LiDAR-Based SLAM

Bharath Raam Radhakrishnan
University of California, San Diego
PID: A69030742

***Abstract*—The objective of this project is to implement simultaneous localization and mapping (SLAM) using encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements from a differential-drive robot. Using the SLAM estimates for the poses of the robot, we try to visualize the trajectory of the robot, and also generate a occupancy grid. The occupancy grid is then textured using the RBGD data from the Kinect camera. We also implement point-cloud registration via iterative closest point, which helps us obtain the relative transformation between 2 given LiDAR scans. SLAM is implemented using a factor graph, with loop closures being established based at fixed interval and based on proximity.**

***Index Terms*—LiDAR SLAM, GTSAM, Occupancy Map, Point Cloud registration, Iterative closest point, Factor Graph**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a computational technique that allows a robot, drone, or autonomous vehicle to build a map of an unknown environment while simultaneously determining its own position within that map. SLAM is essential in robotics, autonomous navigation, and augmented reality applications, enabling systems to move intelligently in dynamic and unstructured environments without relying on pre-existing maps. SLAM integrates data from various sensors such as LiDAR, cameras, and IMUs (Inertial Measurement Units), allowing precise localization even in challenging conditions, which has made it one of the widely adopted technique to solve navigation in unknown environments.

In this project, we aim to construct map of the environment the robot traverses using SLAM. The differential drive robot has wheel encoders, IMU, Hokuyo LiDAR and a Kinect camera, which keeps on collecting data as the robot moves and is being used to solve the SLAM problem. We aim to use Iterative closest point (ICP) algorithm to perform LiDAR scan matching, to improve the initial odometry estimate generated using the motion model. We then formulate the SLAM problem using a factor graphs, to model the constraints between robot poses and sensor measurements enabling efficient estimation of the most probable trajectory and map. We then optimize the factor graph using nonlinear optimization techniques, like Levenberg-Marquardt (LM) optimization algorithm to help reduce drift and improve accuracy in SLAM.

II. Problem Formulation

A. Motion model Odometry

The differential drive robot, is set to move on the XY plane, and has the ability to turn about its Z-axis. The setup provides the linear and angular velocities, which is computed from the encoder values and the IMU respectively. The motion model of a differential drive robot is given as,

$$\begin{bmatrix} R(\theta_{t+1}) & p_{t+1} \\ 0^\top & 1 \end{bmatrix} = \begin{bmatrix} R(\theta_t) & p_t \\ 0^\top & 1 \end{bmatrix} \exp\left(\tau_t \hat{\boldsymbol{\xi}}\right) \tag{1}$$

where,

$$\hat{\boldsymbol{\xi}} = \begin{bmatrix} 0 & -\omega_z & 0 & v_t \\ \omega_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

ω_z, v_t, $R(\theta_t)$, $p_t$is the yaw rate, linear velocity, Rotation matrix of the robot pose, and translation vector of the robot pose at time t respectively. τ_t is the time difference between the timestamp $t+1$ and t.

The trajectory (x_t, y_t, θ_t) of the robot from the pose matrix $T_t \in SE(3)$ at time t, can be extracted as,

$$\begin{aligned} x_t &= T_{1,4} \\ y_t &= T_{2,4} \\ \theta_t &= \arctan 2(T_{2,1}, T_{1,1}) \end{aligned} \tag{3}$$

,where $T_{i,j}$ is the element in i^{th} row and j^{th} column of the pose matrix T, (x_t, y_t) is the position of the robot and θ_t is the orientation of the robot at time t. The intial pose of the robot T_0 is taken to be the identity (I_{4x4})

B. Point cloud Registration

The Point cloud registration problem is in general defined as, Given two sets $\{m_i\}$, the source point cloud, and $\{z_j\}$ of points in $\mathbb{R}^d$,the target point cloud, find the transformation $p \in \mathbb{R}^d$, $R \in SO(d)$, and data association Δ that align them:

$$\min_{R \in SO(d),\, p \in \mathbb{R}^d,\, \Delta} f(R, p, \Delta) := \sum_{(i,j) \in \Delta} w_{ij} \left\| (R z_j + p) - m_i \right\|_2^2 \tag{4}$$

where, $SO(d) := \{R \in \mathbb{R}^{d \times d} \mid R^T R = I, \det(R) = 1\}$. This problem can be solved using the ICP and Kabsch algorithm, which is discussed in the sections below.

C. SLAM

The objective of SLAM is to optimize the robot trajectory over a given number of timestamps (T). The SLAM problem is postulated as, given the initial robot state $\mathbf{x}_0$, sensor measurements $\mathbf{z}_{1:T}$ with observation model h, and control inputs $\mathbf{u}_{0:T-1}$ with motion model f, estimate the robot state trajectory $\mathbf{x}_{1:T}$ and build a map m:

$$\min_{\mathbf{x}_{1:T},m} \sum_{t=1}^{T} \|\mathbf{z}_t - h(\mathbf{x}_t, m)\|_2^2 + \sum_{t=0}^{T-1} \|\mathbf{x}_{t+1} - f(\mathbf{x}_t, \mathbf{u}_t)\|_2^2 \quad (5)$$

.

D. Factor Graph

The SLAM problem given in (5), can be represented or reformulated using a factor graphs, which is a bipartite graph describing the data (z_t, u_t) and variables (x_t, m_j). The nodes/vertices of the Factor graph are the robot poses $\mathcal{V} := T_0, T_1, ..T_T$ and the edges are the relative poses between consecutive poses given as $\mathcal{E} := {}_0T_1, {}_1T_2 ..., {}_{T-1}T_T$. Using the factor graph setup we can reformulate the optimization problem presented in (5) as,

$$\min_{\mathbf{x_i}} \sum_{(i,j)\in\mathcal{E}} \phi_{ij}(e(\mathbf{x_i}, \mathbf{x_j})) \quad (6)$$

where, $\mathbf{x_i}$ are variables associated with the graph vertices $i \in \mathcal{V}$ and factors $e(\mathbf{x_i}, \mathbf{x_j})$ associated with the graph edges $(i, j) \in \mathcal{E}$ and $\phi_{ij} : \mathbb{R}^d \to \mathbb{R}$ is a distance function,

$$\phi_{ij}(e) := e^\top W_{ij}^\top W_{ij} e \quad (7)$$

W_{ij} is the weight between the edges i & j and the initial guess is obtained from odometry. A descent method is then applied for optimization,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \delta\mathbf{x}^{(k)} \quad (8)$$

and The Levenberg-Marquardt algorithm is used for (7)

$$\left(\sum_{ij} J_{ij}^\top W_{ij}^\top W_{ij} J_{ij} + \lambda D\right) \delta\mathbf{x}^{(k)} = -\sum_{ij} J_{ij}^\top W_{ij}^\top e(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}) \quad (9)$$

where $J_{ij} = \frac{\partial e(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}^{(k)}}$ is the Jacobian of $e(\mathbf{x}_i, \mathbf{x}_j)$ with respect to all variables $\mathbf{x}$ evaluated at $\mathbf{x} = \mathbf{x}^{(k)}$.

E. Occupancy Grid and Texture mapping

The occupancy grid is a 2D representation of the environment in which the robot operates. Each cell represents a small area of the environment and contains a value that indicates the likelihood of that area being occupied by an obstacle (or free space). The grid updates as the robot moves through the environment and gathers sensor data (i.e; LiDAR readings) to map obstacles and free space. Given a grid map $M \in R^{nxn}$ (n is the dimension of the map), each cell holds the cummulative odds of it being occupied or not as the robot traverses the environment. This can be briefly expressed as described below. *(Note: please check references for a more elaborate formulation)*

Let X_{ij} be a binary random variable that indicates the true occupancy of the cell at row i and column j, where $X_{ij} = 1$ corresponds to the cell being occupied and $X_{ij} = 0$ being free. Let Y_{ij} be another binary random variable that represents the measured occupancy of that cell. By Bayes' Rule, we have

$$P(X_{ij} \mid Y_{ij}) := \frac{P(Y_{ij} \mid X_{ij}) P(X_{ij})}{P(Y_{ij})} \quad (10)$$

writing Eqn.(10) in terms of log odds we get,

$$\text{logodd}(X_{ij} \mid Y_{ij}) = \log\left(\frac{P(Y_{ij} \mid X_{ij} = 1)}{P(Y_{ij} \mid X_{ij} = 0)}\right) + \text{logodd}(X_{ij}) \quad (11)$$

. We can further simplify the representation of the odds of a cell being occupied or empty as,

$$\text{logodd}(X_{ij} \mid Y_{ij} = 1) = \Delta_{\text{occ}} + \text{logodd}(X_{ij}) \quad (12)$$

$$\text{logodd}(X_{ij} \mid Y_{ij} = 0) = \Delta_{\text{free}} + \text{logodd}(X_{ij}) \quad (13)$$

$$\Delta_{\text{occ}} := \log\left(\frac{P(Y_{ij} = 1 \mid X_{ij} = 1)}{P(Y_{ij} = 1 \mid X_{ij} = 0)}\right) \quad (14)$$

$$\Delta_{\text{free}} := \log\left(\frac{P(Y_{ij} = 0 \mid X_{ij} = 1)}{P(Y_{ij} = 0 \mid X_{ij} = 0)}\right) \quad (15)$$

where $\text{logodd}(X_{ij})$ is the existing log-odd ratio at cell (i, j), $\text{logodd}(X_{ij} \mid Y_{ij} = 1)$ is the updated log-odd ratio after observing the cell (i, j) being occupied, and $\text{logodd}(X_{ij} \mid Y_{ij} = 0)$ is the updated log-odd ratio after observing it being unoccupied.

In texture mapping, the grid map (M) is rasterized using images acquired by the Kinect camera that lead to the textured map $M' \in R^{nxnx3}$, and each cell contains the RGB data corresponding to the coordinate of the map.

III. Technical Approach

A. Sensor Data Pre-processing and Preparation

The robot 4 avenues for acquiring data about the environment and robot movements. The sensors onboard the robot are,

1) Encoders: Each wheel of the robot is equipped with an encoder, which counts the number of wheel rotations. The frequency in which the sensor data acquisition rate is 40 Hz. We use the encoder data to calculate the linear velocity of the robot. The data from then encoder at time t is represented as FR, FL, RR, RL, where FR, FL, RR and RL denotes the encoder reading from Front Right, Front Left, Rear Right and Rear Left wheels respectively. One complete rotation of the wheel corresponds to l meters traversed, where l is the circumference of the wheel. The distance traveled by the wheel, corresponding to one tick on the encoder is:

$$\text{meters per tick} = \frac{\pi \times \text{wheel diameter}}{\text{ticks per revolution}} = 0.0022 \text{ m/tick} \quad (16)$$

where the dimeter of the wheel is 0.254 m and there are 360 ticks per revolution as these are given in the data sheet. The distances traveled by the right (D_r) and left(D_L) wheels are given by,

$$\begin{aligned} D_R &= \frac{(FR + RR)}{2} \times 0.0022 \text{ m} \\ D_L &= \frac{(FL + RL)}{2} \times 0.0022 \text{ m} \end{aligned} \quad (17)$$

The distance traveled by the robot is taken as the arithmetic mean of D_R and D_L. The velocity of the robot at time interval τ_t is given as,

$$\begin{aligned} V_L &= D_L/\tau_t \\ V_R &= D_R/\tau_t \\ V &= \frac{V_L + V_R}{2} \end{aligned} \tag{18}$$

, where V_L and V_R is the velocity of the left wheels and right wheels respectively and V is the linear velocity of the robot.

2) Inertial Measurement Unit: : We rely on the inertial measurement unit to know about the angular velocity of the robot. Since it is a differential drive robot traversing in the XY plane, the yaw rate from the IMU is taken as the angular velocity of the robot.

3) Hokuyo UTM-30LX LiDAR: This is a horizontal LiDAR with a field of view of 270° and a detectable range of 0.1 to 30 meters, providing distance measurements to obstacles in the environment. Since the LiDAR has an FOV of 270°, the maximum and minimum angles of the LiDAR are $+135^\circ$ and -135°, respectively. Each LiDAR scan consists of 1081 measured range values. Therefore, the angular resolution of the LiDAR is given by: $\frac{135^\circ-(-135^\circ)}{1081} = 0.00436332$ rad.The range measurement are some times noisy, and so we remove any range measurement that are not within the detectable range of the LiDAR, and the de-noised range measurements is used moving forward.

The measurements provided by the LiDAR are in polar-coordinates, and convert them to Cartesian coordinates in the sensor frame. Given range measurement at time t, $r_i^t \in \{r_0^t, r_1^t, \ldots, r_{1080}^t\}$

The Cartesian-coordinates are:

$$\begin{bmatrix} x_i^t \\ y_i^t \\ z_i^t \end{bmatrix} = \begin{bmatrix} r_i^t \cos\alpha_i^t \\ r_i^t \sin\alpha_i^t \\ 0 \end{bmatrix} \tag{19}$$

where,

$$\alpha_i^t = -135^\circ + i \times \text{(angular res. of the LiDAR) deg} \tag{20}$$

, the z co-ordinate is set to zero as it a horizontal LiDAR mounted on a robot traversing in the XY plane. We can see that the LiDAR is offset from the center of the robot by (0.133,0,0.514)m, which is applied applied to LiDAR Point cloud generated by eqn (19), to get the LiDAR point cloud in the body frame.

4) Kinect - RGBD Camera: Kinect is a RGBD camera provides RGB images and disparity images. The depth camera is located at $(0.18, 0.005, 0.36)$ m with respect to the robot center and has orientation with roll 0 rad, pitch 0.36 rad, and yaw 0.021 rad. The intrinsic parameters of the depth camera are:

$$K = \begin{bmatrix} fsu & fs\theta & cu \\ 0 & fsv & cv \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix}. \tag{21}$$

The depth camera and RGB camera are not in the same location. There is an x-axis offset between them and it is necessary to use a transformation to match the color to the depth. Given the values d at pixel (i, j) of the disparity image, you can use the following conversion to obtain the depth and the pixel location (rgbi,rgbj) of the associated RGB color:

$$\begin{aligned} dd &= (-0.00304d + 3.31) \\ \text{depth} &= \frac{1.03}{dd} \\ rgbi &= \frac{526.37i + 19276 - 7877.07 \cdot dd}{585.051} \\ rgbj &= \frac{526.37j + 16662}{585.051} \end{aligned} \tag{22}$$

The pixels from the RBGD images are to be used to project onto the world, so we need to get the pixel co-ordinate to the World frame, which is done by first get the pixel coordinates in the optical frame, and then to the regular camera frame coordinates (X_r, Y_r, Z_r), given by ,

$$\begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix} = \mathbf{oR_r}^{-1}\mathbf{K}^{-1} \begin{bmatrix} \text{rgbi} \\ \text{rgbj} \\ 1 \end{bmatrix} \cdot \text{depth} \tag{23}$$

where,

$${}_oR_r = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \tag{24}$$

Then the points in regular camera coordinates is then transformed using the known offset and orientation of the camera with respect to the body frame to get the points the frame of the body of the robot. The (rbgi, rgbj, depth) in the body frame is given as,

$$\begin{bmatrix} X_B \\ Y_B \\ Z_B \\ 1 \end{bmatrix} = {}_B\,T_r \begin{bmatrix} X_r \\ Y_r \\ Z_r \\ 1 \end{bmatrix} \tag{25}$$

where,

$${}_BT_r = \begin{bmatrix} 0.93569047 & -0.02099846 & 0.35219656 & 0.18 \\ 0.01965239 & 0.99977951 & 0.00739722 & 0.005 \\ -0.35227423 & 0 & 0.93589682 & 0.36 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

The body frame coordinates from (25) when multiplied with the robot pose $({}_WT_B)$ at time t, will transform the pixel coordinate (rgbi, rbgj , depth) at time t to its respective world coordinates.

All above mentioned sensor data are not synchronized, and for this project, all the sensor data are synchronized with respect to the encoder timestamps when we are working with the sensor data.

B. Encoder and IMU odometry

As explained in the above section, the linear (v) and angular velocity (ω_z) of the robot at all timestamps is extracted, is given to the motion model expressed in the equation (1) to get

the robot poses. These poses are then used to get the initial estimate of the robot poses and trajectory.

C. Point cloud registration via Iterative closest point (ICP)- Warm up

We use the Iterative closest point algorithm to solve the point cloud registration problem. The ICP algorithm iterates between finding associations Δ based on closest points and applying the Kabsch algorithm to determine p, R for the point clouds m_i and z_j. In the warm up scenario, we are given that the object only rotates in the z axis, so it is a reasonable estimate to pick the initial guess for R=R_0 by discretizing the yaw angles. A reasonable estimate for p=p_0, is the difference of the centroid of the given point clouds. In the data association step, we find correspondences $(i, j) \in \Delta$ based on closest points, given R_k and p_k:

$$i \leftrightarrow \arg\min_j \|m_i - (R_k z_j + p_k)\|_2^2 \tag{27}$$

,where k denotes the k^{th} iteration of ICP.

With the correspondences, from the data association step we apply the Kabsch algorithm on m_i and z_j, and we rewrite (4) as,

$$\min_{R \in SO(d), p \in \mathbb{R}^d} f(R, p) := \sum_i w_i \|(Rz_i + p) - m_i\|_2^2 \tag{28}$$

To solve this, we set $\nabla_p f(R, p) = 0$, , and solve for p we get,

$$\mathbf{p} = \bar{m} - R\bar{z} \tag{29}$$

where,

$$\begin{aligned} \bar{m} &= \frac{\sum_i w_i m_i}{\sum_i w_i} \\ \bar{z} &= \frac{\sum_i w_i z_i}{\sum_i w_i} \end{aligned} \tag{30}$$

. Replacing (29) in (28), finding the optimal rotation reduces to:

$$\min_{R \in \text{SO}(d)} \sum_i w_i \|R\delta z_i - \delta m_i\|_2^2 \tag{31}$$

, where

$$\begin{aligned} \delta m_i &:= m_i - \bar{m} \\ \delta z_i &:= z_i - \bar{z} \end{aligned} \tag{32}$$

are the centred point clouds. This the Wahba's problem, where we 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):

$$\max_{R \in \text{SO}(d)} \text{tr}(Q^T R) \tag{33}$$

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

The Wahba's problem (33),is solved using Kabsch algorithm. We set the weights w_i as 1, and taking the single value decomposition of $Q = U\Sigma V^T$, the optimal $\mathbf{R}$ is given as,

$$\mathbf{R} = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(UV^T) \end{pmatrix} V^T \tag{34}$$

, using this optimal $\mathbf{R}$, recompute $\mathbf{p}$ using (29). In ICP the, above mentioned Data association step and Kabsch step is performed iteratively over K iteration, using the intial estimate of R_0 and p_0 to get the optimal R_k and p_k after K iterations. We run ICP with different initializations as discussed above and choose the output of the iteration with the least squared distance between point clouds after K iterations and use that to shift the point clouds of the dataset drill or container. *Note: In our implementations d in SO(d) is 3, $R \in R^{3x3}$ and $p \in R^3$*

D. Point cloud registration via Iterative closest point (ICP)- Scan Matching

Using ICP we use the odometry data (IMU and Encoder data) and the LiDAR data in cohesion at each timestamp t to achieve at a better trajectory for the robot motion. We take 2 LiDAR scans at time t and $t+1$, and generate the point clouds using the method explained in the section III.A.3. We perform scan matching on the LiDAR point cloud at timestamp t (z_i)and $t+1$ (m_i), using the ICP implementation as detailed above. ICP is used to give a better estitmate for the realtive transformation of the robot poses at t and t+1 ie; ${}_tT_{t+1}$. The initial guess for the relative transformation between the point clouds is given as $T_t^{-1} * T_{t+1}$, where T_t and T_{t+1} are the robot poses from the motion model at time t and $t+1$. The scan matched robot pose at time t+1 is given as,

$$\mathbf{T}_{t+1} = \mathbf{T}_t *_t \mathbf{T}_{t+1} \tag{35}$$

, where ${}_t\mathbf{T}_{t+1}$ is the ICP estimate of the relative transformation of the LiDAR point clouds at t and t+1, and $\mathbf{T}_t$ is the scan matched robot pose for time t.

E. Occupancy and texture mapping

As stated in the problem formulation , we consider a grid map $M \in R^{nxn}$ (n is the dimension of the map). To simplify our log odds update of if a cell is occupied or not, we take $\Delta_{occ} = log(4)$ and $\Delta_{free} = -log(4)$, which are defined in eqn. (14) and (15) respectively. We also clamp the logodds of a cell being occupied at -4*log(4) and 4*log(4). The (x,y) of a point in the LiDAR point cloud is transformed to the grid coordinates (x_M, y_M) as, We consider a grid map of dimension 40x40 meters, with the origin at the center. The range of x_M and y_M is [-20,20] meters. The map resolution is taken to be 0.05 meters. The transformation of (x,y) to (x_M, y_M) is given as,

$$\begin{aligned} x_M &= (x - min(x_m))/\text{map resolution} \\ y_M &= (y - min(y_m))/\text{map resolution} \end{aligned} \tag{36}$$

So, at any given time t, we transform the LiDAR point clouds and current position of the robot, taken from the robot pose at time t, covert them to to map grid, which gives us the end and start points for the bresenham2D algorithm to coordinates of points the LiDAR passes through. This give us the coordinates of all the unoccupied grid cells, while the other coordinates are occupied. Using these coordinates we perform the log odds updates as postulated in (12) and (13) using logodds values discussed in the above paragraph. We repeat this process for

all given LiDAR scans, which are synced to the timestamp of the robot poses.

In texture mapping, we take the pixel coordinates in body frame, which we derived in section III.A.4, and multiply it with the robot pose to get the pixel coordinates in world frame. We defined a texture map $M' \in R^{NXNX3}$, where N is the map dimension. M' has same dimension and resolution as that of M. The pixel coordinates in the world frame is filtered to have only those with z coordinate values below 0.3m, in order to segment only the floor pixels, is then converted to the grid co-ordinate system using (36). Once this is done the corresponding RBG value of the for the given grid coordinate is used to fill in the grid cell. This how the occupancy grid gets rasterized with the RGBD images.

F. Pose graph optimization and loop closure

The factor graph is constructed using GTSAM, where the nodes are the absolute robot poses given by the motion model, and the edges are the relative transformation between robot poses as estimated by ICP. We employ pose graph estimation and loop closures to enhance the trajectory of the robot.

We implement fixed interval loop closure and proximity based loop closure. For fixed interval loop closure, for example at node/time time t=30 and fixed interval of 10, we introduce a edge between the nodes at t=30 and t=20, using the relative pose estimate of ${}_{20}T_{30}$, which is estimated by ICP. The initial pose estimate for the ICP is computed using the motion model poses at t=20 and t=30, as discussed previously.

In proximity based loop closure, at fixed interval, we look at previous poses which are in close proximity to our current pose. If we find a suitable pose which is in close proximity, which measured by if the translation difference between the nodes is less than 0.001m and if so only add an edge between the current node and the node that is in close proximity. The edge is created by the using ICP estimate of the relative transformation between the current pose and pose with close proximity. I also enforce a condition to look for poses with close proximity that are at least 200 timestamps behind the current pose, so as to eliminate adding redundant edges.

Also, by observing the trajectory from the RBGD images, we can see that the robot starts and ends in the same place. So, we add a loop closure between the initial and final pose of the robot trajectory. The factor graph is optimized using the Levenberg Marquard optimization algorithm. This algorithm is run for 4000 iterations.

IV. Results

The results of the robot trajectory for dataset 20 and 21 are in fig.1 and fig.2.

The ICP warm up results of the drill and liq container dataset are presented in figures 3-10. We had used yaw angle discretization by a factor of 4, so the possible yaw angles are $\pi/2, \pi, 3\pi/2$ and 2π. The least LSE on an average after ICP of for the drill and liq container dataset was 52.7m and 285.64m respectively. We can see that the algorithms struggles when the

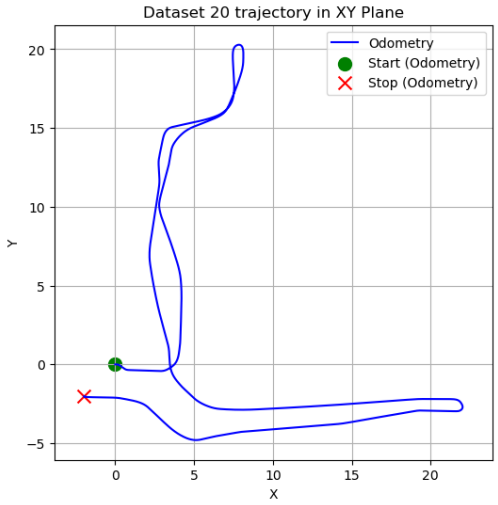

Fig. 1. Motion model robot trajectory for dataset 20

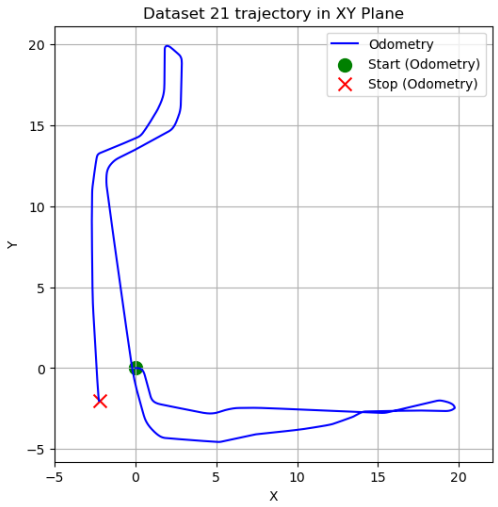

Fig. 2. Motion model robot trajectory for dataset 21

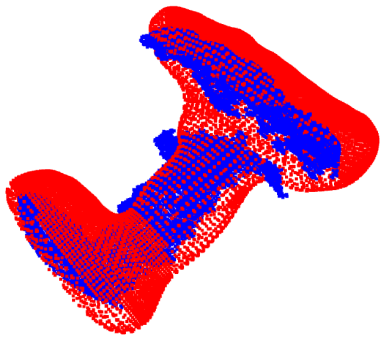

Fig. 3. Drill-1 ICP warmup output

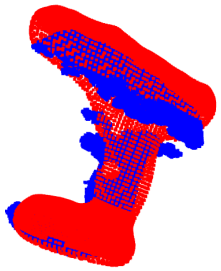

Fig. 4. Drill-2 ICP warmup output

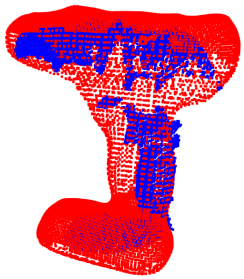

Fig. 5. Drill-3 ICP warmup output

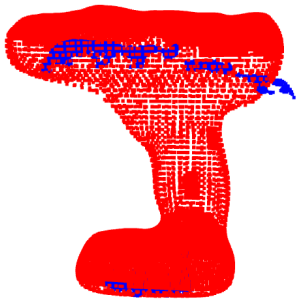

Fig. 6. Drill-3 ICP warmup output

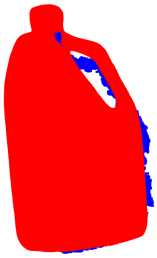

Fig. 7. Liq Container-1 ICP warmup output

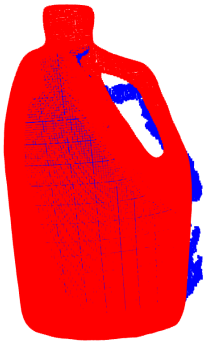

Fig. 8. Liq Container-2 ICP warmup output

density of the source point could is very less, as observeded in the 4th set of drill and liq container datasets.

The scan matching output for dataset 20 and 21 in comparison to the motion model trajectory is presented in fig.11 and 12 respectively.

The occupancy and textured grid map of dataset 20 and 21 after ICP scan matching is presented in figures. The occupancy and textured grid map of dataset 20 and 21 after pose graph optimization and loop closure by proximity and fixed interval is presented in figures.

For dataset 20, we can clearly see that the, pose graph

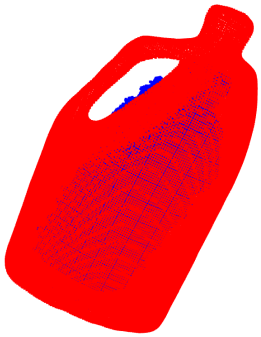

Fig. 9. Liq Container-3 ICP warmup output

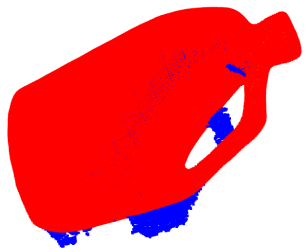

Fig. 10. Liq Container-3 ICP warmup output

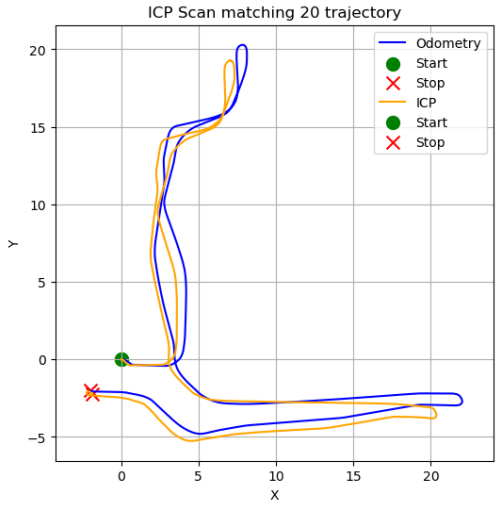

Fig. 11. ICP Scan Matching for dataset 20

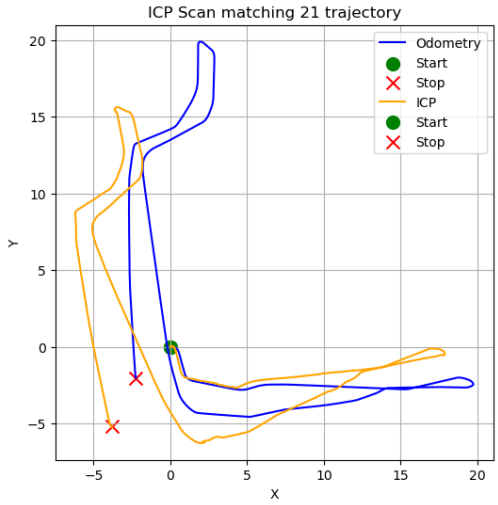

Fig. 12. ICP Scan Matching for dataset 21

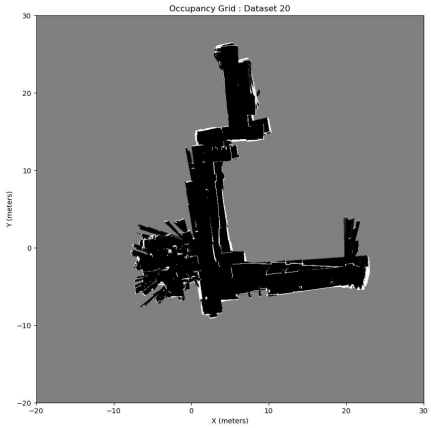

Fig. 13. Occupancy Map for Dataset 20 after ICP scan matching

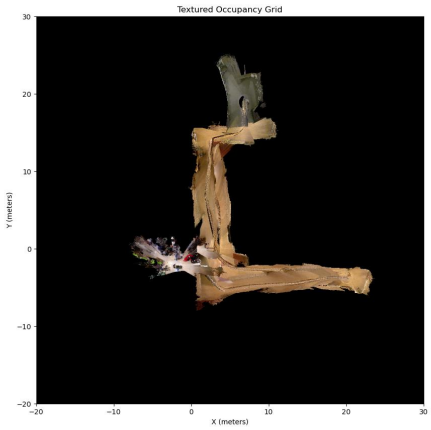

Fig. 14. Textured Map for Dataset 20 after ICP scan matching

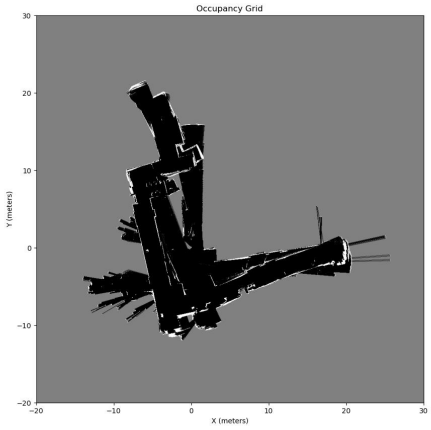

Fig. 15. Occupancy Map for Dataset 21 after ICP scan matching

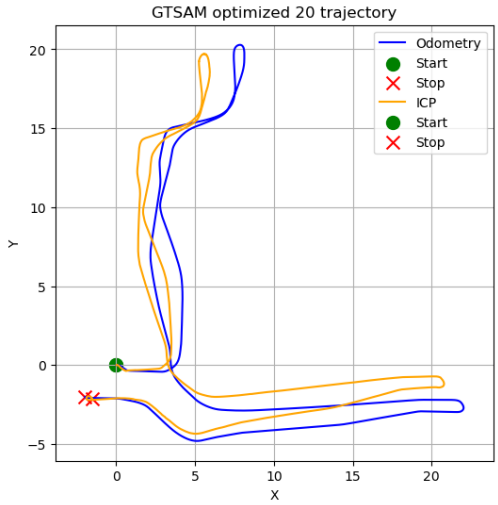

Fig. 16. Trajectory for Dataset 20 after pose graph estimation with proximity loop closure

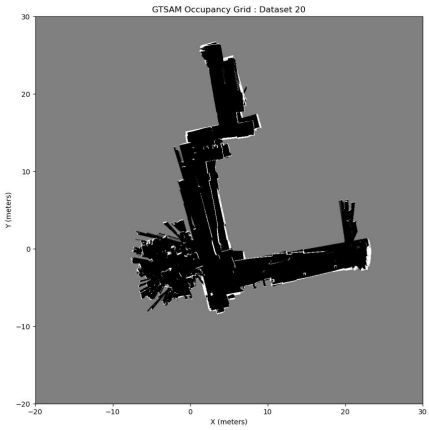

Fig. 17. Occupancy grid after Pose graph optimized output with proximity loop closure- dataset 20

optimized output has a much more cleaner textured map as the hallways are very well defined. The same can be said about the occupancy grid of 21. We can see that occupancy grid of 21 is not very consistent with that of the actual path because, we are able to see two hallways, when in actuality there is only 1 hallway. This can be correct by applying better loop closure and ICP algorithms. But we can see significant improvement in the occupancy grid and texture map for Dataset 21 when we use pose graph optimization and loop closure as compared to when we only use the ICP scan matching.

REFERENCES

1) Class lecture material
2) ICP: https://www.youtube.com/watch?v=4uWSo8v3iQA
3) Lidar datasheet: https://hokuyo-usa.com/products/lidar-obstacle-detection/utm-30lx

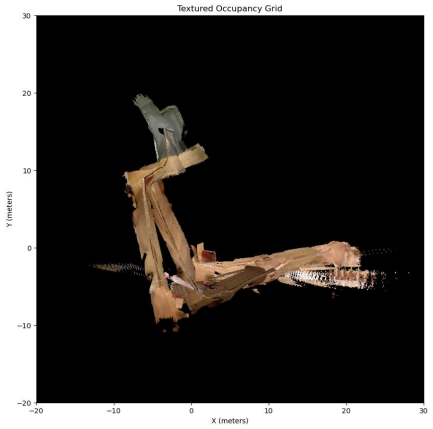

Fig. 18. Texture grid Map of Dataset 20 after ICP

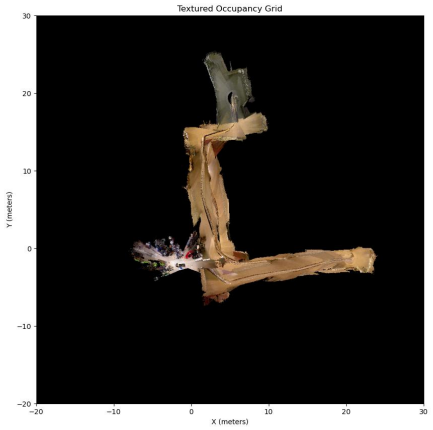

Fig. 19. Texture grid map after Pose graph optimized output with proximity loop closure- dataset 20

4) Occupancy grid (used for the problem formulation of occupancy map log odds update): https://ucb-ee106.github.io/eecs106a-fa23site/assets/labs/lab6.pdf

Acknowledegement : Discuseed with Maitrayee about the theoretical concepts involved in the project implementation.

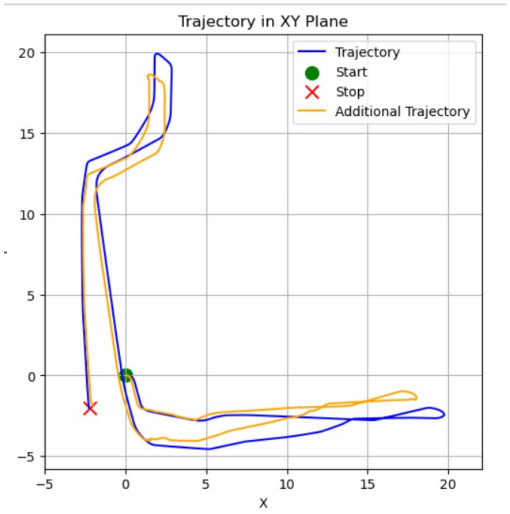

Fig. 20. Robot trajectory after Pose graph optimized output with proximity loop closure- dataset 21. Additional trajectory is GTSAM optimized and trajectory is the encoder trajectory

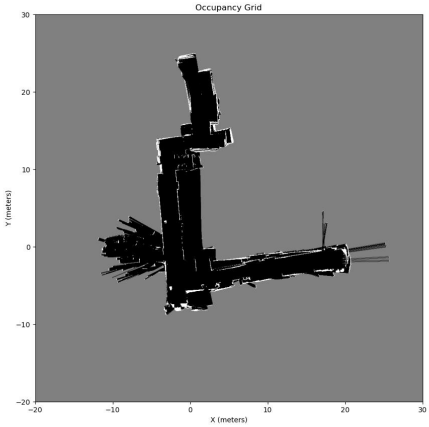

Fig. 21. Occupancy grid for Dataset 21 after pose graph optimization and loop closure

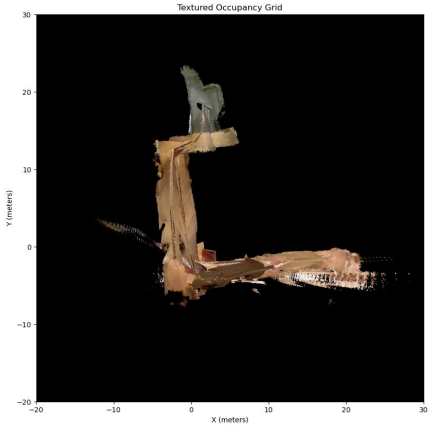

Fig. 22. Textured grid for Dataset 21 after pose graph optimization and loop closure