# Project 2: LiDAR-Based SLAM

Chen-Yeh Lin
Department of Electrical and Computer Engineering
University of California, San Diego
PID: A69036794

***Abstract*—This project implements a LiDAR-based Simultaneous Localization and Mapping (SLAM) system for a differential-drive robot equipped with wheel encoders, an Inertial Measurement Unit (IMU), a 2D LiDAR scanner, and an RGBD camera. The approach integrates odometry-based motion estimation with LiDAR scan matching using the Iterative Closest Point (ICP) algorithm to mitigate drift errors. Additionally, pose graph optimization with loop closure constraints further refines the estimated trajectory, enhancing map consistency. The resulting trajectory is used to construct a 2D occupancy grid and a texture-mapped representation of the floor using RGBD sensor data. Experimental results demonstrate improved localization accuracy and mapping quality, showcasing the potential of LiDAR-based SLAM in autonomous navigation and robotic exploration.**

## I. Introduction

Mapping and localization are fundamental challenges in robotics, particularly for autonomous systems operating in unknown environments. This project aims to implement a LiDAR-based SLAM system that enables a differential-drive robot to construct a 2D map while estimating its trajectory. SLAM is essential in various applications, including autonomous navigation, robotic exploration, and augmented reality, where precise localization and environmental representation are required.

Despite advancements in SLAM techniques, odometry-based localization remains prone to errors due to wheel slippage, sensor noise, and accumulated drift over time. To address these challenges, this project combines multiple sensor modalities, utilizing wheel encoders and an IMU for initial motion estimation while refining localization through LiDAR-based scan matching. The ICP algorithm is employed to align consecutive LiDAR scans, reducing pose estimation errors and improving the overall trajectory.

Further enhancements are achieved through pose graph optimization, which incorporates loop closure constraints to correct long-term drift and inconsistencies in the map. Additionally, an RGBD camera is utilized to generate a texture-mapped floor representation by projecting color data onto the estimated 2D occupancy grid. The final implementation integrates these techniques to provide an accurate and visually informative map.

To demonstrate the effectiveness of this approach, we evaluate the system's performance in terms of trajectory accuracy and mapping consistency. Compared to pure odometry-based localization, the integration of LiDAR scan matching and pose graph optimization significantly improves SLAM accuracy. These findings highlight the potential of LiDAR-based SLAM in real-world applications, particularly for autonomous robotic systems operating in dynamic and unstructured environments.

## II. Problem Formulation

The objective of this project is to implement a LiDAR-based simultaneous localization and mapping (SLAM) system for a differential-drive robot. The system estimates the robot's trajectory while constructing a two-dimensional occupancy grid map of the environment. It integrates odometry, LiDAR scan matching, and pose graph optimization to improve localization accuracy and map consistency. Additionally, an RGBD camera is utilized to generate a texture-mapped representation of the floor. This section presents the mathematical formulation of the problem.

### A. State Estimation Model

Our system utilizes a *differential-drive* robot that operates in the $(x, y)$ plane with orientation $\theta$ along its $z$-axis. The robot is equipped with the following sensors:

- Wheel Encoders for odometry estimation
- IMU for angular velocity measurement
- 2D LiDAR (Hokuyo) for scan matching
- RGBD Camera (Kinect) for texture mapping

The motion model of our robot is given by the Euler discretization model:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \mathbf{x}_t + \tau_t \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix} \tag{1}$$

where:

- $(x_t, y_t, \theta_t)$ represents the robot's position and orientation at time $t$,

- $v_t$ is the linear velocity obtained from the wheel encoders,
- $\omega_t$ is the angular velocity obtained from the IMU, and
- $\tau_t$ is the time interval between two consecutive measurements.

Encoders measure the rotations of the four wheels at a frequency of 40 Hz. Given encoder counts $[FR, FL, RR, RL]$, corresponding to the front-right, front-left, rear-right, and rear-left wheels, the travel distances for the right and left wheels are computed as:

$$\begin{aligned} d_{\text{right}} &= \frac{FR + RR}{2} \times 0.0022, \\ d_{\text{left}} &= \frac{FL + RL}{2} \times 0.0022. \end{aligned} \tag{2}$$

The robot's linear velocity is estimated as:

$$v_t = \frac{1}{2\tau_t} \left( d_{\text{right}} + d_{\text{left}} \right). \tag{3}$$

### B. SLAM Problem Formulation

The goal of SLAM is to estimate the robot's trajectory and simultaneously construct a map of the environment. Given an initial state $\mathbf{x}_0$, sensor measurements $\mathcal{Z} = \{z_t\}_{t=1}^{K}$, an observation model $h(\mathbf{x}_t, m)$, and a motion model $f(\mathbf{x}_t, \mathbf{u}_t)$, we solve the following optimization problem:

$$\min_{\mathbf{x}_{1:K}, m} \sum_{t=1}^{K} \|z_t - h(\mathbf{x}_t, m)\|^2 + \sum_{t=0}^{K-1} \|\mathbf{x}_{t+1} - f(\mathbf{x}_t, \mathbf{u}_t)\|^2. \tag{4}$$

Alternatively, this problem can be reformulated using a factor graph representation. Given a graph $G = (\mathcal{X}, \mathcal{E})$, where nodes $\mathcal{X}$ represent robot poses and edges $\mathcal{E}$ encode relative transformations, the optimization problem becomes:

$$\min_{\{\mathbf{T}_i\}} \sum_{(i,j) \in \mathcal{E}} \|W_{ij} e(\mathbf{T}_i, \mathbf{T}_j)\|^2, \tag{5}$$

where $W_{ij}$ are weights associated with each edge.

### C. LiDAR Scan Matching using ICP

Given a sequence of two-dimensional LiDAR scans, the estimated robot trajectory is refined by aligning consecutive scans using the iterative closest point (ICP) algorithm:

$$T_{t \to t+1} = \arg\min_{T} \sum_{i} \|T p_t^i - p_{t+1}^i\|^2. \tag{6}$$

The updated robot pose is computed as:

$$\mathbf{x}_{t+1} = T_{t \to t+1} \cdot \mathbf{x}_t. \tag{7}$$

### D. Occupancy Grid Mapping

The environment is represented as an occupancy grid where each grid cell $m$ stores a probability of being occupied, updated using a log-odds representation:

$$\ell(m_{x,y}) = \log \frac{P(m_{x,y} = \text{occupied})}{P(m_{x,y} = \text{free})}.$$

### E. Kinect Sensor and Texture Mapping

The RGBD camera provides depth and color images. Depth information is projected into 3D space using intrinsic parameters:

$$\mathbf{P}_{\text{cam}} = d(i,j) K^{-1} \mathbf{p}_{\text{img}}. \tag{8}$$

To map the extracted depth points to the world coordinate system, transformations from the camera to the robot frame and from the robot to the world frame are applied:

$$\mathbf{P}_{\text{world}} = T_{\text{robot} \to \text{world}} \cdot T_{\text{camera} \to \text{robot}} \cdot \mathbf{P}_{\text{cam}}. \tag{9}$$

### F. Pose Graph Optimization for Loop Closure

To correct long-term drift, pose graph optimization is employed by incorporating loop closure constraints. The optimized trajectory is obtained by solving:

$$\hat{\mathbf{x}}_{1:T} = \arg\min_{\mathbf{x}_{1:T}} \sum_{t} \|f(\mathbf{x}_t, u_t) - \mathbf{x}_{t+1}\|^2 + \sum_{(i,j) \in \mathcal{L}} \|g(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}_{i,j}\|^2. \tag{10}$$

Here, $\mathcal{L}$ represents detected loop closures, and $g(\mathbf{x}_i, \mathbf{x}_j)$ is the relative transformation.

This formulation ensures that the robot trajectory is optimized while maintaining map consistency.

## III. Technical Approach

We approach the problem in the following manner:

- *Encoder and IMU Odometry*
- *Point-cloud registration via ICP*
  - Warm-up
  - Scan Matching
- *Occupancy and Texture mapping*
- *Pose graph optimization and loop closure*

### A. Encoder and IMU Odometry

To estimate the robot's trajectory, we use data from wheel encoders and an inertial measurement unit (IMU). The robot follows a differential-drive kinematic model, where the linear velocity $v_t$ is obtained from encoder readings and the angular velocity $\omega_t$ is determined from the IMU yaw rate. Given the encoder counts for the front-right (FR), front-left (FL), rear-right (RR), and rear-left (RL) wheels, the traveled distances for the right and left wheels are computed as:

$$d_r = \frac{(FR + RR)}{2} \times r_e, \quad d_l = \frac{(FL + RL)}{2} \times r_e \tag{11}$$

where $r_e = 0.0022$ m/tick is the conversion factor from encoder ticks to meters. The linear velocity is then calculated as:

$$v_t = \frac{d_r + d_l}{2\Delta t} \tag{12}$$

while the angular velocity is obtained from IMU measurements. Since the IMU and encoder operate at different frequencies, their timestamps are not inherently synchronized. To align the IMU yaw rate $\dot{\theta}_{\text{IMU}}$ with the encoder timestamps $t_t$, we perform linear interpolation:

$$\omega_t \approx \dot{\theta}_{\text{IMU}}(t_{i-1}) + \frac{(t_t - t_{i-1})}{(t_i - t_{i-1})} \left( \dot{\theta}_{\text{IMU}}(t_i) - \dot{\theta}_{\text{IMU}}(t_{i-1}) \right) \tag{13}$$

where $t_i$ and $t_{i-1}$ are the closest IMU timestamps surrounding $t_t$. This ensures smooth and consistent yaw rate estimation.

Using the differential-drive motion model, the robot's pose at time $t$, denoted as $(x_t, y_t, \theta_t)$, is recursively computed as:

$$\theta_t = \theta_{t-1} + \omega_t \Delta t \tag{14}$$

$$\begin{aligned} x_t &= x_{t-1} + v_t \cos(\theta_{t-1}) \Delta t, \\ y_t &= y_{t-1} + v_t \sin(\theta_{t-1}) \Delta t \end{aligned} \tag{15}$$

where $\Delta t = t_t - t_{t-1}$ is the time interval between consecutive encoder readings. To prevent numerical instability caused by duplicate or missing timestamps, we enforce a minimum time threshold:

$$\begin{aligned} \Delta t = \max(t_t - t_{t-1}, \Delta t_{\min}), \\ \text{where } \Delta t_{\min} = 1 \text{ ms} \end{aligned} \tag{16}$$

The initial robot pose is assumed to be $(x_0, y_0, \theta_0) = (0, 0, 0)$. The computed trajectory $(x_t, y_t)$ is visualized to validate the accuracy of the odometry estimation.

### *B. Point-cloud registration via ICP*

*1) Warm-up:* The Iterative Closest Point (ICP) algorithm estimates the transformation between two point clouds by iteratively refining the alignment. Given a reference point cloud $\mathcal{M} = \{\mathbf{m}_i\}_{i=1}^N$ and a source point cloud $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^M$, the goal is to find the optimal rotation $\mathbf{R} \in SO(d)$ and translation $\mathbf{p} \in R^d$ that minimize the alignment error:

$$\min_{\mathbf{p} \in R^d, \mathbf{R} \in SO(d)} \sum_i w_i \|(\mathbf{R}\mathbf{z}_i + \mathbf{p}) - \mathbf{m}_i\|^2 \tag{17}$$

where $w_i$ are correspondence weights. The centroids of both point clouds are computed as:

$$\bar{\mathbf{m}} = \frac{\sum_i w_i \mathbf{m}_i}{\sum_i w_i}, \quad \bar{\mathbf{z}} = \frac{\sum_i w_i \mathbf{z}_i}{\sum_i w_i} \tag{18}$$

Solving for $\mathbf{p}$ by setting $\nabla_{\mathbf{p}} f(\mathbf{R}, \mathbf{p}) = 0$ gives:

$$\mathbf{p}^* = \bar{\mathbf{m}} - \mathbf{R}\bar{\mathbf{z}} \tag{19}$$

Substituting this into the original minimization leads to Wahba's problem:

$$\max_{\mathbf{R} \in SO(3)} \text{tr}(\mathbf{Q}^T \mathbf{R}) \tag{20}$$

where the cross-covariance matrix is:

$$\mathbf{Q} = \sum_i w_i \delta\mathbf{m}_i \delta\mathbf{z}_i^T, \quad \delta\mathbf{m}_i = \mathbf{m}_i - \bar{\mathbf{m}}, \quad \delta\mathbf{z}_i = \mathbf{z}_i - \bar{\mathbf{z}} \tag{21}$$

Using Singular Value Decomposition (SVD), $\mathbf{Q} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, the optimal rotation is:

$$\mathbf{R}^* = \mathbf{U} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{U}\mathbf{V}^T) \end{bmatrix} \mathbf{V}^T \tag{22}$$

With $\mathbf{R}^*$ computed, the translation is updated as $\mathbf{p}^* = \bar{\mathbf{m}} - \mathbf{R}^*\bar{\mathbf{z}}$. The ICP algorithm starts with initial estimates $\mathbf{R}_0 = \mathbf{I}$ and $\mathbf{p}_0 = \bar{\mathbf{m}} - \bar{\mathbf{z}}$, then iterates:

1) Find correspondences using $i \leftrightarrow \arg\min_j \|\mathbf{m}_i - (\mathbf{R}_k \mathbf{z}_j + \mathbf{p}_k)\|^2$.
2) Compute centroids $\bar{\mathbf{m}} = \frac{1}{N}\sum_i \tilde{\mathbf{m}}_i$, $\bar{\mathbf{z}} = \frac{1}{N}\sum_i \mathbf{z}_i$.
3) Compute covariance matrix $\mathbf{Q} = \sum_i \delta\tilde{\mathbf{m}}_i \delta\mathbf{z}_i^T$, solve $\mathbf{Q} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.
4) Update rotation $\mathbf{R}_{k+1}$ using $\mathbf{U}\mathbf{V}^T$ and translation $\mathbf{p}_{k+1} = \bar{\mathbf{m}} - \mathbf{R}_{k+1}\bar{\mathbf{z}}$.
5) Stop when $\|\mathbf{p}_{k+1} - \mathbf{p}_k\| < \epsilon$ or the mean square error change is below a threshold.

This process refines the transformation $(\mathbf{R}, \mathbf{p})$ until convergence. The final transformation $(\mathbf{R}, \mathbf{p})$ is then applied to align the source point cloud to the target.

*2) Scan Matching:* To improve the initial odometry estimate obtained from the motion model, we perform LiDAR scan matching using the Iterative Closest Point (ICP) algorithm. Given two consecutive LiDAR scans $\mathcal{P}_t = \{\mathbf{p}_i\}_{i=1}^N$ and $\mathcal{P}_{t+1} = \{\mathbf{q}_j\}_{j=1}^M$, we estimate the relative transformation $\mathbf{T}_{t,t+1} \in SE(2)$ that best aligns $\mathcal{P}_{t+1}$ to $\mathcal{P}_t$. The transformation consists of a rotation $\mathbf{R} \in SO(2)$ and a translation $\mathbf{t} \in R^2$, represented as:

$$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

Each LiDAR scan contains a set of range measurements $\{r_i\}$ and corresponding angles $\{\theta_i\}$, which are converted into Cartesian coordinates as:

$$x_i = r_i \cos(\theta_i), \quad y_i = r_i \sin(\theta_i) \tag{24}$$

forming the scan point cloud. The transformation is initialized using odometry, where the relative pose between timestamps is estimated as:

$$\mathbf{T}_{\text{odom}} = \begin{bmatrix} \cos(\theta_{t+1} - \theta_t) & -\sin(\theta_{t+1} - \theta_t) & x_{t+1} - x_t \\ \sin(\theta_{t+1} - \theta_t) & \cos(\theta_{t+1} - \theta_t) & y_{t+1} - y_t \\ 0 & 0 & 1 \end{bmatrix} \tag{25}$$

The source scan $\mathcal{P}_{t+1}$ is transformed using $\mathbf{T}_{\text{odom}}$ to provide an initial estimate. For each point in $\mathcal{P}_{t+1}$, the closest point in $\mathcal{P}_t$ is found using a k-d tree:

$$i \leftrightarrow \arg\min_j \|\mathbf{p}_i - (\mathbf{R}\mathbf{q}_j + \mathbf{t})\|^2 \tag{26}$$

After obtaining correspondences, we compute the centroids:

$$\bar{\mathbf{p}} = \frac{1}{N} \sum_i \mathbf{p}_i, \quad \bar{\mathbf{q}} = \frac{1}{N} \sum_i \mathbf{q}_i \tag{27}$$

and the covariance matrix:

$$\mathbf{Q} = \sum_i (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^T \tag{28}$$

Using Singular Value Decomposition (SVD), $\mathbf{Q} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, the optimal rotation is obtained as:

$$\mathbf{R}^* = \mathbf{U} \begin{bmatrix} 1 & 0 \\ 0 & \det(\mathbf{U}\mathbf{V}^T) \end{bmatrix} \mathbf{V}^T \tag{29}$$

and the translation is:

$$\mathbf{t}^* = \bar{\mathbf{p}} - \mathbf{R}^* \bar{\mathbf{q}} \tag{30}$$

The transformation $\mathbf{T}_{t,t+1}$ is iteratively refined until convergence criteria are met:

$$\|\mathbf{t}_{k+1} - \mathbf{t}_k\| < \epsilon, \quad \frac{1}{N} \sum_i \|\mathbf{p}_i - (\mathbf{R}_{k+1}\mathbf{q}_i + \mathbf{t}_{k+1})\|^2 < \tau \tag{31}$$

Once $\mathbf{T}_{t,t+1}$ is computed, the global pose of the robot is updated as:

$$\mathbf{T}_{t+1} = \mathbf{T}_t \cdot \mathbf{T}_{t,t+1} \tag{32}$$

This ensures that the robot's trajectory is continuously refined, reducing drift and improving localization accuracy.

### C. Occupancy and Texture mapping

*1) Occupancy mapping:* To build a 2D occupancy grid map, we integrate LiDAR scans along the robot's trajectory. The map is represented as a discrete grid with a specified resolution $\Delta x$, where each cell $(x, y)$ holds a log-odds value $L(x, y)$ indicating the likelihood of being occupied.

First, each LiDAR scan provides a set of range measurements. These measurements are converted into Cartesian coordinates in the sensor frame:

$$x_i = r_i \cos\theta_i, \quad y_i = r_i \sin\theta_i,$$

with the corresponding homogeneous point given by

$$\mathbf{p}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}.$$

Using the estimated robot pose $\mathbf{T}_t \in R^{3\times3}$ (obtained from encoder+IMU odometry refined via scan matching), the global coordinates of each point are computed as

$$\mathbf{p}_i^{\text{global}} = \mathbf{T}_t \, \mathbf{p}_i.$$

The grid cell corresponding to $\mathbf{p}_i^{\text{global}}$ is determined by

$$\text{cell}(x, y) = \left\lfloor \frac{\mathbf{p}_i^{\text{global}} - \mathbf{m}}{\Delta x} \right\rfloor,$$

where $\mathbf{m}$ is the minimum coordinate of the map.

For each LiDAR measurement, a ray is cast from the robot's current position $\mathbf{r}_t = [x_t, y_t]^T$ to the endpoint cell. Using a ray-tracing algorithm (e.g., Bresenham's algorithm), let $\mathcal{R}$ denote the set of cells along the ray, with the final cell $\mathcal{R}_{\text{occ}}$ corresponding to the detected obstacle.

The occupancy update is performed in the log-odds domain:

$$L_{\text{new}}(x, y) = L_{\text{old}}(x, y) + \begin{cases} \log \frac{P_{\text{occ}}}{1 - P_{\text{occ}}}, & \text{if } (x, y) \in \mathcal{R}_{\text{occ}}, \\ \log \frac{P_{\text{free}}}{1 - P_{\text{free}}}, & \text{if } (x, y) \in \mathcal{R} \setminus \mathcal{R}_{\text{occ}}, \end{cases}$$

where $P_{\text{occ}}$ and $P_{\text{free}}$ are the probabilities assigned to occupied and free cells, respectively. The log-odds values are clipped to a range $[L_{\min}, L_{\max}]$ to prevent divergence. Finally, the occupancy probability for each cell is recovered by

$$P(x, y) = \frac{1}{1 + \exp(-L(x, y))}.$$

This update process is applied sequentially for every LiDAR scan. The first scan is assumed to correspond to the global origin, while subsequent scans are transformed into the global frame using the estimated poses. Cells with measurements that fall outside the defined map boundaries are ignored.

This method allows us to construct an accurate occupancy grid map based solely on the integration of LiDAR scans and the refined robot trajectory.

*2) Texture mapping:* To generate a colored floor map, we project RGB information from Kinect images onto the floor plane using the robot's pose. Each Kinect frame provides a disparity image $d(i, j)$ and a corresponding RGB image. For a pixel $(i, j)$ in the disparity image, we first compute:

$$dd = -0.00304\, d(i, j) + 3.31, \quad \text{depth} = \frac{1.03}{dd}. \tag{33}$$

Next, we determine the corresponding pixel $(\text{rgbi}, \text{rgbj})$ in the RGB image to retrieve its color. The mapping is given by:

$$\begin{aligned} \text{rgbi} &= \frac{526.37\, i + 19276 - 7877.07\, dd}{585.051}, \\ \text{rgbj} &= \frac{526.37\, j + 16662}{585.051}. \end{aligned} \tag{34}$$

Only valid pixels (i.e., $0 \le \text{rgbi} < H_{\text{rgb}}$, $0 \le \text{rgbj} < W_{\text{rgb}}$) are used. Let $(R, G, B)$ be the color at $(\text{rgbi}, \text{rgbj})$. We then convert $(i, j)$ into a 3D point in the camera frame:

$$x_c = \frac{(j - c_u)\,\text{depth}}{f_u}, \quad y_c = \frac{(i - c_v)\,\text{depth}}{f_v}, \quad z_c = \text{depth}. \tag{35}$$

In homogeneous coordinates,

$$\mathbf{p}_{\text{cam}} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}.$$

Using a known rigid transformation $T_{\text{robot}}^{\text{cam}} \in R^{4\times4}$ from camera to robot frames, and the robot pose $T_{\text{world}}^{\text{robot}}$ (obtained via odometry and scan matching), we obtain:

$$\mathbf{p}_{\text{world}} = T_{\text{world}}^{\text{robot}}\, T_{\text{robot}}^{\text{cam}}\, \mathbf{p}_{\text{cam}}. \tag{36}$$

If $\mathbf{p}_{\text{world}}$ is close to the floor plane (e.g., $|z_{\text{world}} - z_{\text{floor}}| < \epsilon$), we project $(x_{\text{world}}, y_{\text{world}})$ to the texture map grid cell:

$$\text{cell}(x, y) = \left\lfloor \frac{\mathbf{p}_{\text{world}}(1:2) - \mathbf{m}}{\Delta x} \right\rfloor, \tag{37}$$

where $\mathbf{m}$ is the minimum map coordinate and $\Delta x$ is the map resolution. The color $(R, G, B)$ is then assigned to this cell. Repeating the above process for each valid pixel across all Kinect frames accumulates a color map of the floor. When combined with the occupancy grid, this results in a textured floor map that enhances the visual representation of the SLAM environment.

### *D. Pose graph optimization and loop closure*

To further refine the robot's trajectory, we construct a pose graph in which each node $\mathbf{T}_i \in SE(2)$ represents the robot pose at time $i$, and each edge encodes a relative pose measurement. We begin by adding edges corresponding to consecutive poses from scan matching:

$$\mathbf{T}_{i+1} \approx \mathbf{T}_i \cdot \mathbf{T}_{i,i+1}^{\text{ICP}}, \tag{38}$$

where $\mathbf{T}_{i,i+1}^{\text{ICP}}$ is the relative transformation obtained by ICP between scans $i$ and $i + 1$. Next, we introduce loop-closure edges for pairs $(i, j)$ that are either detected by a proximity criterion (e.g., checking if the robot has revisited a region) or inserted at a fixed interval (e.g., every 10 poses). Each loop-closure edge imposes:

$$\mathbf{T}_j \approx \mathbf{T}_i \cdot \mathbf{T}_{i,j}^{\text{loop}}, \tag{39}$$

where $\mathbf{T}_{i,j}^{\text{loop}}$ is the estimated relative pose between scans $i$ and $j$. All edges are collected in a factor graph $\mathcal{G} = \{\mathbf{V}, \mathbf{E}\}$, where $\mathbf{V} = \{\mathbf{T}_i\}$ and $\mathbf{E}$ contains both ICP edges (38) and loop-closure edges (39).

To solve for the poses $\{\mathbf{T}_i\}$ that minimize the overall error, we define a cost function:

$$\begin{aligned} \min_{\{\mathbf{T}_i\}} \Bigg[ & \sum_{(i,i+1)\in\mathcal{E}_{\text{ICP}}} \left\| \mathbf{F}\left(\mathbf{T}_i, \mathbf{T}_{i+1}, \mathbf{T}_{i,i+1}^{\text{ICP}}\right) \right\|^2 \\ & + \sum_{(i,j)\in\mathcal{E}_{\text{loop}}} \left\| \mathbf{F}\left(\mathbf{T}_i, \mathbf{T}_j, \mathbf{T}_{i,j}^{\text{loop}}\right) \right\|^2 \Bigg], \end{aligned} \tag{40}$$

where $\mathbf{F}(\cdot)$ measures the difference in $SE(2)$ between the estimated relative pose $\mathbf{T}_i^{-1}\mathbf{T}_j$ and the measured relative pose (ICP or loop closure). We then apply a Gauss-Newton or Levenberg-Marquardt iteration to solve:

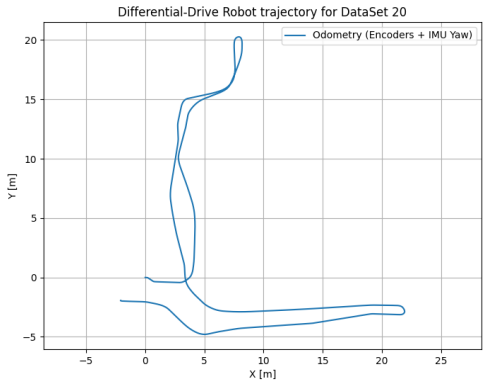
$$\theta_{k+1} = \theta_k + \alpha_k\, \delta\theta, \quad \text{where} \quad \nabla^2\mathcal{L}(\theta_k)\, \delta\theta = -\nabla\mathcal{L}(\theta_k), \tag{41}$$

with $\theta$ parameterizing all poses $\{\mathbf{T}_i\}$ in a suitable minimal coordinate (e.g., $[x, y, \theta]$ per node). The solution converges when $\|\delta\theta\|$ is below a threshold or the error reduction becomes negligible.

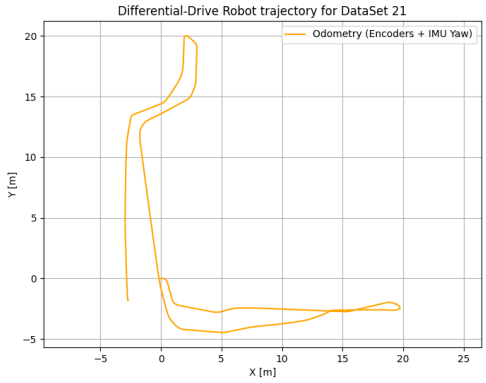
Once the poses are optimized, we obtain a final trajectory $\{\hat{\mathbf{T}}_i\}$ with reduced drift. This refined trajectory is then used to re-construct the occupancy grid and texture map, yielding improved global consistency compared to the raw ICP result.

## IV. Result

### A. Encoder and IMU odometry:




(a) Odometry (Encoders + IMU Yaw) for Dataset 20




(b) Odometry (Encoders + IMU Yaw) for Dataset 21

Fig. 1: Differential-drive robot trajectory estimates using encoders and IMU yaw for Dataset 20 and Dataset 21 .

After computing the trajectory at $T$ timestamps, Figure 1 depicts the robot's motion in the $(x, y)$ plane for both Data Set 20 and Data Set 21.

### B. Warm-up

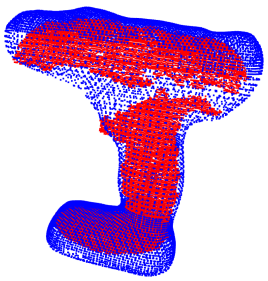


Fig. 2: drill 1 point cloud

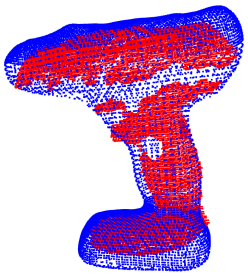


Fig. 3: drill 2 point cloud

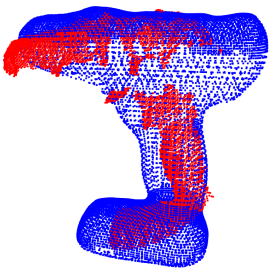


Fig. 4: drill 3 point cloud

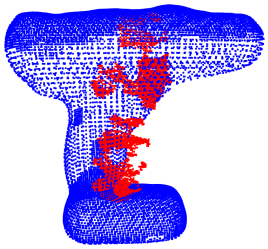


Fig. 5: drill 4 point cloud

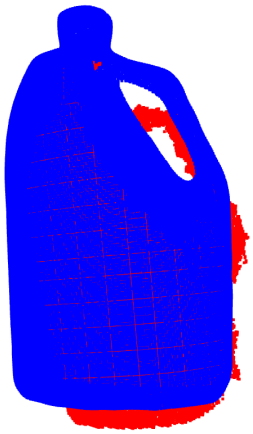


Fig. 6: liquid container point cloud 1

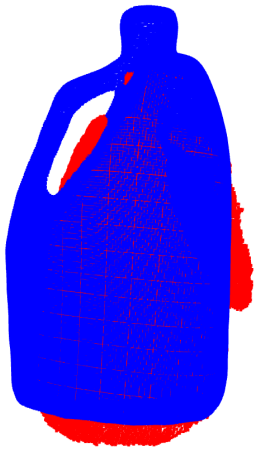


Fig. 7: liquid container point cloud 2

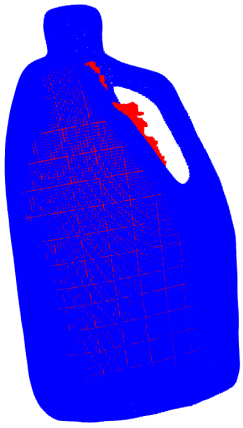


Fig. 8: liquid container point cloud 3

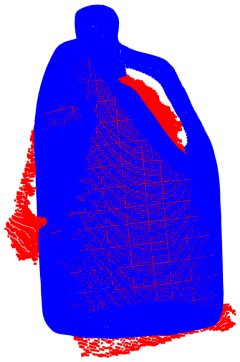
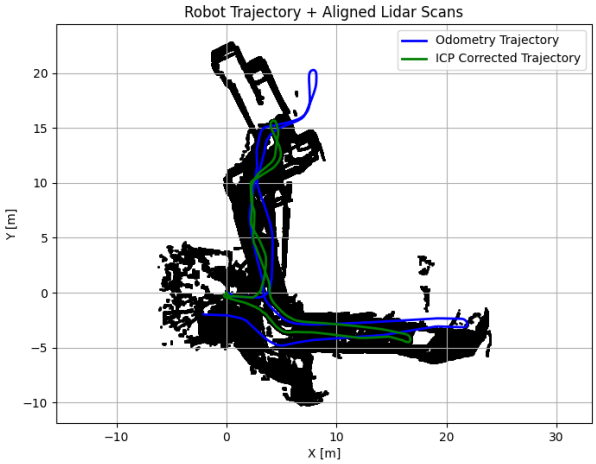


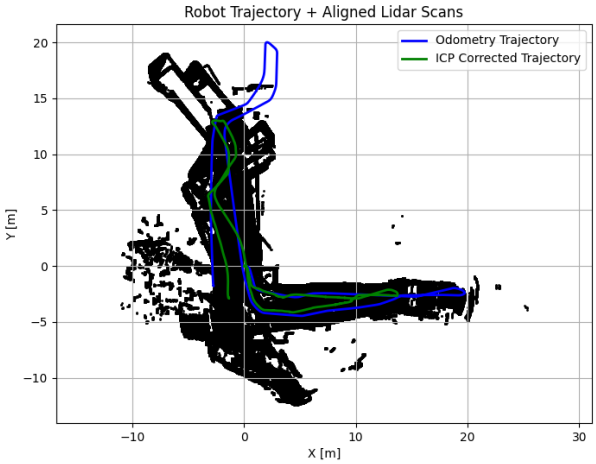
Fig. 9: liquid container point cloud 3

The warm-up ICP results demonstrated effective alignment of point clouds for both the drill and the liquid container. The algorithm successfully aligned the major features of each object by estimating reliable rotations and translations. Significant overlap between corresponding surfaces was observed, confirming that the core ICP routine and Kabsch algorithm are functioning as intended. Minor misalignments were present in complex or symmetrical regions, indicating areas where the algorithm struggles. Sensor noise and incomplete data further contributed to these residual discrepancies. Despite these challenges, the method proved robust even when dealing with partial overlaps and occlusions. The experiments revealed that the ICP approach converges reliably, highlighting its overall effectiveness. Future improvements could focus on mitigating issues arising from symmetry and noise to further refine the alignment.

### *C. Scan Matching*




(a) Trajectory for DataSet 20

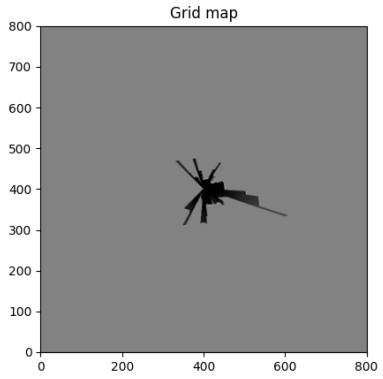



(b) Trajectory for DataSet 21

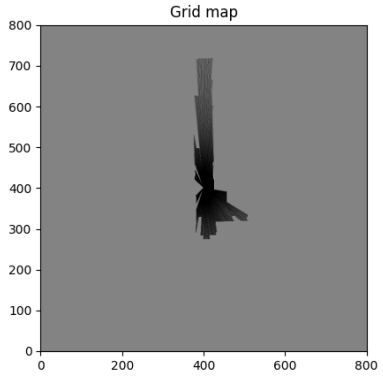
Fig. 10: Robot Trajectory + Aligned Lidar Scans for DataSet 20 and DataSet 21 .

In the results from Data Set 20 and Data Set 21, the raw odometry trajectory (blue) and the ICP-corrected trajectory (green) are shown overlaid on the aligned lidar scans (black). ICP noticeably reduces the drift present in pure odometry, resulting in more consistent global positioning for both datasets. The aligned scans appear better registered, indicating that local scan matching effectively refines the robot's pose estimates. Major structural features in the environment are more accurately captured after ICP, as demonstrated by fewer overlaps or gaps in the map. However, residual misalignments can be observed in certain areas, potentially caused by sparse features or partial occlusions in the lidar scans. Noise and outliers in the lidar data also influence the ICP's convergence, sometimes causing minor discrepancies in alignment. Despite these challenges, the overall improvement from odometry-only to ICP-corrected trajectories is evident. Future enhancements, such as loop closure or advanced outlier rejection, could further improve global accuracy and reduce remaining errors.

### *D. Occupancy Map*

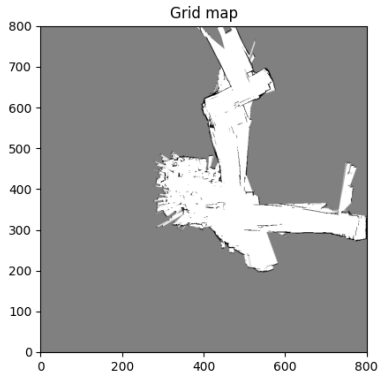



(a) First LiDAR Scan OGM for DataSet 20

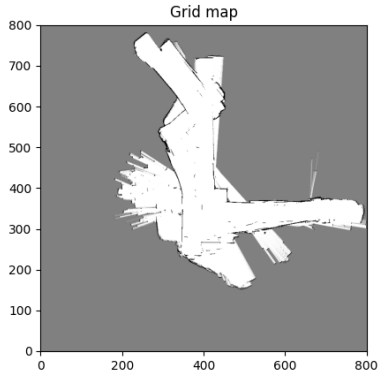



(b) First LiDAR Scan OGM for DataSet 21

Fig. 11: Occupancy grid maps built using only the first LiDAR scan for DataSet 20 and DataSet 21.




(a) Final Occupancy Grid for DataSet 20

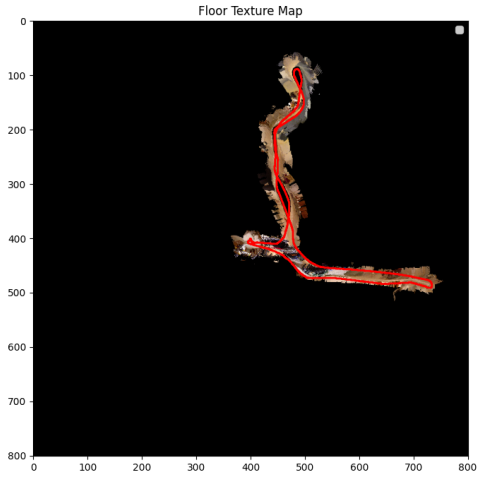



(b) Final Occupancy Grid for DataSet 21

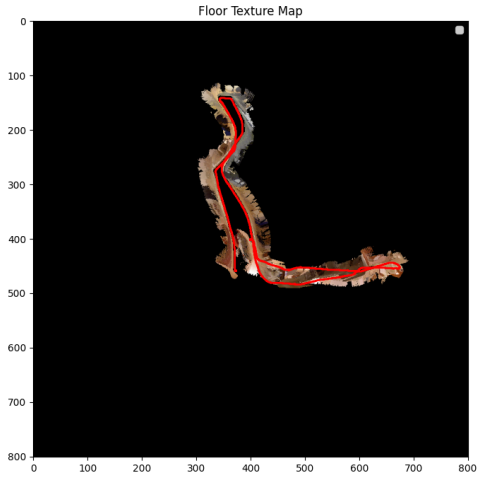
Fig. 12: Full occupancy grid maps for DataSet 20 and DataSet 21 .

The first and second images illustrate occupancy grid maps created from only the first LiDAR scan, capturing just a minimal slice of the environment. Because these scans are limited in coverage, the resulting maps appear relatively sparse and incomplete. This highlights how a single scan provides only an immediate snapshot and cannot represent the entire space. In contrast, the third and fourth images show the final occupancy grid maps for DataSet 20 and DataSet 21, which incorporate sensor data collected throughout the robot's run. These final maps are significantly more comprehensive, with extended corridors and structural details that were missing in the single-scan versions. Nevertheless, some areas remain partially filled or blank, possibly due to occlusions or insufficient coverage. Overall, the mapping approach effectively captures the main layout of the environment, but any drift in pose estimation can still introduce distortions. Techniques such as loop closure or enhanced sensor fusion could help further refine global consistency and fill in remaining gaps.

### E. Texture Map




(a) Floor Texture Map for DataSet 20




(b) Floor Texture Map for DataSet 21

Fig. 13: Floor texture maps for DataSet 20 and DataSet 21.

These figures illustrate the floor texture maps for DataSet 20 and DataSet 21, constructed by projecting Kinect RGB data onto the corresponding floor coordinates. The red lines indicate the final trajectory estimated via ICP and odometry fusion. Overall, the texture mapping successfully captures the visual appearance of the floor, with corridors and structural elements clearly delineated. Most regions where the robot traveled have dense color coverage, reflecting good sensor alignment. However, certain areas remain incomplete or exhibit distorted textures, possibly due to occlusions or insufficient overlap between RGB frames. Noise and slight inaccuracies in pose estimation can also introduce misalignment in the texture. Despite these challenges, the resulting maps provide a visually intuitive representation of the environment's floor layout. Incorporating loop closure and additional sensor fusion methods could further improve global consistency and fill in any remaining gaps in coverage.

### F. Pose Graph Optimization for Loop Closure

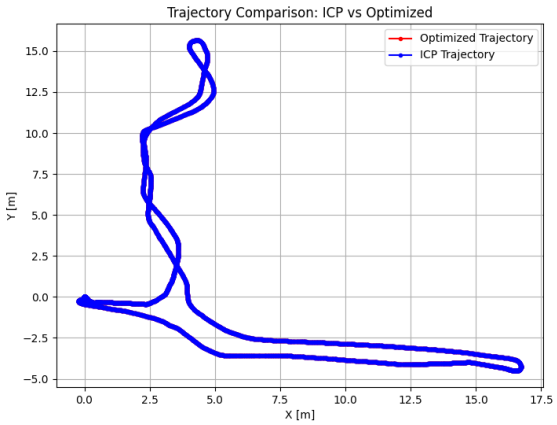



Fig. 14: ICP Trajectory and Optimized Trajectory

The results show that the optimized trajectory obtained from pose graph optimization is nearly identical to the ICP trajectory, indicating minimal improvement. The initial error between poses is close to zero, suggesting that ICP already produced a well-aligned trajectory with minimal drift. Since pose graph optimization relies on correcting errors through loop closure constraints, the lack of significant deviations implies that either no meaningful loop closures were detected, or the initial ICP trajectory was already accurate enough. To address this, I attempted several modifications to the loop closure detection parameters, including increasing the distance threshold to allow for more flexible detection and reducing the minimum index difference to capture loop closures earlier in the trajectory. Despite these efforts, the number of detected loop closures remained at zero, suggesting that either the trajectory did not contain valid revisits or that the detection criteria still needed further adjustments. Additionally, I adjusted the odometry and loop closure noise models to ensure that the optimizer was not overconfident in the initial ICP estimates, but this did not result in significant changes. The optimization process did not introduce noticeable improvements, which may indicate that ICP's relative transformations were already highly reliable, reducing

the need for additional corrections. Despite this, the stability of the trajectory confirms that the ICP and pose graph optimization implementations are functioning correctly, and future refinements should focus on enhancing loop closure detection methods and refining optimization weightings to capture small inconsistencies more effectively.