ECE 276A Project 2: LiDAR-Based SLAM

Winston Chou
PID: A17460970

***Abstract*—This paper presents a comprehensive approach to Simultaneous Localization and Mapping (SLAM) using multi-sensor data fusion and post-analysis optimization. The proposed method integrates data from wheel encoders, Inertial Measurement Units (IMUs), 2-D LiDAR, and RGBD cameras on a differential-drive robot. We develop two parallel trajectory estimations: one based on differential-drive kinematics using wheel encoder and IMU data, and another observation model using LiDAR scans and the Iterative Closest Points (ICP) method. The trajectories are then refined and optimized using Factor Graph techniques and loop closure detection. This multi-faceted approach enhances SLAM performance by leveraging the strengths of each sensor and optimization method, improving accuracy and robustness in diverse environments.**

Keywords—multi-sensor fusion, differential drive, IMU, LiDAR, RGBD camera, pose-graph optimization, iterative closest point (ICP), factor graph, trajectory estimation

I. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) has emerged as a fundamental capability for autonomous mobile robots, enabling them to navigate and map unknown environments without prior knowledge. This paper presents an advanced approach to SLAM, integrating multi-sensor data fusion and post-analysis optimization techniques to enhance accuracy and robustness. SLAM addresses the challenge of a robot building a map of its surroundings while simultaneously determining its location within that map. This interdependent process is crucial for autonomous navigation, as it allows robots to operate in unfamiliar settings with minimal human intervention.

Our research focuses on a differential-drive robot equipped with wheel encoders, an Inertial Measurement Unit (IMU), a 2-D LiDAR scanner, and an RGBD camera. I develop a dual trajectory estimation approach:

1) A motion model based on differential-drive kinematics, utilizing wheel encoder and IMU data.
2) An observation model derived from LiDAR scans using the Iterative Closest Point (ICP) method.

These trajectories are then refined and optimized using Factor Graph techniques and loop closure detection, significantly improving the accuracy of both localization and mapping.

By leveraging the strengths of multiple sensors and advanced optimization methods, our approach aims to overcome common SLAM challenges such as sensor noise, data association errors, and loop closure detection. The proposed system demonstrates enhanced performance in diverse environments, including those with dynamic obstacles and varying terrain.

This paper contributes to the ongoing development of SLAM technologies, which continue to play a vital role in robotics, autonomous vehicles, and other applications requiring precise environmental mapping and self-localization.

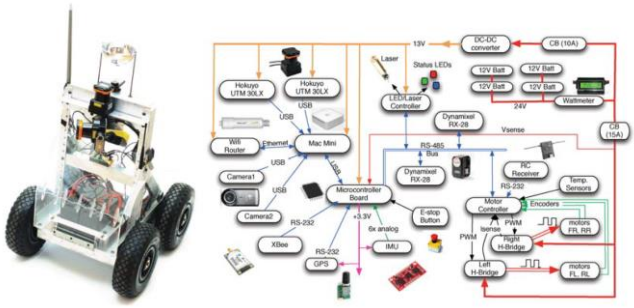


Fig. 1. Sensor Setup. Differential-drive robot, equipped with encoders, IMU, 2-D LIDAR scanner, and an RGBD camera. [Source: ECE276A_PR2.pdf]

II. PROBLEM FORMULATION

The core of the SLAM problem lies in estimating the robot's trajectory and constructing a map of the environment simultaneously. The robot's trajectory is discretized as a set of pose and timestamp tuples, which serve as the foundation for computation. Let the robot's pose at a given timestamp t be denoted as P_t. The pose can be represented in both two-dimensional (2D) and three-dimensional (3D) notation, with the z-axis value fixed at zero for simplicity.

$$P_t = \begin{cases} =^{2D} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \in \mathbb{R}^3 \\ =^{3D} \begin{bmatrix} R_z(\theta) & \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \end{cases}$$

Where x, y is the robot's position in the plane, θ is the orientation (yaw angle) of the robot, and $R_z(\theta)$ is rotation matrix with respect to the z-axis that encodes the yaw angle in 3D space.

Initial Pose, P_0, at time $t = 0$, is defined as the world frame, which serves as the reference frame for all subsequent poses. It is represented as:

$$P_0 = \begin{cases} =^{2D} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3 \\ =^{3D} \begin{bmatrix} R_z(0) & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ 0 & 1 \end{bmatrix} = I_{4\times4} \end{cases}$$

A. Motion Model: Differential-Drive Kinematics & IMU

To estimate the robot's motion, we employ a differential-drive kinematics model combined with data from the wheel encoders and IMU. For a small-time interval, τ, between two consecutive timestamps $t-1$ and t, the wheel encoders and IMU provide the traveled distances δd (in meters, for FL, FR, RL, RR wheels) and the rotation $\delta\theta$ (in radians). The robot's motion is described by its linear and angular velocities:

$$v_L = \frac{\delta d_{FL} + \delta d_{RL}}{2\tau}$$

$$v_R = \frac{\delta d_{FR} + \delta d_{RR}}{2\tau}$$
$$v = \frac{v_R + v_L}{2}$$

where v_L, v_R are the discrete instantaneous left and right velocities computed from the encoder readings, and τ is the time difference between measurements. Differential-drive kinematics shows that the velocity in body frame x-direction, v, can be obtained from left and right velocities.

For orientation, we use angular velocities, ω gained from IMU readings. Specifically, we only care about yaw rate, ω_z. After getting both linear and angular velocities, we formulate a twist vector, and take its matrix form:

$$V_b = \begin{bmatrix} 0 \\ 0 \\ \omega_z \\ v \\ 0 \\ 0 \end{bmatrix}, \qquad \widehat{V_b} = \begin{bmatrix} 0 & -\omega_z & 0 & v \\ \omega_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Using the matrix exponential, the current pose is updated from the previous pose:

$$P_t = P_{t-1} * e^{\widehat{V_b}\tau}$$

Where V_b is the twist vector, and P_{t-1} and P_t are previous and current poses. This formulation ensures accurate integration of both linear and angular velocities over time.

B. Observation Model: LiDAR & Scan Matching with ICP

Scan matching on 2D LiDAR can estimate relative transformations between observations. and is used as observation model.

At timestamp t, a LiDAR scan provides a set of range and angle measurements:

$$L_t = \{(r_i, \phi_i) \mid i = 1, 2, \ldots, n\}$$

These measurements are converted into a point cloud in the LiDAR frame:

$$PC_{LiDAR\,frame} = \left\{ \begin{bmatrix} r_i \cos(\phi_i) \\ r_i \sin(\phi_i) \\ 0 \end{bmatrix}, \forall (r_i, \phi_i) \in L_t \right\}$$

$$\begin{bmatrix} PC_t \\ 1 \end{bmatrix} = {}_{body}\, T_{LiDAR} \begin{bmatrix} PC_{LiDAR\,frame} \\ 1 \end{bmatrix}$$

Due to no prior knowledge of data association between two LiDAR scans across a time step, in order to align consecutive scans, we use the Iterative Closest Point (ICP) algorithm:

At each timestamp, a LiDAR scan produces a homogeneous point cloud:

1) *Source point cloud:* $S = \{PC_s \quad 1\}^T$
2) *Target point cloud:* $T = \{PC_t \quad 1\}^T$

ICP Algorithm Steps:

1) <u>*Apply Initial Guess:*</u>
   Transform the source point cloud using the initial guess, T_0:
   $$S_k = T_0 \cdot S, \qquad k = 1$$

2) <u>*Point Association:*</u>
   For each point $s \in S_k$, find its nearest neighbor in the target point cloud T:
   $$\Delta = \{(s,t): t = argmin_{t \in T}(\|s - t\|),\ s \in S_k\}$$
   where Δ represents the set of correspondences between points in S_k and T.

3) <u>*Estimate Relative Transformation:*</u>
   Using the Kabsch algorithm, compute the optimal transformation T_k that minimizes the alignment error:
   $$T_k = argmin_{T \in SE(3)} \sum_{(s,t)\in\Delta} \|t - T \cdot s\|^2$$

4) <u>*Update Source Point Cloud:*</u>
   Update the source point cloud using the computed transformation:
   $$S_{k+1} = T_k \cdot S$$
   Increment k by 1, and repeat from Step 2 until convergence (i.e., no further improvement in alignment).

5) <u>*Compose Total Transformation and Error:*</u>
   After convergence, compose the total transformation and compute alignment error:
   - Total transformation:
     $${}_tT_s = \prod_k T_k$$
   - Alignment error:
     $${}_t\epsilon_s = \frac{1}{|\Delta|} \sum_{(s,t)\in\Delta} \|t - {}_tT_s \cdot s\|$$

Using ICP results, the relative transformation between consecutive timestamps is computed as:

$${}_{t-1}T_t = ICP(S = PC_{t-1}, T = PC_t)$$

The observation-based trajectory in 3D is then composed iteratively as:

$$P_t = P_0 \cdot \prod_{t'=1}^{t} {}_{t'-1}T_{t'}$$

C. Factor Graph and Loop Closure

A factor graph is a constrained optimization framework used to solve the SLAM problem. It represents the robot's states (poses) as nodes (random variables) and the constraints between these states as edges (factors). The goal of the factor graph is to optimize the robot's trajectory by finding the set of poses that maximizes the likelihood of all factors.

- ***Nodes****:* Represent the robot's states P_t at each timestamp t.
- ***Edges****:* Represent constraints, including motion constraints, observation constraints, and loop closure constraints.

The optimization process minimizes the overall error by adjusting the poses P_t to satisfy all constraints as closely as possible.

Loop closure is a special type of constraint that reduces accumulated error in SLAM. In typical SLAM systems, constraints are derived from consecutive poses based on the

Markov assumption, forming a single chain graph. However, this approach leads to error accumulation over time.

When the robot revisits a previously mapped area, it becomes possible to estimate the relative transformation between temporally distant but spatially close poses. This creates a loop in the factor graph, which helps reduce drift and accumulated error.

Loop Detection Criteria

Loop closure candidates lc are identified using the following criteria:

- *Maximum location difference:*
$$\left\|P_i[xy] - P_j[xy]\right\| < d^*$$
- *Maximum yaw difference:*
$$\left\|P_i[\theta] - P_j[\theta]\right\| < \theta^*$$
- *Minimum time interval:*
$$\|i - j\| > \tau^*$$

The set of loop closure pairs is defined as:

$$lc = \{(i,j): \left\|P_i[xy] - P_j[xy]\right\| < d^*, \left\|P_i[\theta] - P_j[\theta]\right\| < \theta^*, \|i - j\| > \tau^*\}$$

The factor graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is constructed with:

Vertices (V): Representing all robot poses $P_t, \forall t$.

Edges (E): Representing three types of constraints:

- **Motion Constraints**: Derived from differential-drive kinematics for consecutive poses:
$$P_t(2D) \ominus (P_{t-1}(2D) + \ {}_{t-1}O_t), \ \forall t$$
- **Observation Constraints**: Based on scan matching (ICP) for consecutive poses:
$$P_t(3D) \ominus (P_{t-1}(3D) \cdot \ {}_{t-1}T_t), \ \forall t$$
- **Loop Closure Constraints**: Applied to temporally distant but spatially proximate poses using ICP:
$$P_i(3D) \ominus \left(P_i(3D) \cdot ICP\left(S = PC_i, T = PC_j\right)\right), \forall (i,j) \in lc$$

The optimization process minimizes the overall error across all factors by solving:

$$argmin_{P_t}(E(G))$$

Where $E(G)$ is the total error induced by motion, observation, and loop closure constraints.

This approach ensures accurate trajectory estimation by leveraging both local motion models and global corrections from loop closures, significantly reducing drift and improving map consistency.

D. Mapping

Mapping involves reconstructing the surrounding environment using the robot's poses and sensor measurements. In this project, a 2D LiDAR scan is used to generate an occupancy grid map, while an RGBD camera provides texture mapping by combining RGB and depth information.

Occupancy Mapping with 2D LiDAR

The occupancy grid map represents the environment as a grid of cells, where each cell is classified as either occupied or free based on LiDAR measurements. The process is as follows:

1) ***LiDAR Data Conversion****:* Each LiDAR scan provides range and angle measurements (r_i, ϕ_i), which are converted to Cartesian coordinates in the LiDAR frame:

$$(x_i, y_i) = (r_i \cos(\phi_i), r_i \sin(\phi_i)), \quad \forall i$$

These points represent the endpoints of detected obstacles.

2) ***Occupied and Free Space:*** The endpoints are considered occupied. The space between the sensor origin $(0,0)$ in the LiDAR frame and the endpoints is considered free.

3) ***Transformation to World Frame:*** Transform the endpoints and the sensor origin to the robot's body frame using ${}_{body}T_{LiDAR}$. Transform these points further to the world frame using the robot's pose P_t.

4) ***Grid Representation****:* Discretize the world into a grid of cells. Use Bresenham's ray-tracing algorithm to determine which cells lie along the line between the sensor origin and each endpoint. Mark these cells as free, while marking the endpoints as occupied.

5) ***Occupancy Probability****:* For each cell, compute its occupancy probability using logistic regression over all scans. This probabilistic approach accounts for sensor noise and uncertainty.

Texture Mapping with RGBD Camera

Texture mapping uses RGBD camera data to create a detailed floor map by associating RGB values with spatial coordinates. The process includes:

1) ***Depth Calculation:*** For each pixel (u, v) in the disparity image, compute its depth z using a disparity-to-depth function:
$$z = g(d)$$
where d is the disparity value at pixel (u, v).
2) ***3D Point Reconstruction:*** Using the camera's intrinsic matrix K and projection function $\pi(\cdot)$, compute the corresponding 3D point in the camera frame:
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{camera} = {}_{optical}R^{-1}_{camera} \cdot g(d) \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$
3) ***Transformation to World Frame:*** Transform the point cloud from the camera frame to the body frame using ${}_{body}T_{camera}$, and further transform it to the world frame using robot pose P_t.
4) ***Floor Isolation:*** Isolate points near the floor by filtering out points where $\|z\| > z^*$ (a predefined threshold for floor height).
5) ***Texture Mapping:*** Map RGB values from each point to its corresponding cell in the grid based on its (x, y) coordinates in the world frame.

III. TECHNICAL APPROACH

The implementation of the project is presented as follows:

A. Sensor Data Interpolation on Timstamps

Robotic sensors operate at varying frequencies, and synchronization is not enforced. To align measurements temporally, interpolation is critical for accurate sensor fusion and downstream tasks. The following strategies are employed:

All interpolations are done by using “np.interp”.

- Wheel encoder data is linearly interpolated to match the IMU’s timestamp.
- The motion model’s poses (derived from differential-drive kinematics and IMU) are interpolated to LiDAR timestamps when providing a reliable initial guess for the ICP algorithm, accelerating convergence.
- Poses are interpolated at LiDAR timestamps in occupancy mapping.
- Poses are interpolated at camera timestamps in occupancy mapping.

B. Encoders Preprocess

The encoders count the rotations of the four wheels at 40 Hz. The encoder counter is reset after each reading. Therefore, we are actually dealing with a twist with a timestep. Encoder data from the four wheels (FL, FR, RL, RR) is processed to compute the traveled distance in meters. The steps are as follows:

$$\frac{\delta d_{count}}{360\ ticksPerRev} = \delta d_{rev}$$

$$\delta d_{rev} \cdot \pi \cdot d_{wheel} = \delta d_{meters}$$

This is how we convert encoder counts to distance in meters for all four encoders.

C. 2D and 3D Conversion

The poses often need to be converted between 2D and 3D representations.

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

D. Transform Tree

Based on the “RobotConfiguration.pdf”, the following transformations are required for converting reference frames.

$${}_{body}T_{LiDAR} = \begin{bmatrix} I_{3\times3} & \begin{bmatrix} 0.13323 \\ 0.0 \\ 0.51435 \end{bmatrix} \\ 0 & 1 \end{bmatrix}$$

$${}_{body}T_{camera} = \begin{bmatrix} R_z(0.021)R_y(0.36) & \begin{bmatrix} 0.18 \\ 0.0 \\ 0.36 \end{bmatrix} \\ 0 & 1 \end{bmatrix}$$

E. Iterative Closest Point (ICP)

The Iterative Closest Point (ICP) algorithm is a widely used method for aligning two point clouds by iteratively minimizing the distance between corresponding points. In this project, ICP is applied for scan matching to estimate relative transformations between LiDAR scans and refine the robot's trajectory. Below is an elaboration of the ICP process and its performance during warm-up tests and scan matching.

To improve ICP performance in scan matching, several optimizations are implemented:

1) Nearest Neighbor Search:

Two methods were tested for finding point correspondences:

- KDTree from scipy
- NearestNeighbors from sklearn

The NearestNeighbors method was chosen due to its faster response time under identical tolerance conditions.

2) Outlier Filtering:

An ICP threshold version is used to filter outliers by discarding points that are too far apart.

3) Error Calculation with Huber Loss:

A Huber loss function is employed to calculate alignment errors, reducing the influence of outliers and improving overall robustness.

4) Multiple Initial Guesses:

To mitigate sensitivity to initial guesses, multiple guesses with different yaw angles are tested. The guess yielding the lowest RMSE is selected as the starting point for ICP.

F. Factor Graph

In this project, two factor graphs are constructed to compare the performance of SLAM with and without loop closure. The factor graph is designed to represent the robot's trajectory and constraints between poses, including motion, observation, and loop closure constraints.

1) Loop Closure Criteria

The criteria for detecting potential loop closure pose pairs are as follows:

- ***Maximum location difference**: $d^* = 0.1\,m$*
- ***Maximum yaw difference**: $\theta^* = \frac{3}{4}\pi\,rad$*
- ***Minimum time interval**: $\tau^* = 30\,frames$*

These criteria ensure that loop closures are only added when poses are spatially close and temporally distant, reducing the risk of false associations.

2) Constraints

- ***Motion Constraints**: Represented with a static diagonal variance of:*

$$[0.0025, 0.0025, 0.0025]$$

These constraints are derived from the motion model based on differential-drive kinematics and IMU data.

- ***Observation and Loop Closure Constraints**: The variances for these constraints are proportional to the ICP error:*

$$[\epsilon s, \epsilon s, \epsilon s]$$

Here, $\epsilon s \epsilon s$ is the ICP alignment error for the corresponding scan match.

3) Comparison

- ***Without Loop Closure**: The factor graph forms a chain structure where errors accumulate over time due to the lack of global corrections.*
- ***With Loop Closure**: The graph incorporates additional constraints (loops) that connect spatially close but temporally distant poses. This reduces drift and improves the accuracy of both trajectory estimation and mapping.*

G. Mapping

Mapping is performed in two stages: occupancy mapping using LiDAR data and texture mapping using RGB-D data.

1) Discretized Map

The world is discretized into a grid with a resolution of **10 cells per meter** (i.e., each cell represents a 0.1 m×0.1 m area). All points are rounded to their respective cell centers for computational simplicity.

2) LiDAR to Occupancy Mapping

LiDAR scans are used to update the occupancy probabilities of grid cells:

- ***Empty Cells**:* Cells along the line between the LiDAR sensor origin and endpoints decrease their odds ratio by −2.
- ***Occupied Cells**:* Endpoint cells increase their odds ratio by +2.

The occupancy probability is calculated using logistic regression:

$$p(occupancy) = \frac{1}{1 + \exp(-odds\ ratio)}$$

This probabilistic approach accounts for uncertainty in sensor measurements while maintaining computational efficiency.

3) RGB-D Texture Mapping

RGB-D data is used to create a textured map of the environment:

- ***Intrinsic Matrix**:* The camera's intrinsic matrix K is given as:

$$K = \begin{bmatrix} 585.05 & 0 & 241.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}$$

- ***Depth Conversion**:* Disparity values dd are converted to depth z using:

$$dd = 0.00304d + 3.31$$

$$z = g(d) = \frac{1.03}{dd}$$

- ***Coordinate Transformation**:*
  - The optical-to-camera rotation matrix is:

$$_{optical}R^{-1}_{camera} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

  - Floor points are isolated based on height

$$z^* = 0.5$$

- ***RGBD Alignment**:*
  - The disparity camera and RGB camera have an offset along the x-axis.
  - Pixel coordinates in the disparity image (i,j) are mapped to RGB image coordinates (rgbi, rgbj) using:

$$rgbi = \frac{526.37i + 19276 - 7877.07dd}{585.051}$$

$$rgbj = \frac{526.37j + 16662}{585.051}$$

IV. RESULTS

Dataset 20

Motion Model: The motion model provided a solid baseline trajectory (Fig. 1a). While it is not perfect, it serves as a reliable starting point for further optimization.

Observation Model: The observation model improved upon the motion model by refining details.

Factor Graph Without Loop Closure: Not Implemented

Factor Graph With Loop Closure: Somehow did not work will with dataset 20, could be poor loop closure detections due to dense overlap between scans.

Dataset 21

Motion Model: The motion model struggled in areas such as the vertical hallway, introducing errors in the trajectory (Fig. 2a).

Observation Model: The observation model failed to correct these errors and instead converged to a local minimum, worsening the trajectory (Fig. 2b). This highlights the sensitivity of ICP to poor initial guesses.

Factor Graph Without Loop Closure: Not Implemented

Factor Graph With Loop Closure: Loop closure constraints helped correct some errors in the x-coordinate but introduced orientation inconsistencies, leading to unreliable mapping results (Fig. 2d).

Key Observations

ICP Sensitivity:

Both the standard ICP algorithm and its modified version are prone to converging to local minima. A good initial guess for translation and rotation is critical for achieving accurate results. Using multiple initial guesses with varying yaw angles, as done in this project, helps mitigate this issue.

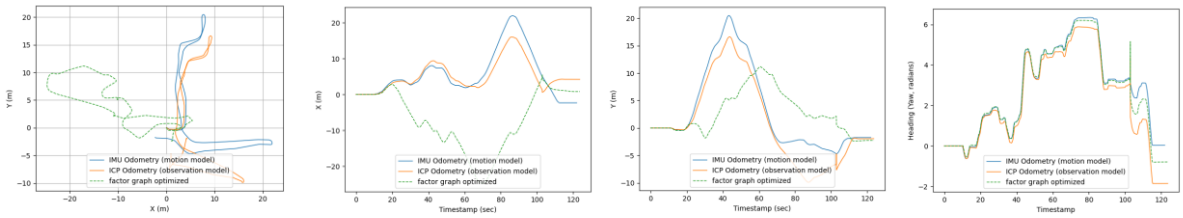
Potential Improvements:

Equipping the robot with a 3D LiDAR or utilizing a full 3D point cloud from an RGB-D camera could enhance ICP performance by providing more data points and an additional dimension for alignment.

These enhancements would reduce reliance on precise initial guesses and improve robustness against local minima.

Conclusion

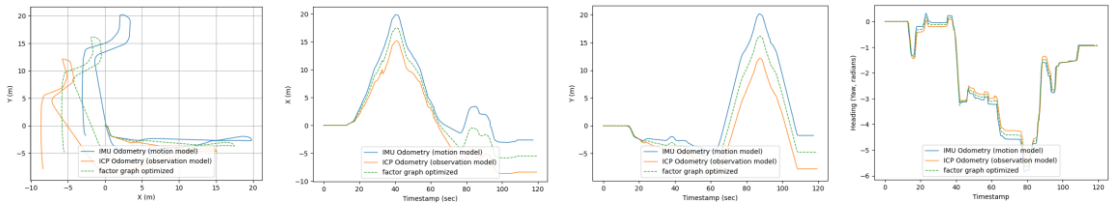
The results demonstrate that while motion models provide a reliable starting point, observation models and factor graphs with loop closure constraints are essential for refining trajectories and correcting drift. However, ICP's sensitivity to initial guesses remains a limitation, particularly in challenging environments like Dataset 21. Future work should explore integrating richer sensor data (e.g., 3D LiDAR) to enhance robustness and accuracy in scan matching and trajectory estimation.



(a) trajectory (b) trajectory x-t (c) trajectory y-t (d) trajectory yaw-t

Unfortunately, my occupancy and texture mapping scripts are not working, graph (e) – (h) not listed

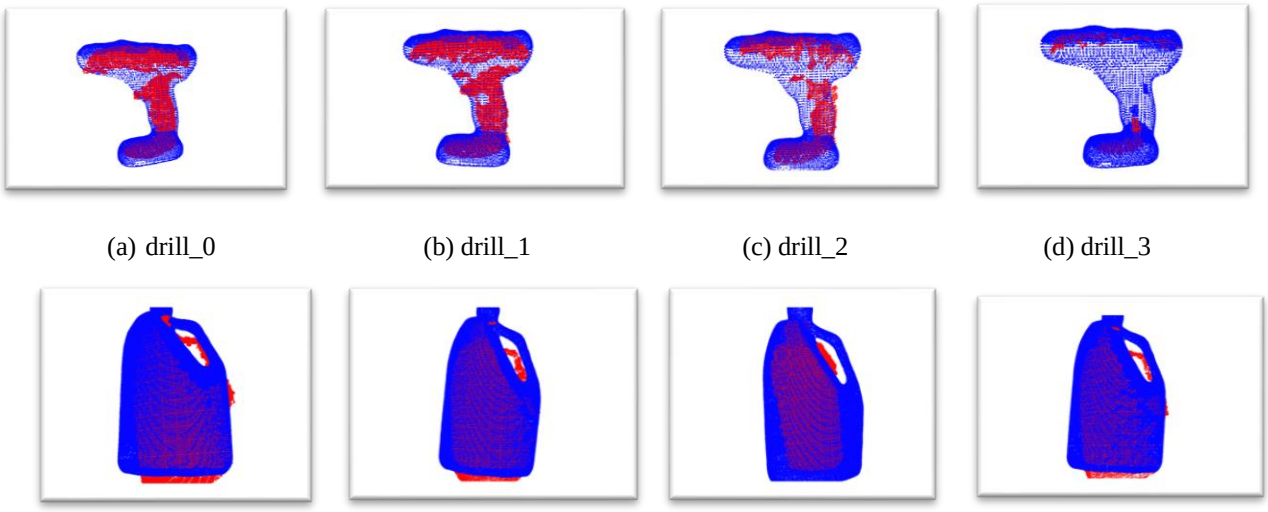
Fig. 2. Dataset 20



(a) trajectory (b) trajectory x-t (c) trajectory y-t (d) trajectory yaw-t

Unfortunately, my occupancy and texture mapping scripts are not working, graph (e) – (h) not listed

Fig. 3. Dataset 21



(a) drill_0 (b) drill_1 (c) drill_2 (d) drill_3

(e) liq_container_0 (f) liq_container_1 (g) liq_container_2 (h) liq_container_3

Fig. 4. ICP warm up images

REFERENCES

[1] N. Atanasov, UCSD ECE276A: Sensing & Estimation in Robotics (Winter 2025), https://natanaso.github.io/ece276a/schedule.html.