LiDAR-Based SLAM

Varun Sankar Moparthi
University of California San Diego
vmoparthi@ucsd.edu

Nikolay Antasnov
University of California San Diego
natanasov@ucsd.edu

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a crucial capability for autonomous robots, enabling them to navigate and map unknown environments without external positioning systems. This project focuses on implementing a LiDAR-based SLAM system for a differential-drive robot equipped with wheel encoders, an Inertial Measurement Unit (IMU), a 2D LiDAR scanner, and an RGBD camera. The methodology begins with odometry estimation using encoder and IMU data, employing the differential-drive motion model. To refine localization accuracy, LiDAR scan matching is performed using the Iterative Closest Point (ICP) algorithm, which corrects incremental pose errors. The SLAM framework then constructs a 2D occupancy grid map from LiDAR data, while RGBD images are processed to generate a texture-mapped floor representation. Further improvements in trajectory estimation are achieved through pose graph optimization with loop closure constraints, implemented via the GTSAM library. Results demonstrated the effectiveness of scan matching and enhancing map accuracy, while pose graph optimization further refines localization consistency. This work highlights the integration of multiple sensor modalities for robust SLAM, contributing to reliable autonomous navigation in complex environments.**

I. Introduction

Autonomous robots must navigate and understand their surroundings without prior knowledge of the environment. This requires solving the fundamental problem of Simultaneous Localization and Mapping (SLAM), where a robot builds a map of an unknown environment while simultaneously estimating its own position. SLAM is a crucial capability for autonomous navigation in various applications, including self-driving cars, warehouse automation, and search-and-rescue operations. Among different SLAM approaches, LiDAR-based SLAM has emerged as a highly effective method due to its ability to provide precise range measurements and detect obstacles with high accuracy. However, real-world challenges such as sensor noise, odometry drift, and inconsistent loop closure detection make robust SLAM implementations difficult.

In this project, we implement a LiDAR-based SLAM system for a differential-drive robot equipped with multiple onboard sensors, including wheel encoders, an Inertial Measurement Unit (IMU), a 2D LiDAR scanner, and an RGBD camera. The SLAM pipeline begins with odometry estimation using encoder and IMU data. The differential-drive motion model provides an initial estimate of the robot's movement, but due to wheel slippage, drift, and cumulative errors, odometry alone is insufficient for accurate localization.

Identify applicable funding agency here. If none, delete this.

To improve localization, we integrate LiDAR-based scan matching using the Iterative Closest Point (ICP) algorithm. ICP aligns consecutive LiDAR scans by estimating the transformation that minimizes alignment errors, effectively refining the robot's estimated trajectory. This scan-matching approach helps correct odometry drift and ensures a more reliable localization over time. Using these refined pose estimates, we construct a 2D occupancy grid map, where free and occupied spaces in the environment are represented based on LiDAR observations.This mapping approach is fundamental for robotic path planning and navigation in unknown terrains.

In addition to basic occupancy mapping, we enhance the environment representation by utilizing an RGBD camera to create a texture-mapped floor representation. By processing depth images and associating them with corresponding color information, we project RGB data onto the occupancy grid. This not only improves the visual quality of the generated map but also enables applications where both geometric structure and visual appearance are essential.

To further refine trajectory estimation and ensure global consistency, we implement pose graph optimization with loop closure detection using the Georgia Tech Smoothing and Mapping (GTSAM) library. Loop closure occurs when the robot revisits a previously mapped area, allowing for corrections to past trajectory estimates. By incorporating loop closure constraints into a factor graph, we optimize the robot's full trajectory, minimizing accumulated errors and improving overall map accuracy.

Through a combination of odometry estimation, LiDAR scan matching, RGBD-based texture mapping, and pose graph optimization, our approach provides a robust SLAM solution for autonomous navigation. The experimental results demonstrate that integrating multiple sensing modalities significantly improves localization accuracy, reduces drift, and enhances the reliability of the generated maps. This work contributes to advancing SLAM methodologies, making them more effective for real-world applications in autonomous robotics.

II. Problem Formulation

A. Encoder and IMU odometry

The goal of this sub-project is to predict the motion of a robot using a differential-drive motion model, combining linear velocity v_t from encoders and yaw rate ω_t from an IMU. The robot starts with an identity pose, and we aim to verify the accuracy of our motion model predictions by plotting the robot's trajectory.

Let $x_t = [x_t, y_t, \theta_t]^T$ denote the robot's pose at time t, where x_t and y_t are the coordinates in the 2D plane, and θ_t is the orientation (yaw angle). The differential-drive motion model can be represented as:

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

where τ_t is the time difference between consecutive measurements.

To estimate the robot's trajectory, we aim to minimize the error between the predicted poses and the actual motion. However, since this problem focuses on predicting the trajectory using the given motion model rather than optimizing a cost function, we will directly apply the differential-drive model to compute the robot's pose at each time step.

B. Point-cloud registration via iterative closest point (ICP)

1) Iterative closest point (ICP): The aim of the sub-project is given a source point cloud $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^N$ representing the canonical object and a target point cloud $\mathcal{T} = \{\mathbf{t}_j\}_{j=1}^M$ obtained from different view, our goal is to find the optimal rotation θ and translation $\mathbf{p}$ that best aligns $\mathcal{S}$ with $\mathcal{T}$.

Let $\mathbf{R} \in SO(3)$ be the rotation matrix and $\mathbf{p} \in \mathbb{R}^3$ be the translation vector that transform $\mathcal{S}$ to align with $\mathcal{T}$. We assume rotation is only around the z-axis, so $\mathbf{R}$ can be parameterized by a single angle θ as:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation of a point $\mathbf{s}_i$ in $\mathcal{S}$ to the transformed point $\mathbf{s}'_i$ is given by:

$$\mathbf{s}'_i = \mathbf{R}(\theta)\mathbf{s}_i + \mathbf{p}$$

We seek to minimize the distance between the transformed source points $\mathbf{s}'_i$ and the target points $\mathbf{t}_j$, which can be formulated as an optimization problem:

$$\min_{\theta, \mathbf{p}} \sum_{i=1}^{N} \min_{j \in \Delta} \|\mathbf{s}'_i - \mathbf{t}_j\|_2^2$$

where Δ denotes the set of closest points in $\mathcal{T}$ to $\mathbf{s}'_i$.

2) Scan matching: The aim of sub-project is given two consecutive LiDAR scans at times t and $t+1$, represented by point clouds $\mathcal{S}_t = \{\mathbf{s}_{t,i}\}_{i=1}^{N_t}$ and $\mathcal{S}_{t+1} = \{\mathbf{s}_{t+1,j}\}_{j=1}^{N_{t+1}}$, our goal is to determine the optimal relative pose transformation that aligns two consecutive LiDAR scans between them using the Iterative Closest Point (ICP) algorithm.

Let tT_{t+1} represent the relative transformation matrix between frames t and $t+1$, which consists of a rotation matrix $tR_{t+1} \in SO(3)$ and a translation vector $tp_{t+1} \in \mathbb{R}^3$. The transformation of a point $\mathbf{s}_{t,i}$ in $\mathcal{S}_t$ to the transformed point $\mathbf{s}'_{t,i}$ is given by:

$$\mathbf{s}'_{t,i} = tR_{t+1}\mathbf{s}_{t,i} + tp_{t+1}$$

The ICP algorithm is used to refine this initial estimate to find the optimal transformation that minimizes the distance between corresponding points in the two scans:

$$\min_{tR_{t+1}, tp_{t+1}} \sum_{i=1}^{N_t} \min_{j \in \Delta} \|\mathbf{s}'_{t,i} - \mathbf{s}_{t+1,j}\|_2^2$$

where Δ denotes the set of closest points in $\mathcal{S}_{t+1}$ to $\mathbf{s}'_{t,i}$.

Once the relative pose change between two consecutive robot frames, represented by the transformation matrix tT_{t+1}, is known, it can be expressed as:

$$tT_{t+1} = \begin{bmatrix} tR_{t+1} & tp_{t+1} \\ \mathbf{0} & 1 \end{bmatrix}$$

The robot pose T_{t+1} at time $t+1$ can then be approximated as:

$$T_{t+1} = T_t \cdot tT_{t+1}$$

C. Occupancy and texture mapping

The aim of the sub-project is given a sequence of LiDAR scans $\{\mathcal{L}_t\}_{t=1}^T$ and RGBD images $\{\mathcal{I}_t\}_{t=1}^T$ collected by a robot traversing an environment, $\mathbf{T}_t \in SE(3)$ represent the robot's pose at time t, estimated using the ICP algorithm. For each LiDAR scan $\mathcal{L}_t = \{\mathbf{p}_i\}_{i=1}^{N_t}$, where $\mathbf{p}_i \in \mathbb{R}^3$, the goal is to estimate $\mathcal{M}_o$ and $\mathcal{M}_t$. Let $\mathcal{L}_t = \{\mathbf{p}_i\}_{i=1}^{N_t}$ and $\mathcal{I}_t = \{(\mathbf{c}_j, d_j)\}_{j=1}^{M_t}$. Estimate $\mathcal{M}_o$ and $\mathcal{M}_t$ given $\{\mathcal{L}_t, \mathcal{I}_t, \mathbf{T}_t\}_{t=1}^T$.

D. Pose graph optimization and loop closure

1) Factor graph in GTSAM: The aim of sub-project is given a sequence of robot poses $\{X_t\}_{t=1}^T$ and relative pose measurements $\{tT_{t+1}\}_{t=1}^{T-1}$ obtained from ICP scan matching, we aim to construct a factor graph $G = (V, E)$ where:

- $V = \{X_t\}_{t=1}^T$ represents the set of nodes, each corresponding to a robot pose at time t.
- $E = \{(X_t, X_{t+1}, tT_{t+1})\}_{t=1}^{T-1}$ represents the set of edges, each corresponding to a relative pose measurement between consecutive poses.

Here, tT_{t+1} denotes the transformation from pose X_t to pose X_{t+1}, as estimated by ICP scan matching. Represented as follows,

$$\begin{tikzcd} X_1 \arrow[r, "1T_2"] & X_2 \arrow[r, "2T_3"] & X_3 \end{tikzcd}$$

The factor graph G encodes the probabilistic relationship between robot poses and measurements, allowing for efficient pose graph optimization and SLAM (Simultaneous Localization and Mapping) using the GTSAM framework.

III. Technical Approach

A. Encoder and IMU odometry

To estimate the robot's trajectory, we employed a motion model for a differential drive robot under Euler discretization. The robot's pose at time $t+1$ is predicted using the current pose $X_t = [x_t, y_t, \theta_t]^T$, linear velocity v_t, and angular velocity ω_t.

The motion model equations are as follows:

$$\mathbf{X}_{t+1} = \mathbf{X}_t + \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix} \tau_t$$

where τ_t is the time interval between consecutive measurements.

The linear velocity of each wheel is computed using the formula:

$$v_{\text{wheel}} = r \cdot \frac{\Delta\text{counts}}{\tau_t}$$

where r is the wheel radius, Δcounts is the change in encoder counts over the time interval τ_t, and τ_t is the time interval between consecutive measurements.

For a differential drive robot, the linear velocities of the left and right wheels are calculated separately:

$$v_{\text{left}} = r \cdot \frac{\Delta\text{counts}_{\text{left}}}{\tau_t}$$

$$v_{\text{right}} = r \cdot \frac{\Delta\text{counts}_{\text{right}}}{\tau_t}$$

The robot's linear velocity v_t is then taken as the average of the left and right wheel velocities:

$$v_t = \frac{v_{\text{left}} + v_{\text{right}}}{2}$$

We processed the encoder data to compute the linear velocity v_t and matched the encoder timestamps with the nearest IMU timestamps to obtain the angular velocity ω_t. The time intervals τ_t were computed as the differences between consecutive encoder timestamps.

The robot's trajectory was estimated by iteratively applying the motion model to predict subsequent poses based on the computed velocities and time intervals.

B. Point-cloud registration via iterative closest point (ICP)

1) Iterative closest point (ICP): First, point cloud preprocessing is crucial for removing noise and outliers from the raw data. This is achieved through statistical outlier removal, where a point cloud $P = \{p_i\}_{i=1}^N$ is cleaned by removing points that are significantly far from their neighbors. The process can be formulated as follows:

$$P_{clean} = \{p_i \in P : \frac{1}{k} \sum_{j \in \mathcal{N}_k(i)} \|p_i - p_j\|_2 < \epsilon\}$$

where $\mathcal{N}_k(i)$ is the set of k nearest neighbors of point p_i, ϵ is the thersholding factor.

The `preprocess_point_cloud` function cleans the input point cloud by removing statistical outliers using Open3D's `remove_statistical_outlier` method with parameters `nb_neighbors=20` and `std_ratio=2.0`, which control the sensitivity of outlier detection.

Following preprocessing, efficient nearest neighbor search is essential for establishing correspondences between the source and target point clouds. Given source points $S = \{s_i\}_{i=1}^M$ and target points $T = \{t_j\}_{j=1}^N$, the nearest neighbor for each source point is found using:

$$NN(s_i) = \arg\min_{t_j \in T} \|s_i - t_j\|_2$$

The nearest neighbor search is accelerated using a `cKDTree`, which is constructed from the target points T without explicitly specifying any parameters in the code, relying on default settings for efficient querying.

Once correspondences are established, the Kabsch algorithm is used to compute the optimal rigid transformation between two sets of corresponding points. For corresponding point sets $S = \{s_i\}_{i=1}^K$ and $T = \{t_i\}_{i=1}^K$, the algorithm proceeds as follows:

1) Compute centroids:

$$\bar{s} = \frac{1}{K}\sum_{i=1}^K s_i, \quad \bar{t} = \frac{1}{K}\sum_{i=1}^K t_i$$

2) Center the point clouds:

$$S' = \{s_i - \bar{s}\}, \quad T' = \{t_i - \bar{t}\}$$

3) Compute SVD, rotation Matrix and translation as follows:

$$\begin{aligned} H &= S'^T T' \\ H &= U\Sigma V^T \\ R &= VU^T \\ t &= \bar{t} - R\bar{s} \end{aligned}$$

If $\det(R) < 0$, flip the last column of V.

The Iterative Closest Point (ICP) algorithm iteratively refines the transformation by initializing it with $T_0 = I_4$ and then iteratively transforming source points $S_k = T_k S$, finding nearest neighbors, sorting distances, trimming correspondences, and computing alignment using the Kabsch algorithm. The trimming step removes a fraction τ of the worst correspondences, enhancing robustness to outliers.

To improve convergence, ICP can be initialized with different yaw angles $\theta \in [\theta_{min}, \theta_{max}]$. This involves initializing rotation as:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and running ICP with $T_0 = \begin{bmatrix} R_\theta & 0 \\ 0 & 1 \end{bmatrix}$. The best result is selected as

$$T_{best} = \arg\min_\theta E_{final}(\theta)$$

$E_{final}(\theta)$ corresponds to the Mean Squared Error (MSE) after running the ICP algorithm with an initial transformation based on a specific yaw angle θ. It is calculated as:

$$E_{final}(\theta) = \frac{1}{|C|} \sum_{(s_i,t_i)\in C} \|s_i - t_i\|_2^2$$

The ICP algorithm is run with a yaw discretization of 144 steps over the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$, a maximum of 50 iterations, a tolerance of 10^{-6} for convergence, and a trim ratio of 0.05 for robustness.

2) Scan matching: It starts with converting LiDAR data into Cartesian coordinates.

Given LiDAR data with parameters such as minimum and maximum angles (θ_{min}, θ_{max}), angle increment ($\Delta\theta$), and range measurements (r_i), the conversion to Cartesian coordinates (x_i, y_i, z_i) can be expressed as:

$$\begin{aligned} x_i &= r_i \cos(\theta_i), \\ y_i &= r_i \sin(\theta_i), \\ z_i &= 0, \end{aligned}$$

where θ_i is the angle corresponding to each range measurement, and $z_i = 0$ since we are dealing with a 2D LiDAR scan.

Once the Cartesian coordinates are obtained, the scan matching process involves aligning successive scans using the Iterative Closest Point (ICP) algorithm. The ICP algorithm iteratively refines the transformation between two point clouds by finding the closest points and computing the optimal rigid transformation.

For each pair of successive scans, the initial pose is estimated based on the robot's trajectory. Given the robot's pose at time t as (x_t, y_t, θ_t) and at time $t+1$ as $(x_{t+1}, y_{t+1}, \theta_{t+1})$, the transformation matrices T_t and T_{t+1} can be represented as the composition of a rotation matrix and a translation vector, combined with a fixed offset. Mathematically, this can be expressed as:

$$T_t = R(\theta_t)@T_{offset} + \begin{bmatrix} x_t \\ y_t \\ 0 \\ 1 \end{bmatrix},$$

$$T_{t+1} = R(\theta_{t+1})@T_{offset} + \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ 0 \\ 1 \end{bmatrix},$$

where $R(\theta)$ is the rotation matrix for angle θ, and T_{offset} is the fixed offset matrix.

The initial pose for ICP is then computed as

$$\text{init\_pose} = T_t^{-1}@T_{t+1}$$

The ICP algorithm refines the initial pose by iteratively aligning the point clouds. This involves computing the relative pose R_{rel}, p_{rel} between two scans using the ICP algorithm:

$$R_{rel}, p_{rel} = \text{ICP}(S, T, \text{init\_pose})$$

The global pose of the robot is updated at each time step t as:

$$T_{global,t} = T_{global,t-1}@\begin{bmatrix} R_{rel} & p_{rel} \\ 0 & 1 \end{bmatrix}$$

The final trajectory of the robot is obtained by concatenating these global poses and stroed in trajectory.

$$\text{Trajectory} = \{T_{global,0}, T_{global,1}, \ldots, T_{global,n}\}$$

C. Occupancy and texture mapping

1) Occupancy Map: Given a set of robot poses $\{\mathbf{T}_t\}_{t=1}^T$, and LiDAR point cloud coordinates $\mathbf{xyz}$, our goal is to continuously update the occupancy map $\mathcal{M}_l$ based on the LiDAR scans.

Let $\mathbf{R}_t$ and $\mathbf{p}_t$ represent the rotation matrix and translation vector of the robot's pose at time t, respectively. The LiDAR points are transformed to the world frame using the transformation:

$$\mathbf{p}'(t) = \mathbf{R}_t \cdot \mathbf{p}(t) + \mathbf{p}_t$$

where $\mathbf{p}(t)$ is a point in the LiDAR frame at time t.

The transformed points are then projected onto the map grid using the map resolution $\mathbf{r}$ and minimum map coordinate $\mathbf{m}_{\min}$:

$$\mathbf{g}(t) = \frac{\mathbf{p}'(t) - \mathbf{m}_{\min}}{\mathbf{r}}$$

The occupancy map is updated continuously using the log-odds formulation:

$$\frac{d}{dt} l(\mathbf{g}(t)) = \begin{cases} \log \frac{p(\mathbf{g}(t)|\mathbf{z}(t))}{1-p(\mathbf{g}(t)|\mathbf{z}(t))} & \text{if } \mathbf{g}(t) \text{ is occupied} \\ -\log \frac{p(\mathbf{g}(t)|\mathbf{z}(t))}{1-p(\mathbf{g}(t)|\mathbf{z}(t))} & \text{if } \mathbf{g}(t) \text{ is free} \end{cases}$$

Here, $p(\mathbf{g}(t)|\mathbf{z}(t))$ is the inverse sensor model.

To account for the path traced by the LiDAR points, we use a continuous version of the Bresenham line algorithm. The map values are updated along this path to reflect the free space:

$$\frac{d}{dt} l(\mathbf{x}(t)) = -\log \frac{p(\mathbf{x}(t)|\mathbf{z}(t))}{1 - p(\mathbf{x}(t)|\mathbf{z}(t))}$$

for points $\mathbf{x}(t)$ along the path from the robot's position to each LiDAR point and Here $\log \frac{p(\mathbf{x}(t)|\mathbf{z}(t))}{1-p(\mathbf{x}(t)|\mathbf{z}(t))}$ is fixed as $log(4)$

Finally, the map values are continuously bounded to ensure they remain within a valid range:

$$l(\mathbf{g}(t)) = \max(-5\log(4), \min(l(\mathbf{g}(t)), 5\log(4)))$$

2) Texture Map: Given a sequence of robot poses $\{\mathbf{T}_t\}_{t\in[0,T]}$, RGB images $\{\mathcal{I}_t\}_{t\in[0,T]}$, and disparity images $\{\mathcal{D}_t\}_{t\in[0,T]}$, The depth $d(u,v,t)$ at each pixel (u,v) is calculated continuously from the disparity image $\mathcal{D}(u,v,t)$ using the formula:

$$d(u,v,t)=\frac{1.03}{-0.00304\cdot\mathcal{D}(u,v,t)+3.31}$$

This step converts the disparity values into depth measurements, which are essential for reconstructing the 3D environment.

For each pixel (u,v), we generate 3D points in the camera frame using the depth information:

$$\mathbf{p}_c(u,v,t)=d(u,v,t)\cdot\mathbf{K}^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}$$

Here, $\mathbf{K}$ is the camera intrinsic matrix that accounts for the camera's focal length and principal point.

To align the optical frame with the robot frame, we apply a rotation transformation using the matrix $\mathbf{R}_{o2r}^{-1}$. This step is crucial for maintaining consistency across different coordinate systems.

Given a point $\mathbf{p}_o$ in the optical frame, we transform it to the robot frame as follows:

$$\mathbf{p}_r=\mathbf{R}_{o2r}^{-1}\cdot\mathbf{p}_o$$

We transform the 3D points from the camera frame to the world frame using the robot's pose $\mathbf{T}_t$ and the camera-to-robot transformation $\mathbf{T}_{c2r}$:

$$\mathbf{p}_w(u,v,t)=\mathbf{T}_t\cdot\mathbf{T}_{c2r}\cdot\mathbf{R}_{o2r}^{-1}\cdot\mathbf{p}_c(u,v,t)$$

This transformation aligns the camera's coordinate system with the world frame, allowing us to project colors onto the correct locations in the environment.

Finally, we project the transformed 3D points onto a 2D texture map:

$$\mathbf{g}(u,v,t)=\frac{\mathbf{p}_w(u,v,t)-\mathbf{m}_{\text{min}}}{\text{res}}$$

Here, $\mathbf{m}_{\text{min}}$ is the minimum map coordinate, and res is the map resolution. The texture map $\mathcal{M}_t$ is updated with the RGB colors from the image:

$$\mathcal{M}_t(g_y,g_x)=\mathbf{c}(u,v,t)$$

This process continuously updates the texture map as the robot moves through the environment, creating a visually coherent and detailed representation.

D. Pose graph optimization and loop closure

1) Factor graph in GTSAM: Given a set of robot poses $\{\mathbf{T}_t\}_{t=1}^{T}$ and relative pose measurements $\{{}^tT_{t+1}\}_{t=1}^{T-1}$, our goal is to optimize the robot's trajectory using the GTSAM framework.

Let $\mathbf{R}_t$ and $\mathbf{p}_t$ represent the rotation matrix and translation vector of the robot's pose at time t, respectively. The relative pose measurements are represented as:

$${}^tT_{t+1}=\begin{bmatrix}\mathbf{R}_{t\to t+1} & \mathbf{p}_{t\to t+1}\\ \mathbf{0} & 1\end{bmatrix}$$

We construct a nonlinear factor graph G with nodes representing the robot poses and edges representing the relative pose measurements. The first pose is constrained by a prior factor:

$$G=\text{PriorFactorPose3}(0,\mathbf{T}_0,\mathbf{\Sigma})$$

where $\mathbf{\Sigma}$ is the noise model.

For each subsequent pose, we add a between factor to the graph:

$$G=\text{BetweenFactorPose3}(t-1,t,{}^tT_{t+1},\mathbf{\Sigma})$$

The graph is then optimized using the Levenberg-Marquardt algorithm to find the maximum likelihood estimate of the poses:

$$\hat{\mathbf{T}}_t=\arg\max_{\mathbf{T}_t}p(\mathbf{T}_t|\mathbf{z})$$

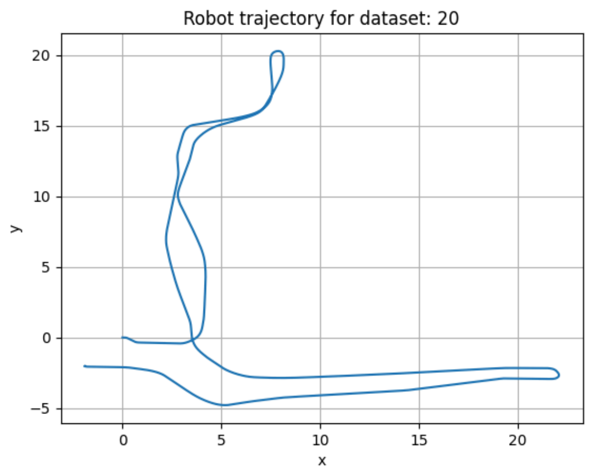
where $\mathbf{z}$ represents the measurements.

IV. RESULTS

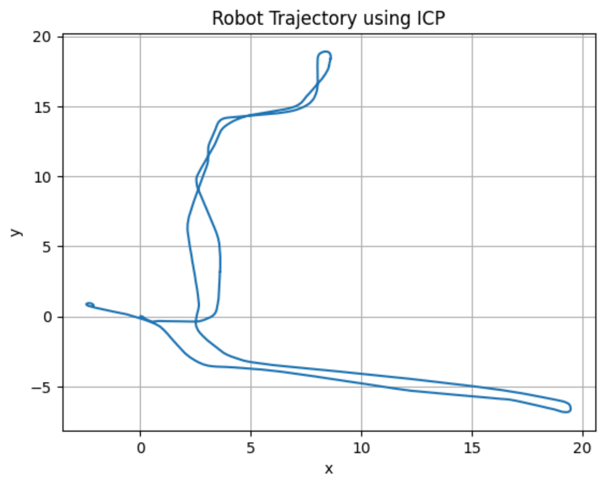
RESULTS

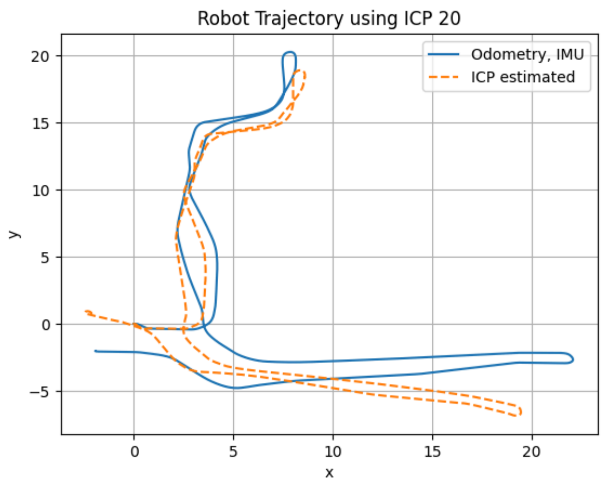
Dataset: 20

TRAJECTORY: (imu, odometry)

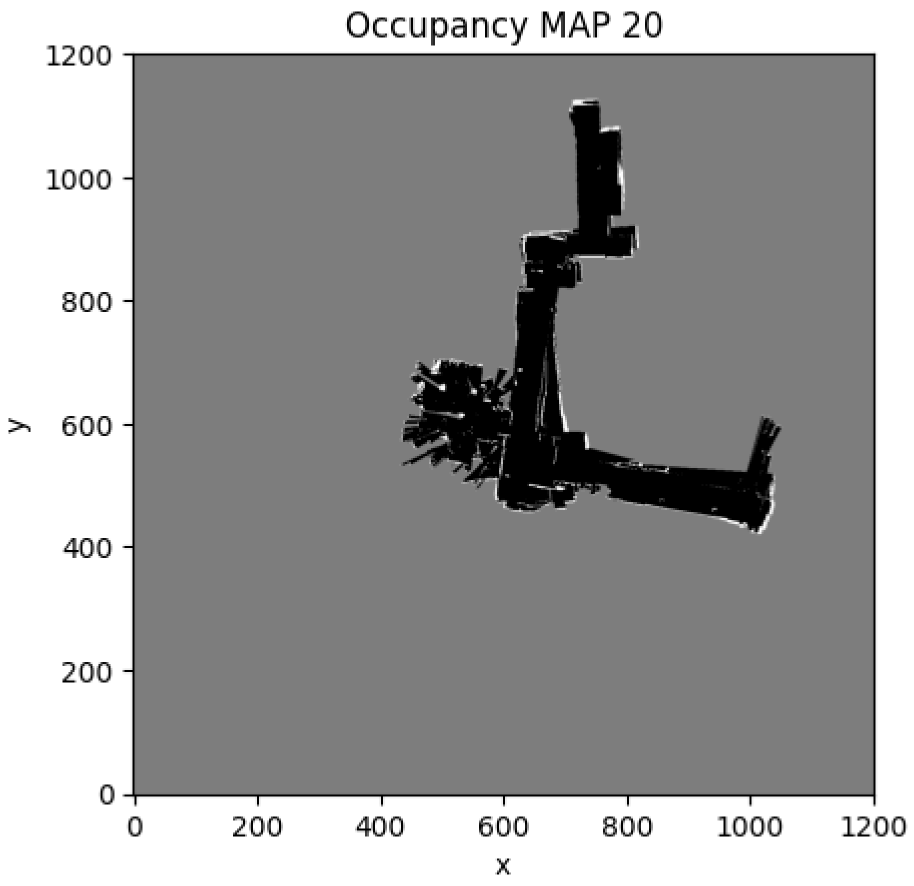


ROBOT TRAJECTORY: (ICP Optimised)

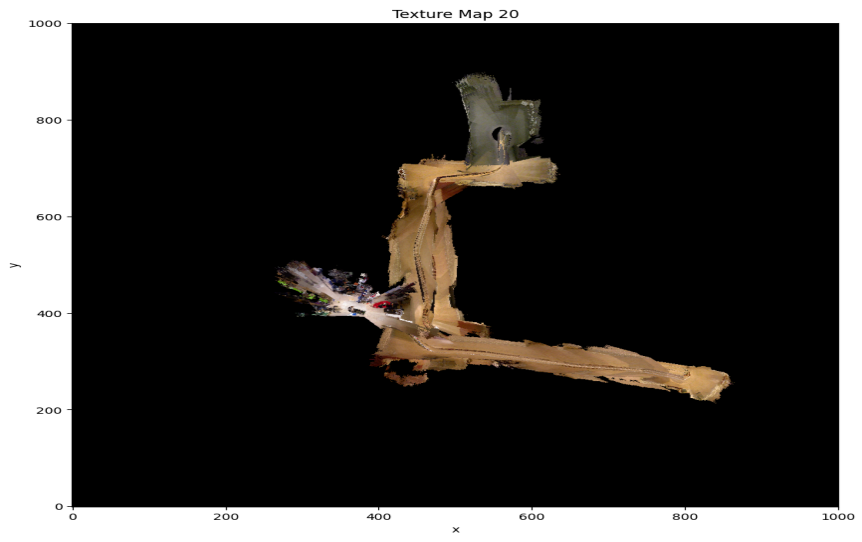




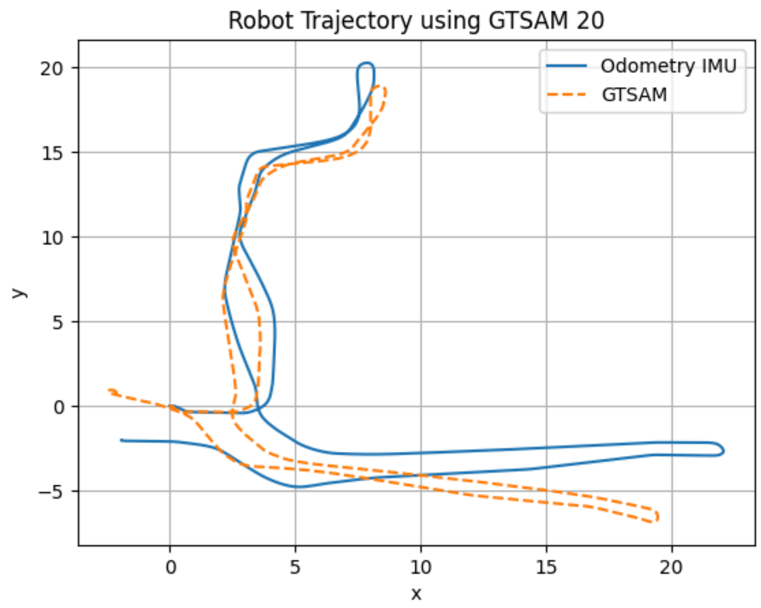
OCCUPANCY MAP:



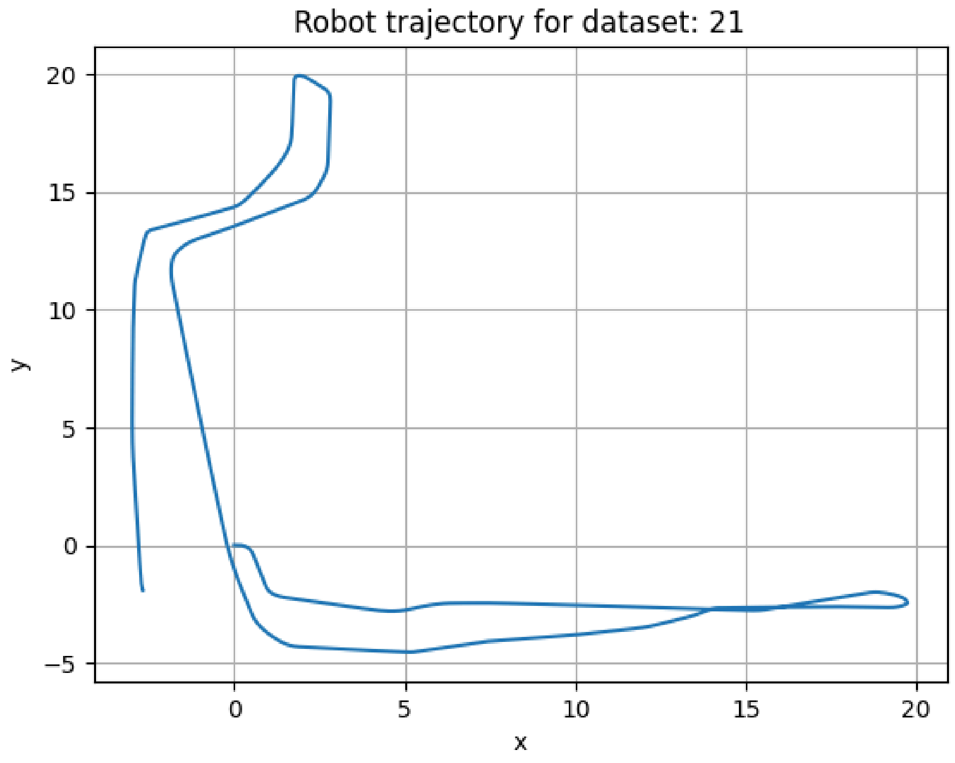
TEXTURE MAP:



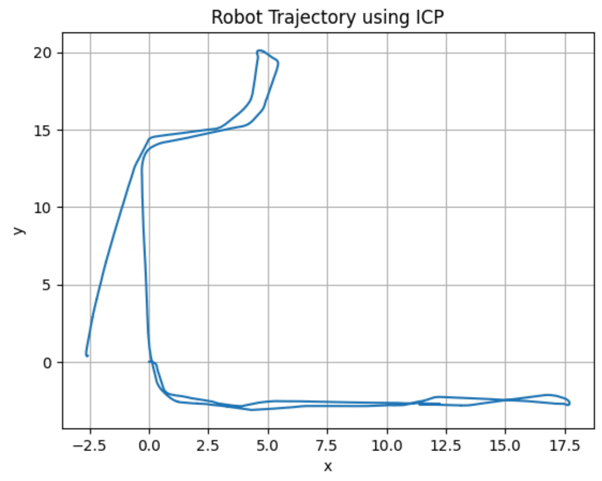
ROBOT TRAJECTORY: (GTSAM Optimised):

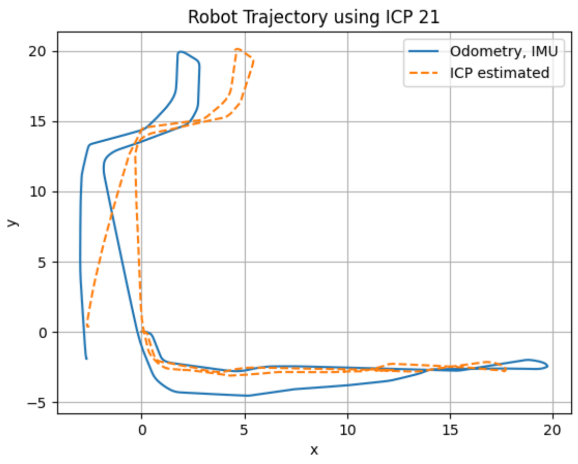


Dataset: 21
TRAJECTORY: (imu, odometry)

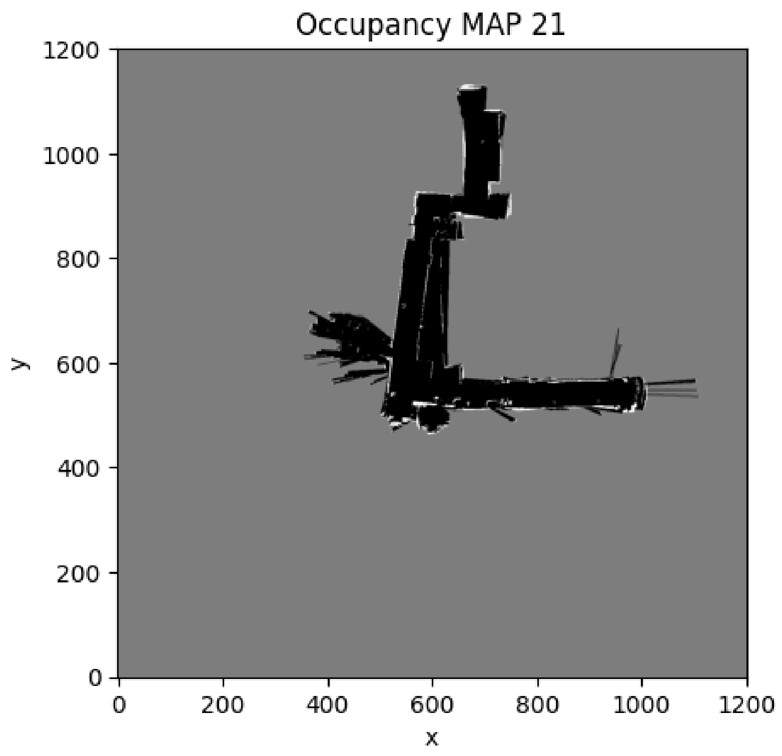


ROBOT TRAJECTORY: (ICP Optimised)





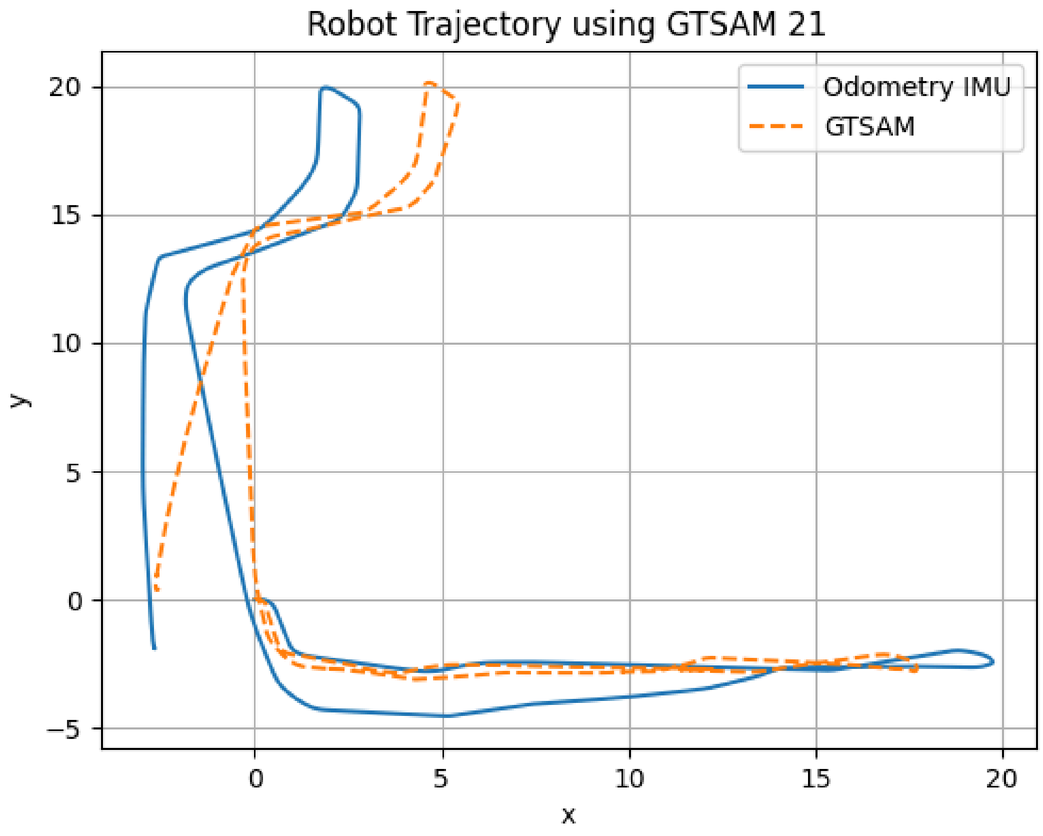
OCCUPANCY MAP:



TEXTURE MAP:

Please run this. I am running out of time to submit, you will get the results.

ROBOT TRAJECTORY: (GTSAM Optimised):


Robot Trajectory using GTSAM 21
Odometry IMU
GTSAM
y
x