$. Hence, the columns of R will be normalized at each step. All ICP matching methods are are iterated over 20 iterations.

In summary, this is the algorithm, where $R = I$ intially:

```
Algorithm 2 ICP
for $i = 1, \dots, 20$ do
    match closest $t_n \to s_n$ using KDTree
    $U\Sigma V^T \leftarrow svd(\sum_{n=1}^{T}(t_n - \mu_t)(s_n - \mu_s)^T)$
    $R \leftarrow RU \begin{bmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \det(UV^T) \end{bmatrix} V^T$
    Normalize columns of $R$
end for=0
```

Then we construct $T = \begin{bmatrix} R & \mu_t - \mu_s \\ 0 & 1 \end{bmatrix}$, and return it.

C. Factor Graph Optimization

To further enhance the accuracy of our trajectory estimation, we employ pose graph optimization with loop closure constraints. Pose graph optimization involves constructing a graph where nodes represent robot poses at different time steps, and edges represent relative pose measurements between these poses obtained from sensor data.

We use the GTSAM library to formulate and solve the pose graph optimization problem. By incorporating loop closure constraints into the graph, we refine the trajectory estimation and correct cumulative errors in the robot's path. We configure the factor graph with $0.1I$ covariance for all observations and positions. Each node will represent a position, and the edges between node t and t' is the transformation from t and t' in the same form as $\mathbf{x}$.

First, we will initialize the graph with the LiDAR optimized model, by inputting the transformations for each t to $t+1$. Next, we will compute the transformation from t to $t+5$, and then add another edge from the tth node to the $t+5$th node, for every fifth node. This adds a loop closure, and ensures that there is not too much discrepancy between the LiDAR measurements up from t to $t+5$ and t directly to $t+5$.

Finally, we will initialize the graph with the LiDAR optimized model, then use the Gauss-Newton optimizer to achieve convergence for the following optimization problem roughly in this form, where E is the edge set.

$$\mathbf{x}^*_{1:T} := \underset{\mathbf{x}_{1:T}\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{(i,j)\in E} \|\mathbf{x}_j -_j T_i \mathbf{x}_i\|$$

D. Occupancy Grid

To create the initial map, given position coordinate (x_t, y_t) and orientation θ_t, we need to compute the lidar points in the world frame. Create a vector $\phi \in \mathbb{R}^{1081}$ such that the ith entry denotes the direction of the ith entry of LiDAR measurement, which should be $-2.35619449 + (i-1)0.00436332$. Let $\oplus$ and $\otimes$ denote the element wise addition and multiplication of the smaller element to each row of larger element, respectively. Let $l \in \mathbb{R}^{1081}$ be the LiDAR distance reading in meters, then LiDAR readings in the world coordinates, l', are:

$$\begin{bmatrix} x_t & y_t \end{bmatrix} \oplus (l \otimes \begin{bmatrix} \cos(\phi + \theta_t) & \sin(\phi + \theta_t) \end{bmatrix})$$

To build the occupancy grid, we consider the LiDAR rays extending from pose $\mathbf{x}_t$ at every time step t. In grid cells where there is a LiDAR reading, we increase the probability that some object is there blocking the view. In grid cells where the ray passes through, the log odds of occupancy is decreased. For any cells where the robot passes through, the occupancy is immediately forced to 0. We would use the given bresenham ray tracing algorithm in order to determine which cells to increment or decrement the odds of occupancy in each cell.

E. Texture Mapping

Finally, we utilize RGBD images captured by the robot's camera to generate a texture map of the floor. RGBD images provide both color and depth information, allowing us to associate RGB values with depth measurements and project colored points onto the floor surface.

We employ geometric transformations to project points from the camera frame to the world frame, ensuring accurate alignment with the robot's trajectory. By combining RGB values with depth information, we create a textured representation of the floor, enhancing the robot's perception capabilities.

Projection of a point from the world frame to the image frame is simple. The pixel coordinates x, y given the observation coordinates $X, Y, Z \in \mathbb{R}$ in the image frame is

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Uusually, the depth coordinate is lost during projection, but given the depth values of each pixel, it is possible to solve the inverse problem and retrace the pixels in the world coordinates.

First, we must find the depth from the disparity image. Let d be the disparity for pixel coordinate x, y

$$Z = \frac{1.03}{0.00304d + 3.31}$$

Now given the depth, we just solve for the image frame coordinates, since K is invertible.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = ZK^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

It is crucial to convert the coordinates back to the world frame, which is just switching the entries. Then translate the points given that the KiNect is tilted downwards by 18 degrees and translated from the center of the robot. Lastly, pick out the points with y coordinates less than 0.15 in the world frame. Associate these floor images with the positions using the time stamps. Paste these points to the map given the position.

IV. Results for ICP

The following photos are the ICP algorithm tested on points clouds of a bottle and drill. The blue represents the source point cloud, and the red represents the target point cloud. Our goal is to find the transformation matrix $T \in \mathbb{R}^{4\times4}$ which transforms the source point cloud to the target.

The matchings are done fairly well given the fact that the target point clouds are 2.5D. Since the mean of the point clouds are offset due to skewness of the points, it very hard to match perfectly when using a standard algorithm. However, the algorithm performed well adjust two of the three axes of rotation.

With more time, using the distances to the closest match as weights and analyzing the surrounding curvature would definitely help improve the 3D matching. However, the performance is satisfactory since the ICP matching for the LiDAR will exclusively be in 2D.

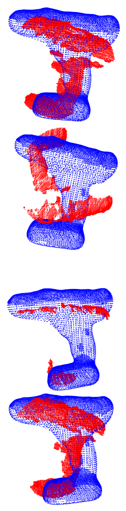


Fig. 2: ICP performance for bottle point cloud

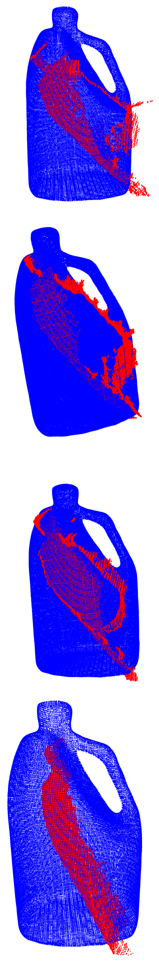


Fig. 3: ICP performance for bottle point cloud

V. Results

The following plots are for data set 20. The scan matching definitely preserved the general shape of the trajectory, but the direction seemed off. The scan matching definitely had some good traces, but exaggerated some turns along the trajectory, which led to a general rotation from the initial trajectory.

Careful inspection of the LiDAR points in the world frame can show that there is an offset of the points when the robot makes a U-turn. With observation of the factor graph optimized trajectory, these portions of the trajectory are put closer together. LiDAR point translation given by the factor graph The factor graph estimates fixes this very well for the right-most section of the map, but struggles with the rest of the map. However, it is a good sign that the factor graph optimization did some good work.

The floor mapping didn't seem too bad since all of the colors had decent correlation with the surrounding and smooth transitions. Note that all axes are in meters.

Given that the algorithms presented in this project were prone to fast rotations, it would be adequate to implement a robust method to prevent such errors from occuring.

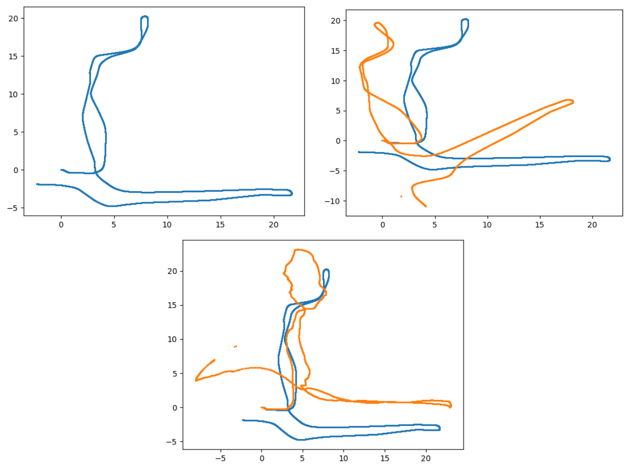


Fig. 4: Motion model (top left), LiDAR scan match (top right), and factor graph model (bottom) estimations

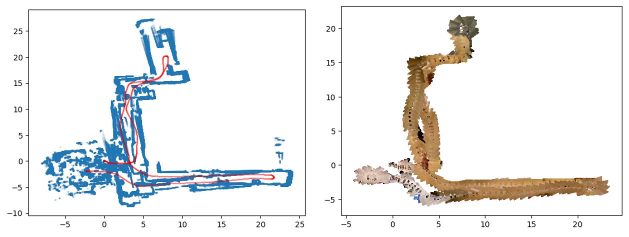


Fig. 5: LiDAR points (left) and floor mapping (right) with motion model estimates

The following plots are for data set 21. Similar to dataset 20, the scan matching model shows a rotated version of the original trajectory. Perhaps the ICP algorithm was a little too sensitive and converged to a point that it shouldn't have.

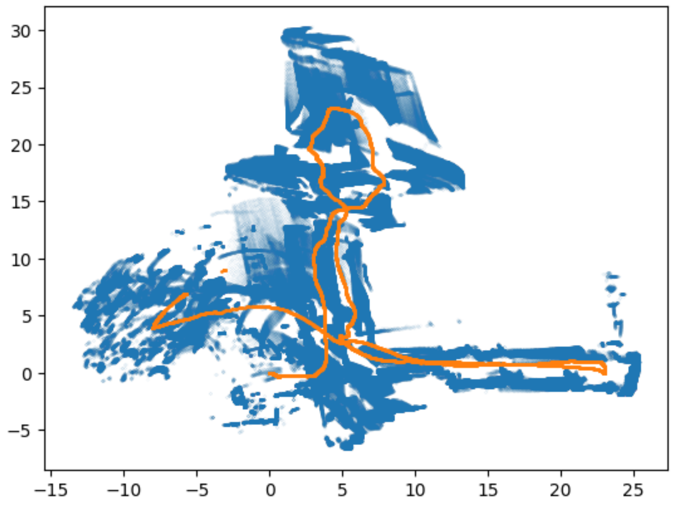


Fig. 6: LiDAR points mapping from factor graph estimates

The factor graph model is definitely not correct, under some assunmptions of the room shape. However, the horizontal portion of the bottom right maintained its shapre, which is a good sign. Perhaps the horizontal hallway had better matching LiDAR points that allowed better orientation tracking.

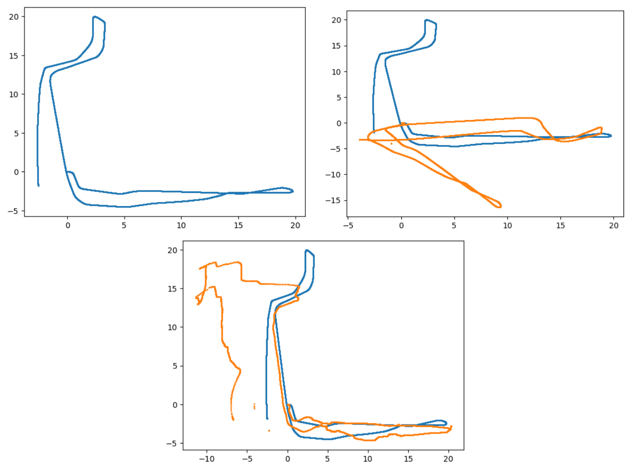


Fig. 7: Motion model (top left), LiDAR scan match (top right), and factor graph model (bottom) estimations

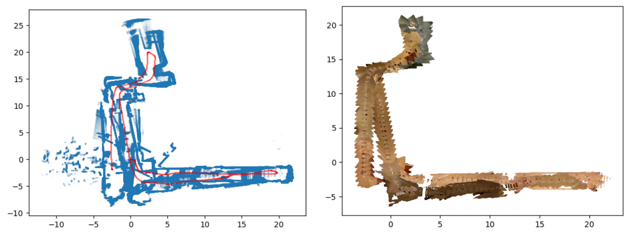


Fig. 8: LiDAR points (left) and floor mapping (right) with motion model estimates

Overall, the scan matching has overdid some of the rotations, which led to instances of the entire trajectory being a rotated version of the original. Furthermore, the factor graph model performed well only under circumstances with good scan matches.

With more time, I would've definitely implement the occupancy grid map along with better implementations for ICP and LiDAR point mapping.