FactorSLAM: Enhanced LiDAR SLAM via Factor Graph Enhancement

Yixin Zhang

I. INTRODUCTION

State estimation, localization, and mapping serve as essential foundations for the effective operation of intelligent mobile robots, underpinning critical functions such as feedback control, obstacle navigation, and strategic planning. Leveraging lidar-based sensing technologies, substantial advancements have been made toward achieving high-precision, real-time localization and mapping capabilities that significantly enhance a mobile robot's ability to understand and interact with its environment. Moreover, lidar-based approaches offer the distinct advantage of being largely unaffected by changes in lighting conditions, a benefit that has been further amplified by the recent development of long-range, high-resolution lidar systems.

In this project, we introduce a framework based on LiDAR for simultaneous localization and mapping (SLAM). For the localization part, the front end utilizes encoders and an IMU to construct an approximate trajectory and pose estimation as an initial guess. This initial odometry estimate is subsequently refined through LiDAR odometry, leveraging scan matching via the iterative closest point (ICP) algorithm to enhance precision. In the back end, the robot's trajectory accuracy is further improved by employing pose graph optimization with loop closure constraints, facilitated by the GTSAM library. For the mapping part, the robot's localization is achieved using odometry and LiDAR measurements, which enables the construction of a detailed 2-D occupancy grid map of the environment. Additionally, RGB-D images are utilized to colorize the 2-D floor map, providing a richer and more informative representation of the surroundings.

The main contributions of our work can be summarized as follows:

- We effectively leverage the synergistic potential of encoder, IMU, and LiDAR information, harnessing their combined strengths to refine pose estimation capabilities within our framework.
- Owing to the computational demands of ICP algorithms, coupled with their susceptibility to being misled by initial guesses and local minima, we have developed an advanced ICP algorithm. This enhanced approach incorporates multiple techniques, including uniform downsampling, multi-resolution strategies, and a priority KD-tree for precise data association. Visual results from warm-up parts demonstrate the effectiveness of these improvements.
- To address the inherent drift and cumulative errors associated with scan matching over time, our backend implementation employs a factor graph in conjunction with loop closure techniques. This method effectively mitigates these errors, ensuring more reliable trajectory estimation.
- Our framework has been rigorously tested in two distinct indoor scenarios: one featuring a flat floor, and another with obstacles that include slopes and declines. In both environments, our system has exhibited remarkable robustness and reliability.

II. PROBLEM FORMULATION

This section introduces the mechanical system and the SLAM algorithm. The mechanical system discussion focuses on the sensors equipped and the model of a differential-drive robot. In contrast, the SLAM portion delves into the algorithmic formulation for localization and mapping.

A. Mechanical System

The robot is equipped with a comprehensive sensor suite: Encoders, IMU, Hokuyo, and Kinect. The encoders, operating at 40 Hz, monitor the rotation of the four wheels, with their count reset after each reading. From the IMU, we specifically utilize the yaw rate as the angular velocity in our differential-drive model. The Hokuyo, a horizontal LiDAR, provides a 270° field of view and a maximum range of 30 meters, offering distance measurements to obstacles with each scan comprising 1081 range values. The Kinect, an RGBD camera, supplies RGB and disparity images. Positioned at (0.18, 0.005, 0.36) meters relative to the robot's center, the camera is aligned with a roll of 0 rad, a pitch of 0.36 rad, and a yaw of 0.021 rad.

Fig. 1: Differential-drive robot equipped with encoders, IMU, 2-D LIDA, and RGBD camera. Note their sensing frequencies are not the same.

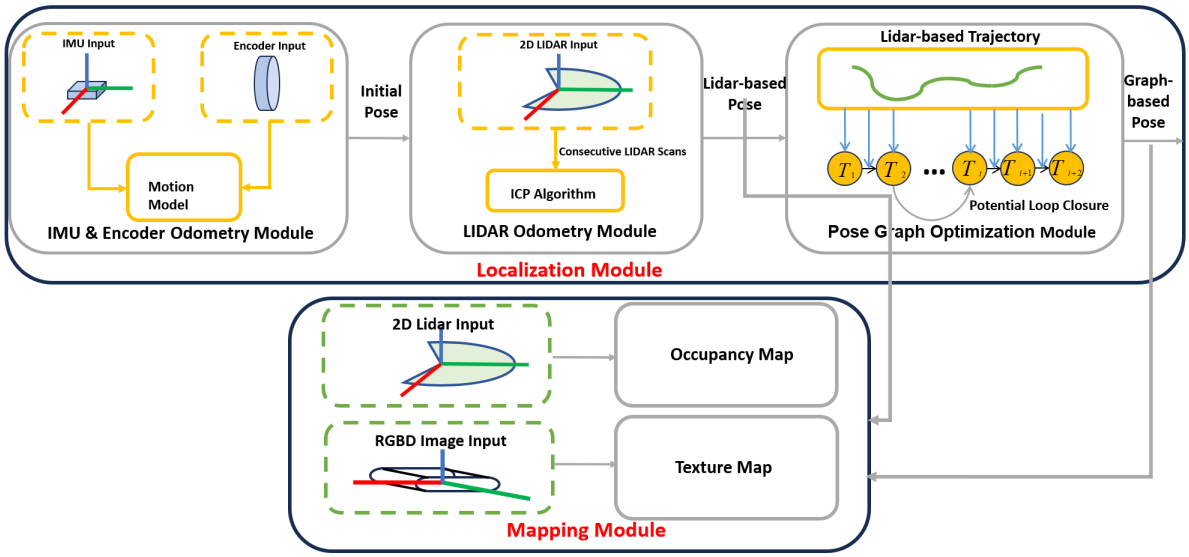


Fig. 2: The overall SLAM (Simultaneous Localization and Mapping) framework is composed of two main modules: the Localization Module and the Mapping Module. The figure illustrates the data relationships between these modules, highlighting how they interact and contribute to the SLAM process.

B. SLAM Framework

Our SLAM framework addresses two intertwined challenges: localization and mapping, with localization divided into front-end and back-end components, and mapping differentiated into occupancy and texture mapping for enhanced visualization and intuitive understanding, thereby validating the robustness of our algorithms. Detailed inputs and outputs can be found in Fig.2.

1) Localization: The robot state $\mathbf{x_t}$ can be written as

$$\mathbf{x_t} = T_t = \begin{bmatrix} R_t & P_t \\ 0 & 1 \end{bmatrix}$$

where $R_t \in SO(2)$ is a rotation matrix, and $P_t \in \mathbb{R}^2$ is a position vector. For localization, our objective is to estimate the pose sequence $\{\mathbf{x_t}\}_1^N$ over time, leveraging data from the IMU, encoders, and LiDAR scans. Given the 2D nature of the problem, we utilize only the yaw rate from the IMU, denoted as $\{w_t\}_1^N$. The encoders are reset after each reading, producing tick counts for each interval, represented as $\{fr_t, fl_t, rr_t, rl_t\}_1^N$, which correspond to the tick counts for the front-right, front-left, rear-right, and rear-left wheels, respectively. The LiDAR inputs are denoted as $\{r_{t,1}, r_{t,2}, r_{t,3}, ..., r_{t,1081}\}_1^N$, where $r_{t,i}$ represents a specific range scan corresponding to a particular scan angle at each time instance. Each LiDAR scan comprises 1081 measured range values, accumulating to N scans over time.

a) Front-end: Using the data from the Inertial Measurement Unit (IMU) and encoders, we initiate the process by applying the robot's motion model to generate an initial estimation of its trajectory. Following this preliminary step, we leverage the initial guess to incorporate LiDAR scans from consecutive frames. This data serves as the basis for implementing Iterative Closest Point (ICP) algorithms, aimed at re-estimating the relative poses between frames.

b) Back-end: Acknowledging the cumulative error in LiDAR scan matching and IMU-encoder readings, we foresee a trajectory drift over time. To address this, our back-end employs a graph approach, with each trajectory estimate as a node and LiDAR scan matches as edges, representing relative movements. Utilizing the GTSAM library, we optimize this graph to correct errors and enhance trajectory accuracy, ensuring a more precise navigation path for the robot.

2) Mapping: Given the estimated trajectory $\{\mathbf{x_t}\}_1^N$ over a certain period, along with LiDAR inputs $\{r_{t,1}, r_{t,2}, r_{t,3}, ..., r_{t,1081}\}_1^N$, and RGBD data inputs $\{depth_i\}_1^N$ and $\{rgb_i\}_1^N$ over time, our goal is to construct both an occupancy map and a texture map of the floor over this duration.

a) Occupancy Mapping: The inputs for this process encompass both the estimated trajectory and LiDAR data. By leveraging these components, we're able to construct a dynamic occupancy map. This evolving map articulates the spatial configuration, distinguishing between occupied and unoccupied areas.The outcome of this process is denoted as $\{H, W, m_i\}$, where H represents the image height, W is the image width, and m_i symbolizes the probability mass function (PMF) probability.

b) Texture Mapping: The process of texture mapping utilizes the estimated trajectory and data from an RGBD camera as its inputs. The objective of this procedure is to generate a dynamic texture map. This map enhances the spatial structure determined by the trajectory estimates with intricate texture details extracted from the RGBD camera data. The result of this operation is denoted as $\{H, W, rgb_i\}$, where rgb_i represents the corresponding RGB values for each pixel.

III. TECHNICAL APPROACH

A. Data Synchronization

Given two datasets D_1 and D_2, with $D_1 = \{(t_{1i}, x_{1i})\}_{i=1}^{n}$ and $D_2 = \{(t_{2j}, x_{2j})\}_{j=1}^{m}$, where:

- t_{1i} and t_{2j} represent the timestamps for each data point in their respective datasets,
- x_{1i} and x_{2j} are the data points (signals) collected at these timestamps,
- n and m are the total number of observations in each dataset, respectively.

The goal is to synchronize D_1 and D_2 onto a common time base $T = \{t_k\}_{k=1}^{p}$, where p is the number of time points in the common time base.

1) Selection of T: The common time base T can be selected based on the higher frequency dataset, a lower frequency dataset, or an entirely new frequency that is suitable for the intended analysis. The choice depends on the specific requirements of the analysis or the nature of the signals.
2) Interpolation/Resampling: For each time point t_k in T, we find corresponding values in D_1 and D_2 by interpolation or resampling. The interpolated values $\hat{x}_{1k}$ and $\hat{x}_{2k}$ are computed as follows:
   - For D_1, $\hat{x}_{1k} = \text{interp}(t_k, D_1)$ where $\text{interp}(\cdot)$ denotes an interpolation function applied to D_1 at time t_k.
   - Similarly, for D_2, $\hat{x}_{2k} = \text{interp}(t_k, D_2)$.

   This step requires choosing an interpolation method (e.g., linear, spline) that is appropriate for the data.
3) Merging: Create a synchronized dataset $D_{sync} = \{(t_k, \hat{x}_{1k}, \hat{x}_{2k})\}_{k=1}^{p}$ that contains the common time base T and the corresponding interpolated values from both D_1 and D_2.

B. ICP Algorithms

The refined ICP algorithms are decribes as below:

- FindClosestPoint: we dealwith this data association issue via Kdtree algorithms to faster the training process;
- Kabsh Algorithms: This problem can be formulated as below:

$$T = \arg\min_{T} \left\{ \sum_i w_i \left\| T(z_i) - m_i \right\|^2 \right\}$$

$$= \arg\min_{R,\mathbf{p}} \left\{ \sum_i w_i \left\| (R\mathbf{z}_i + \mathbf{p}) - \mathbf{m}_i \right\|^2 \right\}$$

Let the point cloud centroids be:

$$\overline{\mathbf{m}} := \frac{\sum_i w_i \mathbf{m}_i}{\sum_i w_i} \quad \overline{\mathbf{z}} := \frac{\sum_i w_i \mathbf{z}_i}{\sum_i w_i}$$

Solving $\nabla_{\mathbf{p}} f(R, \mathbf{p}) = \mathbf{0}$ for $\mathbf{p}$ leads to:

$$\mathbf{p} = \overline{\mathbf{m}} - R\overline{\mathbf{z}}$$

Replace $\mathbf{p} = \overline{\mathbf{m}} - R\overline{\mathbf{z}}$ in objective function :

$$f(R, \overline{\mathbf{m}} - R\overline{\mathbf{z}}) = \sum_i w_i \left\| R(\mathbf{z}_i - \overline{\mathbf{z}}) - (\mathbf{m}_i - \overline{\mathbf{m}}) \right\|_2^2$$

Define the centered point clouds:

$$\delta\mathbf{m}_i := \mathbf{m}_i - \overline{\mathbf{m}} \quad \delta\mathbf{z}_i := \mathbf{z}_i - \overline{\mathbf{z}}$$

Finding the optimal rotation reduces to:

$$\min_R \sum_i w_i \left\| R\delta\mathbf{z}_i - \delta\mathbf{m}_i \right\|_2^2$$

The objective function can be simplified further:

$$\sum_i w_i \left\| R\delta\mathbf{z}_i - \delta\mathbf{m}_i \right\|_2^2$$

$$= \sum_i w_i (\delta\mathbf{z}_i^\top \underbrace{R^\top R}_{I} \delta\mathbf{z}_i - 2\delta\mathbf{m}_i^\top R\delta\mathbf{z}_i + \delta\mathbf{m}_i^\top \delta\mathbf{m}_i)$$

To determine the rotation R that aligns two associated centered point clouds $\{\delta\mathbf{m}_i\}$ and $\{\delta\mathbf{z}_i\}$, we need to solve a linear optimization problem:

$$\max_R \text{tr}\left(Q^\top R\right)$$

where $Q := \sum_i w_i \delta\mathbf{m}_i \delta\mathbf{z}_i^\top$ let $Q = U\Sigma V^\top$ be the singular value decomposition of Q. The maximum is achieved with:

$$R = U \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det\left(UV^\top\right) \end{bmatrix} V^\top$$

Algorithm 1 Refined Iterative Closest Point (ICP) Algorithm with Distance Filtering

Require: Two point clouds: $M = \{m_i\}_{i=1}^{n}$ and $Z = \{z_i\}_{i=1}^{n}$
Require: Initial transformation guess: T_0
Require: Filter threshold: A
Ensure: Optimal transformation T, aligning M to Z

$T \leftarrow T_0$
while not converged **do**
    Point Cloud Down-sampling
    Calculate all distances $\{d_i\}$ for $i = 1$ to n, where $d_i = \|m_i - T \cdot z_i\|$
    Sort $\{d_i\}$ in ascending order
    $d_{\max} \leftarrow$ the Ath(eg. 80th) percentile of $\{d_i\}$, ensuring only the smallest A%(eg. 80%) of distances are retained
    for $i = 1$ to n **do**
        $m_i \leftarrow \text{FindClosestPointIn}(M, T \cdot z_i)$
        if $\|m_i - T \cdot z_i\| \leq d_{\max}$ **then**
            $w_i \leftarrow 1$
        else
            $w_i \leftarrow 0$
        end if
    end for
    $T \leftarrow \arg\min_T \sum_i w_i \|T \cdot z_i - m_i\|^2$
end while

To enhance the robustness of our Iterative Closest Point (ICP) algorithms, we incorporate the following techniques:

- Multi-resolution Downsampling of Point Clouds: At the initiation of our algorithm, we downsample the entire

point cloud into multiple resolutions. This approach accelerates convergence and reduces the likelihood of being trapped in local minima.

- Distance Filtering: During each data association step, we filter out data points that exceed a specified distance threshold, treating them as outliers. This practice improves the accuracy of our algorithms.

C. Motion Model and Pose Estimation

The state of robot can be desribed as $\mathbf{x} = (\mathbf{p}, \theta)$, where $\mathbf{p} = (x, y) \in \mathbb{R}^2$ is the position and $\theta \in (-\pi, \pi]$ is the orientation (yaw angle) in the world frame. And the control input of robot can be denoted as $\mathbf{u} = (v, \omega)$, where $v \in \mathbb{R}$ is the linear velocity and $\omega \in \mathbb{R}$ is the angular velocity (yaw rate) in the body frame. The Continuous-time model can be decribed as below:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) := \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$$

The model is obtained using 2D pose kinematics with body-frame twist $\zeta = (v, 0, \omega)^\top$:

$$\begin{bmatrix} \dot{R}(\theta) & \dot{\mathbf{p}} \\ \mathbf{0} & 0 \end{bmatrix} = \begin{bmatrix} R(\theta) & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} 0 & -\omega & v \\ \omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We use discrete-time pose kinematics to approximate the robot's kinematics:

$$\begin{bmatrix} R(\theta_{t+1}) & \boldsymbol{p}_{t+1} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R(\theta_t) & \boldsymbol{p}_t \\ \mathbf{0} & 1 \end{bmatrix} \exp\left(\tau_t \begin{bmatrix} 0 & -\omega_t & v_t \\ \omega_t & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right)$$

D. Front-end Localization

1) IMU & Encoder-based Localizaiton: Given the encoder dataset $\{fr_t, fl_t, rr_t, rl_t\}_1^N$ and IMU yaw rate dataset $\{w_t\}_1^N$, we can calculate the velocity as below:

$$v_t = \left(\frac{fr_t + fl_t + rr_t + rl_t}{4 \cdot \tau_t} \right) \cdot 0.0022$$

where $\tau_t = t_{t+1} - t_t$ After completing the data synchronization for v_t and w_t, we are equipped to utilize the previously discussed motion model to initiate the trajectory estimation.

2) Lidar-based Localization: In our data conversion process, we transform LiDAR scan data into Cartesian coordinates as follows:

$$\begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} = \begin{bmatrix} r_{t,i} \cdot \cos(\theta_i) \\ r_{t,i} \cdot \sin(\theta_i) \end{bmatrix}$$

where θ_i represents the corresponding angle of $r_{t,i}$ for each scan point. We represent each scan by $\{x, y\}_t$. We define a continuous LiDAR scan by $\{x, y\}_t$ and $\{x, y\}_{t+1}$.

The initial step involves synchronizing the data with the LiDAR sensor and IMU&Encoder based estimation to find the relative pose T_t^{t+1}, which is derived from IMU and Encoder-based Localization.

Subsequently, we employ the Iterative Closest Point (ICP) algorithm, treating $\{x, y\}_{t+1}$ as the source point cloud and $\{x, y\}_t$ as the target point cloud. This process enables us to obtain a refined relative pose T_t^{t+1}, enhancing the accuracy of our localization and mapping efforts.

In this phase of our analysis, it is crucial to acknowledge that outliers invariably exist, particularly in Dataset 21, which encompasses obstacles such as slopes and declines that the robot encounters while navigating the terrain. These outliers, often resulting from point clouds that are anomalously distant or markedly close, necessitate removal to ensure the integrity of our data. Furthermore, we implement a distance thresholding technique, essential for addressing visibility inconsistencies during the robot's maneuvers, such as turning. Objects that are visible in one scan may become obscured in the subsequent scan due to the robot's orientation or position changes. By setting our distance threshold to 80

E. Back-end Opimization

Given a set of lidar scan matching trajectories $\{T_t\}_1^N$, we initiate the construction of the factor graph. Each node's value is denoted as $node_i = T_i$, and the edge between consecutive nodes is represented by $edge_i^{i+1} = T_i^{-1} T_{i+1}$, computed using ICP scan matching.

Subsequently, we introduce loop closures between intervals of 10 steps. New constraints in the form of $edge_i^{i+10}$ are added, and ICP is performed again to refine these constraints. Our ICP algorithm, prioritizing robustness, effectively handles outliers resulting from the robot's motion. When the robot turns, previously detected features may not be visible in current scans, but our algorithm effectively deals with such scenarios by setting priority distance thresholds when adding constraints. Finally, we optimize the entire graph using gtsam.GaussNewtonOptimizer. The model noise is specified as $[0.01, 0.01, 0.01]$, and the prior noise is set to $[1e-6, 1e-6, 1e-4]$.

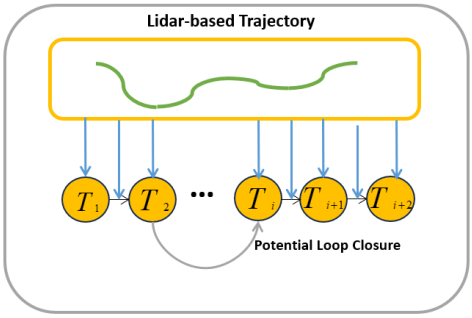


Fig. 3: Visualization of Pose Graph

F. Mapping Module

1) Occupancy Mapping: In the preliminary phase of our methodology, we commence by filtering out LiDAR scan points based on their proximity to the robot, excluding those that are anomalously close or far. This selective process ensures that our analysis remains focused on environmentally significant features. Following this filtration, we transform the LiDAR points from the local coordinate frame to the global frame, thus standardizing our spatial references. This transformation is critical for maintaining consistency across our dataset. Subsequently, these global coordinates are converted into map-cell coordinates to align with our map's discretized representation.

To enhance the accuracy and efficiency of our occupancy detection for each scan update, we introduce a novel algorithm that diverges from the traditional reliance on Python loops. The sequence of operations in this algorithm includes:

- **Bounding Box Calculation**: The algorithm computes the minimum and maximum x and y coordinates from the polygon's vertices, establishing a bounding box. This box delineates the search area for points within the polygon.
- **Polygon Path Formation**: By utilizing the Path class, commonly found in graphical libraries like Matplotlib, the polygon's vertices are transformed into a path object. This conversion is instrumental in efficiently determining point containment within the polygon.
- **Grid Generation Within Bounding Box**: A grid of points is generated within the bounding box using the `np.meshgrid` function. This grid, formed by flattening and merging two-dimensional arrays of x and y coordinates, facilitates a thorough evaluation of coordinate pairs.
- **Point Containment Verification**: Applying the `contains_points` method of the Path object, the algorithm assesses whether points fall within the polygon, yielding a boolean array that indicates containment status for each point.
- **Selection of Interior Points**: Points confirmed as interior, based on the boolean array, are selectively extracted. This ensures the algorithm focuses exclusively on points verified to be within the polygon.
- **Update Log Odds**: The log-odds value $\lambda_{i,t+1} = \lambda_{i,t} \pm \log(4)$ is modified to update occupancy within the grid map. An increment is applied for interior points, while a decrement is used for points along the bounding box contour. To prevent overconfident estimations, constraints $\lambda_{\text{MIN}} \leq \lambda_{i,t} \leq \lambda_{\text{MAX}}$ are imposed, with $\lambda_{\text{MIN}} = -30\log(4)$ and $\lambda_{\text{MAX}} = 30\log(4)$.

Finally, the map's probability mass function (pmf), $\gamma_{i,t}$, is derived from the log-odds, $\lambda_{i,t}$, through the logistic sigmoid function:

$$\gamma_{i,t} = p(m_i = 1 | \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \sigma(\lambda_{i,t}) = \frac{\exp(\lambda_{i,t})}{1 + \exp(\lambda_{i,t})}$$

2) Texture Mapping: Utilizing RGBD imagery and the robot's estimated trajectory, our methodology extends beyond occupancy mapping to generate a two-dimensional color map of the floor texture. The process unfolds through the following steps:

1) **Depth Data Acquisition**: We read the disparity image as a 16-bit unsigned integer array and extract the corresponding RGB values, following the specifications detailed by the Kinect sensor documentation.

$$\begin{aligned} dd &= (-0.00304d + 3.31) \\ z = \text{ depth } &= \frac{1.03}{dd} \\ \text{rgbi } &= (526.37i + 19276 - 7877.07dd)/585.051 \\ \text{rgbj } &= (526.37j + 16662)/585.051 \end{aligned}$$

2) **Coordinate Transformation Sequence**: The pixel coordinates from the RGB images undergo a series of frame transformations to be accurately projected onto the world frame:
   - *Intrinsic Calibration*
   - *Optical Frame Transformation*
   - *Camera Frame Transformation*
   - *Body Frame Transformation*:
   - *World Frame Transformation*

$$p_o = K^{-1} \cdot \begin{bmatrix} u \cdot z \\ v \cdot z \\ z \end{bmatrix},$$

$$\widetilde{p}_w = T \cdot \widetilde{p}_o$$

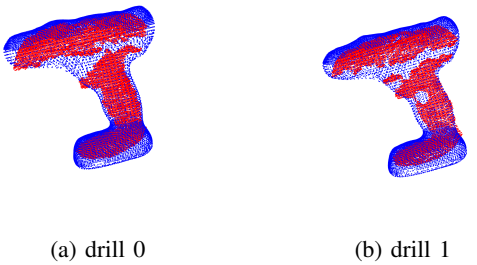
   where $\widetilde{p}_w$ denotes the homogeneous world coordinates, $\widetilde{p}_o$ denotes the homogeneous optical coordinates.The matrix K denotes intrinsic calibration, The matrix T embodies a series of transformations, encompassing Optical Frame Transformation, Camera Frame Transformation, Body Frame Transformation, and finally, World Frame Transformation.
3) **Mapping and Thresholding**: Within the transformed dataset, we identify the plane corresponding to the occupancy grid's floor plane through height-based thresholding, where a height of 0 m is indicative of the floor plane.
4) **Color Map Construction**: An array initialized with zeros, shaped according to the world frame limits, serves as the canvas for our color map. The RGB values associated with the identified floor height are inserted into this array, resulting in a detailed 2-D color representation.

The integration of depth and color information culminates in a nuanced environmental representation, offering insights into not only the spatial distribution of obstacles but also the textural nuances of the floor, pivotal for a holistic understanding of the robot's operational domain.

IV. Results and Comparative Analysis

A. ICP Warm-up

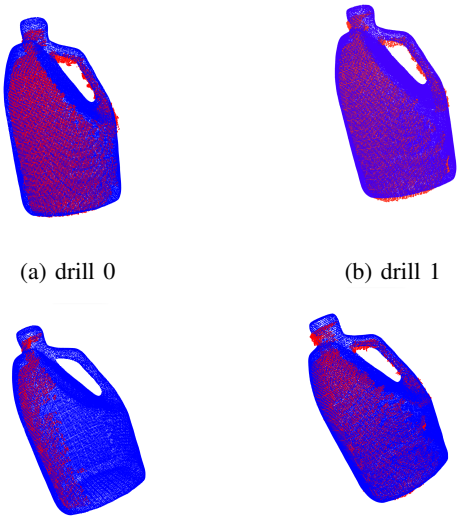
We observed a notable trend within our dataset, spanning from categories 0 to 3: a gradual decrease in the quantity of source point clouds. And the liquid container dataset demonstrated a significantly higher count of point clouds compared to that of the drill. Despite these disparities between the source and target point cloud counts, as well as the total volume of point clouds across categories, our results clearly demonstrate that our ICP algorithm maintains a high level of robustness and accuracy.



(a) drill 0 (b) drill 1

(c) drill 2 (d) drill 3

Fig. 4: ICP warm up results for drills



(a) drill 0 (b) drill 1

(c) drill 2 (d) drill 3

Fig. 5: ICP warm up results for liquid containers

B. Front-end Estimation Result

1) Trajectory Visualization: The blue trajectory, sourced from IMU and encoder data, alongside the red trajectory, generated via scan matching, both manifest progressive drift and error accumulation as time advances, culminating in noticeable divergence from their intended courses. In the context of occupancy and texture map visualizations, this is particularly evident in dataset20, which displays a marked superimposition of colors. This overlap is attributed to the compounded errors that induce drift, consequently leading to misalignments in the mapping process.

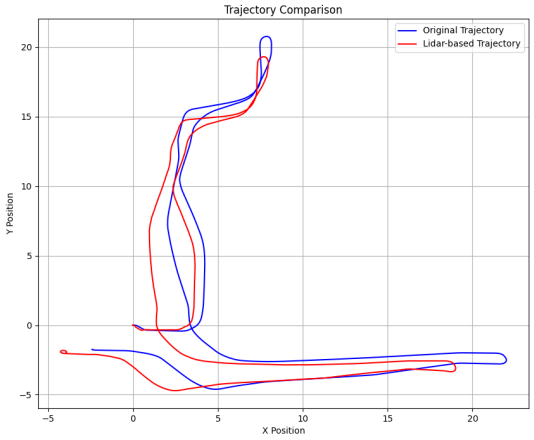


Fig. 6: Trajectory for dataset20

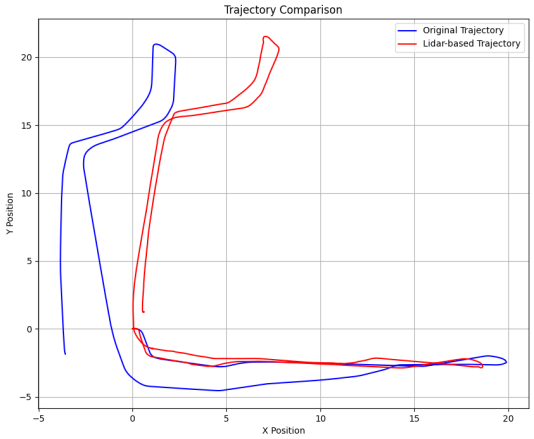


Fig. 7: Trajectory for dataset20

2) Occupancy Map Visualization: The mapping utilizes a color-coded system where the yellow regions denote occupied spaces, and purple signifies unoccupied areas. Green marks are used to represent regions with a probability of 0.5, indicating that these spaces remain unexplored or are beyond the detection capabilities of the sensors.

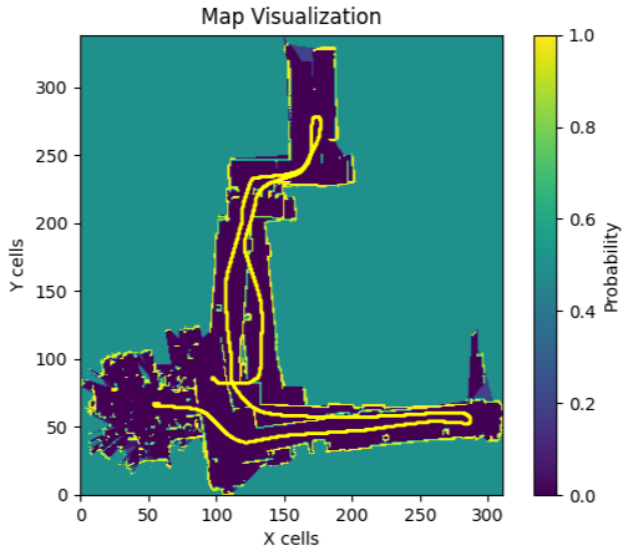


Fig. 8: Occupancy Map for dataset20

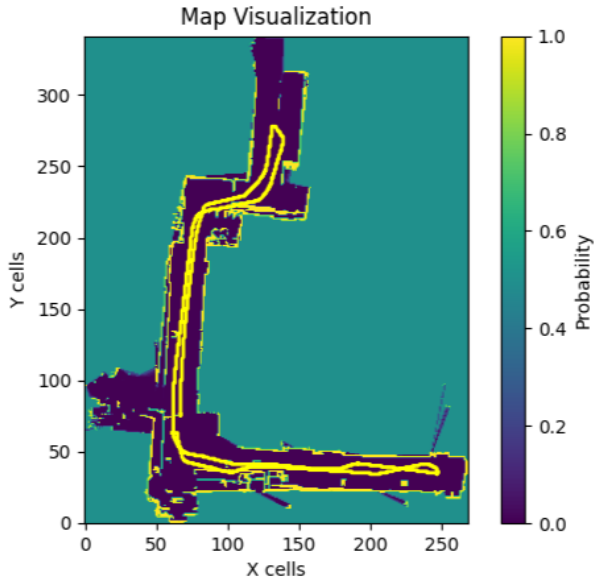


Fig. 9: Occupancy Map for dataset21

3) Texture Map Visualization: The texture map for dataset 20 reveals bad localization performance, evident from the incongruent mappings of the same area. Conversely, the map for dataset 21 illustrates displacement in the mapping that can be caused by drift in the trajectory.

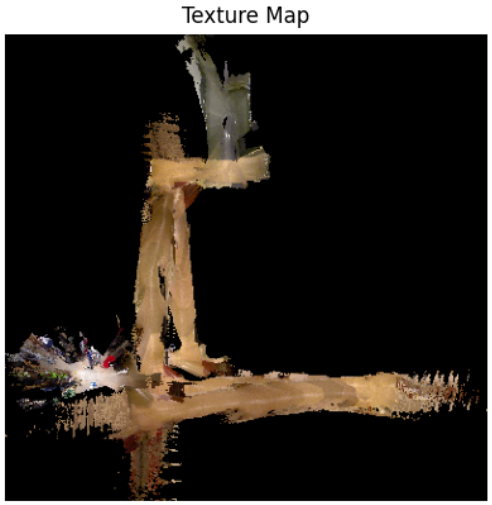


Fig. 10: Texture Map for dataset20

Fig. 11: Texture Map for dataset21

C. GTSAM-based Estimation Result

It should be highlighted that datasets 20 and 21 encompass distinct scenarios; dataset 21, in particular, features obstacles that the robot must navigate over and around. Despite these challenges, our results demonstrate robustness in obstacle-rich environments. Furthermore, post-GTSAM optimization, the noticeable drift and cumulative error present in both datasets have been significantly mitigated, resulting in an accurate localization that faithfully positions datasets 20 and 21 in their correct contexts.

1) Trajectory Visualization: The blue trajectory delineates the initial scan matching results, while the red trajectory illustrates the refined outcomes post-GTSAM optimization. Through the application of GTSAM, the previously accumulated errors have been effectively rectified.

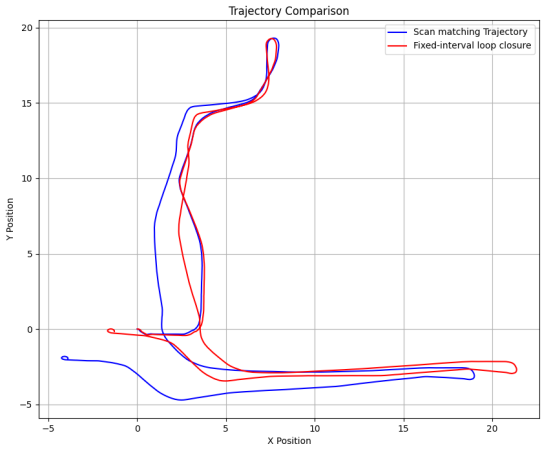


Fig. 12: Trajectory for dataset20

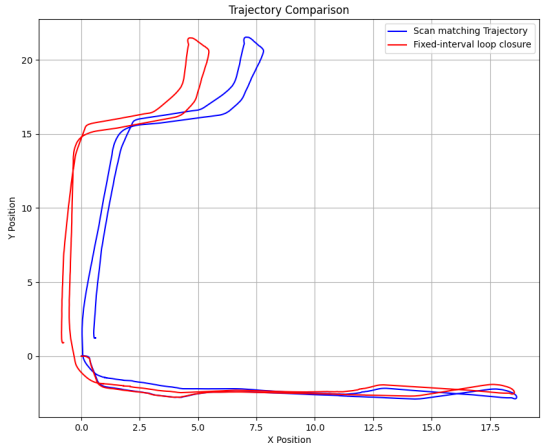


Fig. 13: Trajectory for dataset21

2) Occupancy Map Visualization: The mapping outcomes display a substantial enhancement attributed to GTSAM optimization, which has notably diminished the accumulation of error.

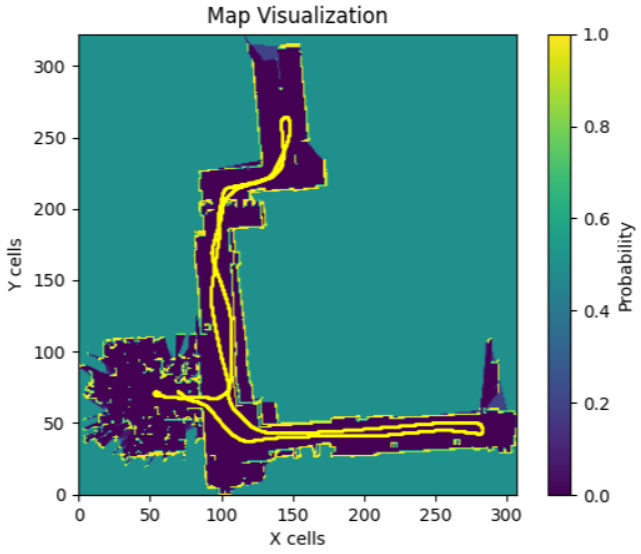


Fig. 14: Occupancy Map for dataset20

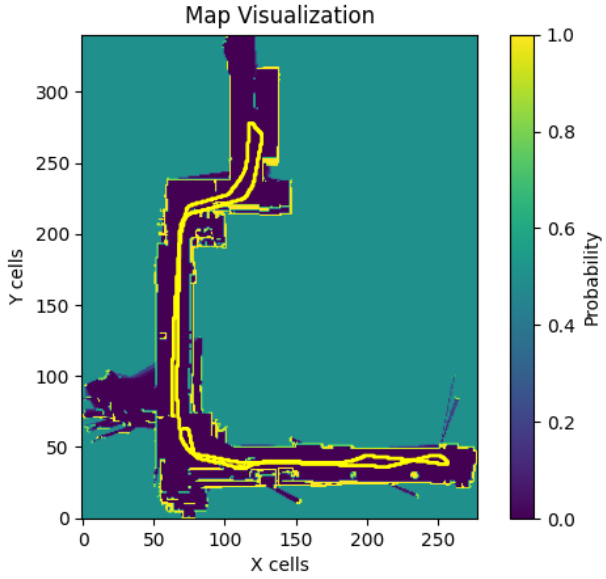


Fig. 15: Occupancy Map for dataset20n

3) Texture Map Visualization: The mapping outcomes display a substantial enhancement attributed to GTSAM optimization, which has notably diminished the accumulation of error.

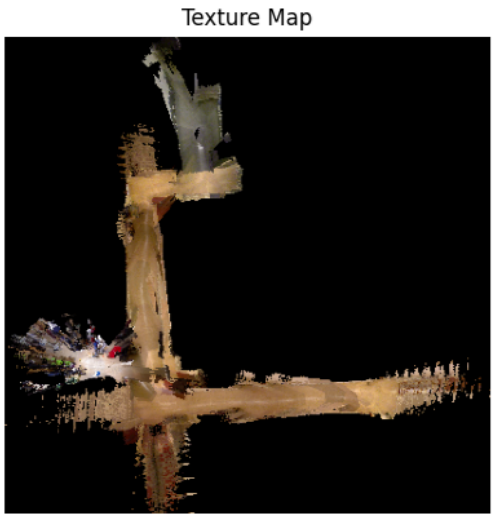


Fig. 16: Texture Map for dataset20

Fig. 17: Texture Map for dataset21

V. CONCLUSIONS

Our SLAM framework skillfully navigates the complexities of localization and mapping, blending occupancy and texture mapping to create a nuanced environmental model. While the approach yields encouraging outcomes, it also unveils areas for improvement, such as the current reliance on two-dimensional plane constraints and the susceptibility to sensor noise, which could be mitigated by advancing into three-dimensional mapping and refining sensor fusion methodologies. Additionally, future work could benefit from exploring proximity-based loop closure methods within the GTSAM implementation to enhance pose accuracy. This study serves as a pivotal step towards sophisticated autonomous navigation solutions.

ACKNOWLEDGMENT

The author would also like to thank for the constructive advice from Professor Nikolay Atanasov, all teaching assistants, and everyone who shared their ideas on Piazza and during office hours.