LiDAR-Based SLAM and Texture Mapping

Jiajun Li
Email: jil186@ucsd.edu
Department of ECE, UCSD

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a cornerstone of autonomous robotics. This project implements LiDAR-based SLAM using encoder-IMU odometry, iterative closest point (ICP) scan matching, and pose graph optimization with GTSAM. We estimate robot trajectories, construct occupancy grid maps, and generate textured floor maps using RGBD data. Loop closure detection and factor graph optimization significantly improve mapping accuracy by reducing cumulative odometry errors.**

I. Introduction

Autonomous navigation in unknown environments is a fundamental capability for robots, enabling applications from search and rescue to automated inspection. A core challenge lies in accurately estimating the robot's pose while simultaneously reconstructing the environment—a problem known as Simultaneous Localization and Mapping (SLAM). Sensor noise, motion uncertainties, and cumulative errors make this task particularly demanding for real-world deployment.

This project addresses these challenges by integrating encoder-IMU odometry, LiDAR scan matching, and RGBD sensing. We first predict robot motion using wheel encoders and IMU data, refine the trajectory through iterative LiDAR alignment, and optimize it globally using factor graphs with loop closure constraints. The resulting trajectory enables the construction of precise occupancy grids and textured floor maps. Our approach demonstrates how sensor fusion and optimization techniques mitigate drift and enhance mapping accuracy, providing a robust foundation for autonomous navigation in unstructured environments.

II. Problem Formulation

Given a differential-drive robot equipped with encoders, IMU, LiDAR, and RGBD sensors, we aim to estimate its trajectory $\{T_t\}_{t=1}^{N} \in SE(2)$ (where $T_t = [x_t, y_t, \theta_t]^T$) and reconstruct a 2D occupancy grid map $\mathcal{M}$ with floor texture. The inputs include:

- **Encoder counts**: $\mathbf{e}_t \in \mathbb{Z}^4$ (four wheels, 40 Hz).
- **IMU data**: Yaw rate $\omega_t \in \mathbb{R}$.
- **LiDAR scans**: $\mathcal{L}_t = \{\mathbf{p}_i\}_{i=1}^{1081}$, where $\mathbf{p}_i$ are 2D obstacle coordinates.
- **RGBD images**: $\mathcal{D}_t$ providing depth and color data.

The problem is formulated as minimizing a cost function with three components:

- **Motion model**: Penalize deviations between consecutive poses T_t, T_{t+1} and encoder-IMU odometry predictions.
- **Observation model**: Minimize LiDAR-to-map alignment errors using iterative closest point (ICP).
- **Loop closure**: Enforce consistency between temporally distant poses via scan matching.

The optimization objective is:

$$\begin{aligned}\min_{\{T_t\},\mathcal{M}} &\sum_{t=1}^{N-1} \|T_{t+1} \ominus f(T_t, \mathbf{e}_t, \omega_t)\|^2_{\Sigma_o} \\ &+ \sum_{t=1}^{N} \|\mathcal{L}_t \ominus h(T_t, \mathcal{M})\|^2_{\Sigma_l} + \sum_{(i,j)\in\mathcal{C}} \|T_j \ominus T_i\|^2_{\Sigma_c},\end{aligned} \tag{1}$$

where $f(\cdot)$ is the differential-drive motion model, $h(\cdot)$ projects LiDAR scans to the map $\mathcal{M}$, $\mathcal{C}$ denotes loop closure pairs, and $\Sigma_o, \Sigma_l, \Sigma_c$ represent covariance matrices for odometry, LiDAR, and loop closure uncertainties.

III. Technical Approach

A. Encoder-IMU Odometry

Before getting into the core algorithm, we need to handle the input data. The raw data from encoder mean the number of ticks it counted in certain time interval. However, we need linear speed instead of counts for the rest of the problem. Therefore, we have to convert the data.

1) **Wheel Displacement**: Convert encoder counts $\mathbf{e}_t = [FR, FL, RR, RL]$ to linear displacements:

$$\Delta s_{\text{right}} = \frac{(FR + RR)}{2} \cdot 0.0022,$$
$$\Delta s_{\text{left}} = \frac{(FL + RL)}{2} \cdot 0.0022.$$

2) **Velocity Estimation**: Compute linear (v_t) and angular (ω_t) velocities:

$$v_t = \frac{\Delta s_{\text{right}} + \Delta s_{\text{left}}}{2\Delta t}, \quad \omega_t = \text{IMU yaw rate}.$$

where Δt is the time interval between two timestamp of the measurement. One more problem we may have here is the timestamp difference. IMU and encoder have different timestamp. So to make data synchronized, we take the closest data to form a pair.

With linear and angular velocities, we can calculate the position and orientation of the robot with respect to the world frame in all time through motion model.

3) **Pose Update**: Apply differential-drive motion model:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_t \Delta t \cos\left(\theta_t + \frac{\omega_t \Delta t}{2}\right) \\ y_t + v_t \Delta t \sin\left(\theta_t + \frac{\omega_t \Delta t}{2}\right) \\ \theta_t + \omega_t \Delta t \end{bmatrix}.$$

B. LiDAR Scan Matching via ICP

Position prediction of the robot barely based on the motion model is not enough. To improve the accuracy about our prediction, we implement ICP with LiDAR scan.

1) **Point Cloud Generation**: Convert LiDAR ranges to Cartesian points with respect to robot frame:

$$\mathbf{p}_i = r_i \begin{bmatrix} \cos\phi_i \\ \sin\phi_i \end{bmatrix}, \quad \phi_i \in [\theta_{\min}, \theta_{\max}].$$

where r_i is reading of the sensor which represents the distance of one beam.

2) **Initial Alignment**: Initialize relative pose ${}_{i+1}T_i$ using encoder-IMU odometry. To do this, we need to $T_i * T_{i+1}^{-1}$, where T is

$$T(x, y, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x \\ \sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transformation stands for change from robot frame to world frame.

3) **Pose Interpolation** The timestamp of LiDAR sensor does not match with the other sensors. Therefore, we need to interpolate new position prediction for each LiDAR reading. Given:

- Trajectory poses: $\{\mathbf{p}_k\}_{k=1}^{M}$, where

$$\mathbf{p}_k = [x_k, y_k, \theta_k]^T \in SE(2)$$

- Encoder and IMU timestamps: $\{t_k^{\text{enc}}\}_{k=1}^{M}$
- LiDAR timestamp: t^{LiDAR}

The interpolated pose $\mathbf{p}_{\text{interp}}$ at t^{LiDAR} is computed as:

$$\mathbf{p}_{\text{interp}} = \begin{cases} \mathbf{p}_1 & \text{if } t^{\text{LiDAR}} \le t_1^{\text{enc}}, \\ \mathbf{p}_M & \text{if } t^{\text{LiDAR}} \ge t_M^{\text{enc}}, \\ \mathbf{p}_{i-1} + \alpha(\mathbf{p}_i - \mathbf{p}_{i-1}) & \text{otherwise,} \end{cases}$$

where:

$$i = \min\left\{ j \in \{1, \ldots, M\} \mid t_j^{\text{enc}} \ge t^{\text{LiDAR}} \right\},$$

$$\alpha = \frac{t^{\text{LiDAR}} - t_{i-1}^{\text{enc}}}{t_i^{\text{enc}} - t_{i-1}^{\text{enc}}}.$$

Now, we have a more precise position prediction in the LiDAR timestamp.

4) **ICP Iteration**: ICP is trying to match the common place of the points cloud to its prevous one. This strengthens position prediction by adding extra reference instead of only motion odometry readings.

Input: Source point cloud $\mathcal{L}_{i-1} = \mathcal{S} = \{\mathbf{s}_i \in \mathbb{R}^3\}_{i=1}^{N}$, target point cloud $\mathcal{L}_i = \mathcal{T} = \{\mathbf{t}_j \in \mathbb{R}^3\}_{j=1}^{M}$, initial relative pose ${}_{i+1}T_i \in SE(3)$, max iterations K, tolerance ϵ.

Output: Optimal transformation $T^* \in SE(3)$, final alignment error E^*.

a) Repeat for $k = 1$ to K:

i) **Transform & Match**: Transform the target points cloud to source, and find the closest points in source with respect to the target points.

$$\mathcal{T}' = \{T \cdot \mathbf{t}_j \mid \mathbf{t}_j \in \mathcal{T}\}$$
$$\mathbf{s}_i \leftarrow \underset{s \in S}{argmin} \|\mathbf{s} - \mathbf{t}'_j\|, \quad \forall \mathbf{t}'_j \in \mathcal{T}'$$

ii) **Compute Error**:

$$E = \frac{1}{|\mathcal{T}|} \sum \|\mathbf{s}_i - \mathbf{t}'_j\|^2$$

iii) If $|E - E_{\text{prev}}| < \epsilon$, break

iv) **Calculate Transformation**: Using singular value decomposition (SVD) algorithm to find the new translation of the target points cloud which minimize the error.

$$\boldsymbol{\mu}_s = \text{mean}(\mathbf{s}_i), \quad \boldsymbol{\mu}_t = \text{mean}(\mathbf{t}'_j)$$
$$H = \sum (\mathbf{t}'_j - \boldsymbol{\mu}_t)(\mathbf{s}_i - \boldsymbol{\mu}_s)^\top$$
$$[U, \Sigma, V^\top] = \text{SVD}(H)$$
$$R = VU^\top \quad (\text{ensure} \det(R) = 1)$$
$$\mathbf{t} = \boldsymbol{\mu}_s - R\boldsymbol{\mu}_t$$

v) **Update Pose**:

$$T \leftarrow \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \cdot T$$

b) Return $T^* = T$, $E^* = E$

ICP can help us find the optimal transformation matrix for target points cloud to source points cloud which minimize their error. This transformation matrix will be our new relative pose and position prediction.

C. Occupancy Grid Mapping

a) **Log-Odds Update**: To draw a grid map, we need to continuously updating the status probability of cells in the grid. For each LiDAR scan $\mathcal{L}_t$ at pose T_t:

$$l(m_{i,j}) = l(m_{i,j}) + \begin{cases} \log \frac{p_{\text{occ}}}{1 - p_{\text{occ}}} & \text{(occupied cells)}, \\ \log \frac{1 - p_{\text{free}}}{p_{\text{free}}} & \text{(free cells)}. \end{cases}$$

where $m_{i,j}$ is the cell in i_{th} column and j_{th} row, and we will classify a cell be occupied or not base on the final likelihood the cell, $l(m_{i,j})$.

b) **Bresenham Ray Tracing**: Bresenham algorithm help us to find the line in 2D from starting point to end point. For LiDAR point $\mathbf{p}_k$, update free cells along the ray from robot position, T_t to $\mathbf{p}_k$.

D. Texture Mapping with RGBD Data

RGBD data also have different timestamp than other sensors. Therefore, we match the closest data of LiDAR and RGB image as pairs. Moreover, data are coming from two cameras, one camera takes the rgb image and the other one takes the disparity image. To align the RGB image with the disparity image, we project RGB pixel onto the depth that disparity image captured.

a) **Depth Projection**: For RGBD pixel (u, v) with disparity d:

$$z = \frac{1.03}{-0.00304d + 3.31}, \quad \mathbf{p}_{\text{camera}} = z \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

where K is the camera parameter matrix

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

b) **Image Conversion**: Although we have convert the RGB to disparity image, it is in the frame of optical frame. We need to convert it into regular frame. Moreover, we have to convert the image to the robotic frame instead of the camera frame because the camera is tilting in our robot. Optical frame to regular frame matrix is taking the inverse of matrix

$$\mathbf{oRr} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

and transformation matrix from camera to robot comes from

$$\text{depth\_cam\_pos} = \begin{bmatrix} 0.18 \\ 0.005 \\ 0.36 \end{bmatrix}$$

$$\text{depth\_cam\_rot} = \begin{bmatrix} 0 \\ 0.36 \\ 0.021 \end{bmatrix}$$

to form matrix

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\text{roll}) & -\sin(\text{roll}) \\ 0 & \sin(\text{roll}) & \cos(\text{roll}) \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\text{pitch}) & 0 & \sin(\text{pitch}) \\ 0 & 1 & 0 \\ -\sin(\text{pitch}) & 0 & \cos(\text{pitch}) \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\text{yaw}) & -\sin(\text{yaw}) & 0 \\ \sin(\text{yaw}) & \cos(\text{yaw}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z R_y R_x$$

c) **Floor Extraction**:Filter points with $z_{\text{world}} < 0.1$ m.

d) **Color Assignment**: Map floor points to grid cells using nearest-neighbor interpolation.

E. Pose Graph Optimization with GTSAM

a) **Factor Graph Construction**:

- **Odometry Factors**: For consecutive poses T_t and T_{t+1}, create a between-factor encoding the relative transformation $\hat{T}_{t\to t+1}$ estimated by ICP:

$$\text{Factor}(T_t, T_{t+1}) \propto \exp\left(-\frac{1}{2}\|\log(T_t^{-1} T_{t+1} \ominus \hat{T}_{t\to t+1})\|^2_{\Sigma_o^{-1}}\right) \tag{2}$$

where Σ_o is the ICP estimation covariance matrix.

- **Loop Closure**: Detect revisited locations using:

$$\text{Proximity: } \|T_i^{xy} - T_j^{xy}\| < \delta_{\text{pos}}$$
$$\text{Scan Match: } \mathcal{L}_i \leftrightarrow \mathcal{L}_j \text{ via ICP}$$
$$\text{Error Check: } E_{\text{ICP}} < \tau$$

Valid pairs (T_i, T_j) generate loop closure factors:

$$\text{Factor}(T_i, T_j) \propto \exp\left(-\frac{1}{2}\|\log(T_i^{-1} T_j \ominus \hat{T}_{i\to j})\|^2_{\Sigma_l^{-1}}\right)$$

b) **Optimization**: Solve using Levenberg-Marquardt algorithm:

$$T^*_{1:N} = \arg\min_{T_{1:N}} \underbrace{\sum_{t=1}^{N-1} \|e_t^{\text{odo}}\|^2_{\Sigma_o^{-1}}}_{\text{Odometry Terms}} + \underbrace{\sum_{(i,j)\in\mathcal{C}} \|e_{ij}^{\text{loop}}\|^2_{\Sigma_l^{-1}}}_{\text{Loop Closure Terms}}$$

Where:

- $e_t^{\text{odo}} = \log(T_t^{-1} T_{t+1} \ominus \hat{T}_{t\to t+1})$ (odometry residual)
- $e_{ij}^{\text{loop}} = \log(T_i^{-1} T_j \ominus \hat{T}_{i\to j})$ (loop closure residual)
- $\log : SE(2) \to \mathbb{R}^3$ maps transformations to tangent space coordinates
- $\ominus$ operator: $A \ominus B = A^{-1} \cdot B$ (transformation composition)

IV. Results

A. Trajectory Estimation

Figure 1 compares the estimated trajectories across three stages:

- **Encoder-IMU Odometry**: Accumulates significant drift due to wheel slippage and IMU biases
- **ICP-Corrected**: Reduces drift through scan matching but shows residual errors at loop closures
- **GTSAM-Optimized**: Produces globally consistent trajectory by balancing odometry and loop closure constraints

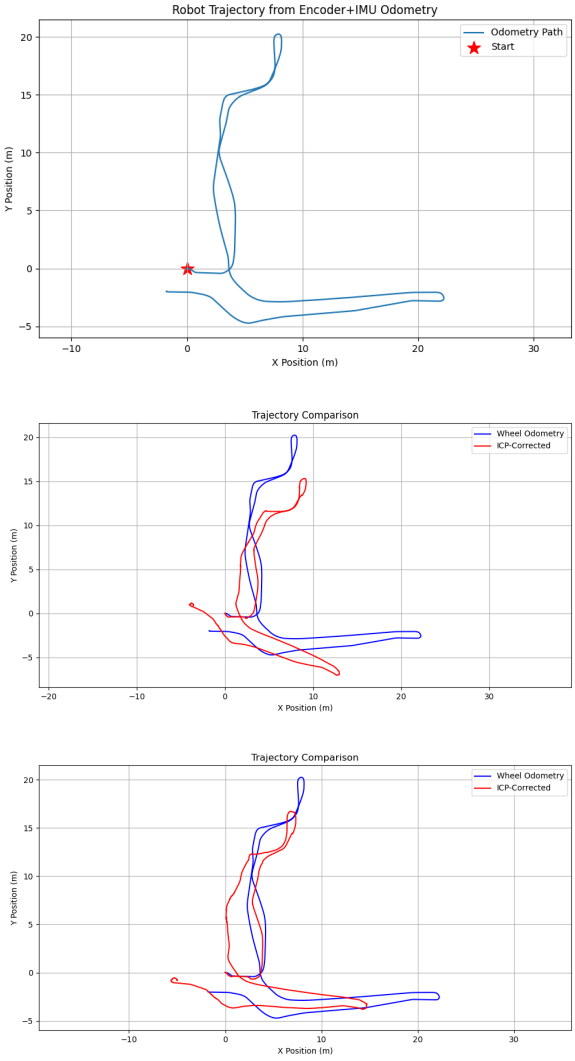

Fig. 1. Trajectory comparison: (top) Raw odometry, (middle) ICP, (bottom) GTSAM

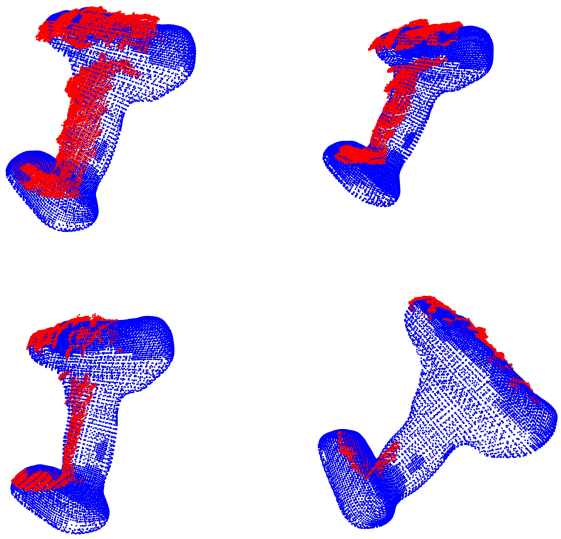

Fig. 2. Alignment Performance of Points Cloud in Different Directions

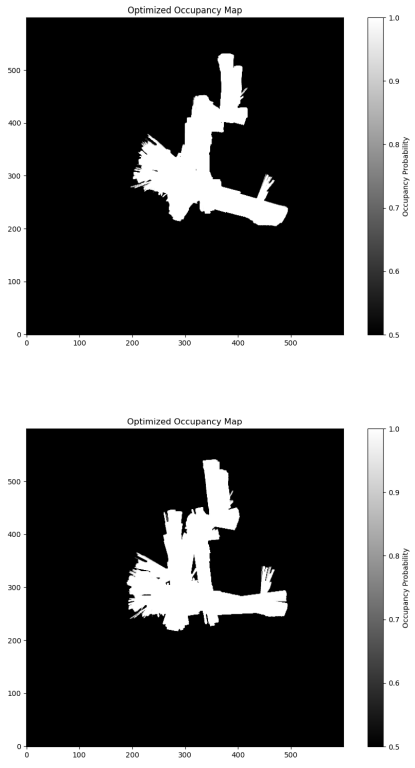

Fig. 3. Occupancy grid maps: (top) Unoptimized map, (bottom) Optimized map

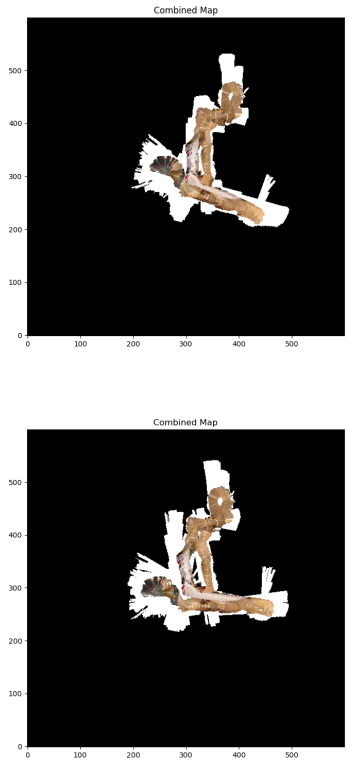

Fig. 4. Textured floor map:(top) Unoptimized map, (bottom) Optimized map

B. ICP Warm Up

Figure 2 demonstrates a warm up practice with aligning points cloud of a drill. The target (red) aligns to the surface of the drill (blue) closely. This made a good foundation for the following ICP problem.

C. Occupancy Grid Mapping

Figure 3 demonstrates mapping improvements through GTSAM. The loop closure detection seeks to minimize the discrepancy between the measured relative poses and estimated absolute poses, thus refining the robot trajectory.

- **Before Optimization**: Two paths count as one and duplicated features due to trajectory drift
- **After Optimization**: Clearer corridors and consistent wall alignments

D. Texture Mapping Performance

Similar to occupancy grid map, the texture map avoid closure loop problem and mapping out the paths with clear corridors and consistent wall alignments.

E. Discussion

ICP scan matching improved odometry accuracy, while GTSAM optimization further reduced pose error. Texture mapping occasionally misassigned pathes due to depth noise, but floor regions were reconstructed coherently.

V. Further Work

- Incorporate semantic segmentation to improve texture mapping.
- Implement probabilistic loop closure detection using deep learning.
- Fuse LiDAR and RGBD data for 3D mapping.
- Optimize ICP with GPU acceleration for real-time performance.
- Extend to multi-robot SLAM.

VI. Conclusion

This project demonstrated a complete LiDAR-based SLAM pipeline integrating encoder-IMU odometry, ICP scan matching, and factor graph optimization. The results validated the effectiveness of loop closure in reducing trajectory drift. The textured maps provide actionable environmental representations for navigation tasks. Future work will focus on real-time performance and 3D extensions.

References

[1] GTSAM Development Team, “Georgia Tech Smoothing and Mapping Library,” 2025. [Online]. Available: https://gtsam.org
[2] Nikolay Atanasov, “Motion and Observation Models,” ECE276A: Sensing & Estimation in Robotics, UCSD, 2025. [11-14].
[3] Nikolay Atanasov, “Localization and Odometry from Point Features,” ECE276A: Sensing & Estimation in Robotics, UCSD, 2025. [12-33].
[4] Nikolay Atanasov, “Particle Filter SLAM,” ECE276A: Sensing & Estimation in Robotics, UCSD, 2025. [25-34].
[5] Nikolay Atanasov, “Factor Graph SLAM,” ECE276A: Sensing & Estimation in Robotics, UCSD, 2025.