Report for Project two ECE276A

Amirhosein Javadi

March 4, 2024

1 Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in robotics that involves the task of a robot navigating in an unknown environment while simultaneously constructing a map of that environment. LiDAR-Based SLAM, in particular, utilizes LiDAR sensors to perceive the surroundings and solve the SLAM problem. In this project, we aim to implement LiDAR-Based SLAM using encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements from a differential-drive robot, with a specific emphasis on the critical aspect of data synchronization. The problem we are addressing in this project is the implementation of SLAM using LiDAR data, encoder and IMU odometry, and RGBD measurements. The goal is to accurately localize the robot in its environment while simultaneously constructing a detailed map of the surroundings. This involves integrating data from various sensors and algorithms to estimate the robot's pose and map the environment efficiently. LiDAR-Based SLAM is crucial in many real-world applications, such as autonomous navigation, robotic exploration, and mapping of indoor and outdoor environments. By accurately localizing the robot and generating detailed maps, we enable robots to operate autonomously in complex environments, perform tasks efficiently, and adapt to dynamic surroundings.

2 Problem Formulation

Given encoder and IMU odometry measurements, 2-D LiDAR scans, and RGBD measurements from a differential-drive robot, the task is to perform Simultaneous Localization and Mapping (SLAM) using the following steps:

1. Estimate the robot's odometry using encoder and IMU measurements.
2. Improve the initial odometry estimates by aligning LiDAR scans via the iterative closest point (ICP) algorithm.
3. Use the estimated robot trajectory to generate a 2-D occupancy grid map of the environment from LiDAR scans and a 2-D texture map of the floor using RGBD images.
4. Optimize the robot trajectory estimates further using the GTSAM library with loop-closure constraints.

Let $\mathbf{X} = \{x_1, x_2, ..., x_N\}$ represent the robot's trajectory over time, where each x_i is the pose of the robot at time i. Let $\mathbf{Z} = \{z_1, z_2, ..., z_M\}$ represent the set of sensor measurements, including encoder and IMU odometry, LiDAR scans, and RGBD measurements. The goal is to estimate the trajectory $\mathbf{X}$ and construct a map of the environment $\mathbf{M}$ that maximizes the likelihood of the sensor measurements $\mathbf{Z}$, i.e.,

$$P(\mathbf{X}, \mathbf{M}|\mathbf{Z})$$

subject to constraints imposed by the robot's motion model, sensor models, and any loop-closure constraints in the environment.

3 Technical Approach

3.1 Data Synchronization

Data synchronization is a critical aspect of our project, as it involves aligning and integrating data from multiple sensors to ensure accurate localization and mapping. Each sensor, including encoders, IMU, LiDAR, and RGBD Kinect, reports measurements at its own frequency, resulting in unsynchronized data streams. To address this challenge, we leverage Unix timestamps associated with each sensor measurement. First, for each data sample from every sensor, we calculate the corresponding Unix timestamp. Next, to synchronize the data, we employ a method that involves finding the data sample in other sensors with the smallest distance in timestamp. This process allows us to align the sensor data accurately, facilitating the integration of information from multiple sources.

3.2 Encoder and IMU odometry

Encoders are sensors that count the rotations of the wheels on a robot and provide information about the distance traveled by the robot. In this project, the encoders provide readings at a frequency of 40 Hz, with each reading indicating the number of rotations of the wheels since the last reset. By knowing the wheel diameter and the number of ticks per revolution, the distance traveled by each wheel can be calculated from the encoder counts. This information is used to estimate the linear velocity of the robot.
IMU (Inertial Measurement Unit) provides data on linear acceleration and angular velocity. For this project, we are specifically interested in the yaw rate, which represents the rate of change of orientation around the vertical axis. This yaw rate data is used to predict the robot's angular velocity in the differential-drive motion model.
To implement encoder and IMU odometry, the following approach is used:

1. Calculate Linear Velocity: Using the encoder counts, the linear velocity of the robot is estimated. The distance traveled by each wheel is calculated based on the number of ticks and wheel diameter. The average linear velocity of the right and left wheels is then determined.

2. Predict Robot Motion: Combine the linear velocity obtained from encoders with the yaw rate obtained from the IMU to predict the motion of the robot using the differential-drive motion model. This model relates the linear and angular velocities to the motion of the robot.
   Let v_t denote the linear velocity obtained from the encoders and ω_t denote the yaw rate obtained from the IMU at time t. The robot's motion can be predicted using the differential-drive motion model. We aim to estimate the robot's pose at each time step based on the initial identity pose. The robot odometry estimation problem can be formulated as:

$$x_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = x_t + \tau_t \begin{bmatrix} v_t sinc(\frac{w_t \tau_t}{2}) \cos(\theta_t + \frac{w_t \tau_t}{2}) \\ v_t sinc(\frac{w_t \tau_t}{2}) \sin(\theta_t + \frac{w_t \tau_t}{2}) \\ w_t \end{bmatrix}$$

3. Trajectory Plotting: Verify the accuracy of the motion model predictions by plotting the trajectory of the robot. This allows for visual inspection of the predicted motion and comparison with the actual path taken by the robot.

3.3 Point-cloud registration via iterative closest point (ICP)

3.3.1 Kabsch Algorithm

The Kabash algorithm aims to find the optimal transformation $p \in \mathbb{R}^d$ and rotation matrix $R \in SO(d)$ between two sets of associated points, denoted as $\{m_i\}$ and $\{z_i\}$. The problem is formulated as

minimizing the objective function $f(R, p)$ given by:

$$f(R,p) = \sum_i w_i \|(Rz_i + p) - m_i\|_2^2$$

where w_i represents the weight associated with each point.

1. Optimal Translation Calculation: The optimal translation p is obtained by setting the gradient of $f(R, p)$ with respect to p to zero, yielding:

$$0 = \nabla_p f(R,p) = 2\sum_i w_i \left((Rz_i + p) - m_i\right)$$

The centroids of the point clouds are defined as:

$$\bar{m} = \frac{\sum_i w_i m_i}{\sum_i w_i} \quad \text{and} \quad \bar{z} = \frac{\sum_i w_i z_i}{\sum_i w_i}$$

Solving $\nabla_p f(R, p) = 0$ for p leads to:

$$p = \bar{m} - R\bar{z}$$

where $\bar{m}$ and $\bar{z}$ are the centroids of the point clouds $\{m_i\}$ and $\{z_i\}$, respectively. This expression represents the optimal translation that aligns the centroids of the two point clouds after applying the optimal rotation R. Substituting $p = \bar{m} - R\bar{z}$ into $f(R, p)$ yields:

$$f(R, \bar{m} - R\bar{z}) = \sum_i w_i \|R(z_i - \bar{z}) - (m_i - \bar{m})\|_2^2$$

Define the centered point clouds as $\delta m_i = m_i - \bar{m}$ and $\delta z_i = z_i - \bar{z}$.

2. Optimal Rotation Calculation: The problem reduces to finding the optimal rotation R that aligns the two associated centered point clouds $\{\delta m_i\}$ and $\{\delta z_i\}$. This is formulated as a linear optimization problem in $SO(d)$, known as Wahba's problem:

$$\max \quad \text{tr}(Q^T R)$$

where $Q = \sum_i w_i \delta m_i \delta z_i^T$. Let $Q = U\Sigma V^T$ be the singular value decomposition of Q. The singular vectors U and V and singular values Σ satisfy certain properties. The optimal rotation matrix R is then calculated as:

$$R = U \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(UV^T) \end{pmatrix} V^T$$

By following the Kabash algorithm, we can efficiently determine the optimal transformation and rotation between two sets of associated points, which is crucial for accurate point-cloud registration in SLAM applications.

3.3.2 Unknown Data Association with Iterative Closest Point (ICP)

To address the problem of unknown data association between sets of points $\{m_i\}$ and $\{z_j\}$, we employ the Iterative Closest Point (ICP) algorithm. This iterative approach iterates between finding associations based on closest points and determining the transformation p and rotation R using the Kabsch algorithm.

1. Initialize with initial guesses p_0 and R_0.
2. Given p_k and R_k, the algorithm iterates between finding correspondences based on closest points and applying the Kabsch algorithm to determine the optimal transformation.
3. For each point m_i, find the corresponding point z_j that minimizes the squared Euclidean distance:

$$j \leftrightarrow \arg\min_j \|m_i - (R_k z_j + p_k)\|_2^2$$

   This step involves associating each point m_i with the closest point z_j based on the current transformation R_k and p_k.
4. Given the correspondences (i, j) determined in the previous step, apply the Kabsch algorithm to find the optimal transformation p_{k+1} and rotation R_{k+1} that minimize the overall squared Euclidean distance between the associated points.
5. Update p_k and R_k with the newly computed p_{k+1} and R_{k+1}. Iterate the process until convergence criteria are met, such as a maximum number of iterations or a sufficiently small change in the transformation parameters.

By iteratively refining the transformation between the point sets using the ICP algorithm, we can effectively address the problem of unknown data association and accurately align the two sets of points, enabling further processing for tasks such as point-cloud registration in SLAM applications.

3.3.3 Warm-up

In order to determine the initial pose, I make the assumption that rotation occurs solely around the z-axis. To accomplish this, I discretize the angles into 360-degree increments between 0 and 360. Subsequently, I construct a rotation matrix based on the chosen yaw angle, and identify the optimal translation vector given this rotation. Utilizing the NearestNeighbors function from the sklearn library, I calculate the distances between corresponding points. The angle yielding the lowest mean squared distance between pairs is then chosen as the initial estimate for the ICP algorithm. Following the selection of the initial estimate, I proceed to execute the ICP algorithm. This involves running the algorithm to determine the rotation matrix and translation vector required to align the source point cloud with the target point cloud.
The results of the sections referenced as 4.3 and 4.4 demonstrate the effectiveness of the Iterative Closest Point (ICP) algorithm in improving the initial pose estimation by discretizing the yaw angle. However, the algorithm encounters a challenge as it becomes trapped in a local optimum. This limitation is likely attributed to the heavy reliance of ICP on the initial pose. In many instances, the assumption that an object only rotates around the z-axis may not hold true, leading to rotations around other axes as well. Consequently, the initial pose estimation may not be sufficiently accurate, particularly given the presence of a dense point cloud. As a result, the impact of an imperfect initial pose becomes significant and cannot be ignored.

3.3.4 Scan matching

This technique involves aligning consecutive LiDAR scans to estimate the robot's motion between two consecutive poses. The theoretical foundation of lidar scan matching lies in the computation of relative pose transformations between successive poses, which is crucial for accurate localization and mapping. In the given scenario, a horizontal LiDAR system with a 270° field of view and a maximum range of 30 meters is utilized to detect obstacles in the environment. Each LiDAR scan provides 1081 measured range values, covering an angular range from $-135°$ to 135°. However, measurements falling below 0.1 meters or exceeding 30 meters are considered erroneous and are filtered out. To calculate the coordinates of the obstacles detected by the LiDAR, the following equations can be employed:

$$x = \cos(\alpha_i) \cdot r$$
$$y = \sin(\alpha_i) \cdot r$$

Where: x and y represent the Cartesian coordinates of the obstacle, α_i is the angle corresponding to the i-th range measurement, r is the measured range value corresponding to the i-th measurement. For each pair of consecutive poses x_t and x_{t+1}, I identify the LiDAR frames with the smallest timestamp gap between the IMU and LiDAR data. Using the implemented ICP algorithm, I compute the relative pose between these poses. Firstly, I filter the LiDAR's range measurements as previously mentioned and compute the coordinates of obstacles in the LiDAR frame. With knowledge of the translation vector between the LiDAR and the robot center, I can then calculate the coordinates of obstacles in the robot center frame. Next, I determine the initial guess of the rotation matrix (R_0) and translation vector (P_0). If $x_t = [x_t, y_t, \theta_t]$ and $x_{t+1} = [x_{t+1}, y_{t+1}, \theta_{t+1}]$, the relative pose is calculated as follows:

$$R_0 = \begin{bmatrix} \cos(\delta\theta) & -\sin(\delta\theta) & 0 \\ \sin(\delta\theta) & \cos(\delta\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1}$$

$$P_0 = \left(\begin{bmatrix} \cos(\theta_t) & -\sin(\theta_t) & 0 \\ \sin(\theta_t) & \cos(\theta_t) & 0 \\ 0 & 0 & 0 \end{bmatrix}^T \right) \begin{bmatrix} x_{t+1} - x_t \\ y_{t+1} - y_t \\ 0 \end{bmatrix} \tag{2}$$

The relative rotation matrix (R_0) represents a rotation around the yaw angle with an angle equal to the difference in yaw angles between the two consecutive poses ($\delta\theta = \theta_{t+1} - \theta_t$). The initial translation vector (P_0) is the rotated translation from the world frame to the frame at time t. Subsequently, the ICP algorithm is employed to compute the optimal rotation and translation. I extract the yaw angle from this rotation matrix and calculate the updated rotation angle for time $t + 1$. Additionally, I rotate the best-calculated translation from frame t to the world frame and then compute the $t + 1$ transition array based on it.
The impact of lidar scan matching on our project's outcomes is significant, as it enables accurate estimation of the robot's motion and improves the overall SLAM performance. By leveraging lidar scan matching, we can effectively address the challenges posed by dynamic environments and sensor noise, ultimately enhancing the reliability and efficiency of our autonomous navigation system.

3.4 Occupancy and texture mapping

3.4.1 Probabilistic Occupancy Grid Mapping

Occupancy Grid Mapping is one of the simplest and widely adopted representations for modeling environments. It involves dividing the environment into a regular grid with n cells. Each cell in the grid is represented by a binary value in a vector $m \in \mathbb{R}^n$, where a value of -1 indicates a free

cell and a value of 1 indicates an occupied cell. The objective of occupancy grid mapping is to estimate the occupancy grid m given the trajectory of the robot $x_{0:t}$ and a sequence of observations $z_{0:t}$. Since the map is unknown and the measurements are uncertain, a probability mass function $p(m|z_{0:t}, x_{0:t})$ is maintained over time. To track the probability of each cell being occupied, denoted as $\gamma_{i,t} := p(m_i = 1|z_{0:t}, x_{0:t})$, we model the map cells m_i as independent Bernoulli random variables. Each cell m_i is assigned a value of $+1$ (occupied) with probability $\gamma_{i,t}$ and a value of -1 (free) with probability $1 - \gamma_{i,t}$.
The odds ratio of the Bernoulli random variable m_i is updated using Bayes' rule as follows:

$$o(m_i|z_{0:t}, x_{0:t}) := \frac{p(m_i = 1|z_{0:t}, x_{0:t})}{p(m_i = -1|z_{0:t}, x_{0:t})} = \frac{\gamma_{i,t}}{1 - \gamma_{i,t}} = \frac{p_h(z_t|m_i = 1, x_t)}{p_h(z_t|m_i = -1, x_t)} \frac{\gamma_{i,t-1}}{1 - \gamma_{i,t-1}}$$

The observation model odds ratio $g_h(z_t|m_i, x_t)$ specifies the ratio of true positives to false positives, indicating the level of trust in the observations. It is calculated as:

$$g_h(z_t|m_i, x_t) = \frac{p(m_i = 1|z_t, x_t)}{p(m_i = -1|z_t, x_t)} \frac{p(m_i = -1)}{p(m_i = 1)}$$

Additionally, the inverse observation model odds ratio $\frac{p(m_i=-1)}{p(m_i=1)}$ represents the prior occupancy odds ratio, which reflects the initial belief about the likelihood of occupancy for each cell. The log-odds of the Bernoulli random variable m_i (λ_i, t) is calculated as:

$$\lambda_{i,t} := \log o(m_i|z_{0:t}, x_{0:t})$$

The log-odds occupancy grid mapping involves accumulating the log-odds ratio $\Delta\lambda_{i,t}$ of the inverse measurement model over time:

$$\lambda_{i,t} = \lambda_{i,t-1} + (\Delta\lambda_{i,t} - \lambda_{i,0})$$

If the map prior is uniform ($\lambda_{i,0} = \log 1 = 0$), the log-odds ratio $\Delta\lambda_{i,t}$ of the inverse measurement model specifies the level of trust in the measurements. For instance, for an 80% correct sensor, $\Delta\lambda_{i,t} = \log 4$ if z_t indicates m_i is occupied, and $\Delta\lambda_{i,t} = -\log 4$ if z_t indicates m_i is free.

3.4.2 LiDAR Occupancy Grid Mapping

To create an occupancy map, I begin by associating each LiDAR frame with the best corresponding pose, determined by the closest timestamps between the LiDAR and IMU data. Similar to the ICP process, I calculate the coordinates of obstacles in the LiDAR frame, transfer them to the robot frame, and then to the world frame using the corresponding pose. Using Bresenham's line rasterization algorithm, I identify the cells that the LiDAR beams pass through. For each observed cell i, I adjust the log-odds value accordingly: I decrease the log-odds if the cell was observed to be free or increase the log-odds if the cell was observed to be occupied. This adjustment is typically performed by adding or subtracting 1. To prevent overconfident estimations, I set upper and lower bounds on the log-odds values (λ_{MIN} and λ_{MAX}, respectively) to constrain them within a reasonable range, typically between -100 and 100. After updating the map with all LiDAR scans, I recover the probability mass function (pmf) $\gamma_{i,t}$ from the log-odds $\lambda_{i,t}$ using the logistic sigmoid function:

$$\gamma_{i,t} = p(m_i = 1|z_{0:t}, x_{0:t}) = \sigma(\lambda_{i,t}) = \frac{\exp(\lambda_{i,t})}{1 + \exp(\lambda_{i,t})}$$

This process ensures that the occupancy grid map accurately reflects the probability of each cell being occupied based on the observed LiDAR scans and robot trajectory over time.

3.4.3 Texture Mapping

An RGBD camera, consisting of RGB and depth sensors, is utilized in the setup. The depth camera is positioned at coordinates $(0.18, 0.005, 0.36)$ meters relative to the robot center, with orientation angles roll of 0 radians, pitch of 0.36 radians, and yaw of 0.021 radians. The intrinsic parameters of the depth camera, denoted by K, are specified as:

$$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}$$

Additionally, there exists an x-axis offset between the depth and RGB cameras, necessitating a transformation to synchronize the color data with the depth data. To obtain the depth and pixel location (rgbi, rgbj) of the associated RGB color from the disparity image, the following conversion equations are applied:

$$dd = (-0.00304d + 3.31)$$

$$\text{depth} = \frac{1.03}{dd}$$

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

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

My approach to generating the texture map involves processing each RGBD sample and mapping it onto the world frame. Specifically, I focus on creating a floor map by identifying particles with height coordinates close to zero. For each camera image, I locate the closest depth map based on timestamps and compute the pixel coordinates (rgbi, rgbj) in the depth map. Utilizing the depth map, which provides the depth of each pixel, I determine the location of each pixel in the optical camera frame.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = zK^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Subsequently, I apply the transformation matrix ${}_oR^{-1}$ to convert the coordinates from the optical frame to regular camera frame coordinates.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Since the camera frame exhibits rotation and translation relative to the robot center, all particles undergo a transformation accordingly.

$$R = R_z(\psi)R_y(\theta)R_x(\phi) = R_z(0.021)R_y(0.36)R_x(0)$$

$$P = \begin{bmatrix} 0.18 \\ 0.005 \\ 0.36 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + P$$

Next, I identify the optimal pose based on the nearest timestamp and calculate the particle's coordinates in the world frame. If the height value of each particle falls below a predefined threshold (e.g., 0.05), I proceed to map the RGB value of the particle onto the texture map.

3.5 Pose graph optimization and loop closure

To enhance the accuracy of the robot trajectory estimation, we will employ pose graph optimization with loop closure constraints using GTSAM. Factor Graph SLAM is a powerful framework for simultaneous localization and mapping in robotics. It involves the construction of a factor graph representing the relationships between the variables to be estimated (robot states and landmark positions) and the measurements obtained from sensors. In Factor Graph SLAM, the factor graph is divided into two main components: the front-end and the back-end. The front-end is responsible for constructing the factor graph using various sources of information such as odometry, laser-scan matching, and feature matching. It involves creating nodes for the variables to be estimated, which include robot poses (T_i) and landmark states (m_j), and defining factors representing the relationships between these variables. The back-end of Factor Graph SLAM performs graph optimization to estimate the variables (robot poses and landmark positions) based on the factor graph constructed in the front-end. The optimization problem aims to minimize the error between the measured relative poses and the estimated absolute poses, thus refining the trajectory and improving the accuracy of the mapping:

$$\min_{\{x_i\}} \sum_{(i,j)\in E} \phi_{ij}(e(x_i, x_j))$$

where $\phi_{ij} : \mathbb{R}^d \to \mathbb{R}$ is a distance function, e.g., $\phi_{ij}(e) = e^T \Omega_{ij} e$ with positive-definite Ω_{ij}.
In Pose Graph Optimization, the variables are the robot poses (T_i), and the measurements include relative poses obtained from odometry and loop closures ($\bar{T}_{ij}$). Loop closure factors are introduced to account for revisits to previously seen areas, generating factors between non-successive robot poses. The optimization problem seeks to minimize the discrepancy between the measured relative poses and the estimated absolute poses, improving the accuracy of the robot trajectory.

$$e(T_i, T_j) = \log(\bar{T}_{ij}^{-1} T_{i-1} T_j)^\vee$$

$$\min_{\{T_i\}} \sum_{(i,j)\in E} \|W_{ij} \log(\bar{T}_{ij}^{-1} T_{i-1} T_j)^\vee\|_2^2$$

Factor Graph Optimization involves optimizing the factor graph with respect to the variables. This optimization process utilizes an initial guess obtained from sensor measurements and landmark initialization.
For this problem, I construct a Nonlinear Factor Graph with the number of parameters equal to the poses calculated after optimizing the poses using ICP. The initial guess for these variables is the poses calculated after applying ICP. The constraints between consecutive poses in the graph are determined as described in Equation 1. To handle loop closures, I establish edges between each pose and the subsequent 10th pose. The relative pose between these pairs is calculated using ICP. I model the noise for pose estimation as white noise with a diagonal covariance matrix equal to 0.1. Once the graph is created and the constraints are set, I solve the optimization problem using the Gauss-Newton optimizer and visualize the optimized trajectory alongside the previously calculated trajectory. These new poses allow for the creation of less noisy versions of the occupancy and texture maps of the environment.

4 Result

4.1 Dataset 20

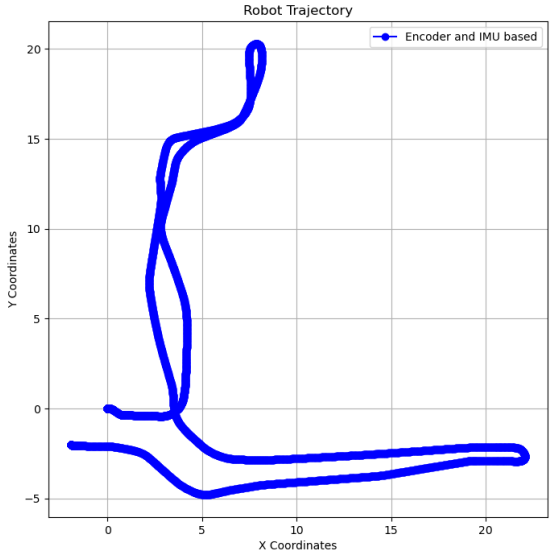


Figure 1: Trajectory of Encoder and IMU odometry

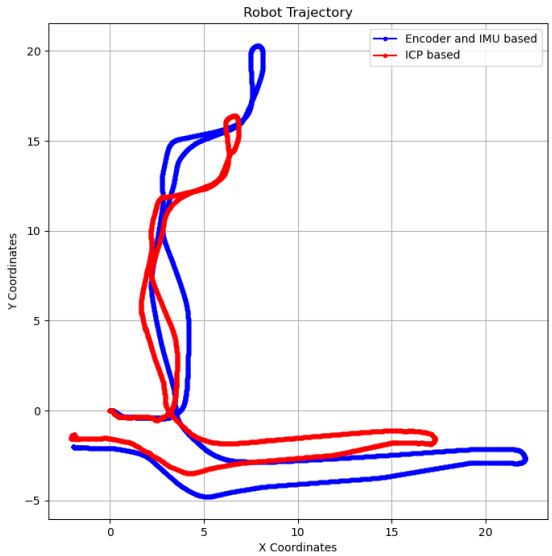
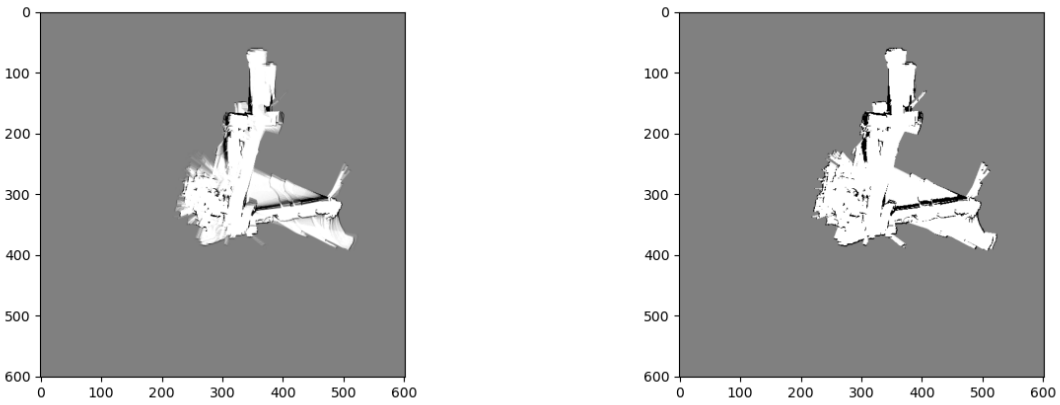


Figure 2: Trajectory of Point-cloud registration via iterative closest point (ICP)

As evident from the trajectory, there has been a slight alteration while preserving the general shapes, albeit with a slight reduction along the y-axis.



(a) log-odds matrix (b) probability map

Figure 3: Occupancy map after ICP

The Occupancy map appears satisfactory, albeit slightly noisy due to the imperfections in the trajectory. However, it aligns well with the trajectory and appears to be accurate.

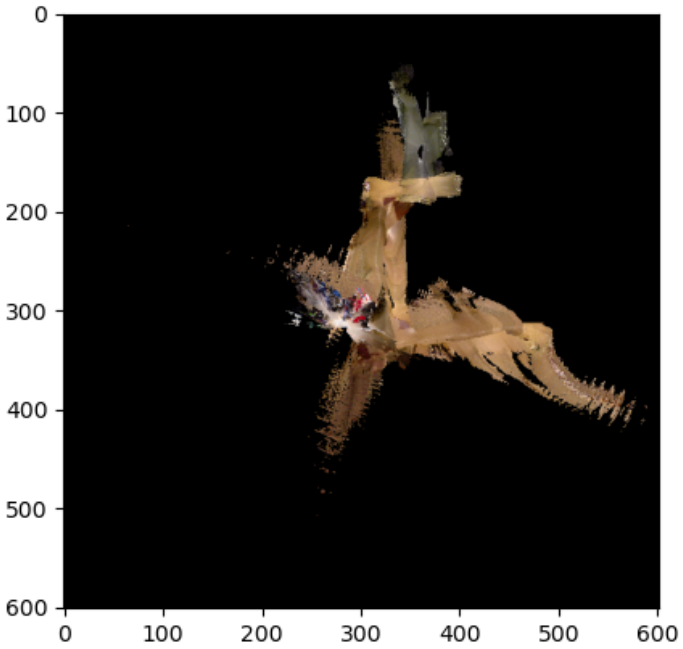
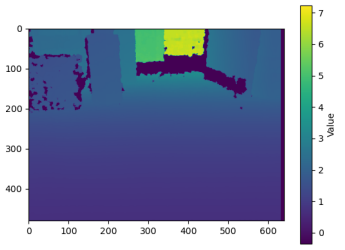
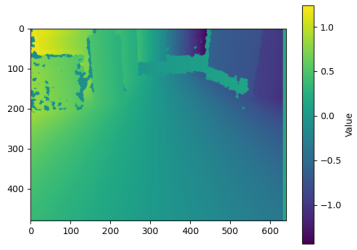


Figure 4: Texture map after ICP

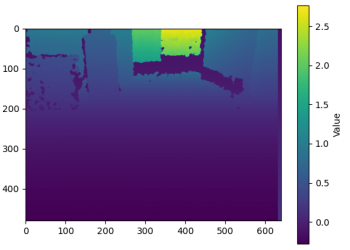
The texture map also appears to be acceptable, aligning well with both the Occupancy map and the trajectory. While some noise is present, the overall quality seems satisfactory.



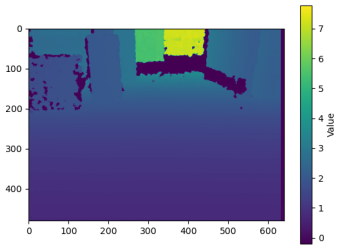
(a) x map in camera frame



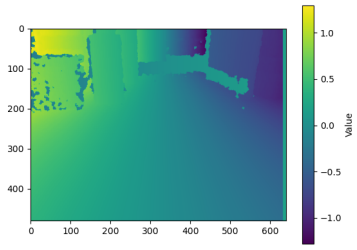
(b) y map in camera frame



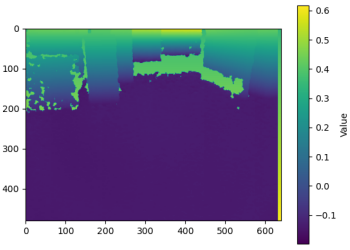
(c) z map in camera frame



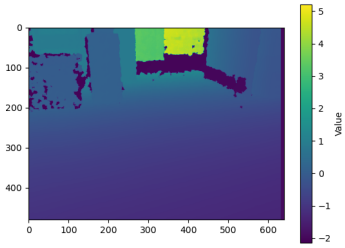
(d) x map in robot frame



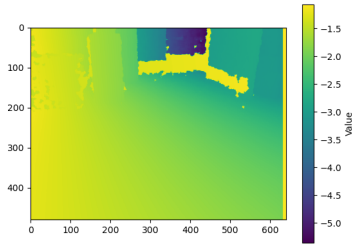
(e) y map in robot frame



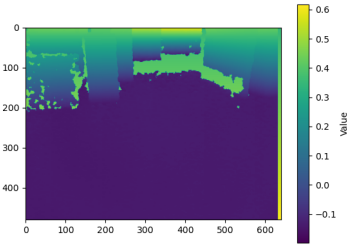
(f) z map in robot frame



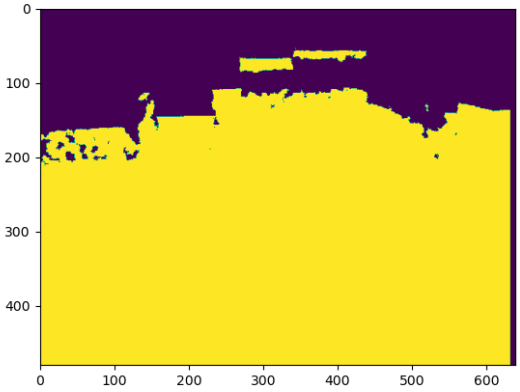
(g) x map in world frame



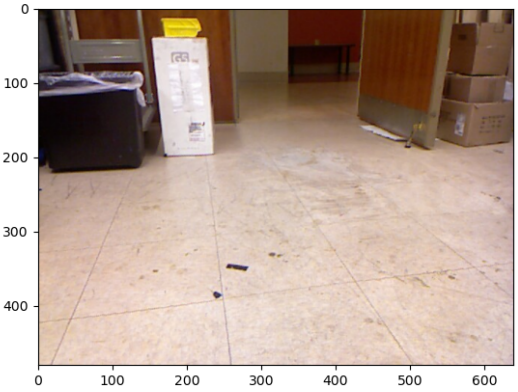
(h) y map in world frame



(i) z map in world frame



(j) calculated floor



(k) rgb image

Figure 5: Last RGBD image of dataset 20 with their heat map in three frames

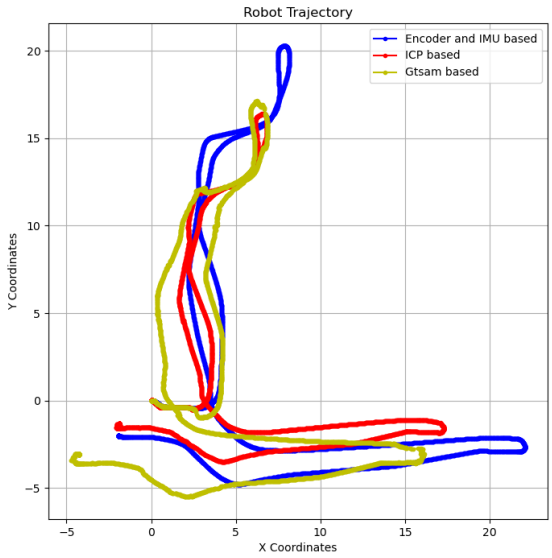
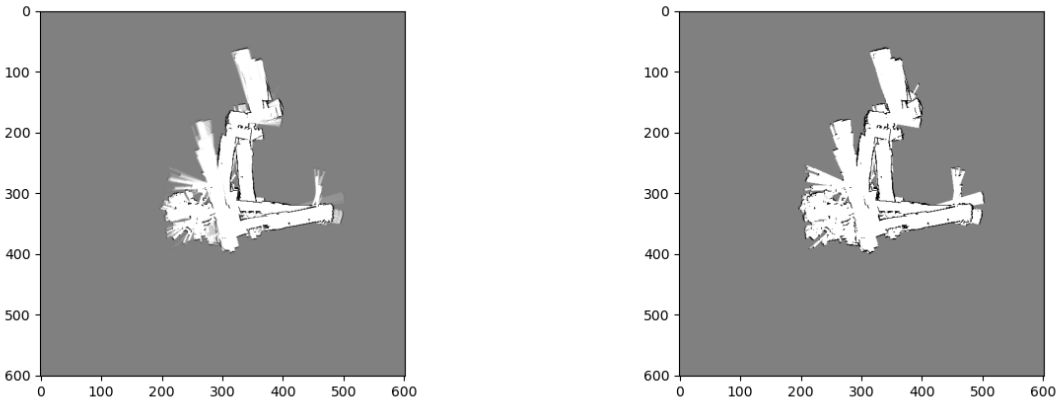


Figure 6: Trajectory of Pose graph optimization and loop closure

The GTSAM algorithm has the potential to generate a more accurate trajectory. It appears that employing Fixed-interval loop closure could effectively mitigate accumulated errors, leading to improved trajectory accuracy.



(a) log-odds matrix (b) probability map

Figure 7: Occupancy map after GTSAM

The occupancy map following GTSAM optimization appears to exhibit reduced noise compared to the occupancy map generated after ICP, highlighting the effectiveness of GTSAM in improving map quality.

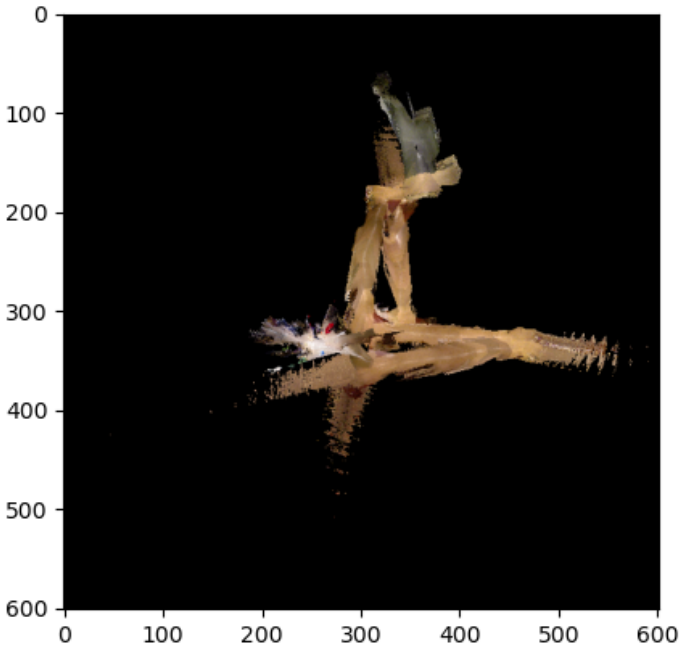


Figure 8: Texture map after GTSAM

After applying GTSAM optimization, it appears that a more accurate texture map has been generated, with many color misalignments resolved. This indicates the effectiveness of GTSAM in improving the alignment and overall quality of the texture map.

4.2 Dataset 21

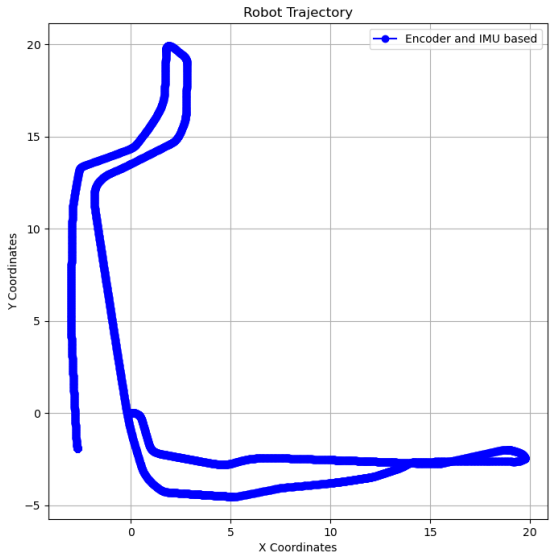


Figure 9: Trajectory of Encoder and IMU odometry

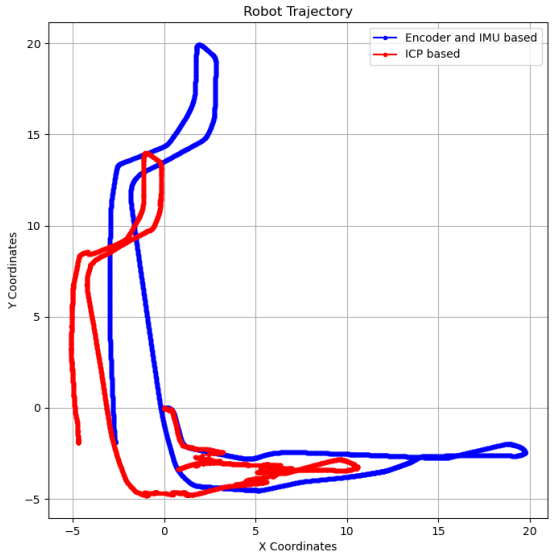
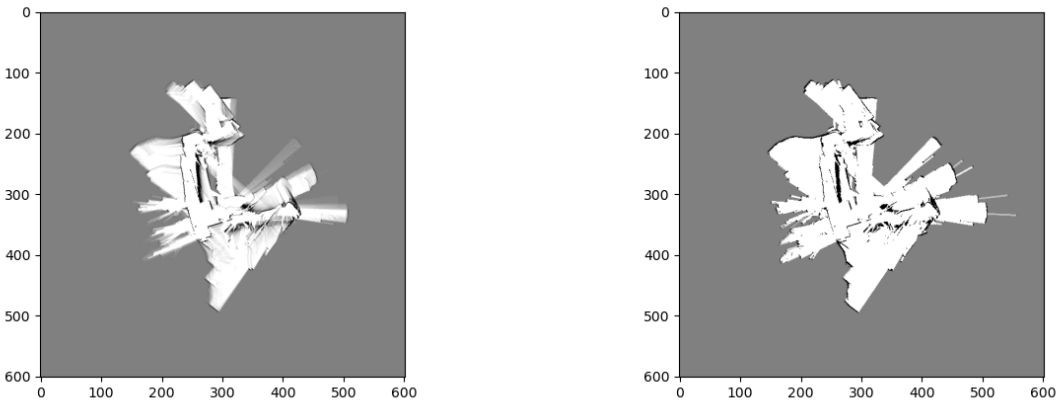


Figure 10: Trajectory of Point-cloud registration via iterative closest point (ICP)



(a) log-odds matrix (b) probability map

Figure 11: Occupancy map after ICP

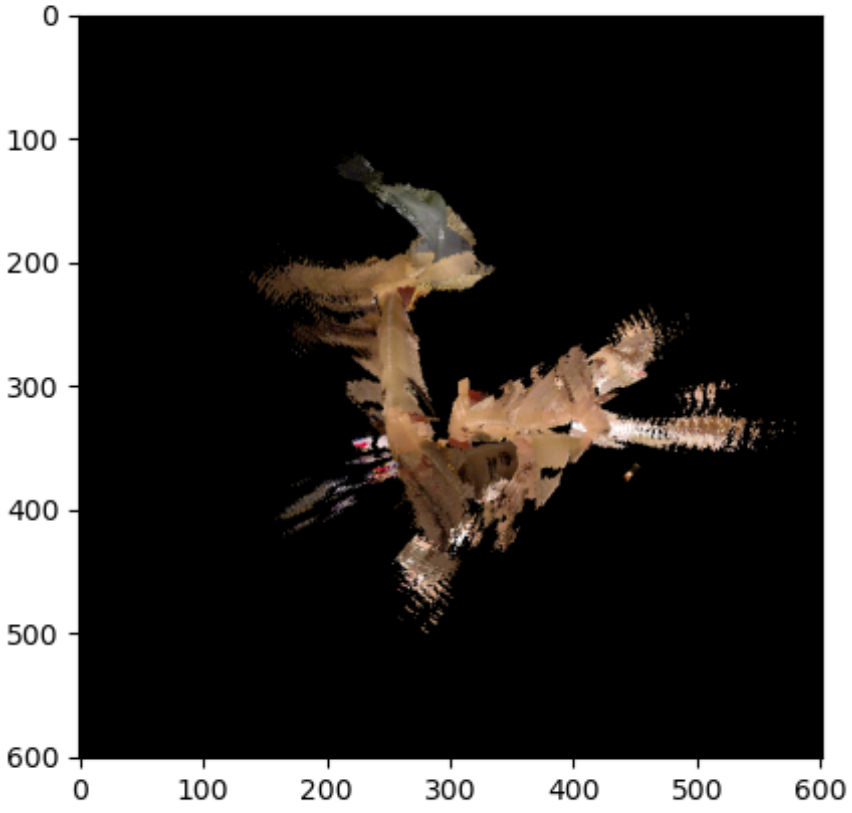
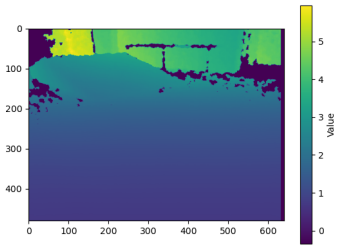
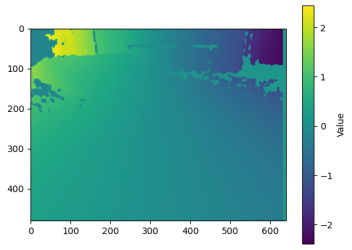


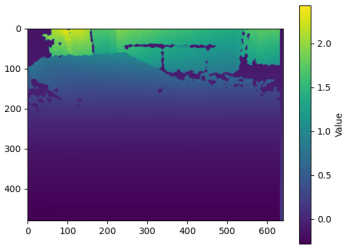
Figure 12: Texture map after ICP



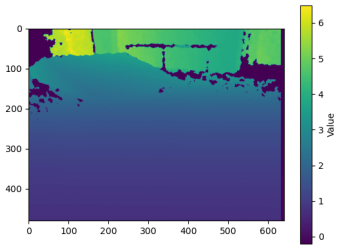
(a) x map in camera frame



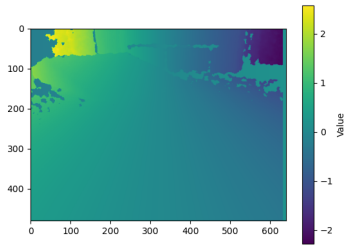
(b) y map in camera frame



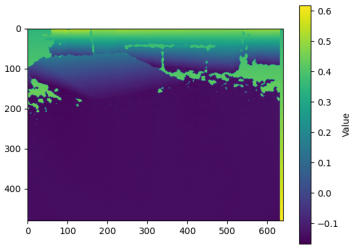
(c) z map in camera frame



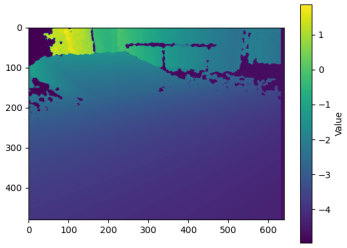
(d) x map in robot frame



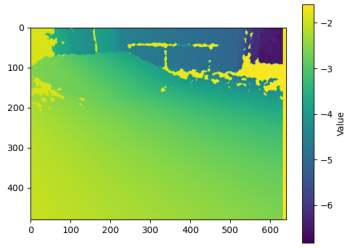
(e) y map in robot frame



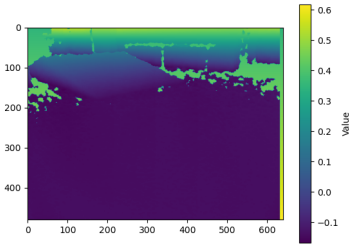
(f) z map in robot frame



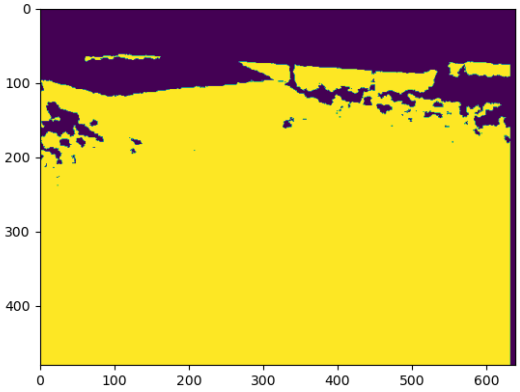
(g) x map in world frame



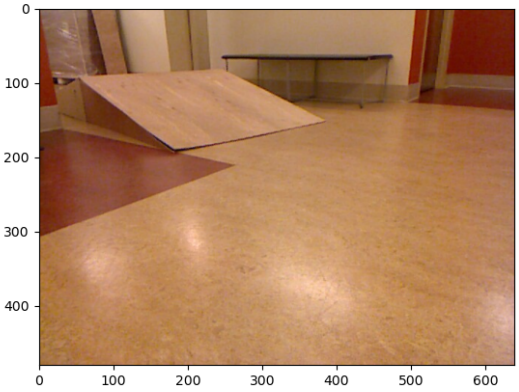
(h) y map in world frame



(i) z map in world frame



(j) calculated floor



(k) rgb image

Figure 13: Last rgbd image of dataset 21 with their heat map in three frames

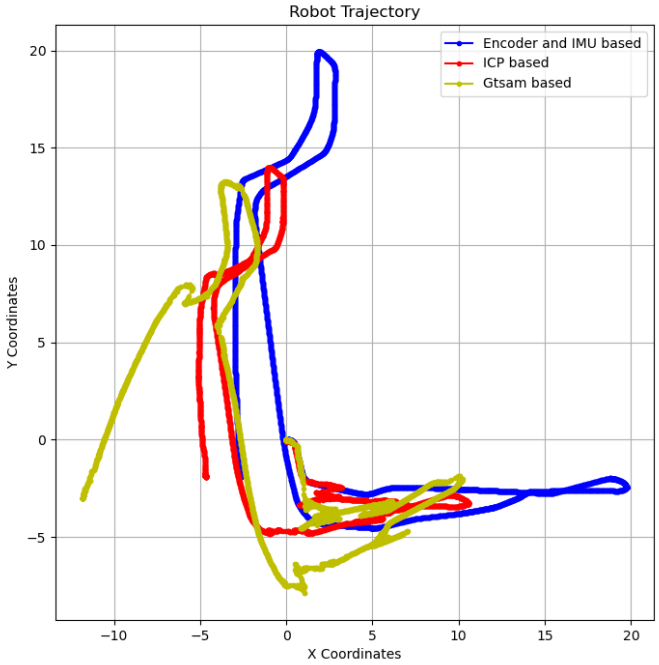
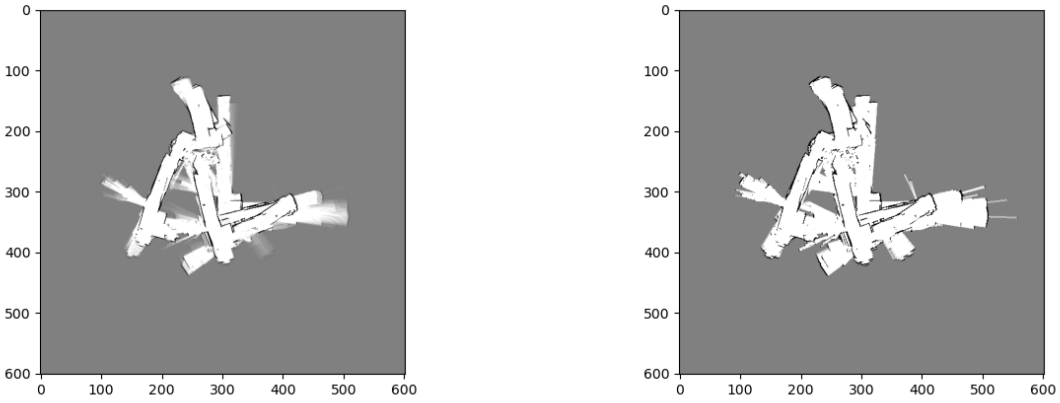


Figure 14: Trajectory of Pose graph optimization and loop closure



(a) log-odds matrix

(b) probability map

Figure 15: Occupancy map after GTSAM

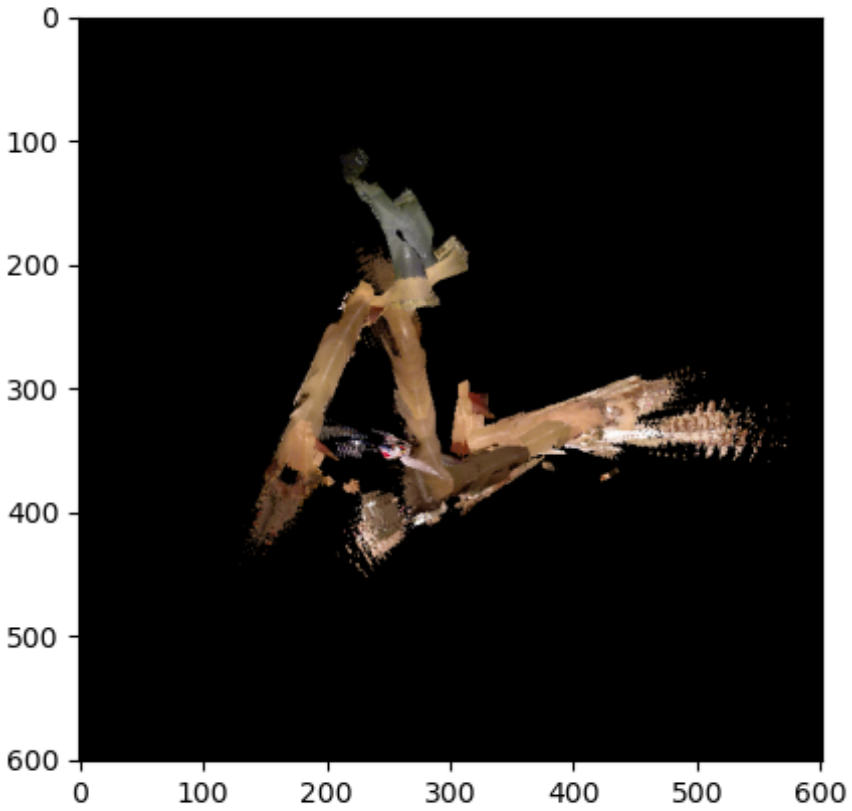


Figure 16: Texture map after GTSAM

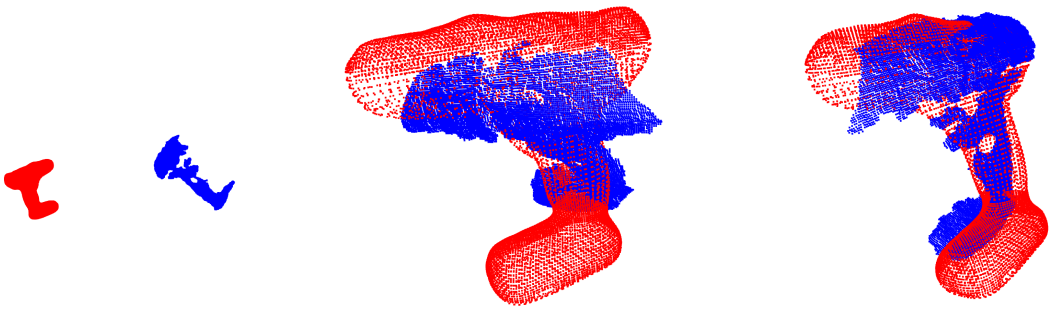
4.3 Drill ICP

(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 17: ICP Result for source 1

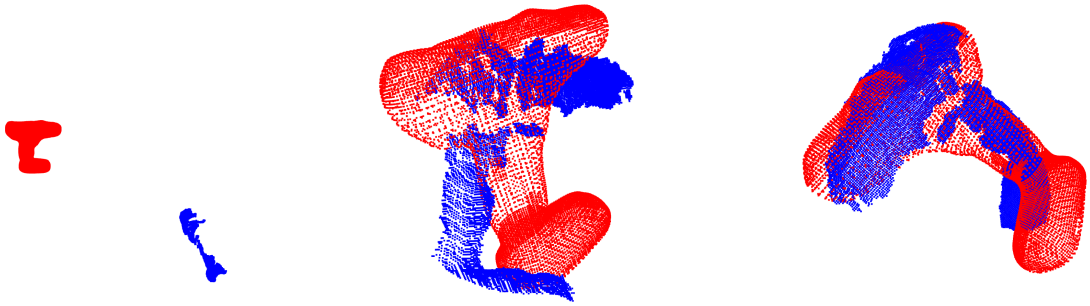


(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 18: ICP Result for source 2

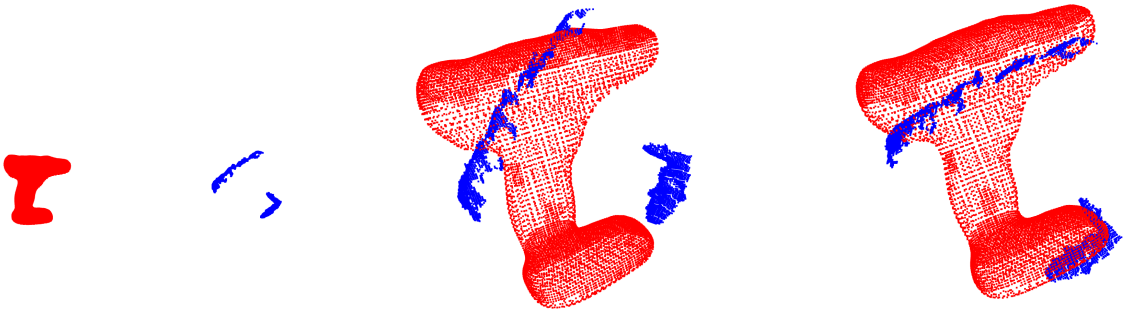


(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 19: ICP Result for source 3



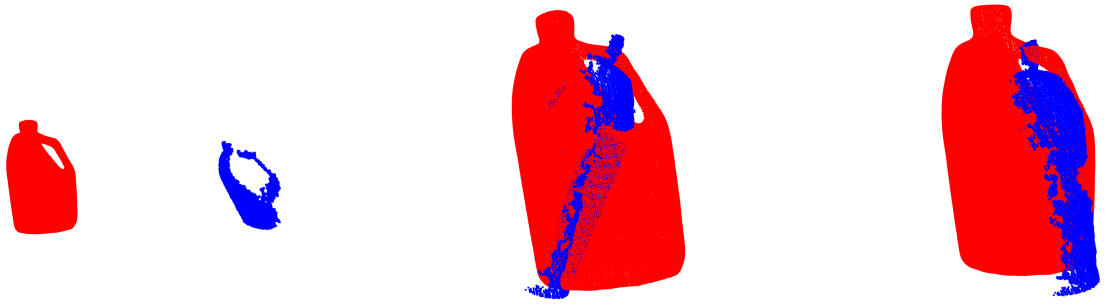
(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 20: ICP Result for source 4

4.4 Liquid Container ICP

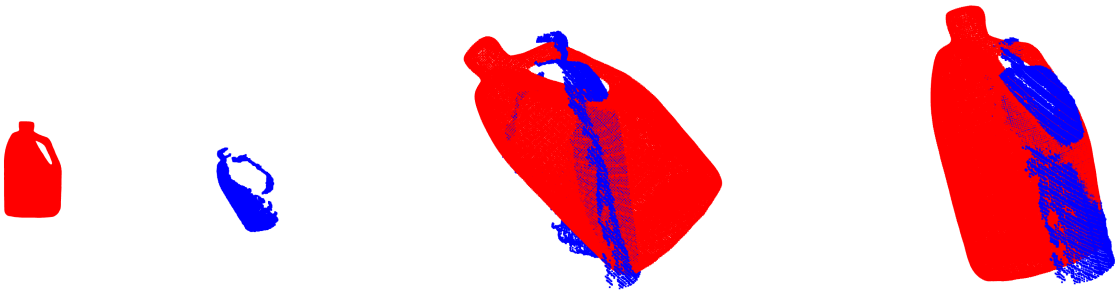


(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 21: ICP Result for source 1



(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

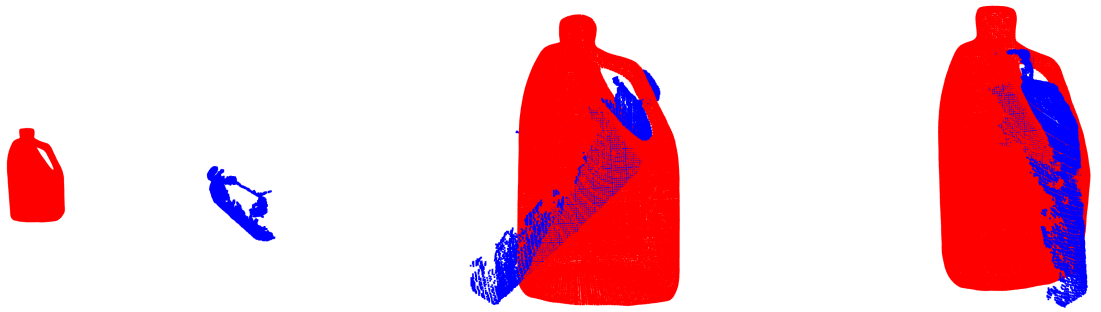
Figure 22: ICP Result for source 2

(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 23: ICP Result for source 3



(a) source and target point cloud

(b) rotated source along z axis with best score and target point cloud

(c) rotated source after ICP and target point cloud

Figure 24: ICP Result for source 4

5 Conclusion

In conclusion, our findings highlight the efficacy of the methodologies employed in this project. Through the utilization of LiDAR-based SLAM, lidar scan matching, and GTSAM optimization, we were able to achieve significant improvements in trajectory accuracy, occupancy map quality, and texture map alignment. Specifically, the results demonstrate the power of GTSAM in producing more accurate trajectories and occupancy maps, as well as in resolving color misalignments in texture maps. Looking ahead, future research could explore additional refinements to the applied methodologies to further enhance their performance. This may involve investigating alternative sensor fusion techniques, refining loop closure strategies, or exploring advanced optimization algorithms. Additionally, there is potential for integrating machine learning approaches to improve map interpretation and decision-making capabilities in complex environments.
Overall, our findings underscore the importance of continuous research and development in the field of robotic mapping and localization, with the aim of advancing the capabilities of autonomous systems for real-world applications.