Project 2: LiDAR-Based SLAM

Shou-Yu Wang
Department of Electrical and Computer Engineering
UC San Diego
PID: A69030868

***Abstract*—This report presents the development and implementation of a LiDAR-based Simultaneous Localization and Mapping (SLAM) system for a differential-drive robot. The robot is equipped with encoders, an IMU, a Hokuyo LiDAR, and a Kinect RGB-D camera. Our approach integrates encoder and IMU odometry for an initial pose estimate, refines the trajectory using Iterative Closest Point (ICP) scan matching, and constructs both an occupancy grid map and a textured map of the environment. Experimental results demonstrate that the fusion of multiple sensor modalities effectively reduces drift and improves mapping accuracy.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental capability for autonomous robots operating in unknown environments. By combining observations from multiple sensors, a robot can incrementally construct a map while estimating its own trajectory within that map. In this project, we focus on a differential-drive robot equipped with:

- **Encoders and IMU:** Providing odometry estimates (linear velocity and yaw rate).
- **Hokuyo LiDAR:** Capturing 2-D range data for scan matching and occupancy grid mapping.
- **Kinect RGB-D:** Supplying depth and color images for floor texture mapping.

While integrating encoder and IMU data offers an initial odometry estimate, cumulative errors can become significant over time. To mitigate these errors, we employ LiDAR-based Iterative Closest Point (ICP) scan matching between consecutive scans. Furthermore, to address large-scale drift and ensure global consistency, we introduce pose graph optimization with loop closure constraints using GTSAM. This approach not only refines local estimates but also corrects global inconsistencies by detecting and closing loops in the trajectory. In the following sections, we detail our problem formulation and discuss each step of the pipeline, including odometry, ICP scan matching, occupancy grid construction, texture mapping, and pose graph optimization with loop closure.

II. Problem Formulation

The goal of this project is to accurately estimate the robot's trajectory and construct a 2-D occupancy grid map enriched with floor texture. To achieve this, we decompose the problem into five main components: (1) odometry estimation, (2) LiDAR-based scan matching, (3) occupancy grid mapping, (4) texture mapping, and (5) pose graph optimization with loop closure. Each component is outlined below.

A. Odometry Estimation

Let the robot pose at time step t be

$$T_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix},$$

where (x_t, y_t) is the position in the plane and θ_t is the heading. From wheel encoders, we derive the linear velocity v_t, while the IMU provides the yaw rate ω_t. These values are integrated using a differential-drive motion model:

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

This odometry model yields a baseline trajectory that will later be refined.

B. Scan Matching via ICP

To reduce cumulative drift in odometry, we apply Iterative Closest Point (ICP) to align consecutive 2-D LiDAR scans. Each scan is converted from polar to Cartesian coordinates, yielding a set of points:

$$\{\mathbf{p}_i\}_{i=1}^{N}.$$

Given two consecutive scans, we estimate a rigid transformation $(\mathbf{R}, \mathbf{t})$ that minimizes the sum of squared distances between corresponding points:

$$\min_{\mathbf{R},\mathbf{t}} \sum_i \left\| \mathbf{p}_i^{(\text{target})} - \left(\mathbf{R}\,\mathbf{p}_i^{(\text{source})} + \mathbf{t}\right) \right\|^2,$$

where $\mathbf{R} \in SO(2)$ is a 2-D rotation and $\mathbf{t} \in \mathbb{R}^2$ is a translation vector. The resulting transform refines the relative pose between consecutive scans.

C. Occupancy Grid Mapping

Using the refined robot poses, each LiDAR scan is transformed into the global coordinate frame to update a 2-D occupancy grid map. Let $m_{x,y}$ denote the state of a cell in the grid (occupied or free). We store the log-odds representation:

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

When a laser beam endpoint falls into a particular cell, we increment its occupancy log-odds; when a beam passes through a cell, we decrement it. Over time, this generates a consistent occupancy map of the environment.

D. Texture Mapping

We further enhance the occupancy map using Kinect RGB-D data. By converting each depth image into 3-D points in the camera frame and then transforming them into the world frame, we can identify floor points and project their corresponding RGB values onto the occupancy grid. The result is a textured map that provides both geometric and visual information about the environment.

E. Pose Graph Optimization and Loop Closure

Although ICP reduces drift locally, the trajectory may still accumulate global errors in large-scale environments. To address this, we employ pose graph optimization with loop closure constraints via GTSAM:

1) **Factor Graph Construction:** We create nodes for robot poses at each time step and add edges (factors) representing measurements. These include odometry factors derived from encoders/IMU, ICP factors from matching consecutive scans, and additional loop closure factors.
2) **Loop Closure Detection:** We periodically compare current scans with past scans. If a match is found (e.g., via ICP) that indicates revisiting the same area, we add a loop closure factor linking the corresponding nodes in the graph.
3) **Optimization:** We solve a nonlinear least-squares problem over all poses, minimizing the sum of errors from each factor. This globally adjusts the trajectory to ensure consistency and corrects large-scale drift.

Once optimized poses are obtained, we re-map the environment using the updated trajectory to produce a more accurate occupancy and texture map.

III. Technical Approach

A. Odometry Estimation

The first step in our SLAM pipeline is to estimate the robot's pose using encoder and IMU data. In a differential-drive robot, the motion is characterized by the distance traveled by the left and right wheels and the resulting change in orientation. Given the encoder counts and the corresponding time stamps, we compute the incremental motion as follows.

Assume that the robot is equipped with four wheel encoders, with counts given as:

$$\text{Counts} = \begin{bmatrix} \text{FR} \\ \text{FL} \\ \text{RR} \\ \text{RL} \end{bmatrix},$$

where FR, FL, RR, and RL represent the front-right, front-left, rear-right, and rear-left wheel counts, respectively. Each count is converted to a linear distance using the conversion factor:

$$\ell = 0.0022 \text{ m/tick}.$$

Thus, the distance traveled by the right wheels and left wheels over an interval are given by:

$$d_{\text{right}} = \frac{\text{FR} + \text{RR}}{2}\,\ell, \quad d_{\text{left}} = \frac{\text{FL} + \text{RL}}{2}\,\ell.$$

The average distance traveled during a time interval Δt is then:

$$d = \frac{d_{\text{right}} + d_{\text{left}}}{2}.$$

This leads to the linear velocity:

$$v = \frac{d}{\Delta t}.$$

In parallel, the IMU provides the yaw rate ω (angular velocity about the z-axis), which is sampled at a higher frequency than the encoders. For each encoder interval, we choose the yaw rate corresponding to the midpoint of the time interval.

The differential-drive motion model then updates the robot's pose $\mathbf{x}_t = \begin{bmatrix} x_t & y_t & \theta_t \end{bmatrix}^T$ using:

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

where:

- v_t is the linear velocity computed from the encoder data,
- ω_t is the yaw rate from the IMU,
- $\Delta t = t_{i+1} - t_i$ is the time difference between consecutive encoder readings.

This integration process is performed sequentially over all time intervals, starting from an initial pose $\mathbf{x}_0 = [0,\, 0,\, 0]^T$. The result is an initial trajectory estimate, which forms the basis for further refinement using LiDAR scan matching and pose graph optimization.

Mathematically, this can be summarized as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t \oplus \Delta\mathbf{x}_t,$$

where the motion increment is:

$$\Delta\mathbf{x}_t = \begin{bmatrix} v_t\,\Delta t\,\cos(\theta_t) \\ v_t\,\Delta t\,\sin(\theta_t) \\ \omega_t\,\Delta t \end{bmatrix}.$$

The operator $\oplus$ denotes the composition of the current pose with the incremental motion, yielding the updated pose.

This odometry estimation provides a baseline trajectory that, despite being subject to cumulative errors, is later refined by incorporating LiDAR scan matching and pose graph optimization with loop closure constraints.

B. ICP (Iterative Closest Point) Algorithm

The ICP algorithm is used to perform rigid registration between two point clouds. Its objective is to find the optimal rotation matrix R and translation vector t such that the transformed source point cloud aligns with the target point cloud, minimizing the mean squared error between corresponding points.

1) Objective and Fundamental Principle: The basic steps of the ICP algorithm are:

1) **Correspondence Search:** For each point in the source point cloud, a nearest-neighbor search (e.g., using a KD-Tree) is performed in the target point cloud to find the closest point. To eliminate outliers, only correspondences with a distance less than a threshold d_{th} are retained.
2) **Rigid Transformation Estimation:** Given the set of correspondences, compute the centroids of the source and target point clouds:

$$\bar{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i, \quad \bar{\mathbf{q}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{q}_i.$$

The point clouds are then centered by subtracting their respective centroids:

$$\mathbf{p}_i' = \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{q}_i' = \mathbf{q}_i - \bar{\mathbf{q}}.$$

Next, form the covariance matrix:

$$H = \sum_{i=1}^{N}\mathbf{p}_i'(\mathbf{q}_i')^T.$$

Perform Singular Value Decomposition (SVD) on H:

$$H = U\Sigma V^T.$$

The optimal rotation matrix and translation vector are computed as:

$$R = V\,U^T, \quad t = \bar{\mathbf{q}} - R\,\bar{\mathbf{p}}.$$

If $\det(R) < 0$, the SVD result is adjusted to ensure a proper rotation.
3) **Iterative Update:** The transformation (R, t) is applied to the source point cloud, and the correspondence search and transformation estimation are repeated until the change in mean squared error is below a predefined tolerance or the maximum number of iterations is reached.

This optimization can be formulated as:

$$\min_{R,\,t}\sum_{i\in\mathcal{C}}\left\|\mathbf{q}_i - (R\,\mathbf{p}_i + t)\right\|^2,$$

where $\mathcal{C}$ is the set of valid correspondences.

2) Implementation Details: Our implementation, as described in `test_icp.py` and `utils.py`, includes the following components:

- **Voxel Downsampling:** To reduce computational load, we use Open3D's voxel downsampling. By setting a voxel size (e.g., 0.05 m for coarse alignment or 0.01 m for fine alignment), we reduce the number of points while preserving the overall structure.
- **Rigid Transformation Computation:** For a given pair of corresponding points, we compute the centroids and form the covariance matrix H. The SVD of H yields the rotation R and translation t. This step is implemented in the function `compute_rigid_transform_3d`, which includes normalization to ensure that R has a positive determinant.
- **ICP Main Loop:** In the function `icp_3d`, we build a KD-Tree for the target point cloud to accelerate nearest-neighbor search. In each iteration, the source point cloud is transformed, and correspondences are filtered using a distance threshold. If insufficient inliers are found, the threshold is dynamically increased. The incremental transformation is then computed and accumulated until convergence.
- **Multi-Scale ICP:** To improve registration accuracy, we perform ICP in two stages:
  1) **Coarse Alignment:** Use a larger voxel size (e.g., 0.05 m) and a looser threshold (e.g., 0.2 m) to obtain a rough transformation.
  2) **Fine Alignment:** Apply the coarse result to the original source point cloud, then perform ICP with a smaller voxel size (e.g., 0.01 m) and a tighter threshold (e.g., 0.05 m) for refinement.

  The final transformation is obtained by composing the coarse and fine transformations.

C. Scan Matching

After implementing the ICP algorithm, we apply it for scan matching between consecutive LiDAR scans. The process is as follows:

1) **Data Conversion:** Each LiDAR scan is originally in polar coordinates. We convert the scan data into 2D Cartesian coordinates using:

$$x = r\cos(\theta), \quad y = r\sin(\theta),$$

where r is the measured range and θ is the corresponding angle. Points outside the valid range (between `range_min` and `range_max`) are discarded.
2) **Initial Guess from Odometry:** Using odometry data (from encoders and IMU), an initial relative transformation $(R_{\text{init}}, t_{\text{init}})$ between consecutive scans is computed. This initial estimate accelerates convergence and helps avoid local minima.
3) **Applying ICP for Refinement:** The initial guess is used as the starting point for the ICP algorithm. The function `icp_2d` (or its 3D counterpart) refines the alignment by computing the optimal rigid transformation $(R_{\text{ICP}}, t_{\text{ICP}})$ that minimizes the mean squared error between the matched points.
4) **Global Trajectory Integration:** The refined transformation from ICP is then accumulated with the previous global pose to update the robot's overall trajectory:

$$T_{t+1} = T_t \cdot \begin{bmatrix} R_{\text{ICP}} & t_{\text{ICP}} \\ 0 & 1 \end{bmatrix}.$$

This accumulation is performed for each pair of consecutive scans, yielding a refined global trajectory.

In our `scan_matching.py` module, the following steps are executed:

- LiDAR scan data is loaded and converted from polar to Cartesian coordinates using the function `polar_to_cartesian`.
- The corresponding odometry data (from `odometry.py`) is used to obtain an initial pose guess for each scan.
- For each consecutive pair of scans, ICP is applied to compute the refined relative transformation. In some experiments, we even swap the source and target inputs to verify the impact on trajectory estimation.
- The relative transformation is accumulated to produce the global ICP-refined trajectory.
- Finally, the raw odometry trajectory and the ICP-refined trajectory are plotted for comparison.

By combining the precise point cloud registration from ICP with the initial odometry estimates, our scan matching process effectively reduces drift and enhances the overall accuracy of the SLAM system.

D. Occupancy and Texture Mapping

To build a complete map of the environment, our system constructs a 2D occupancy grid map from LiDAR data and overlays texture from Kinect RGB-D images. The process is divided into two parts: occupancy mapping and texture mapping.

1) Occupancy Mapping: **1) LiDAR Data Conversion:** Convert each LiDAR scan from polar to Cartesian coordinates:

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

where r_i is the range and θ_i is the angle of the i-th measurement.

2) Coordinate Transformation: Transform points from the LiDAR (sensor) frame to the world frame using the homogeneous transformation matrix T obtained from the robot's pose:

$$\mathbf{p}^w = T\,\mathbf{p}^s, \quad \text{with} \quad T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

where R is the rotation matrix and t is the translation vector.

3) Grid Map Update: The occupancy grid map is represented by the log-odds of occupancy:

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

For each transformed LiDAR point:

- Increase $\ell(m_{x,y})$ for cells hit by a laser beam.
- Decrease $\ell(m_{x,y})$ for cells along the beam's path (using, e.g., Bresenham's algorithm).

The update rule is:

$$\ell_{\text{new}}(m_{x,y}) = \ell_{\text{old}}(m_{x,y}) + \Delta\ell$$

with $\Delta\ell > 0$ for occupied measurements and $\Delta\ell < 0$ for free space.

2) Texture Mapping: **1) 3D Point Reconstruction:** For each Kinect RGB-D frame, compute the 3D point in the camera frame using the intrinsic matrix K:

$$\mathbf{p}^c = d \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

where d is the depth at pixel (u, v).

2) Frame Transformation: Transform the reconstructed point cloud from the camera frame to the world frame:

$$\mathbf{p}^w = T_{wc}\,\mathbf{p}^c$$

where T_{wc} is the camera-to-world transformation.

3) Floor Plane Extraction: Select points corresponding to the floor by thresholding the z-coordinate (e.g., $z \approx 0$).

4) Color Projection: Project the floor points onto the occupancy grid. For each grid cell, assign a color $C(m_{x,y})$ based on the projected RGB values. A simple approach is to compute the weighted average:

$$C(m_{x,y}) = \frac{1}{N} \sum_{i=1}^{N} C_i$$

where C_i are the RGB colors of the N points projected into cell $m_{x,y}$.

Final Integration: The occupancy grid map with log-odds $\ell(m_{x,y})$ is combined with the texture map $C(m_{x,y})$ to form a textured 2D map of the environment, which can be visualized for navigation and further analysis.

E. Pose Graph Optimization and Loop Closure

Even with accurate odometry and ICP-based scan matching, long-term drift may accumulate in large-scale environments. To address this, we introduce pose graph optimization with loop closure constraints. The fundamental idea is to treat each robot pose as a node in a factor graph, and to add edges (factors) that represent relative pose measurements between pairs of nodes. Once loop closure constraints are incorporated, the factor graph can be optimized to reduce global inconsistencies.

1) Factor Graph Construction: Let $\mathbf{x}_t \in \mathcal{SE}(2)$ (or $\mathcal{SE}(3)$ in a 3D formulation) denote the robot pose at time t. We place a node in the factor graph for each $\mathbf{x}_t$. The edges (factors) capture measurements or constraints between pairs of poses:

1) **Odometry Factors:** Each consecutive pair $(\mathbf{x}_t, \mathbf{x}_{t+1})$ is connected by a factor derived from odometry or ICP scan matching, typically modeled as:

$$\mathbf{z}_{t,t+1} = f(\mathbf{x}_t, \mathbf{x}_{t+1}) + \boldsymbol{\epsilon},$$

where $\mathbf{z}_{t,t+1}$ is the measured relative transform, f is the motion model, and $\boldsymbol{\epsilon}$ is measurement noise.

2) **Loop Closure Factors:** For non-consecutive poses $(\mathbf{x}_i, \mathbf{x}_j)$, a loop closure factor is added if we detect that the robot has revisited the same physical location. The relative pose measurement $\mathbf{z}_{i,j}$ then enforces an additional constraint:

$$\mathbf{z}_{i,j} = f(\mathbf{x}_i, \mathbf{x}_j) + \boldsymbol{\epsilon}.$$

These loop closure factors provide global corrections by "closing the loop" and thereby reduce large-scale drift.

2) Optimization Formulation: Once the factor graph is constructed, the goal is to estimate all poses $\{\mathbf{x}_t\}$ that minimize the sum of squared errors over all factors:

$$\min_{\{\mathbf{x}_t\}} \sum_{(i,j)\in\mathcal{E}} \rho\Big(\|\mathbf{z}_{i,j} - f(\mathbf{x}_i, \mathbf{x}_j)\|^2\Big),$$

where $\mathcal{E}$ is the set of all edges (odometry and loop closure), and $\rho(\cdot)$ is a robust cost function that can reduce the impact of outliers. This non-linear least-squares problem is typically solved using iterative solvers such as Gauss-Newton or Levenberg-Marquardt. The result is a globally consistent set of poses $\{\hat{\mathbf{x}}_t\}$.

3) Loop Closure Detection: A critical aspect of pose graph optimization is detecting loop closures. Several strategies can be employed:

- **Fixed-Interval Detection:** Every k poses, compare the current pose with one from k steps earlier. If the difference in scans or estimated poses is below a threshold, a loop closure factor is added.
- **Proximity-Based Detection:** If the current pose is spatially close to a previously visited region, compare LiDAR scans or other measurements to confirm whether the robot has revisited the same area. A loop closure factor is added when a match is found.

4) Refined Trajectory and Mapping: After solving for $\{\hat{\mathbf{x}}_t\}$, the refined trajectory is used to construct a more accurate occupancy grid and texture map. By applying each LiDAR scan and RGB-D data to the optimized poses, we obtain a globally consistent map that corrects large-scale drift. This final map incorporates both local alignment (from ICP) and global consistency (from loop closures), resulting in improved overall accuracy.

IV. Results

This section presents the experimental outcomes of our SLAM system, covering each major component: Odometry Estimation, ICP (Iterative Closest Point) Algorithm, Scan Matching, Occupancy and Texture Mapping, and Pose Graph Optimization with Loop Closure. We provide result plots and then discuss the performance, highlighting both successes and challenges.

A. Odometry Estimation

1) Brief Explanation: Our differential-drive odometry model integrates encoder counts (to compute linear velocity) and IMU yaw rate (to compute angular velocity). At each time step, the robot pose is updated with encoders and IMU data, respectively. This integration yields an initial estimate of the robot's global trajectory without any additional sensor corrections.

2) Result Plots: Figure 1 and Figure 2 show the odometry-only trajectories for Dataset 20 and Dataset 21, respectively. Each trajectory is plotted in the (x, y) plane, with the robot starting at the origin $(0, 0)$.

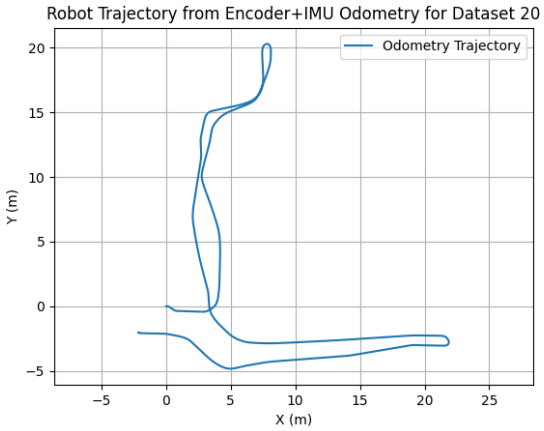

Fig. 1: Robot Trajectory from Encoder+IMU Odometry for Dataset 20.

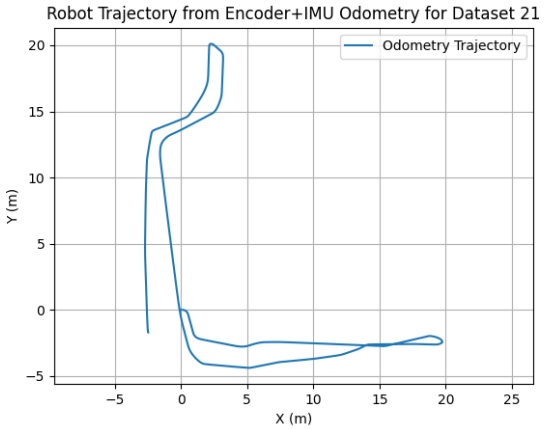

Fig. 2: Robot Trajectory from Encoder+IMU Odometry for Dataset 21.

3) Discussion and Analysis: From these plots, we observe that the robot traverses a relatively large area in both datasets, reaching over 20 m along the x-axis and up to 20 m in the y-axis. In both cases, the trajectory meanders and exhibits increasing drift over time. This behavior is expected since odometry is susceptible to accumulating errors due to wheel slip, encoder resolution limits, and IMU noise.

Overall, while odometry provides a feasible initial pose estimate, the accumulated drift highlights the need for additional sensor fusion (e.g., ICP scan matching, loop closure) to correct long-term errors and improve global consistency.

B. ICP (Iterative Closest Point) Algorithm

Brief Explanation: The ICP algorithm is applied to align point clouds obtained from different views of the same object. In this experiment, we test ICP on two canonical models: a drill and a liquid container. For each object, the algorithm refines the initial pose (obtained from a rough alignment)

by iteratively minimizing the distance between corresponding points in the source and target point clouds.

1) Drill: **Result Plots:** Figures 3–6 show the ICP alignment results for the drill model from four different viewpoints.

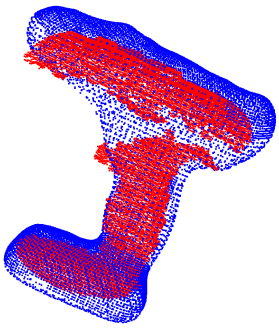

Fig. 3: Drill ICP result - View 0.

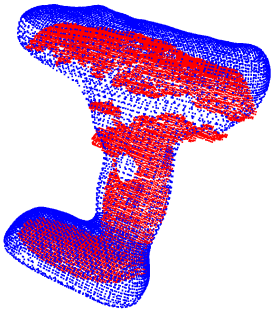

Fig. 4: Drill ICP result - View 1.

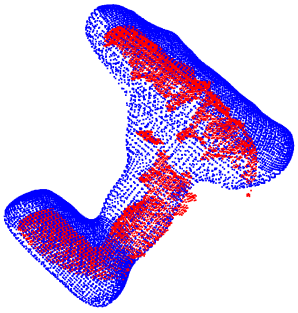

Fig. 5: Drill ICP result - View 2.

Discussion and Analysis: The drill results indicate that the ICP algorithm is able to successfully converge to a precise alignment, as evidenced by the minimal residual errors observed in Views 0–2. Minor misalignments occur in regions with less geometric structure, likely due to a lack of distinct features that aid correspondence matching. Notably, the fourth image (Figure 6) exhibits a significant deviation from the expected alignment. This pronounced misalignment may be attributed to several factors: an inadequate initial pose

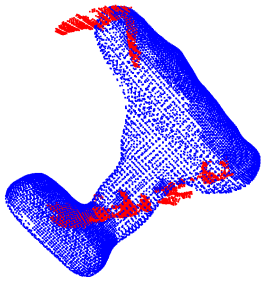

Fig. 6: Drill ICP result - View 3.

estimation for this particular view, occlusions that reduce the number of reliable correspondences, and insufficient overlap between the source and target point clouds. These issues can lead to ambiguous or erroneous correspondences, thereby degrading the quality of the transformation estimated by ICP. Overall, while the algorithm performs robustly when provided with a good initial guess, the result in Figure 6 underscores the sensitivity of ICP to initial conditions and the geometric richness of the scene.

2) Liquid Container: **Result Plots:** Figures 7–10 present the ICP results for the liquid container model from four different views.

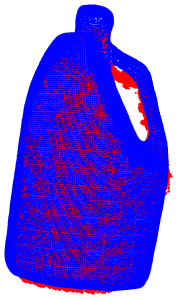

Fig. 7: Liquid Container ICP result - View 0.

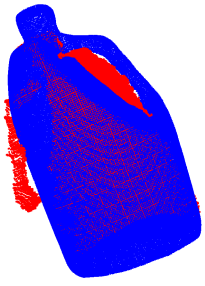

Fig. 8: Liquid Container ICP result - View 1.

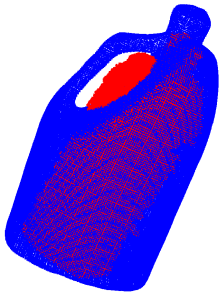

Fig. 9: Liquid Container ICP result - View 2.

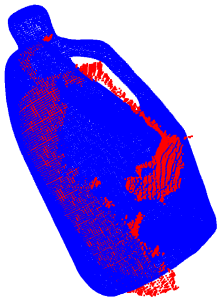

Fig. 10: Liquid Container ICP result - View 3.

Discussion and Analysis: The liquid container alignment results demonstrate that the ICP algorithm performs well across different viewing angles, achieving a tight registration between source and target point clouds. However, compared to the drill, the liquid container sometimes exhibits slightly higher residual errors, possibly due to smoother surfaces and fewer distinctive features, which can challenge the correspondence matching process. This suggests that incorporating feature-based refinement could further enhance the registration for objects with less texture.

C. Scan Matching

1) Brief Explanation: After estimating the robot's initial trajectory via odometry, we apply ICP-based scan matching between consecutive LiDAR scans. The odometry pose serves as the initial guess for each scan pair, and ICP refines the relative transformation. Accumulating these refined transformations produces an improved global trajectory, which is generally less affected by odometry drift.

2) Result Plots: Figures 11 and 12 show the odometry trajectory (in red, dashed) and the ICP-refined trajectory (in blue, solid) for Dataset 20 and Dataset 21, respectively.

3) Discussion and Analysis: In both datasets, the blue ICP trajectory stays closer to the center of the robot's overall path, reducing large deviations seen in the raw odometry (red). For Dataset 20 (Figure 11), ICP corrects several sharp turns that are likely caused by cumulative wheel slip and IMU noise. Similarly, in Dataset 21 (Figure 12), the refined path

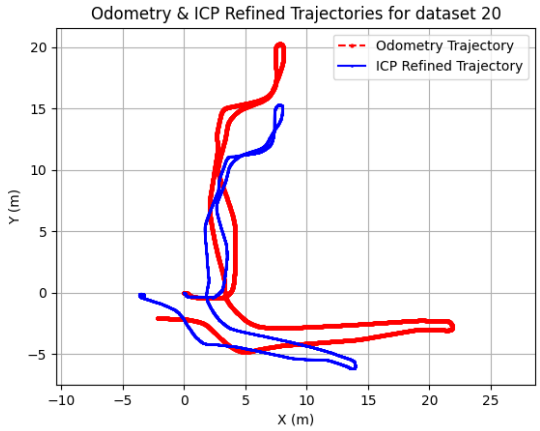

Fig. 11: Odometry and ICP Refined Trajectories for Dataset 20.

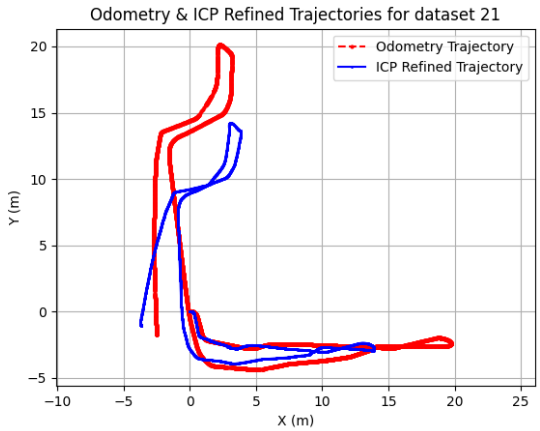

Fig. 12: Odometry and ICP Refined Trajectories for Dataset 21.

exhibits fewer erratic loops, suggesting that ICP successfully compensates for odometry drift. However, small discrepancies remain, especially in areas with limited LiDAR features or narrow corridors. These minor errors may compound over long distances, motivating the use of loop closure to achieve global consistency.

D. Occupancy and Texture Mapping

Brief Explanation: This module builds a 2D occupancy grid from LiDAR scans and, in a later step, overlays color information from the Kinect RGB-D sensor to produce a texture map. We divide our results into three parts:

1) Occupancy map generated from the first LiDAR scan
2) Occupancy map constructed using the full ICP-refined trajectory
3) Texture mapping using RGB-D data

1) Occupancy Map from First LiDAR Scan: At the initial stage, we use only the first LiDAR scan to test the correctness of our coordinate transforms and grid update logic. Each

valid LiDAR range measurement is converted to Cartesian coordinates and projected onto the occupancy grid. Free cells (traversed by the laser beam) have their log-odds decreased, while occupied cells (endpoints of the beam) have their log-odds increased.

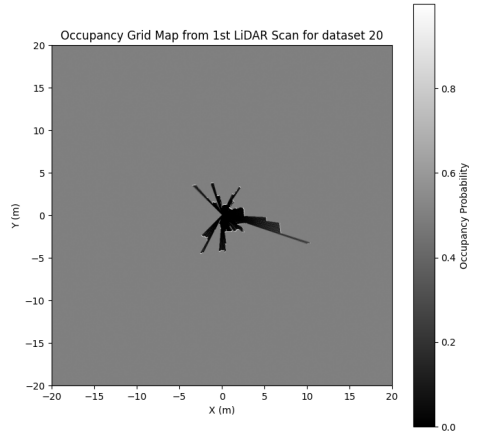

Fig. 13: Occupancy grid map from the first LiDAR scan for dataset 20.

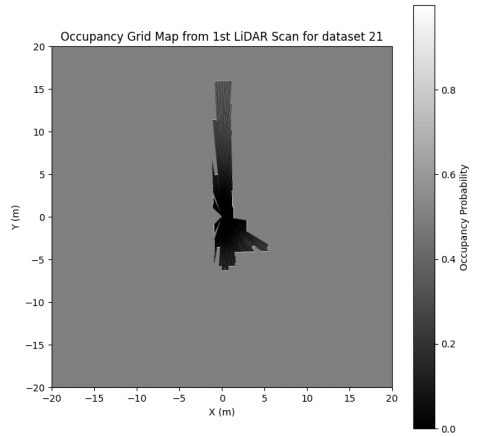

Fig. 14: Occupancy grid map from the first LiDAR scan for dataset 21.

Discussion and Analysis: As shown in Figures 13 and 14, the occupancy grid for a single scan covers only a narrow angular slice of the environment. Despite its limited coverage, this test confirms that the LiDAR data is being properly transformed into the grid coordinates, and that the log-odds updates are functioning correctly. Some artifacts may appear near the edges of the scan range due to sensor noise or maximum range cutoff.

2) Occupancy Map from ICP-Refined Trajectory: Once the ICP algorithm refines the robot trajectory, all LiDAR scans are integrated into the occupancy grid using the corrected poses. This approach reduces drift and yields a more coherent global map. We again employ a log-odds update rule for each LiDAR beam.

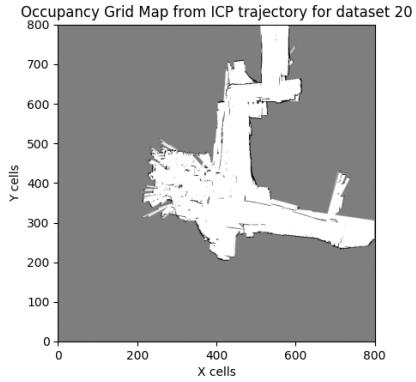

Fig. 15: Occupancy grid map from the ICP-refined trajectory for dataset 20.

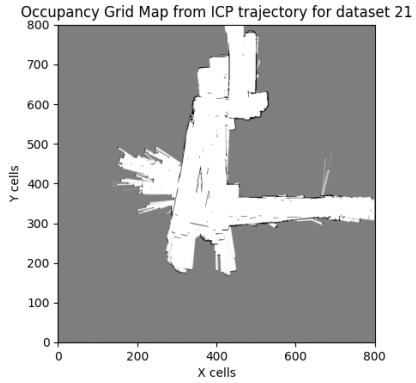

Fig. 16: Occupancy grid map from the ICP-refined trajectory for dataset 21.

Discussion and Analysis: Figures 15 and 16 show significantly larger coverage and improved consistency compared to the single-scan maps. The integrated scans fill out the environment in more detail, with fewer distortions caused by odometry drift. However, some local inconsistencies or "blurring" may still appear in regions where ICP struggled to find a good alignment (e.g., feature-sparse corridors). Future loop closure or global optimization can further enhance map quality.

3) Texture Mapping: In addition to occupancy information, we overlay RGB-D data from the Kinect camera to generate a color-textured map of the floor. The process consists of the following steps:

1) **Point Cloud Generation:** Convert depth images into 3D points in the camera frame using the Kinect intrinsic

parameters.

2) **Transformation to World Frame:** Apply the robot's pose transformation to align the points with the occupancy grid.
3) **Texture Projection:** Assign RGB values from the Kinect images to the transformed 3D points and project them onto the occupancy grid.
4) **Blending and Visualization:** Merge textures from multiple frames to create a consistent floor map representation.

Result Plots: Figures 17 and 18 display the generated texture maps for Dataset 20 and Dataset 21.

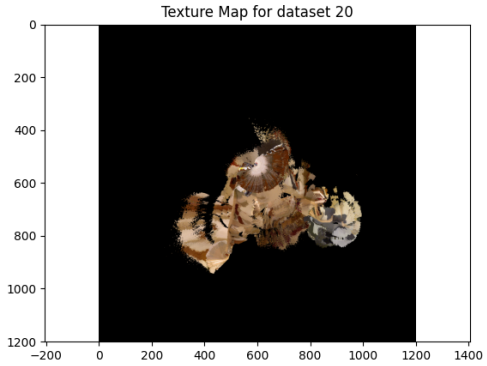

Fig. 17: Texture map generated for Dataset 20.

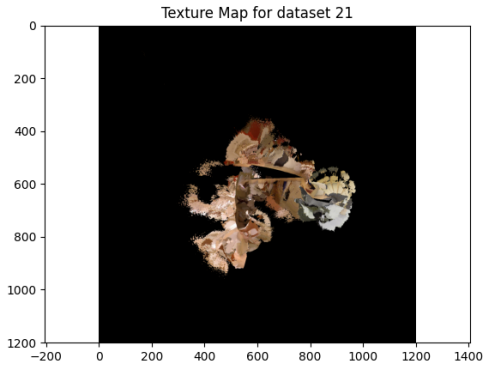

Fig. 18: Texture map generated for Dataset 21.

Discussion and Analysis:
Potential Solutions:

- **Camera Calibration Refinement:** Verify and refine the intrinsic and extrinsic calibration of the Kinect camera relative to the LiDAR and robot frame.
- **Improved Pose Estimation:** Enhance trajectory estimation using loop closure and pose graph optimization to reduce errors in point cloud alignment.
- **Texture Blending Enhancements:** Implement weighted blending techniques to prioritize textures from stable viewpoints and prevent texture artifacts from occluded or inconsistent frames.

Addressing these issues will improve the accuracy and visual quality of the final textured map, providing better environmental representation for navigation and visualization.

E. Pose Graph Optimization and Loop Closure

Pose graph optimization and loop closure play a crucial role in minimizing drift and improving the global consistency of the estimated robot trajectory. Without these refinements, accumulated errors from odometry and scan matching can lead to significant trajectory deviations over time. The process consists of three main components:

1) **Factor Graph Representation:** The trajectory is represented as a factor graph, where each node corresponds to a robot pose at a specific timestep, and edges encode constraints between poses. These constraints arise from odometry, scan matching, and loop closure detections.
2) **Loop Closure Detection:** Detecting when the robot revisits a previously explored location is key to correcting accumulated errors. We consider two methods:
   - *Fixed-interval loop closure:* Introducing loop closure constraints at regular pose intervals to improve global consistency.
   - *Proximity-based loop closure:* Identifying loop closures by comparing LiDAR scans at nearby poses and applying a similarity metric.
3) **Graph Optimization:** Given the factor graph, optimization techniques such as non-linear least squares (e.g., Levenberg-Marquardt) or g2o/GTSAM frameworks are used to find the most probable set of robot poses that satisfy all constraints.

Discussion and Future Work: While the implementation framework for pose graph optimization and loop closure has been outlined, the full implementation and evaluation are still pending. A key challenge in this process is ensuring robust loop closure detection, especially in environments with repetitive structures where false positives may occur. Additionally, efficient optimization is required to handle large-scale datasets without excessive computational overhead. Future improvements may include integrating additional sensor modalities (e.g., visual features) to strengthen loop closure constraints and employing robust outlier rejection methods to enhance pose graph stability.

F. Overall Discussion and Conclusions

This project successfully implemented a LiDAR-based SLAM system that integrates multiple sensor modalities, including encoders, an IMU, a Hokuyo LiDAR, and a Kinect RGB-D camera. Each component of the SLAM pipeline was designed and evaluated individually, contributing to a more robust localization and mapping system.

Summary of Key Findings:

- **Odometry Estimation:** The differential-drive odometry model provided a reasonable initial estimate of the robot

trajectory. However, as expected, significant drift accumulated over time due to wheel slip and sensor noise.

- **ICP Scan Matching:** ICP effectively refined the trajectory by aligning consecutive LiDAR scans. While the method reduced drift compared to odometry alone, its accuracy was highly dependent on good initial alignment and sufficient overlapping features between scans.
- **Scan Matching:** The comparison of odometry-based and ICP-refined trajectories confirmed that ICP successfully corrected many local errors, but some global inconsistencies remained, motivating the need for loop closure.
- **Occupancy Mapping:** The occupancy grid map was constructed using both raw LiDAR data and the ICP-refined trajectory. The latter resulted in a more accurate and consistent map, demonstrating the benefits of trajectory refinement.
- **Texture Mapping:** The texture mapping results revealed potential misalignment issues, likely due to incorrect camera calibration or trajectory errors. Further improvements in pose estimation and synchronization between RGB-D and LiDAR data are required.
- **Pose Graph Optimization and Loop Closure:** Although the theoretical framework for pose graph optimization was established, a complete implementation is still pending. This step is expected to significantly improve global trajectory consistency and further reduce drift.

Challenges and Limitations:

- **Sensor Synchronization:** The lack of precise synchronization between the LiDAR, IMU, and RGB-D camera led to inconsistencies in the texture mapping step.
- **ICP Limitations:** While ICP improved local scan alignment, it occasionally failed in feature-sparse environments where scans had insufficient overlap.
- **Loop Closure Implementation:** The absence of a fully implemented loop closure mechanism meant that large-scale drift was not fully corrected. Future work should focus on developing an effective loop closure detection strategy.

Future Work and Improvements:

- Implement pose graph optimization with loop closure constraints to achieve a globally consistent trajectory.
- Improve texture mapping by refining Kinect-LiDAR calibration and ensuring accurate depth-to-color alignment.
- Enhance ICP robustness by incorporating feature-based registration techniques or using multi-resolution approaches.
- Experiment with alternative scan matching techniques, such as NDT (Normal Distributions Transform), for better performance in feature-sparse environments.
- Optimize computational efficiency to allow real-time SLAM processing.

Overall, the integration of multiple SLAM components demonstrated the effectiveness of LiDAR-based localization and mapping while highlighting the necessity of loop closure for long-term accuracy. The results provide a strong foundation for further refinement and enhancement of the system.