LiDAR-Based SLAM

Chung-Pang, Wang
University of California San Diego
Department of Electrical and Computer Engineering

***Abstract*—In this project, we will implement simultaneous localization and mapping (SLAM) using encoder and IMU odometry, 2-D LiDAR scans, and RGBD measurements from a differential-drive robot. Use the odometry and LiDAR measurements to localize the robot and build a 2-D occupancy grid map of the environment. Use the RGBD images to assign colors to the 2-D map of the floor.**

***Index Terms*—Point-Cloud Registration, SLAM, Factor Graph Optimization**

I. Introduction

In this project, we will implement simultaneous localization and mapping (SLAM) using encoder and IMU odometry to construct motion model for a differential-drive robot and improve the robot trajectory by implementing LiDAR scan matching using iterative Closest Point algorithm to optimize the trajectory with both motion model and observation model. With a better trajectory, we can build a occupancy grid map of the environment that specify where the robot have navigated (scanned) before with 2D bresenham ray tracing algorithm. Furthermore, we will build a texture map that stitches the images obtained from RGBD measurements, showing what the environment of robot traversed looks like. Finally, We will enhance the accuracy of the robot trajectory estimation through pose graph optimization with fixed-interval loop closure constraints, constructing nodes with IMU odometry, edges with scan matching and loop constraints with scan matching using Georgia Tech Smoothing and Mapping library (GTSAM).

II. Problem Formulation

A. Encoder and IMU Odometry

In the first part of the project, our goal is to construct the motion model of a differential-drive robot. With IMU angular velocity ω_t measurements and Encoders counts z ticks, we can derive linear velocity v_t at time t of the robot and plug in v_t into motion model to obtain the trajectory of the robot. Obtained discrete-time differential-drive kinematic model of the robot by utilizing Euler discretization over time interval τ_t, we have :

$$\boldsymbol{x_{t+1}} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_t \end{bmatrix} = \boldsymbol{x_t} + \tau_t \begin{bmatrix} v_t cos(\theta_t) \\ v_t sin(\theta_t) \\ \omega_t \end{bmatrix} \tag{1}$$

Where ω_t is the yaw rate obtained from IMU, θ_t is the orientation of the robot obtained from last robot state $\boldsymbol{x_t}$. After computing robot states for all time steps $\boldsymbol{x_{0:t}}$ with motion model, we have the whole robot trajectory.

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

The robot trajectory we have now constructed exclusively from motion model, which can be very inaccurate. Therefore, we also need observation model to improve the accuracy of the trajectory, meaning that we can utilize what robot observe to correct the trajectory. With 2D LiDAR scans, we can perform a point-cloud registration using iterative closest point algorithm (ICP).

1) Point-cloud Registration: Given two sets of points $\{m_i\}$and$\{z_i\}$ in $\mathbb{R}^3$, find the transformation $\boldsymbol{p} \in \mathbb{R}^3$, $\boldsymbol{R} \in SO(3)$ and data associtation Δ that align them. However, in most SLAM problem and in this project, we only have the ranges values of the LiDAR data but not the associations between scans.

2) Iterative Closest Point Algorithm: Find the transformation $\boldsymbol{p}$, $\boldsymbol{R}$ between sets $\{m_i\}$and$\{z_i\}$ of points with unknown data association Δ. Iterates between finding associations Δ based on closest points and applying the Kabsch algorithm to determine $\boldsymbol{p}$, $\boldsymbol{R}$. ICP first initialize with p_0, R_0 and find correspondences $(i, j) \in \Delta$ based on closest points:

$$i \quad \leftrightarrow \quad \arg\min_j \|\mathbf{m}_i - (R_k \mathbf{z}_j + \mathbf{p}_k)\|_2^2 \tag{2}$$

where p_k and R_k are obtained from updated pose. After getting the data association, we can solve the point cloud registration problem with Kabsch algorithm.

3) Kabsch Algorithm: A Known data association point cloud registration problem can be formulated as the following. Find the transformation $\boldsymbol{p} \in \mathbb{R}^3$, $\boldsymbol{R} \in SO(3)$ between $\{m_i\}$and$\{z_i\}$ with known association.

$$\min_{R \in SO(3), \mathbf{p} \in \mathbb{R}^3} f(R, \mathbf{p}) := \sum_i w_i \|(R\mathbf{z}_i + \mathbf{p}) - \mathbf{m}_i\|_2^2 \tag{3}$$

(3) can be rewritten as

$$\max_{R \in SO(3)} \operatorname{tr}\left(Q^\top R\right) \tag{4}$$

where $Q = \sum_i \delta\mathbf{m}_i \delta\mathbf{z}_i^\top$. This equation is known as Wahba's problem and can be solved by Kabsch algorithm. From Kabsch Algorithm, the optimal rotation R^* can be written as:

$$R^* = UAV^T \tag{5}$$

where U and V are obtained from Sigular Value Decomposition of Q ($Q = U\Sigma V^T$), and A is a matrix to avoid reflections. Now we have the optimal rotation R, we can then update the translation apply to the point cloud and associate data and apply Kabsch iteratively to get the final optimal

transformation T. We can further improve the robot trajectory by using scan matching with ICP algorithm.

C. Occupancy and texture mapping

We can construct an occupancy map with robot states $(x_{0:t}, y_{0:t}, \theta_{0:t})$ and LiDAR scans for all time steps and further construct a texture map with robot states and RGBD measurements from Kinect depth camera. For building a occupancy map, we first need to transform the LiDAR scans from LiDAR frame to robot body frame and transform scans to world frame based on the state of the robot. Then, transform the unit of the coordinates from meters to grids and apply 2D bresenham ray tracing algorithm to detect free grids and occupied grids. Add log odds to occupied grids and subtract log odds to free grids. Finally apply Sigmoid function to the whole map to convert the map into probabilistic map. For building texture map, we also need to transform images from camera frame top robot body frame. However, before that we need to transform RGB color values and pixels coordinate to optical frame, and transform optical frame to regular camera frame. With the pixels in camera frame, we can now transform it into robot body frame and then world frame using robot states $(x_{0:t}, y_{0:t}, \theta_{0:t})$. Filter out the pixels positions that does not represent floor by using the z (height) obtained from the pixels coordinates in the world frame. Finally, insert the image RGB color values to the map at image's pixels positions in the world frame to get a texture map.

D. Pose Graph Optimization and Loop Closure

With IMU odometry obtained from motion model and the improved trajectory from scan matching by combining the information from motion model and observation model, we can construct a factor graph with close loop constraints to perform a poseSLAM method to construct more accurate trajectory. Adding close loop constraints enable the robot to know which area that the robot had seen before and therefore be able to utilize more information than scan matching and IMU odometry. I use the poses of IMU odometry T_t to construct nodes and the relative pose of scan matching ${}_tT_{t+1} = T_tT_{t+1}^{-1}$ and use fix-interval loop closure method with scan matching poses ${}_tT_{t+interval} = T_tT_{t+interval}^{-1}$ to construct loop constraint. Finally apply Levenberg-Marquardt optimizer to optimize the trajectory.

III. Methods

A. Data Synchronization

Since the data for all sensors does not synchronized, we need to implement data synchronization to match the time stamps of different sensors. I use unsupervised k nearest neighbors learning algorithm to perform data synchronization. I use the data that has more time stamps to construct nearest neighbors with $k = 1$ and use the data that has less time stamps to find the closest time stamps in euclidean distance. Utilize the indices of the closest time stamps of the data that has more time stamps to synchronize two data.

$$KNN = NearestNeighbors(more\,time\,stamps) \tag{6}$$

$$indices = KNN(less\,time\,stamps) \tag{7}$$

$$Synchronized = more\,time\,stamps(indices) \tag{8}$$

B. Motion Model and Pose Estimation

Given encoder counts $[FR, FL, RR, RL]$ corresponding to the front-right, front-left, rear-right, and rear-left wheels and the wheel travels 0.0022 meters per tic. We can compute linear velocity v_t of the robot:

$$\begin{aligned} V_{left} &= \frac{(FL + RL)}{2} * time\ interval \\ V_{right} &= \frac{(FR + RR)}{2} * time\ interval \\ v_t &= \frac{V_{left} + V_{right}}{2} \end{aligned} \tag{9}$$

where time interval is obtained by $time\ interval = time\ stamps_{1:t} - time\ stamps_{0:t-1}$. With linear velocity v_t of the robot at time t and yaw rate obtained from IMU, we can obtain robot trajectory by plugging linear velocity v_t into motion model (1). After obtaining the robot states $\boldsymbol{x_{t+1}}$, I constructed homogeneous pose matrix P_t at time t to represent the robot state in the world frame at time t:

$$P_t = \begin{bmatrix} R_t & p_t \\ 0 & 1 \end{bmatrix} \tag{10}$$

where R_t is a rotation matrix of the robot orientation θ_t at time t and P_t is the position of the robot $(x_t, y_t, 0)$at time t. Since the robot is a ground robot, z position and the angle except for yaw angle are 0.

C. Iterative Closest Point (ICP) Algorithm

To ensure the implementation of the ICP algorithm is correct, I test ICP on two different 3D point cloud objects - drill and liquid container with 4 partial point cloud and different poses. Since ICP algorithm is very sensitive to initial guess and initial translation can be easily obtained from the centroids of two point cloud:

$$p_0 = \bar{m} - \bar{z} \tag{11}$$

where $\bar{m}$ is the centroid of target point cloud and $\bar{z}$ is centroid of source point cloud. We need to choose initial rotation properly. I limit the initial rotation to be yaw angles only based on the fact that we know the 3D point cloud model was placed on the ground. Therefore, I try 6 different yaw angles through out $360\,\mathrm{deg}$:

$$\Theta_i = \frac{2\pi}{6} i,\ i = 1, ..., 6 \tag{12}$$

Then construct the yaw angle in rotation matrix to apply onto point cloud. As mentioned above, the first step of ICP is data association which is associate two point clouds. I use KNN

to associate point clouds, similar to data synchronization, I use the point cloud that has more points which is the source point cloud to build/initialize the Nearest Neighbor and use target point cloud to find the closest points in source point cloud. With know data association, we can solve the point cloud registration problem with Kabsch algorithm in (5). We first need to find $Q = \sum_i \delta \mathbf{m}_i \delta \mathbf{z}_i^\top$ where $\delta \mathbf{m}_i = m_i - \bar{m}$ and $\delta \mathbf{z}_i = z_i - \bar{z}$. With Q, we can apply Singular Value Decomposition and get the optimal rotation R_i^* from (5) at iteration i. Then we can update translation $p = \bar{m} - R\bar{z}$. To get the final transformation T, we need to accumulate all of the rotation and translation, final transformation T can be written as:

$$T_t = \begin{bmatrix} R_n ... R_0 & R_n(R_{n-1}p_{t-1} + p_t) \\ 0 & 1 \end{bmatrix} \tag{13}$$

Finally, we apply final transformation T on source point cloud to make source point cloud as close as target point cloud as possible.

D. Lidar Scan Matching

With ICP algorithm from previous section, we can implement LiDAR scan matching, combining both the infomation from motion model and observation model to refine the overall trajectory estimation. We first need to convert LiDAR ranges data into the points in x,y coordinates.

$$\begin{aligned} x &= range * cos(\theta_i) \\ y &= range * sin(\theta_i) \end{aligned} \tag{14}$$

Since the liDAR scan from -135 degrees to +135 degrees, θ ranges from -135 to +135 with the number of LiDAR scans. Then, we transform the LiDAR Scans from LiDAR frame to robot body frame by subtracting by the distance between LiDAR sensor and robot center. Now, we can use the relative pose of IMU odometry ${}_tP_{t+1} = P_t P_{t+1}^{-1}$ to get a good initial guess R_0 and p_0 extract from ${}_tP_{t+1}$. We can now obtain the poses of every time step by updating current poses using ICP:

$$\begin{aligned} P'_{t+1} &= P_{tt} T_{t+1} \\ P'_{t+2} &= P'_{t+1t+1} T_{t+2} \end{aligned} \tag{15}$$

Overtime, we will get the whole new refined trajectory.

E. Mapping

1) Occupancy Grid Mapping: With robot trajectory and LiDAR scans, we can build a occupancy grid map that shows where the robot had "seen" before. We first need to transform the LiDAR scans from LiDAR frame to robot body frame as the way we did in scan matching. Then, transform scans to world frame based on the state of the robot:

$$lidar\ scans_w = P_t \cdot lidar\ scans_{lidar} \tag{16}$$

Then, we need to transform the unit of the coordinates from meters to grids:

$$\begin{aligned} x_{grid} &= x - MAP_{xmin} \\ y_{grid} &= x - MAP_{ymin} \end{aligned} \tag{17}$$

Now we can apply 2D bresenham ray to compute from a starting points (sx,sy) to a end points (ex,ey), where did a ray pass through from starting point to end point. since the input starting points are in grids, the output of 2D bresenham algorithm will also in grids. tracing algorithm to detect free grids and occupied grids. Add log odds to the location of occupied grids in the map and subtract log odds to location of free grids in the map.

$$\begin{cases} free\ grids : -log(2) \\ occupied\ grids : +log(4) \end{cases} \tag{18}$$

Finally apply sigmoid function to make it a probabilistic map.

$$sigmoid = \frac{1}{1 + e^{-x}} \tag{19}$$

2) Texture Mapping: Similar to Occupancy Grid Mapping, we first need to transform RGB color values and pixels coordinate from image coordinate to optical frame with intrinsic matrix K, and transform optical frame to regular camera frame. With the pixels in camera frame, we can now transform it into robot body frame with robot poses:

$$camera\ images_w = P_t \cdot camera\ images_{body} \tag{20}$$

We now have pixels coordinates in the world frame (x_w, y_w, z_w). Since we are stitching floor texture to the texture map, I set the z axis pixels coordinates $z_w < 0.2(m)$. Finally, we get the texture map that show the environment of where robot had navigated.

$$K = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

F. Factor Graph SLAM

With IMU odometry obtained from motion model and the improved trajectory from scan matching by combining the information from motion model and observation model, we can construct a factor graph with close loop constraints to perform a poseSLAM method to construct more accurate trajectory. Adding close loop constraints enable the robot to know which area that the robot had seen before and therefore be able to utilize more information than scan matching and IMU odometry. I use the poses of IMU odometry T_t to construct nodes and the relative pose of scan matching ${}_tT_{t+1} = T_t T_{t+1}^{-1}$ and use fix-interval loop closure method with scan matching poses ${}_tT_{t+interval} = T_t T_{t+interval}^{-1}$ to construct loop constraint. Finally apply Levenberg-Marquardt optimizer to optimize the trajectory. (All factor graph optimization are implement with GTSAM) Levenberg-Marquardt optimizer:

$$\left(\sum_{ij} J_{ij}^\top W_{ij}^\top W_{ij} J_{ij} + \lambda D \right) \delta \mathbf{x}^{(k)} = - \sum_{ij} J_{ij}^\top W_{ij}^\top \mathbf{e} \left(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)} \right) \tag{22}$$

IV. Results

A. IMU Odometry

The results showed that the goal of the experiment should be making the starting point and the end point of the trajectory aligned.

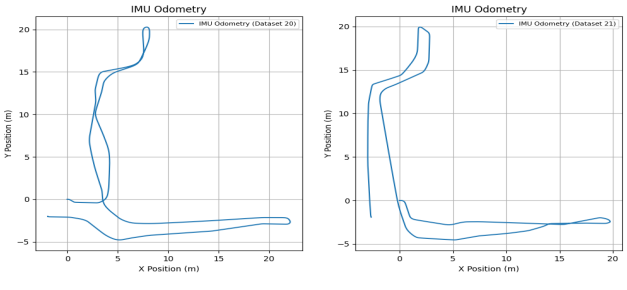

Fig. 1. IMU Odometry Trajectory

B. Point-cloud Registration via ICP

Since ICP is very sensitive to initial guess, some of the point cloud are not aligned. I think it can only be fixed by adjusting initial guess manually.

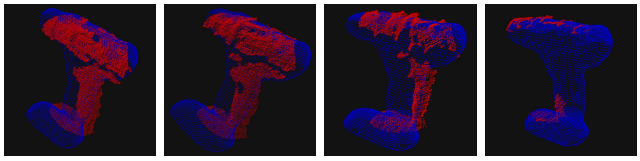

Fig. 2. ICP for 4 different drill poses

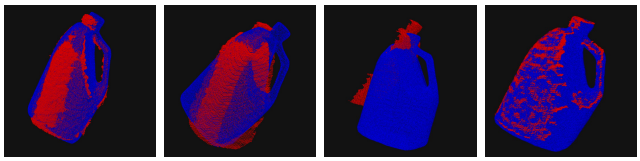

Fig. 3. ICP for 4 different drill poses

C. Lidar Scan Matching

We can see from Fig.4 that scan matching is a little bit off from the IMU odometry, meaning that scan matching algorithm did correct the IMU trajectory but we will further see that scan matching algorithm is not the best result we can get.

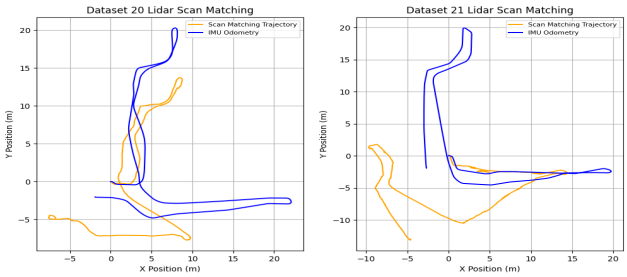

Fig. 4. ICP for 4 different drill poses

D. Texture Mapping and Occupancy Grid Mapping

Texture Mapping and Occupancy Grid Mapping give us a good sense of the environment of what robot had been. It also shows the trajectory optimized with GTSAM has the best results.

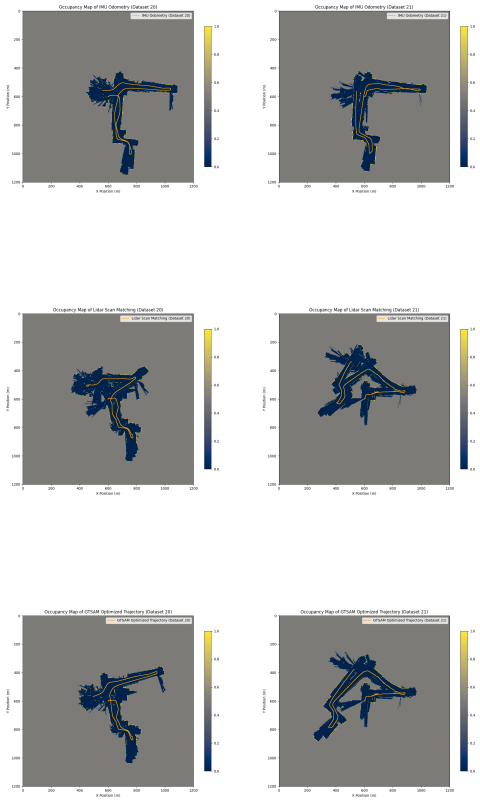

Fig. 5. Occupancy Grid Map

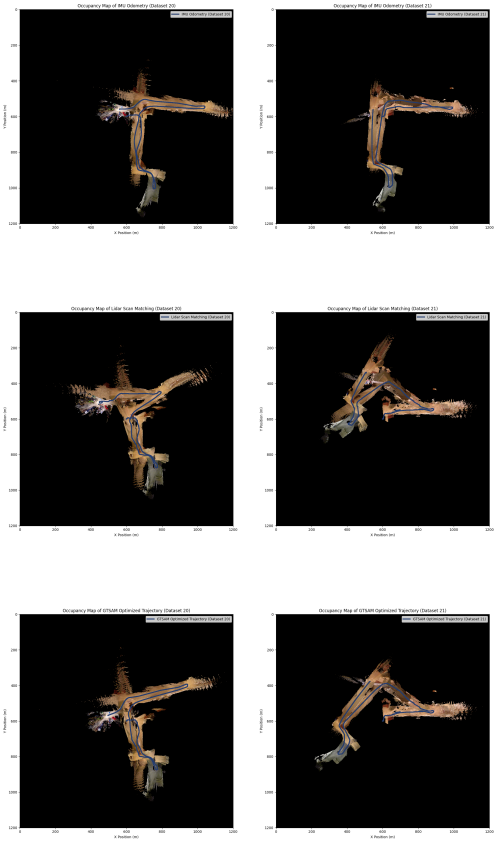

Fig. 6. Texture Map

E. Factor Graph Optimization

Since we do not have the ground truth of the robot trajectory, we can not build a good factor graph with proper covariance of odometry model and for loop closure model. In theroy, the trajectory from scan matching should be better than imu odometry but we can tell from the result that this is not the case. I believed the reason is that our ICP has large error in it to make the trajectory inaccurate.

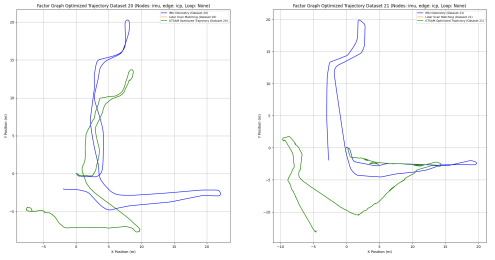

Fig. 7. GTSAM Optimized Trajectory with no loop

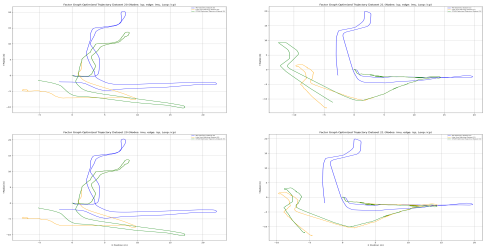

Fig. 8. GTSAM Optimized Trajectory with loop

REFERENCES

[1] UCSD ECE276A slides on motion models and observation models https://natanaso.github.io/ece276a/ref/ECE276A_4_MotionAndObservationModels.pdf

[2] UCSD ECE276A slides on locoalization Modelshttps://natanaso.github.io/ece276a/ref/ECE276A_6_LocalizationOdometry.pdf

[3] UCSD ECE276A slides on Factor Graph SLAM Modelshttps://natanaso.github.io/ece276a/ref/ECE276A_5_FactorGraphSLAM.pdf

[4] astropy library https://docs.astropy.org/en/stable/api/astropy.coordinates.spherical_to_cartesian.html

[5] Jax library https://jax.readthedocs.io/en/latest/