ECE 276A Project 2: LiDAR-Based SLAM

Jean Gorby Calicdan
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
jcalicda@ucsd.edu

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a critical capability for autonomous mobile robots operating in unknown environments. This study presents a LiDAR-based SLAM system that integrates odometry, scan matching, occupancy grid mapping, and factor graph optimization. The proposed approach combines encoder and IMU-based odometry with LiDAR scan registration to improve localization accuracy. An occupancy grid map is generated to represent the environment, and texture mapping is applied using RGBD imaging for enhanced visualization. To further reduce localization errors, factor graph optimization with loop closure detection is implemented using the GTSAM library, ensuring global consistency in the estimated trajectory. The effectiveness of the approach is evaluated through experimental results, demonstrating improved trajectory estimation and high-fidelity environmental mapping. This study provides a comprehensive framework for robust SLAM implementation, addressing common challenges such as sensor drift and environmental uncertainty.**

***Index Terms*—localization, mapping, differential drive robot, lidar, kinect**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in mobile robotics that requires a robot to concurrently estimate its trajectory and construct a map of an unknown environment. Accurate SLAM is essential for autonomous navigation in diverse applications such as robotic exploration, autonomous vehicles, and search-and-rescue operations. This project implements a LiDAR-based SLAM pipeline utilizing multiple sensor modalities, including odometry, LiDAR, and RGBD imaging, to achieve robust localization and mapping.

The primary challenge in SLAM arises from sensor noise and drift in motion estimation. Odometry, which estimates motion using encoders and an inertial measurement unit (IMU), accumulates errors over time, leading to significant localization drift. To address this issue, LiDAR scan matching is employed to refine the odometry estimates by aligning consecutive LiDAR scans. This improves the trajectory estimation and mitigates the effects of drift. Additionally, an occupancy grid map is constructed from LiDAR data to represent the environment, enabling obstacle detection and path planning. Texture mapping is further applied using RGBD data to enhance the visual representation of the map, adding a layer of semantic information to the occupancy grid.

To further improve accuracy, factor graph optimization is incorporated into the SLAM pipeline. By modeling the trajectory as a probabilistic graph and introducing loop closure constraints, the system can correct cumulative errors and ensure global consistency in localization. Loop closure detection is achieved by identifying previously visited locations and refining the trajectory accordingly. The optimization process, implemented using the GTSAM library, significantly enhances the accuracy of the estimated trajectory and results in a more consistent and reliable map.

II. Problem Formulation

A. Encoder and IMU Odometry

Accurate localization of a mobile robot requires estimating its motion using data from encoders and an inertial measurement unit (IMU). The objective is to determine the robot's pose $x_t = [x_t, y_t, \theta_t]^T$ at each time step t using control inputs derived from these sensor measurements.

The primary inputs to the system are encoder readings, which provide wheel displacement, and IMU measurements, which provide yaw rate. Given a time interval τ_t, these inputs are used to compute the robot's motion using a differential-drive kinematic model.

The state update equations governing the robot's motion are given by:

$$x_{t+1} = x_t + v_t \cos(\theta_t)\tau_t, \tag{1}$$

$$y_{t+1} = y_t + v_t \sin(\theta_t)\tau_t, \tag{2}$$

$$\theta_{t+1} = \theta_t + \omega_t \tau_t, \tag{3}$$

where v_t represents linear velocity, and ω_t, angular velocity

B. Point-cloud Registration via Iterative Closest Point (ICP)

Given two sets of points, $\{\mathbf{m}_i\}$ and $\{\mathbf{z}_j\}$, representing a reference and a new scan, respectively, the goal is to find the optimal transformation consisting of a rotation matrix $R \in SO(d)$ and a translation vector $\mathbf{p} \in \mathbb{R}^d$. Additionally, a data association Δ must be determined to match corresponding points. The optimization problem is formulated as follows:

$$\min_{R \in SO(d), \mathbf{p} \in \mathbb{R}^d, \Delta} f(R, \mathbf{p}, \Delta) := \sum_{(i,j) \in \Delta} w_{ij} \|(R\mathbf{z}_j + \mathbf{p}) - \mathbf{m}_i\|_2^2. \tag{4}$$

C. Occupancy and Texture mapping

The goal of occupancy and texture mapping is to construct a representation of the environment using a sequence of sensor measurements and the estimated robot trajectory. Given a robot

state trajectory $x_{0:\tau}$ and a sequence of measurements $z_{0:\tau}$, the objective is to build a map m that encodes environmental information.

For occupancy mapping, the sequence of measurements consists of LiDAR scans. The environment is represented as an occupancy grid $\mathbf{m} \in \mathbb{R}^n$, where each grid cell m_i is assigned a value indicating its state:

$$m_i = \begin{cases} 1, & \text{if the cell is occupied} \\ -1, & \text{if the cell is free} \end{cases} \tag{5}$$

For texture mapping, the sequence of measurements consists of RGBD images and the map is instead assigned color values associated with the floor of the map traversed by the robot.

D. Pose Graph Optimization and Loop Closure

Pose graph optimization aims to refine the estimated robot trajectory by incorporating constraints between robot poses derived from odometry and loop closure detection. The optimization problem is formulated as:

$$\min_{\{T_i\}} \sum_{(i,j)\in E} \|W_{ij} \log(\bar{T_{ij}}^{-1} T_i^{-1} T_j)^\vee\|_2^2, \tag{6}$$

where T_i represents the robot's pose at time step i, $\bar{T_{ij}}$ is the relative pose measurement between nodes i and j, and W_{ij} is a weighting matrix that accounts for the confidence of the measurement, log is the matrix log map, and $\vee$ is the vee map which is an operator to retrieve a vector from a skew-symmetric matrix.

Loop closure detection introduces additional constraints between non-sequential robot poses by recognizing previously visited locations. These constraints help correct accumulated errors in the trajectory estimation and improve the overall consistency of the SLAM solution. By solving the optimization problem, the estimated trajectory is refined, resulting in a more accurate and globally consistent map of the environment.

III. Technical Approach

A. Differential Drive Odometry

The technical implementation of differential-drive odometry utilizes encoder readings and IMU measurements to estimate the robot's trajectory over time. The process begins with data synchronization, ensuring that encoder and IMU timestamps are aligned to facilitate accurate motion estimation. The encoder operates at 40 Hz, and each reading is reset after measurement. Given encoder counts $[FR, FL, RR, RL]$ corresponding to the front-right, front-left, rear-right, and rear-left wheels, the right and left wheel travel distances are computed as:

$$d_R = \frac{FR + RR}{2} \times 0.0022, \tag{7}$$

$$d_L = \frac{FL + RL}{2} \times 0.0022. \tag{8}$$

The factor 0.0022 meters per tick is derived from the wheel diameter (0.254 m) and 360 ticks per revolution. The time difference between consecutive encoder timestamps is computed as $\tau_t = \text{encoder\_stamps}[t] - \text{encoder\_stamps}[t-1]$.

Using these values, the linear velocity is estimated as the average of the right and left wheel displacements:

$$v_t = \frac{d_R + d_L}{2\tau_t}. \tag{9}$$

The angular velocity is directly obtained from the IMU measurement:

$$\omega_t = \text{synced\_imu\_angular\_velocity}[2, t]. \tag{10}$$

The new robot pose is computed using Euler integration:

$$x_{t+1} = x_t + \tau_t \cdot v_t \cos(\theta_t), \tag{11}$$

$$y_{t+1} = y_t + \tau_t \cdot v_t \sin(\theta_t), \tag{12}$$

$$\theta_{t+1} = \theta_t + \tau_t \cdot \omega_t. \tag{13}$$

The computed (x, y, θ) values are stored iteratively to build the estimated robot trajectory. This approach incrementally updates the robot's position based on encoder and IMU measurements.

B. ICP Warm-up

The Iterative Closest Point (ICP) algorithm is used to align two sets of point clouds by estimating the optimal transformation that minimizes alignment errors. The objective is to determine the optimal rotation matrix R and translation vector $\mathbf{p}$ that minimize the cost function:

$$\min_{R\in SO(d), \mathbf{p}\in\mathbb{R}^d, \Delta} f(R, \mathbf{p}, \Delta) := \sum_{(i,j)\in\Delta} w_{ij} \|(R\mathbf{z}_j + \mathbf{p}) - \mathbf{m}_i\|_2^2. \tag{14}$$

To compute the data association Δ, we use a KD-Tree to efficiently find the nearest neighbors between the source and target point clouds. With Δ established, the problem reduces to solving for the transformation parameters using the Kabsch algorithm. First, the centroids of both point clouds are computed as:

$$\bar{\mathbf{m}} = \frac{\sum_i w_i \mathbf{m}_i}{\sum_i w_i}, \quad \bar{\mathbf{z}} = \frac{\sum_i w_i \mathbf{z}_i}{\sum_i w_i}. \tag{15}$$

The translation component is then determined as:

$$\mathbf{p} = \bar{\mathbf{m}} - R\bar{\mathbf{z}}. \tag{16}$$

By centering the point clouds, we define the new zero-mean variables:

$$\delta\mathbf{m}_i = \mathbf{m}_i - \bar{\mathbf{m}}, \quad \delta\mathbf{z}_i = \mathbf{z}_i - \bar{\mathbf{z}}, \tag{17}$$

reducing the problem to Wahba's problem:

$$\max_{R\in SO(d)} \text{tr}(Q^T R), \quad \text{where } Q = \sum_i w_i \delta\mathbf{m}_i \delta\mathbf{z}_i^T. \tag{18}$$

This is solved using singular value decomposition (SVD), where:

$$R = U \begin{bmatrix} 1 & & \\ & \ddots & \\ & & \det(UV^T) \end{bmatrix} V^T. \tag{19}$$

ICP iterates over the following steps: (1) Finding correspondences using nearest neighbor search, (2) Estimating the optimal transformation using the Kabsch algorithm, and (3) Updating the transformation until convergence. This ensures alignment of the point clouds and reduces localization errors.

In terms of practical implementations, there are a couple of things that were attempted to improve the results of the iterative algorithm due to the fact that ICP can be sensitive to getting stuck in a local minima.

1) Downsampling the source
   - The intuition behind comes from certain results in which certain orientations may result in a very small loss but it is apparent due to visual inspection that this is isn't quite right. Such a solution can be regarded as a local minimum but has poor alignment. This is not a stable solution; therefore, downsampling points can allow the algorithm to leave this local minimum and get better results. In terms of some results that were already good, a subsample should still allow for it to be the minimum.
   - Figure 1 presents one such example. Loss evaluation may claim that this is the optimal orientation, but visual inspection suggests otherwise.

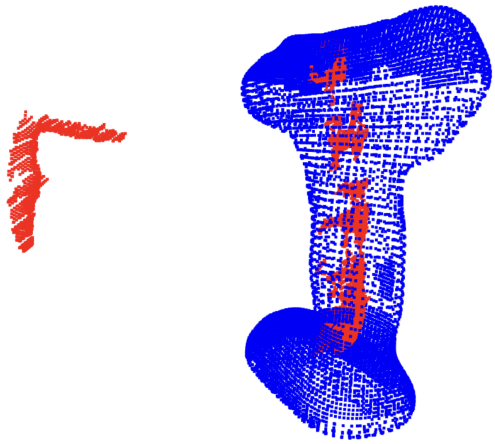

Fig. 1. Local minimum example

2) Initializing multiple yaw angles
   - ICP is known to be very sensitive to the initial guess. One way to get around this is initialize several angles for the orientation. Sufficient inializations can result in finding better optimal poses given a reasonable loss function evaluation. For this project, a simple mean square error is used.

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{m}_i - (R\mathbf{z}_i + \mathbf{p})\|_2 \tag{20}$$

3) Swapping the source and the target
   - Interestingly, one of the approaches that can result in better orientations is the swapping of what is considered the source and target. The intuition behind this come from understand what the algorithm is doing. In basic terms, we are rotating the source to match the orientation of the target. In figure 1, we can imagine that rotating the red point cloud would allow it to converge to this local maximum more easily because it is missing a lot of points that represent the object. Rotating the more complete point clouds can avoid this problem. It does need to be noted that the relative transformations need to make sense as well as this is relevant to the next section.

C. Scan Matching

Given a sequence of LiDAR scans S_t obtained at different time steps, the goal is to estimate the relative transformation ${}_t\mathbf{T}_{t+1}$ between two consecutive scans, which consists of a rotation R and a translation $\mathbf{p}$.

Each LiDAR scan provides a set of range measurements at different angles. These measurements are converted into a point cloud representation using the known sensor parameters, including the minimum angle $\alpha_{\min}$, the angular increment $\Delta\alpha$, and the measured distances r_i. The Cartesian coordinates of the points are computed as:

$$x_i = r_i \cos(\alpha_{\min} + i \cdot \Delta\alpha), \tag{21}$$
$$y_i = r_i \sin(\alpha_{\min} + i \cdot \Delta\alpha). \tag{22}$$

These points define the current scan S_t, which is aligned to the previous scan S_{t-1} using the Iterative Closest Point (ICP) algorithm.

To compute the transformation between two scans, we use the ICP algorithm.

The computed transformation $(R, \mathbf{p})$ is applied to the source scan S_t, and the process is iterated until convergence. The final transformation $\mathbf{T}_{t,t+1}$ is applied to update the global pose:

$$\mathbf{T}_{t+1} = \mathbf{T}_t *_t \mathbf{T}_{t+1}. \tag{23}$$

This refines the trajectory estimate and improves localization accuracy.

To improve convergence, ICP is initialized using the odometry estimate obtained from encoder and IMU measurements. The odometry-based relative transformation ${}_t\mathbf{T}_{t+1}^{(\text{odom})}$ serves as an initial guess for ICP:

$${}_t\mathbf{T}_{t+1} = \text{ICP}(S_{t-1}, S_t, {}_t\mathbf{T}_{t+1}^{(\text{odom})}). \tag{24}$$

D. Occupancy Grid Mapping

The map is represented as a discrete grid where each cell maintains a probability estimate of being occupied or free. Given a sequence of LiDAR scans $z_{0:t}$ and the estimated robot trajectory $x_{0:t}$, the objective is to determine the occupancy probability of each cell.

The environment is discretized into an $n \times n$ occupancy grid, where each cell m_i is modeled as a Bernoulli random variable:

$$m_i = \begin{cases} +1, & \text{if occupied} \\ -1, & \text{if free} \end{cases} \tag{25}$$

Each cell maintains a log-odds representation to track occupancy probabilities efficiently:

$$\lambda_{i,t} = \log \frac{p(m_i = 1 | z_{0:t}, x_{0:t})}{p(m_i = -1 | z_{0:t}, x_{0:t})}. \tag{26}$$

The LiDAR measurements are transformed from the sensor frame to the world frame using the robot pose:

$$\begin{bmatrix} x_w \\ y_w \end{bmatrix} = \mathbf{R}(\theta) \begin{bmatrix} x_s \\ y_s \end{bmatrix} + \begin{bmatrix} x_r \\ y_r \end{bmatrix}. \tag{27}$$

where $\mathbf{R}(\theta)$ is the rotation matrix based on the robot's heading, and (x_r, y_r) is the robot's global position.

To update the occupancy grid, the Bresenham algorithm is employed to trace each LiDAR beam from the sensor origin to the detected obstacle. The cells traversed by the ray are updated as free, while the endpoint is updated as occupied.

The log-odds occupancy values are updated recursively using Bayes' rule:

$$\lambda_{i,t} = \lambda_{i,t-1} + (\Delta\lambda_{i,t} - \lambda_{i,0}), \tag{28}$$

where $\Delta\lambda_{i,t}$ is the inverse sensor model update:

$$\Delta\lambda_{i,t} = \begin{cases} +\log 4, & \text{if cell is occupied} \\ -\log 4, & \text{if cell is free.} \end{cases} \tag{29}$$

To prevent overconfidence, the log-odds values are constrained within predefined bounds $\lambda_{\min} \leq \lambda_{i,t} \leq \lambda_{\max}$.

Once the log-odds map is computed, the final occupancy probability for each cell is obtained using the sigmoid function:

$$p(m_i = 1 | z_{0:t}, x_{0:t}) = \sigma(\lambda_{i,t}) = \frac{\exp(\lambda_{i,t})}{1 + \exp(\lambda_{i,t})}. \tag{30}$$

E. Texture Mapping

The key steps in texture mapping include projecting depth images to 3D, transforming points from the optical frame to the robot and world frames, filtering out floor points, and projecting textures onto the map.

Each depth pixel $d(i, j)$ in the disparity image is converted into a corresponding 3D point using the depth-to-disparity conversion:

$$dd = -0.00304 \cdot d + 3.31, \tag{31}$$

$$depth = \frac{1.03}{dd}. \tag{32}$$

The pixel coordinates (i, j) are then mapped to metric coordinates in the camera frame using the intrinsic calibration matrix K:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \text{depth} \cdot K^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}, \tag{33}$$

where:

$$K = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}. \tag{34}$$

Once the depth points are obtained in the optical frame, they must be transformed into the world frame. The transformation follows a sequence of coordinate changes:

$$\mathbf{p}_r = {}_o\mathbf{R}_r \mathbf{p}_o, \quad \text{(Optical to Regular Camera Frame)} \tag{35}$$

$$\mathbf{p}_b = {}_b\mathbf{R}_r \mathbf{p}_r + {}_b\mathbf{p}_r, \quad \text{(Camera to Robot Frame)} \tag{36}$$

$$\mathbf{p}_w = {}_w\mathbf{R}_b \mathbf{p}_b + {}_w\mathbf{p}_b, \quad \text{(Robot to World Frame).} \tag{37}$$

Here, ${}_o\mathbf{R}_r$ represents the fixed rotation from the optical to the regular camera frame, while ${}_b\mathbf{R}_r$ and ${}_w\mathbf{R}_b$ correspond to the transformations from the camera to the robot and from the robot to the world frame, respectively. These transformations incorporate known extrinsic calibrations and estimated robot poses from SLAM.

To accurately align RGB values with the depth image, a mapping process is performed. First, we create a grid of pixel indices for the depth image. The corresponding pixel indices in the RGB image are computed using intrinsic calibration parameters:

$$\text{rgb}_i = \frac{526.37 \cdot i_{\text{idxs}} + 19276 - 7877.07 \cdot dd}{585.051}, \tag{38}$$

$$\text{rgb}_j = \frac{526.37 \cdot j_{\text{idxs}} + 16662}{585.051}. \tag{39}$$

The corresponding RGB values are then retrieved:

$$\mathbf{c}_w = \text{RGB}(\text{rgb}_i, \text{rgb}_j). \tag{40}$$

This process ensures that each 3D point retains its correct color information from the original RGB image.

Once the 3D points are in the world frame, a floor thresholding operation is applied to remove points below a certain height, filtering out unwanted ground plane projections. The remaining points are then projected onto the 2D occupancy grid using:

$$(u, v) = \text{world\_to\_map}(x_w, y_w), \tag{41}$$

where (x_w, y_w) are the transformed world coordinates, and the function world_to_map converts these coordinates into grid cell indices. The corresponding RGB values are assigned to the texture map, creating a visually interpretable representation of the environment.

F. Factor Graph Optimization

Factor graph optimization is employed to refine the estimated robot trajectory by incorporating constraints derived from odometry and loop closure detection. In our implementation, we represent odometry generated robot poses as nodes in the factor graph, while edges encode relative transformations obtained from ICP-based scan matching. The optimization process seeks to minimize the discrepancy between the measured relative poses and estimated absolute poses, thus refining the overall trajectory.

A pose graph is defined with robot poses ${}_tT$ as nodes and relative pose constraints ${}_tT_{t+1}$ as edges. Given the relative pose measurements from odometry and scan matching, we define the error function:

$$e(T_t, T_{t+1}) = \log(\bar{T}_{t,t+1}^{-1} T_t^{-1} T_{t+1})^\vee, \tag{42}$$

where $\bar{T}_{t,t+1}$ is the observed relative transformation, and the operator $\vee$ converts the logarithm of a transformation matrix into a minimal parameterization. The graph optimization minimizes the weighted error over all constraints:

$$\min_{\{T_i\}} \sum_{(i,j)\in E} \|W_{ij} e(T_i, T_j)\|^2, \tag{43}$$

where W_{ij} represents the information matrix that encodes measurement uncertainty.

We utilize the GTSAM library to perform optimization using the Levenberg-Marquardt algorithm. Given an initial trajectory estimate obtained from odometry and ICP, the optimization iteratively updates the trajectory using:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} \delta x^{(k)}, \tag{44}$$

where $\delta x^{(k)}$ is computed by solving:

$$\left(\sum_{ij} J_{ij}^T W_{ij}^T W_{ij} J_{ij} + \lambda D \right) \delta x^{(k)} = - \sum_{ij} J_{ij}^T W_{ij}^T e(x_i^{(k)}, x_j^{(k)}). \tag{45}$$

Here, J_{ij} represents the Jacobian of the error function, and λ is a damping factor that helps balance between gradient descent and Gauss-Newton updates.

G. Loop Closure Detection

Loop closure detection introduces additional constraints in the factor graph by recognizing previously visited locations. This process corrects accumulated drift in trajectory estimation and improves localization consistency. Our implementation supports two loop closure methods:

1) Fixed-Interval Loop Closure: In this approach, loop closure constraints are introduced every fixed number of steps (e.g., every 10 poses). The relative transformation between these poses is computed using ICP:

$${}_iT_j = \text{ICP}(S_i, S_j), \tag{46}$$

where S_i and S_j represent the LiDAR scans at poses i and j. If the computed transformation satisfies a confidence threshold, it is added as an additional edge in the factor graph.

2) Proximity-Based Loop Closure: Instead of using a fixed interval, proximity-based loop closure detects loop closures dynamically by evaluating whether two poses are spatially close. If the Euclidean distance between two non-successive poses is below a threshold:

$$\|\mathbf{t}_i - \mathbf{t}_j\| < d_{\text{thresh}}, \tag{47}$$

then their corresponding LiDAR scans are aligned using ICP. If the scan alignment error is sufficiently low, a loop closure constraint is introduced in the factor graph:

$${}_iT_j = \text{ICP}(S_i, S_j), \quad \text{if error} < \epsilon_{\text{thresh}}. \tag{48}$$

3) Effect of Loop Closure on Trajectory Estimation: The introduction of loop closure constraints significantly improves trajectory accuracy by reducing accumulated drift. The final optimized trajectory is obtained by solving the factor graph optimization problem with all odometry, ICP, and loop closure constraints included:

$$\hat{T} = \arg\min_T \sum_{(i,j)\in E} \|W_{ij} e(T_i, T_j)\|^2. \tag{49}$$

This refined trajectory is then used to construct a more accurate occupancy grid map and texture map of the environment.

IV. Discussion

The project involved iterative development, debugging, and refinement to achieve the final results. Several challenges were encountered and resolved during the process:

A. Encoder and IMU Odometry

The biggest issue I ran into with this section came from my attempt at using the exact integration results describing the differential drive kinematic model. These equations were given by:

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d(\mathbf{x}_t, \mathbf{u}_t) := \mathbf{x}_t + \tau_t \begin{bmatrix} v_t \,\text{sinc}\left(\frac{\omega_t \tau_t}{2}\right) \cos\left(\theta_t + \frac{\omega_t \tau_t}{2}\right) \\ v_t \,\text{sinc}\left(\frac{\omega_t \tau_t}{2}\right) \sin\left(\theta_t + \frac{\omega_t \tau_t}{2}\right) \\ \omega_t \end{bmatrix} \tag{50}$$

In attempt to implement this version of the kinematic model, I ran into scaling issues. The initial suspect where the calculation of the velocities. Sometimes the odometry result would look correct but the scan matching would be incorrectly scaled. After debugging, I opted to just use the simpler Euler discretized model.

B. ICP Warmup

The ICP warm up seemed simple to implement but applying it with actual point clouds proved to be difficult. Initially, I kept getting results that looked like Figure 14.

Fig. 2. Incorrect Transforms

In debugging and visualizing the transformations at each iteration, it became obvious that the total transformations I was calculating was incorrect. Given n iterations of the algorithm, pose composition should be written as such:

$$_nT_0 =_n T_{n-1} * ... *_2 T_1 *_1 T_0$$

Since the algorithm was iterative, it was important that the order of matrix multiplication was correct. By following the above formula, it meant that the most recent relative transformation has to be multiplied on the left side which may be unintuitive at first. I had initially had the opposite order because I wasn't expressing the transformations in a relative way which involve two frames but rather just took it as transformations at each time steps. This was a massive error which leads to what is show in Figure 2.

C. Scan Matching

Scan matching was relatively simple to implement after ICP and odometry are finalized. As soon as the LIDAR scans dataset was understood, it was as simple as applying ICP repeatedly. In truth, the biggest struggle with this section was ensuring the ICP algorithm is implemented in a way that allows for the best convergence. As discussed in the technical approach for ICP warmup, there are a lot of ways for the algorithm to hit a local minimum. One of the issues I found in my implementation was that I was stopping the iteration when the loss has reached a small threshold. In truth, this threshold wasn't small enough so the algorithm was stopping quite early, and thus not close to the optimal transformation. Even when setting this threshold very low, it is still important to give the ICP more iterations. I observed that while more iterations does lead to more computation time, it does allow for a better-looking trajectory

D. Occupancy Grid and Texture Mapping

The occupancy grid map was simpler to implement than the texture map. The biggest issue I found with the occupancy grid was that I wasn't properly scaling my map. For a while, I was setting my robot origin incorrectly and thus I was only seeing a fraction of the entire trajectory. This was fixed by manually inspecting the scale and making sure that my map sized for the operations I was doing.

The hardest part of this section definitely has to do with texture mapping. The biggest struggle had to do with applying the transformations correctly. Thankfully, the code for projecting the RGBD images into a 3D space was given so this wasn't too much of an issue. The problem was formulating the proper transformations. As described in the technical approach section, the transformation has to be regular frame to optical frame to camera frame to robot frame to world frame. At some point, the transformations were clearly not correct and finding out which one wasn't quite simple. In end, by inspecting the point clouds at each step, it became clear that the perspective wasn't correct. It looks like the camera was staring at the floor rather than a little higher. When thresholding, this results in a lot of the floor being filtered out despite being a floor. In Figure 3, it is clear that there is floor that goes beyond the door. In my original transformation, this wasn't captured as seen in Figure 4. This resulted in a map that didn't fully capture the whole floor map.

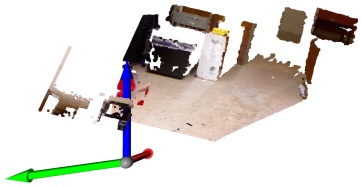

Fig. 3. First RGBD Point Cloud

Fig. 4. Incorrect First Floor Texture

E. Factor Graph Optimization

Unfortunately, I couldn't quite implement factor graph optimization correctly. I suspect that it has to do with how my nodes and edges are defined. I looked into the noise model as well and learned that the prior need to have more noise than the ICP enhanced edges. This was a tweak that I made as my original didn't follow this trend. However, this didn't quite correct the issues. There is probably also issues with the way my loop closures were implemented. The fixed interval method results in a different trajectory than ICP but the proximity criteria simply generates the same trajectory as the ICP enhanced trajectory.

V. Results

A. Encoder and IMU Odometry

See next page

B. ICP Warm-UP

See next page

C. Scan Matching

See next page

D. Occupancy Grid Mapping

See next page

E. Texture Mapping

See next page

F. Factor Graph Optimization

See next page

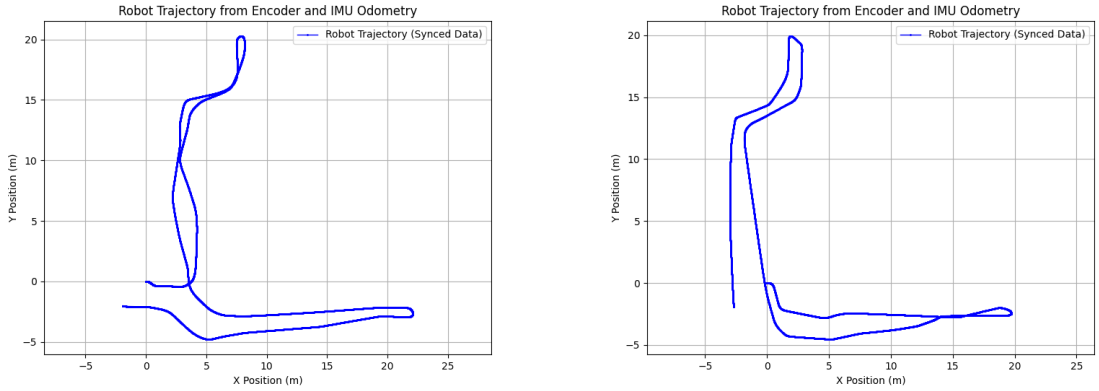

Fig. 5. Odometry Generated Trajectories for Datasets 20 and 21

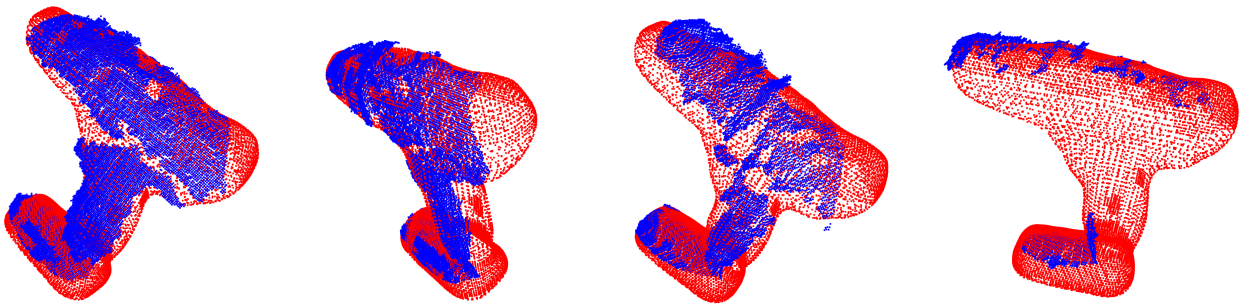

Fig. 6. ICP Drill Warm-Up Results (images 0-3 from left to right)

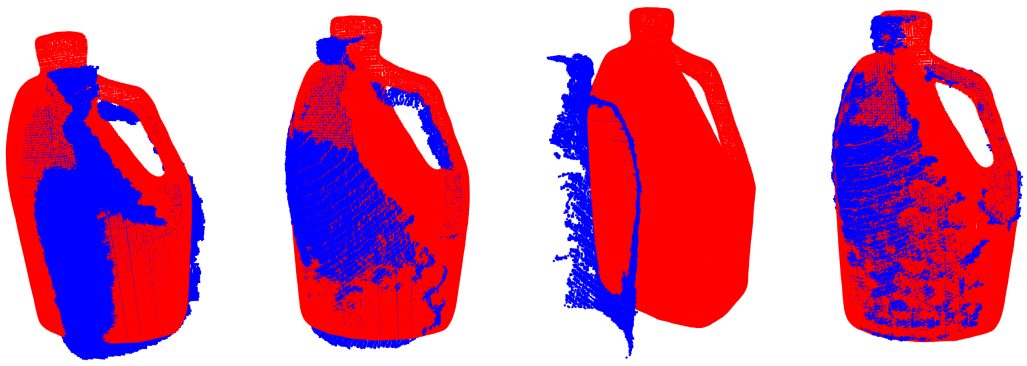

Fig. 7. ICP Liquid Container Warm-Up Results (images 0-3 from left to right)

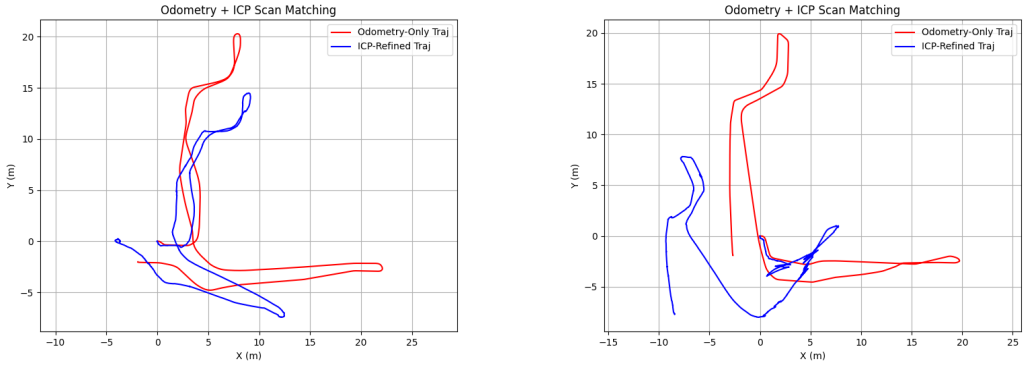

Fig. 8. ICP Enhanced Trajectories for Datasets 20 and 21

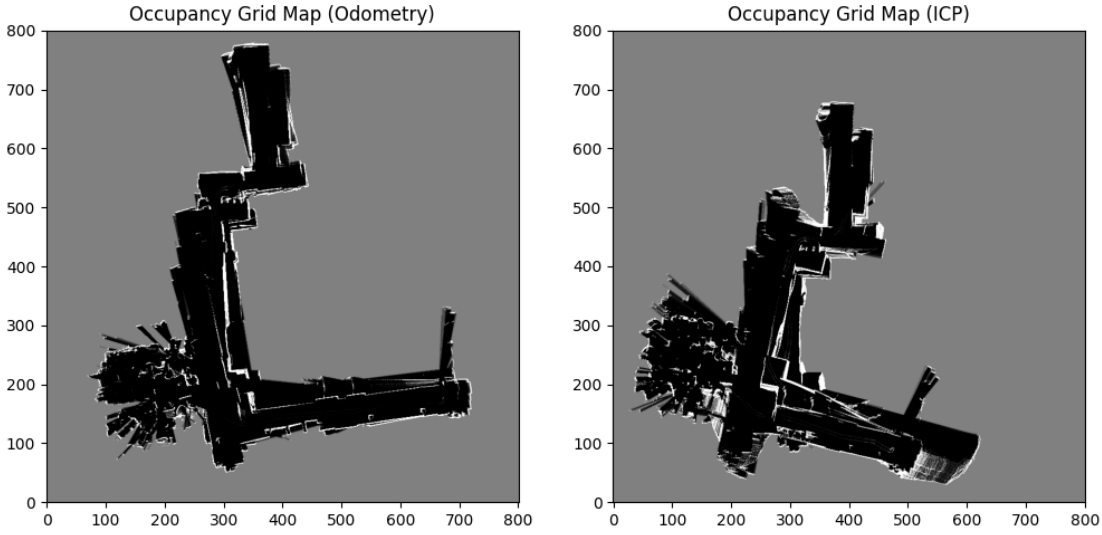

Fig. 9. Occupancy Grid Maps for Dataset 20

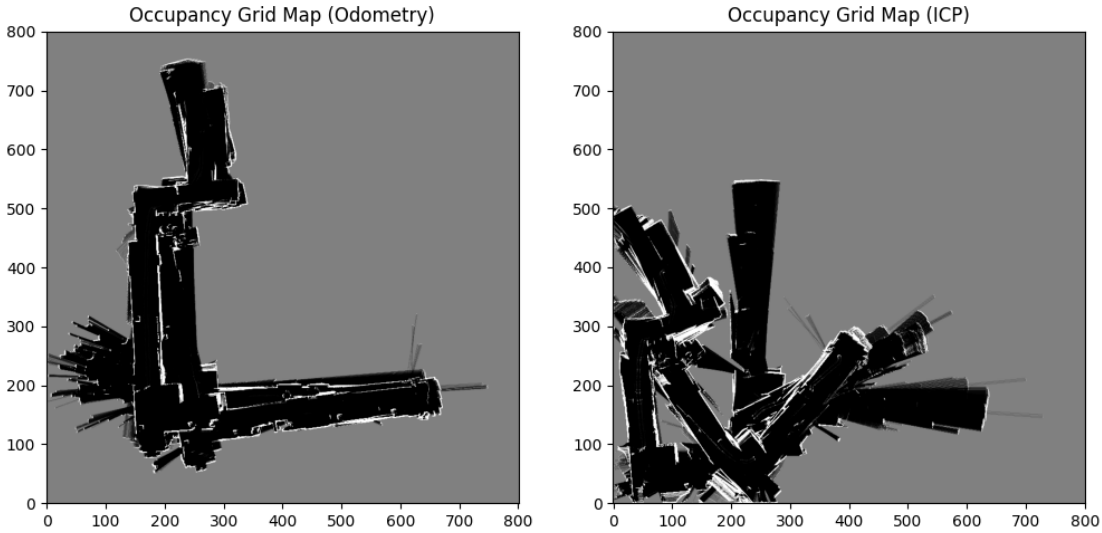

Fig. 10. Occupancy Grid Maps for Dataset 21

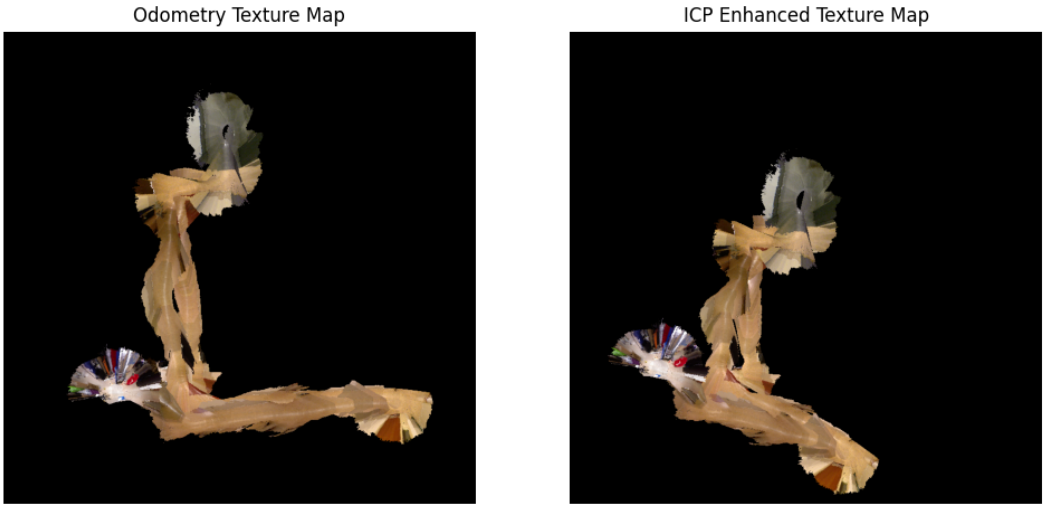

Fig. 11. Texture Maps for Dataset 20

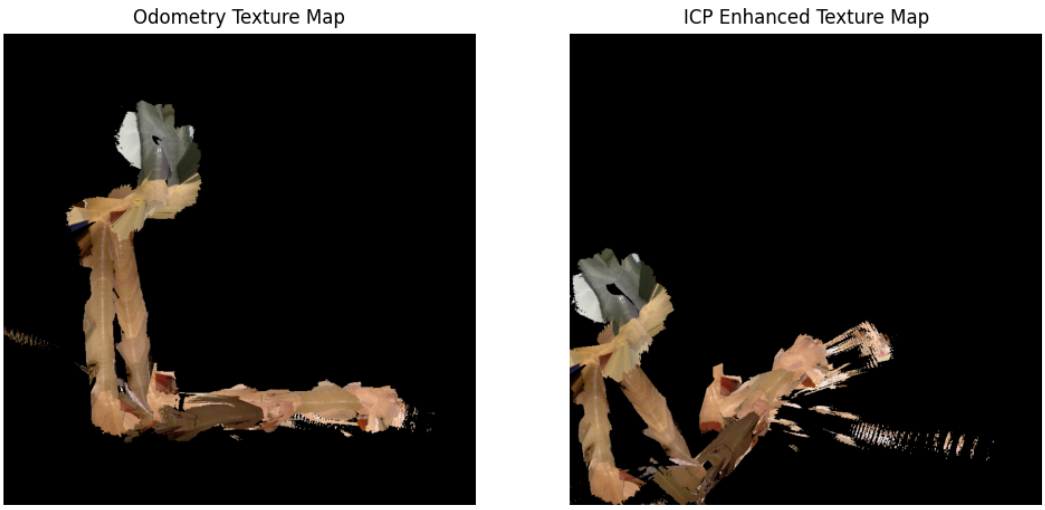

Fig. 12. Texute Maps for Dataset 21

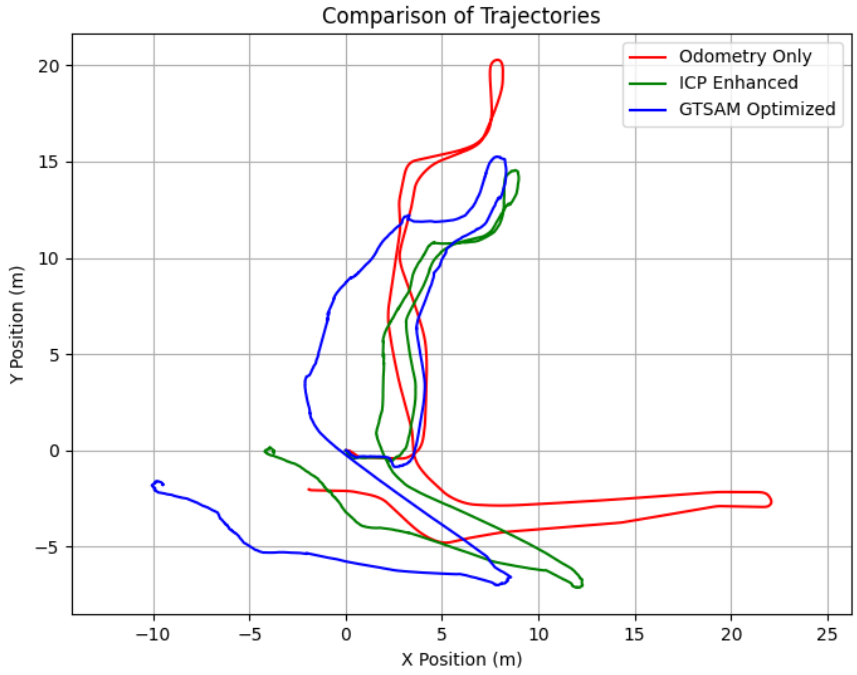

Fig. 13. Factor Graph Optimized (Fixed Interval Loop Closure) Trajectory Dataset 20

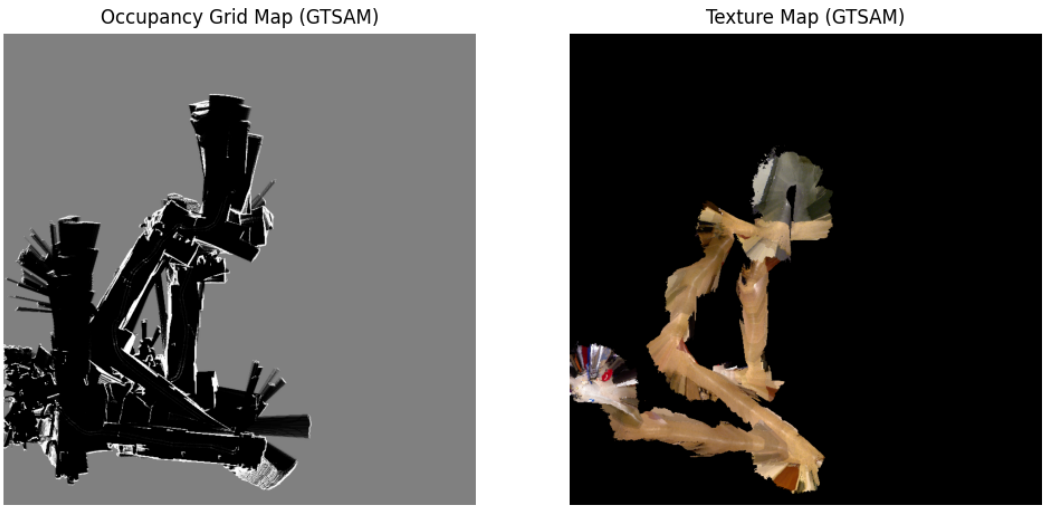

Fig. 14. Factor Graph Optimized Grid (Fixed Interval Loop Closure) and Texture Map for Dataset 20

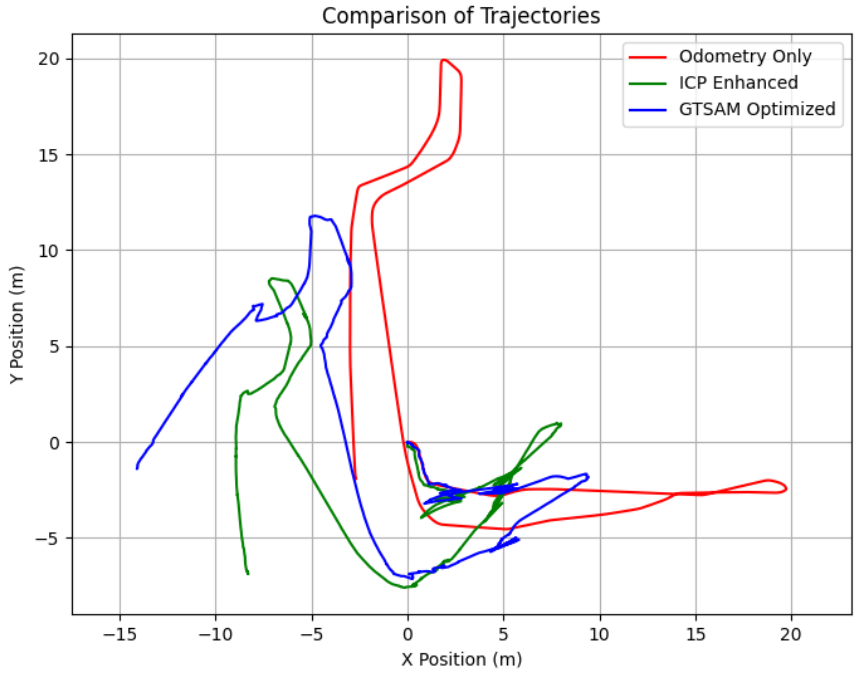

Fig. 15. Factor Graph Optimized (Fixed Interval Loop Closure) Trajectory Dataset 21

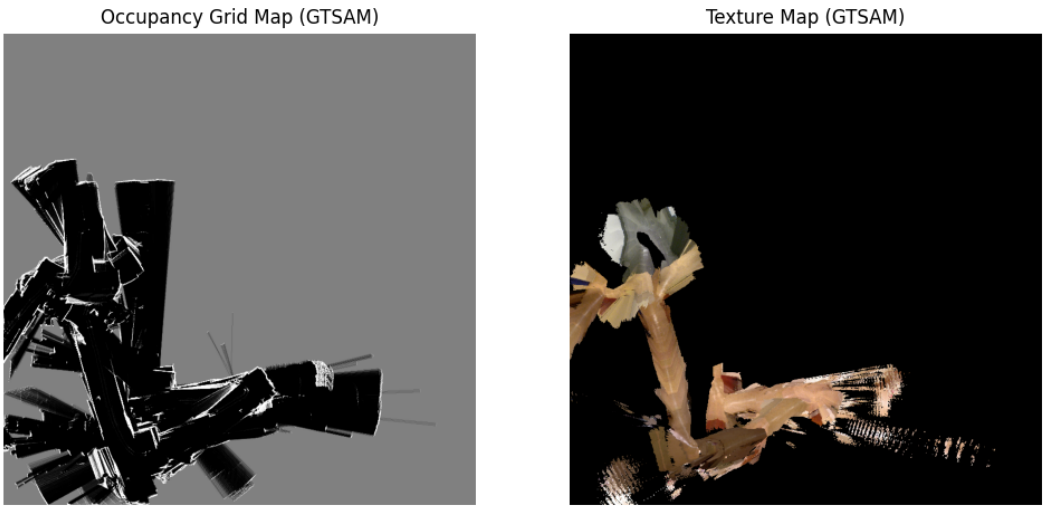

Fig. 16. Factor Graph Optimized (Fixed Interval Loop Closure) Grid and Texture Map for Dataset 21

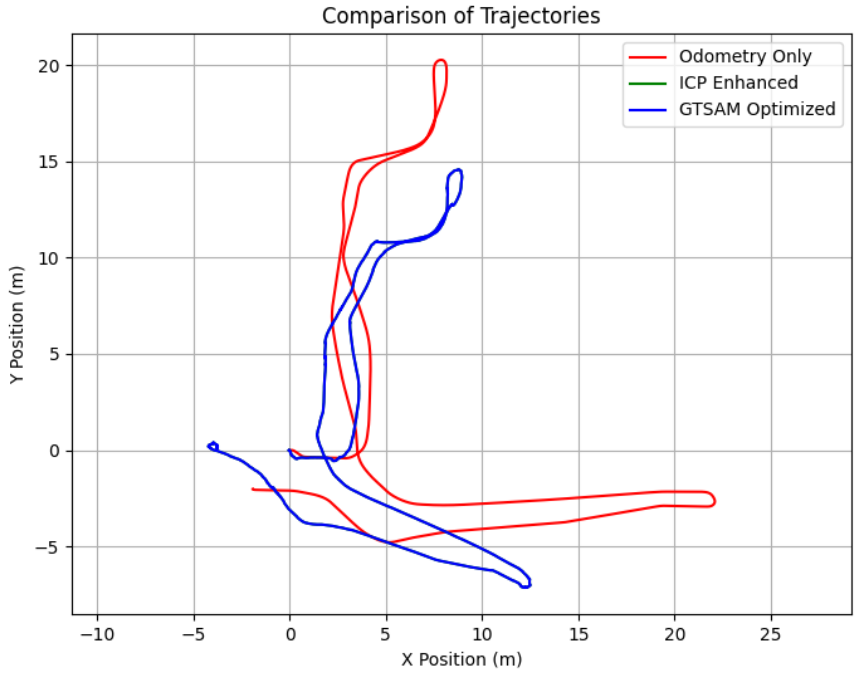

Fig. 17. Factor Graph Optimized (Proximity Based Loop Closure) Trajectory Dataset 20

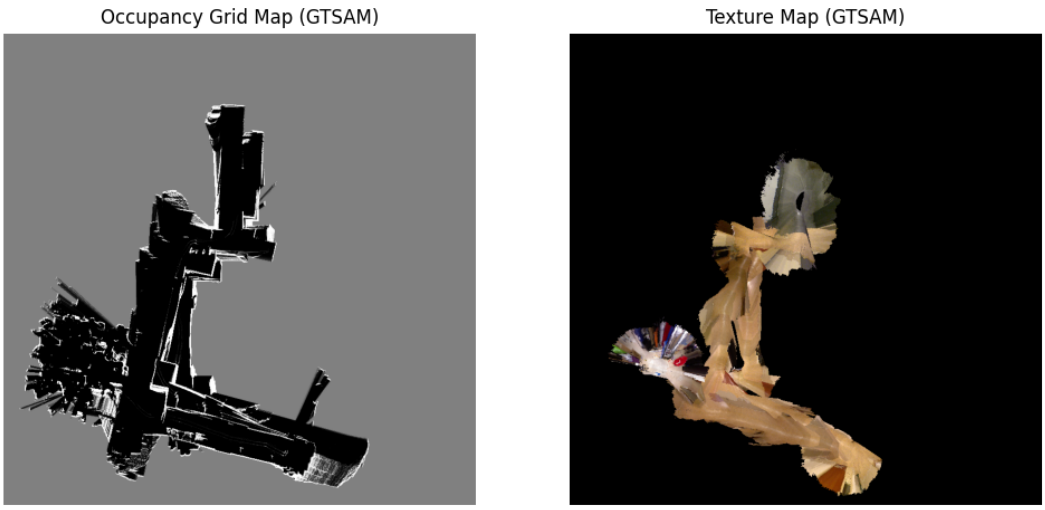

Fig. 18. Factor Graph Optimized Grid (Proximity Based Loop Closure) and Texture Map for Dataset 20

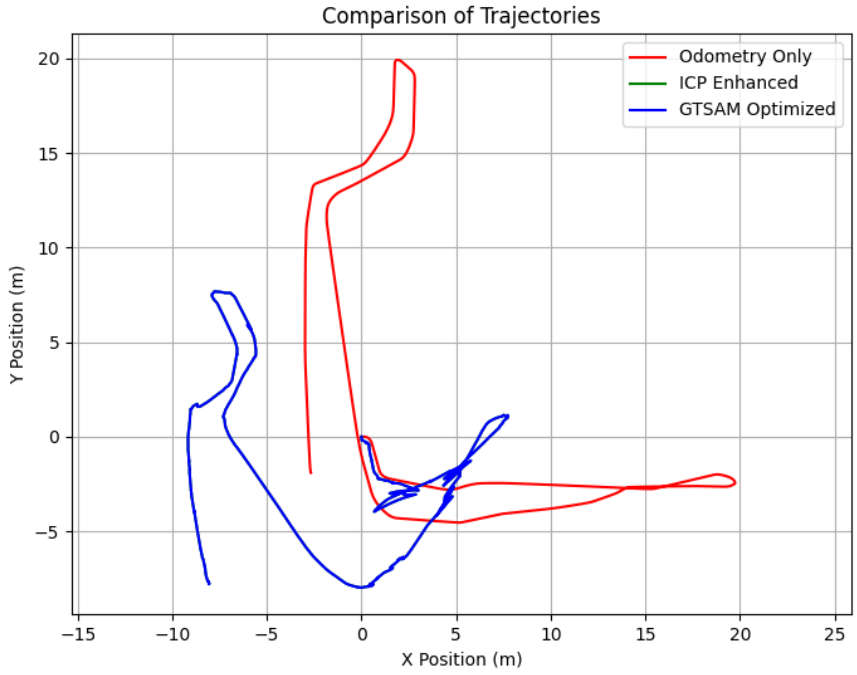

Fig. 19. Factor Graph Optimized (Proximity Based Loop Closure) Trajectory Dataset 21

Fig. 20. Factor Graph Optimized (Proximity Based Loop Closure) Grid and Texture Map for Dataset 21