LiDAR-Based SLAM

Manoj Kumar Reddy Manchala
Department of Mechanical and Aerospace Engineering
University of California San Diego
La Jolla, U.S.A.
mmanchala@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California San Diego
La Jolla, U.S.A.
mmanchala@ucsd.edu

***Abstract*—This report presents the development and implementation of a LiDAR-based Simultaneous Localization and Mapping (SLAM) system for a differential-drive robot. Utilizing odometry, IMU, 2-D LiDAR scans, and RGBD images. The methodology encompasses odometry estimation, Iterative Closest Point (ICP) for scan matching, occupancy grid and texture mapping, and trajectory optimization with GTSAM incorporating loop-closure constraints. Results demonstrate the system's effectiveness in robustly navigating and mapping unknown environments, showcasing potential applications in autonomous robotics.**

***Index Terms*—SLAM, Factor-Graph, LiDAR Scan Matching, Occupancy & Texture Mapping**

I. Introduction

The interest in LiDAR-based SLAM stems from its critical role in enabling autonomous robots to navigate and understand their environments effectively. This technology is foundational for various applications, including autonomous vehicles, robotic mapping, and navigation in complex, unstructured environments where GPS is unreliable or unavailable. Our approach to addressing this problem involves integrating multiple sensors, including odometry, IMU, 2-D LiDAR scans, and RGBD images, to achieve accurate localization and mapping. The method combines odometry estimation to track the robot's movement, scan matching via Iterative Closest Point (ICP) to align and update maps, and the creation of detailed environmental maps through occupancy grids and texture mapping. Furthermore, we employ GTSAM for pose graph optimization, enhancing the system's accuracy through loop-closure detection.

II. Problem Formulation

Consider a differential drive robot navigating an unknown environment whose state at time t is represented by x_t. Let ω_t and v_t be the measured angular velocity and estimated linear velocity obtained from the IMU and the encoder respectively.

A. Encoder model

A magnetic encoder consists of a rotating gear, a permanent magnet, and a sensing element. The sensor has two output channels with offset phase to determine the direction of rotation. A microcontroller counts the number of transitions adding or subtracting 1 to the counter. The distance traveled by the wheel, corresponding to one tick on the encoder is:

$$\text{meters per tick} = \frac{\pi \times (\text{wheel diameter})}{\text{ticks per revolution}}$$

The distance traveled during time τ for a given encoder count z, wheel diameter d, & n ticks on the sensor per revolution is:

$$\tau v \approx \frac{\pi d z}{n}$$

and can be used to measure the linear velocity for a differential-drive model.

B. Scan Matching

Given two sets $\{m_i\}$ and $\{z_j\}$ of points containing landmark positions and relative measurements respectively with unknown data association Δ, find the transformation p, R between sets

- ICP algorithm: iterates between finding associations Δ based on closest points and applying the Kabsch algorithm to determine p, R
- Initialize with p_0, R_0 (sensitive to initial guess) and iterate
  1) Given p_k, R_k, find correspondences $(i, j) \in \Delta$ based on closest points:
  $$i \leftrightarrow \arg\min_j \|m_i - (R_k z_j + p_k)\|^2$$
  2) Given correspondences $(i, j) \in \Delta$, find p_{k+1}, R_{k+1} via Kabsch algorithm

The above ICP algorithm is used to find the relative transformations between consecutive LiDAR scans. ICP is initialised using the odometry estimated from the IMU and encoder. Once the relative pose change between two consecutive robot frames $T_t \to T_{t+1}$ is known, the robot pose T_{t+1} at time $t+1$ can be approximated as $T_{t+1} = T_t \cdot T_{t \to t+1}$.

C. Occupancy Grid & Texture Mapping

Based on the trajectory obtained from Scan matching, occupancy grid and texture maps are created for visualization as follows:

LiDAR points are transformed from the LiDAR frame to the world frame. Bresenham2D function is used to obtain the occupied cells and free cells that correspond to the LiDAR scan, whose log odds are increased if occupied and decreased if free in the occupancy-grid map. The process is repeated for all the LiDAR scans to create an occupancy grid map.

The depth of each RGB pixel is obtained as described in the Kinect description. The colored points are then projected

from the depth camera frame to the world frame. The plane that corresponds to the occupancy grid in the transformed data is found via thresholding on the height and the map cells are colored with the RGB values according to the projected points.

D. Optimization

The obtained icp based trajectory is further optimized through pose graph optimization based on loop closure to improve accuracy. The optimization is performed based on the relative poses between scans $\bar{T}_{ij}$ and the poses T_i as follows:

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

A loop closure factor is added observing previously seen areas between non successive poses.

Optimization: $\phi_{ij}(e) = e^\top W_{ij}^\top W_{ij} e = \|W_{ij} e\|_2^2$:

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

The optimization is performed over all the robot poses.

III. Technical Approach

A. Encoder & IMU odometry

The robot motion is predicted via the discrete time differential-drive motion model using linear velocity v_t obtained from the encoders and yaw rate ω_t obtained from the IMU as follows:

Euler discretization over a time interval of length τ_t:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = f_d(\mathbf{x}_t, \mathbf{u}_t) := \mathbf{x}_t + \tau_t \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}$$

where $\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$ represents the state at time t, including position (x_t, y_t) and orientation θ_t, v_t is the linear velocity, ω_t is the angular velocity

Starting with x_0 = [0, 0, 0], a simple integration of the state x_t is performed, computing $x_{t+1} = f_d(x_t, u_t)$ to get the motion model trajectory.

B. Scan Matching

1) Kabsch Algorithm: Finding the transformation $p \in \mathbb{R}^d$, $R \in SO(d)$ between sets $\{m_i\}$ and $\{z_i\}$ of associated points:

$$\min_{R\in SO(d), p\in\mathbb{R}^d} f(R,p) := \sum_i w_i \|(Rz_i + p) - m_i\|_2^2$$

The optimal translation is obtained by setting $\nabla_p f(R,p)$ to zero:

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

Let the point cloud centroids be:

$$\bar{m} := \frac{\sum_i w_i m_i}{\sum_i w_i}, \quad \bar{z} := \frac{\sum_i w_i z_i}{\sum_i w_i}$$

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

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

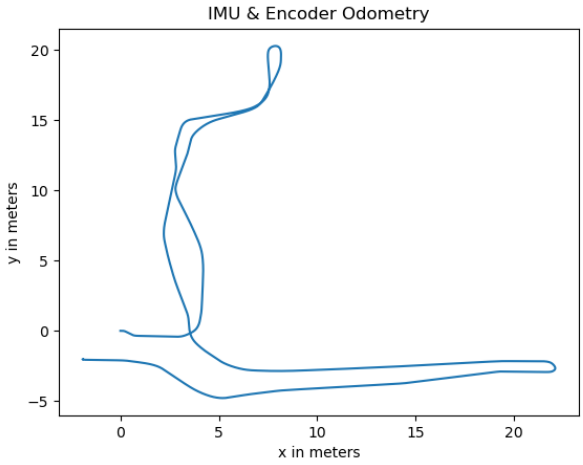

Fig. 1. 2D trajectory based on the IMU odometry - Dataset 20

Substituting $p = \bar{m} - R\bar{z}$ in $f(R,p)$ reduces the optimization to a Wahba's problem: to determine the rotation R that aligns two associated centered point clouds $\{\delta m_i\}$ and $\{\delta z_i\}$, we need to solve a linear optimization problem in $SO(d)$:

$$\max_{R\in SO(d)} \text{tr}(Q^\top R)$$

where $Q := \sum_i w_i \delta m_i \delta z_i^\top$

Let $Q = U\Sigma V^\top$ be the singular value decomposition of Q. Then the rotation matrix R is given by

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

2) Iterative closest point algorithm:

- ICP algorithm: iterates between finding associations Δ based on closest points and applying the Kabsch algorithm to determine p, R
- Initialize with p_0, R_0 (sensitive to initial guess) and iterate
  1) Given p_k, R_k, find correspondences $(i,j) \in \Delta$ based on closest points:
  $$i \leftrightarrow \arg\min_j \|m_i - (R_k z_j + p_k)\|^2$$
  2) Given correspondences $(i,j) \in \Delta$, find p_{k+1}, R_{k+1} via Kabsch algorithm

To find the data association δ, KDtree is being used. A target KDtree is created and searched for closest points from the source.

The icp algorithm is sensitive to initial guess and hence is initialized by the poses obtained from the Scan Matching. The algorithm is terminated after the maximum number of iterations is reached or if the error between the transformed source and the target is less than the threshold.

Algorithm 1 Iterative Closest Point (ICP) Algorithm

Require: source, target, transformation, dimension, iterations, tolerance

```
prev_error ← 0
target_tree ← KDTree(target)
for i ← 1 to iterations do
    homogeneous_source ← np.hstack((source, np.ones((len(source), 1))))
    transformed_source ← homogeneous_source @ transformation.T
    transformed_source ← transformed_source[:, :-1]
    distances, indices ← target_tree.query(transformed_source)
    target_new ← target[indices,:]
    new_transformation ← estimate_transformation(source, target_new, dimension)
    error ← np.mean(distances ** 2)
    if |error - prev_error| < tolerance then
        break
    end if
    prev_error ← error
    transformation ← new_transformation
end for
return transformation, error
```

C. Occupancy Grid & Texture Mapping

LiDAR ranges are filtered so as to remove the points that are too close to the robot or too far away from the robot. An empty map M is created and for each iteration

1) Bresenham2D function: The Bresenham2D function is an algorithm that determines which points in a grid should be marked to form a straight line between two points. It operates in integer space and is particularly efficient because it uses only addition, subtraction, and bit shifting. In mathematical terms:

Bresenham's 2D line algorithm calculates the grid cells a line path intersects. Given two points (x_0, y_0) and (x_1, y_1), the algorithm determines the points (x, y) for the line approximation on a discrete grid.

2) Occupancy Grid: For each point in time (indexed by i), the algorithm:

1) Computes the angles and ranges from the LiDAR data, filtering based on predefined thresholds.
2) Transforms the polar coordinates (angle, range) into Cartesian coordinates (x_{coords}, y_{coords}), and then into the point cloud pc.
3) Adjusts the point cloud by the robot's pose to map coordinates, represented by $T_{\text{list}}[i]$.
4) Uses the Bresenham2D algorithm to trace a line from the robot's position to the observed point, marking the cells along this line as free.
5) The last cell, where an object is detected, is marked as occupied.
6) Applies a sigmoid function to convert log-odds values in the occupancy grid to probabilities.

Mathematically, the transformation from LiDAR readings to map coordinates can be represented as:

$$\text{transformed\_points} = \begin{bmatrix} pc & \mathbf{1} \end{bmatrix} \times T_{\text{list}}[i]^{\top}$$

where $T_{\text{list}}[i]$ is the transformation matrix at time i, and $\begin{bmatrix} pc & \mathbf{1} \end{bmatrix}$ is the homogeneous representation of the point cloud. The occupancy grid is updated based on the transformed points and the Bresenham2D line tracing algorithm. The log-odds values are converted to probabilities using the sigmoid function:

$$\text{occupancy\_grid} = \frac{1}{1 + e^{-\text{occupancy\_grid}}}$$

Finally, the grid map is visualized, with the trajectory of the robot superimposed.

3) Texture Mapping: RGB, encoder, and disparity timestamps to synchronize data.

The algorithm:

1) Matches timestamps to synchronize RGB and depth data.
2) Initializes a texture grid with dimensions based on map resolution and size.
3) Converts disparity to depth with the relation $z = 1.03/(-0.00304 \times \text{disparity} + 3.31)$.
4) Transforms the 3D points into the robot's body frame using T_{camera}.
5) Maps the points onto the world frame using pose indices and T_{list}.
6) Applies a height filter to focus the texture mapping on the floor.
7) Assigns RGB values to the corresponding points on the texture grid.

D. Pose Graph Optimization

For each pair of scans (indexed by i and j), the initial relative pose guess T_{01} is refined by the ICP algorithm to find the best alignment between the source and target point clouds. A factor graph is initialized, and nodes are populated with initial pose estimates. Each edge, representing odometry or loop closure, is weighted by the corresponding noise model. For consecutive poses, odometry factors are added to the graph based on the transformations obtained from ICP, indicating the relative motion.

Every fixed interval, loop closure factors are added, significantly improving the map's consistency by correcting accumulated drift errors. The Levenberg-Marquardt algorithm optimizes the graph, minimizing the discrepancy between estimated poses and measurements, resulting in an optimized trajectory. The optimized trajectory is then plotted alongside the initial ICP-based and state-based trajectories for comparison.

In mathematical terms, the ICP algorithm can be expressed as: ICP algorithm:

$$T_{01}^{\text{ICP}}, \_ = \text{ICP}(\text{source\_pc}, \text{target\_pc}, T_{01}, 3, 50)$$

Optimization problem:

$$\min_{T} \sum \|\text{odometry\_noise} \cdot (T_{\text{prev}}^{-1} \cdot T_{\text{current}} - T_{\text{icp}})\|^2 + \|\text{loop\_closure\_noise} \cdot (T_{\text{loop\_start}}^{-1} \cdot T_{\text{loop\_end}} - T_{\text{loop\_closure}})\|^2$$

IV. Results

A. Dataset - 20

The algorithm has been tested on two datasets, the results for the dataset - 20 is discussed in this section.

1) Trajectory: From Fig.2 and Fig.3, we can see that odometry has accumulated error in the distance travelled and ICP seems to have an accumulated error in the orientation of the robot, which is intuitive.

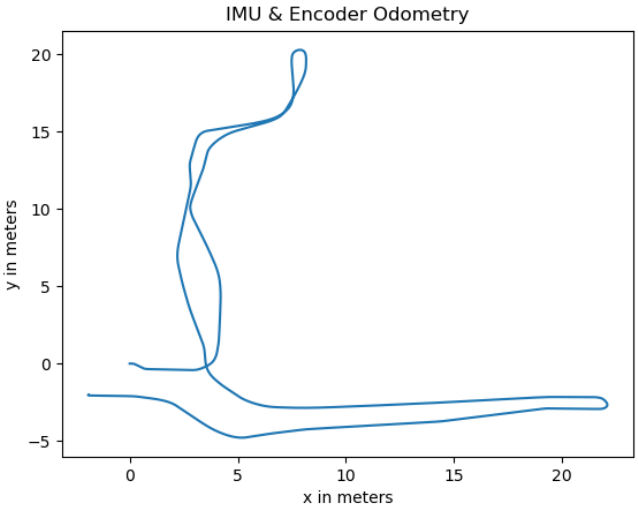

Fig. 2. IMU and Encoder based Odometry

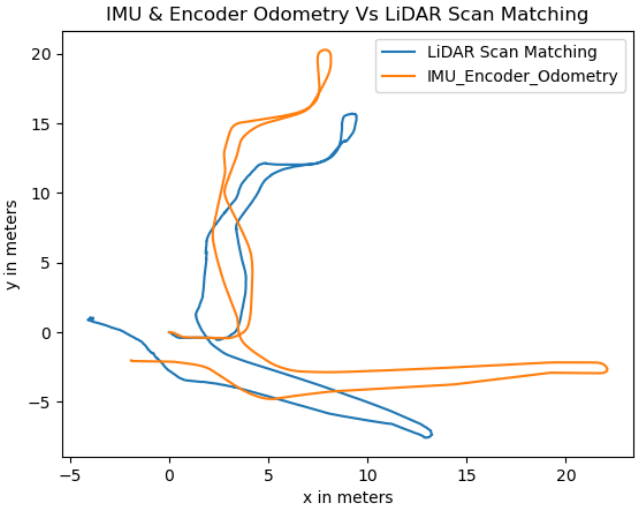

Fig. 3. IMU Encoder Odometry Vs LiDAR based Scan Matching

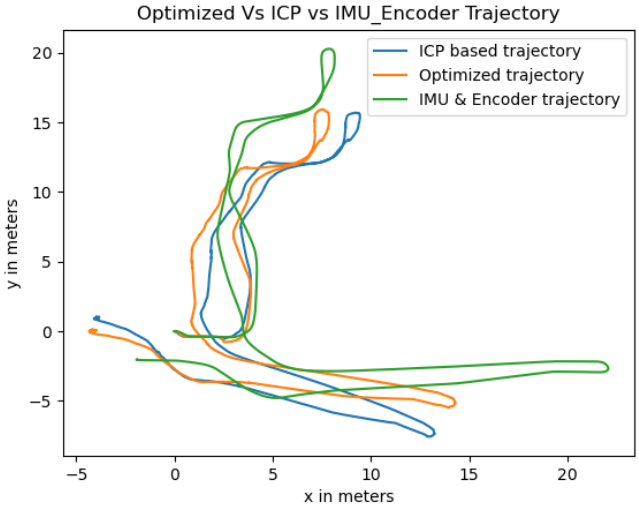

Fig. 4. IMU Encoder Vs Scan Matching Vs Optimized trajectory

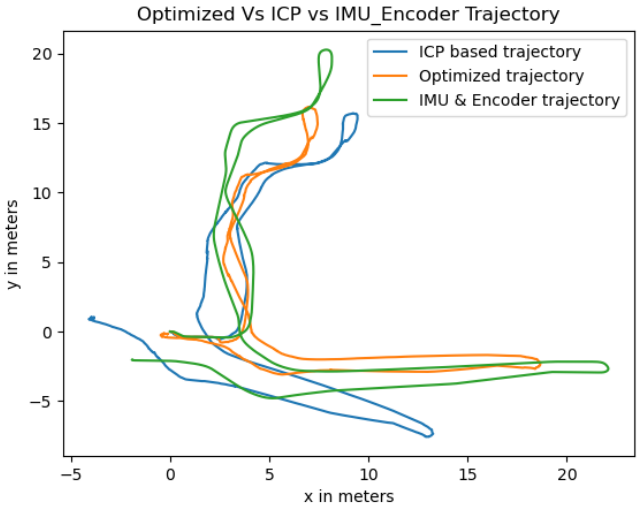

Fig. 5. Optimized trajectory - additional loop closure between first and last

From the figures 3 and 4 we can see that pose graph optimization is helping in making the trajectory more accurate by reducing the accumulation error in yaw.

From the figures 4,5 we can see that adding an additional loop closure between the first and the last state can result in even better optimization.

2) Occupancy Grid & Texture Mapping: From figures 6 & 7, we can see the corrected occupancy grip map based on the optimized trajectory

From the Figures 8 & 9 we can see the corrected texture maps based on the optimized trajectory.

We can see that the optimization indeed corrected the errors in the system, both accumulated error in the distance and the yaw angle which resulted in better grip and texture maps.

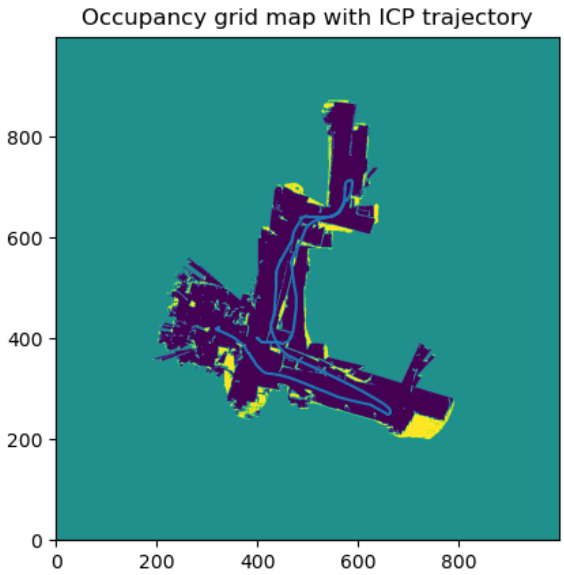

Fig. 6. LiDAR Scan Matching based Occupancy Grid

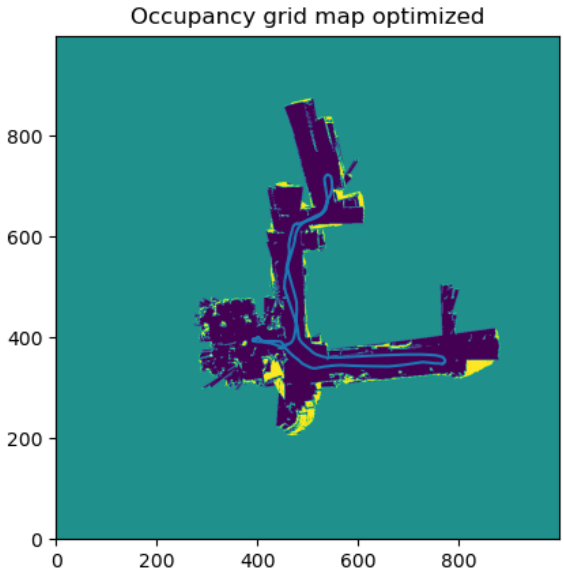

Fig. 7. Occupancy Grid based on the Optimized poses

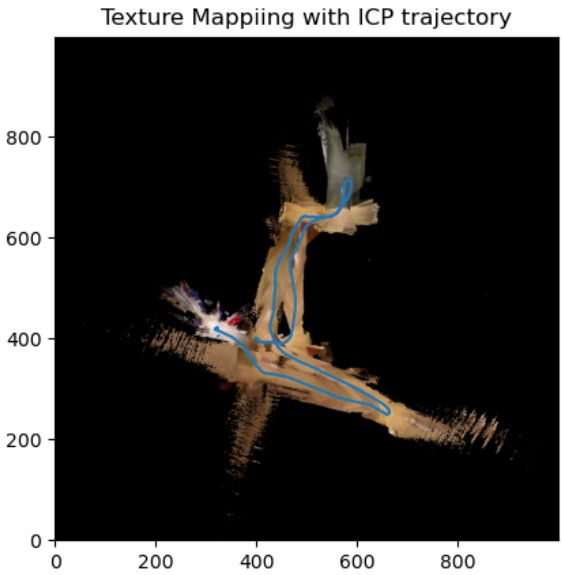

Fig. 8. LiDAR Scan Matching based Texture Mapping

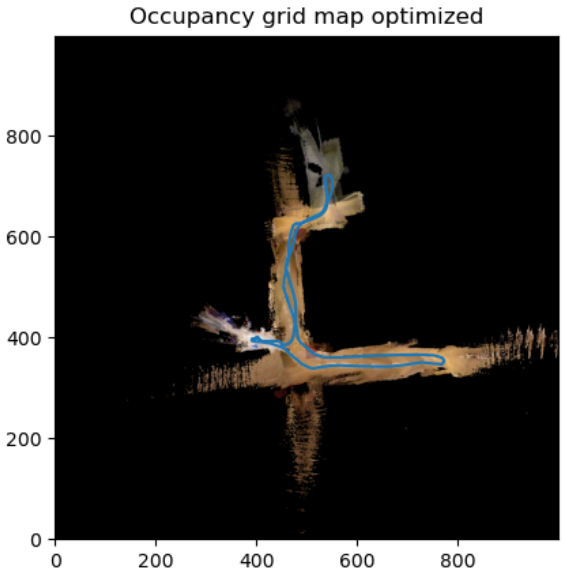

Fig. 9. Optimization based Texture Mapping

B. Dataset - 21

The algorithm has been tested on two datasets, the results for the dataset - 21 is discussed in this section.

For the dataset 21, the ICP algorithm seems to be converging to local minima, hence we have introduced one more constraint as to the relative transformation from ICP should not be too far from the odometry based relative transformation.

1) Trajectory: From Fig.10 and Fig.11, we can see that odometry has accumulated error in the distance travelled.

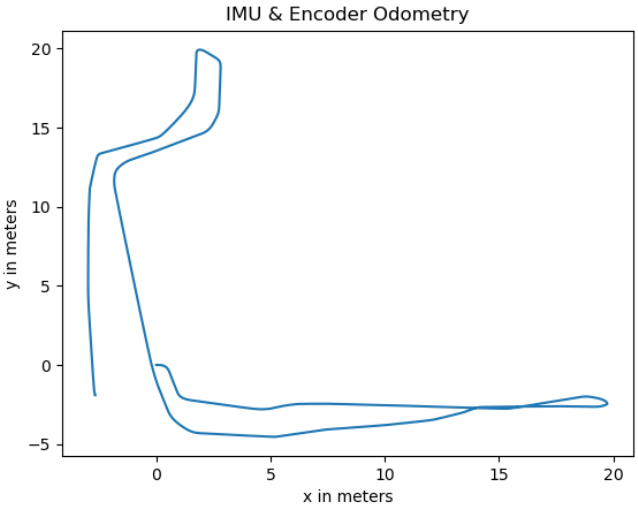

Fig. 10. IMU and Encoder based Odometry

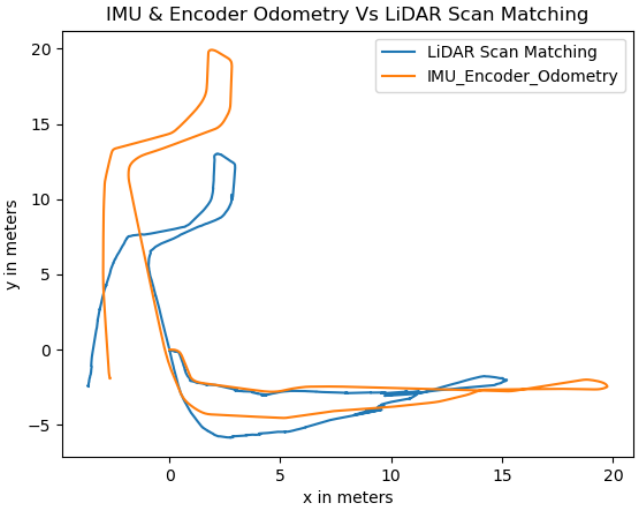

Fig. 11. IMU Encoder Odometry Vs LiDAR based Scan Matching

From the figures 11 and 12 we can see that pose graph optimization is helping in making the trajectory more accurate by reducing the accumulation error in yaw.

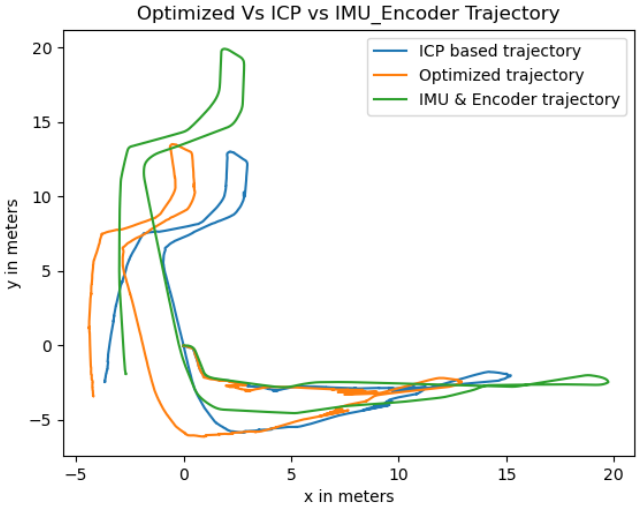

Fig. 12. IMU Encoder Vs Scan Matching Vs Optimized trajectory

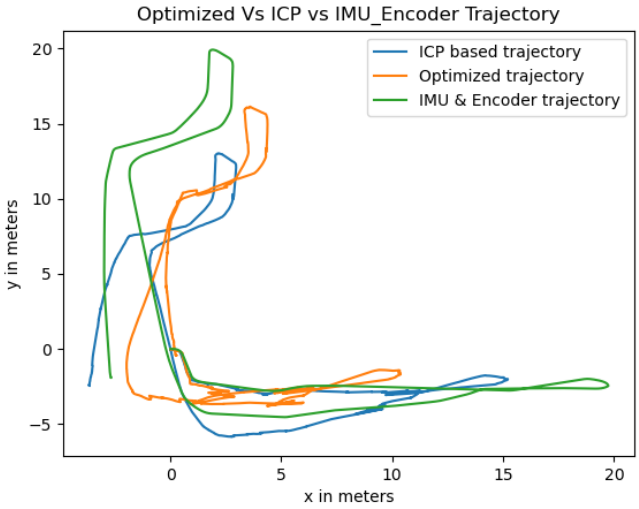

Fig. 13. Optimized trajectory - additional loop closure between first and last

From the figures 12, 13 we can see that adding an additional loop closure between the first and the last state can result in even better optimization.

2) Occupancy Grid & Texture Mapping: From figures 14 & 15, we can see the corrected occupancy grip map based on the optimized trajectory. We can see that optimizatoin here did not essentially create better results, as there is slight noise at the end of the trajectory.

From the Figures 16 & 17 we can see the corrected texture maps based on the optimized trajectory.

We can see that the optimization indeed corrected the errors in the system, both accumulated error in the distance and the yaw angle which resulted in better grip and texture maps.

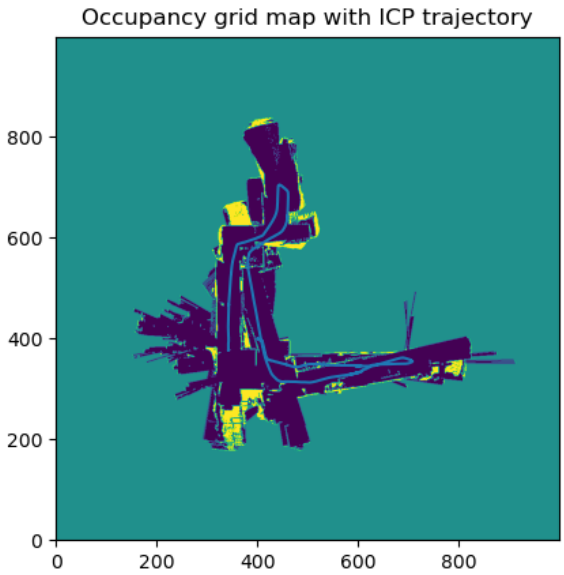

Fig. 14. LiDAR Scan Matching based Occupancy Grid

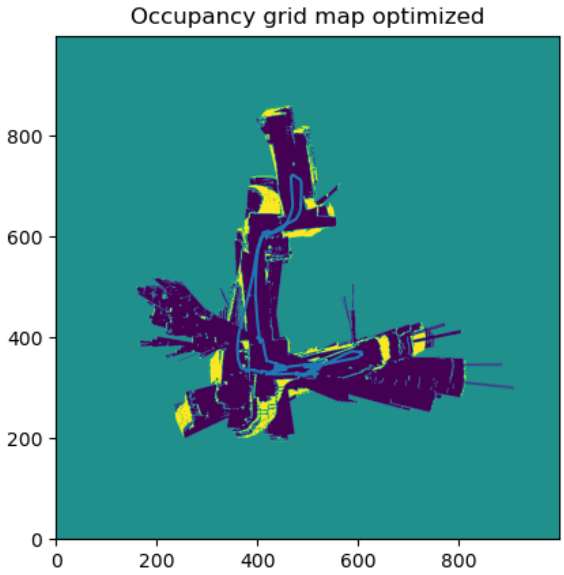

Fig. 15. Occupancy Grid based on the Optimized poses

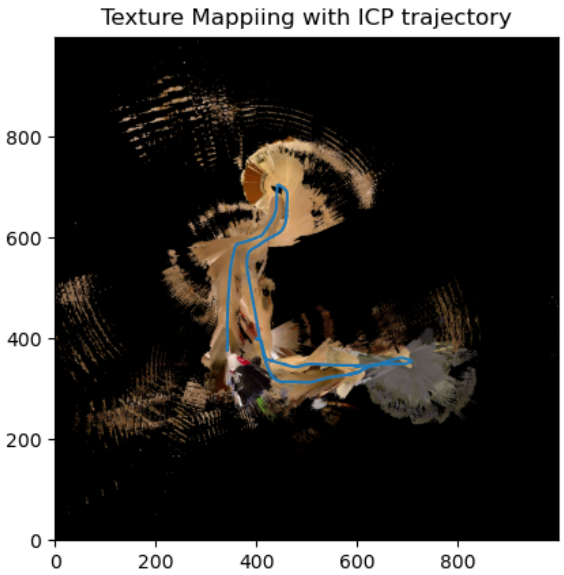

Fig. 16. LiDAR Scan Matching based Texture Mapping

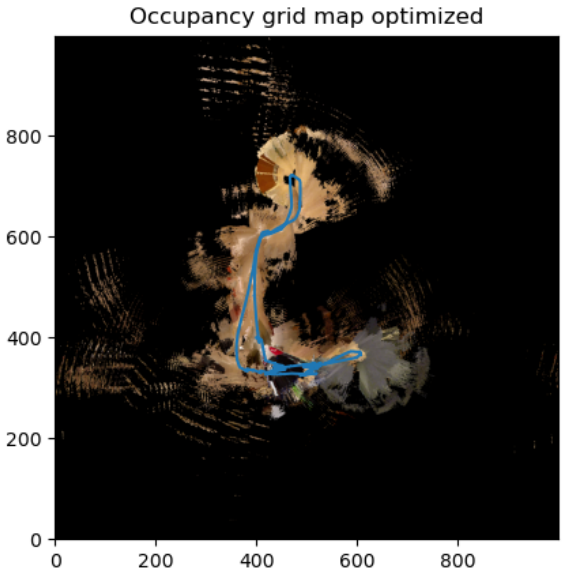

Fig. 17. Optimization based Texture Mapping

REFERENCES

[1] piazza
[2] Lecture 6: Localization and Odometry from Point Features
[3] Lecture 5 - Factor Graph SLAM