Differential-Drive Iterative Closest Point (ICP)

Kevin Shin
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, USA
d3shin@ucsd.edu

***Abstract*—This project focuses on implementing a Simultaneous Localization and Mapping (SLAM) system for a differential-drive robot equipped with encoders, an Inertial Measurement Unit (IMU), a 2-D LiDAR scanner, and an RGBD camera. The primary objective is to estimate the robot's trajectory and construct a 2-D occupancy grid map of the environment while assigning floor textures using RGBD data. The robot's motion is initially predicted using encoder-derived linear velocity and IMU-measured yaw rate, following a differential-drive motion model. LiDAR scan matching is then employed to refine the trajectory estimates through the Iterative Closest Point (ICP) algorithm, which aligns consecutive LiDAR scans to compute relative pose changes. The ICP algorithm is initialized using odometry estimates from the encoders and IMU, and the resulting trajectory is used to generate an occupancy grid map of the environment. Additionally, RGBD images are processed to create a 2-D texture map of the floor by projecting depth and color data onto the world frame. Finally, pose graph optimization with loop closure constraints is implemented using the Georgia Tech Smoothing and Mapping (GTSAM) library to further enhance trajectory accuracy and map quality. The project report details the problem formulation, technical approach, and results, including trajectory plots, occupancy maps, and textured floor maps. The work is carried out individually, with collaboration limited to discussion, and strict adherence to academic integrity guidelines is maintained.**

***Index Terms*—Sensor Fusion, Rotations, Motion Model, Observation Model, Projected Gradient Descent**

I. Introduction

In robotics, accurately estimating the position and orientation of a moving robot is a fundamental challenge with applications ranging from autonomous navigation to environmental monitoring. This project addresses the problem of localization and odometry refinement using data from a differential-drive robot equipped with encoders, an Inertial Measurement Unit (IMU), and a 2-D LiDAR scanner. The encoders and IMU provide odometry measurements, while the LiDAR captures environmental data that can be used to refine the robot's trajectory. By combining these data sources, we aim to estimate the robot's pose (position and orientation) over time using motion model predictions and refine these estimates through LiDAR scan matching with the Iterative Closest Point (ICP) algorithm.

The ability to accurately localize a robot is critical for tasks such as autonomous exploration, warehouse automation, and search-and-rescue operations. LiDAR sensors are widely used for localization due to their high precision in measuring distances to obstacles. However, sensor measurements are prone to noise, drift, and synchronization issues, which can degrade the accuracy of pose estimates. To address these challenges, we employ a multi-sensor fusion approach that integrates encoder and IMU odometry with LiDAR scan matching using ICP. The motion model predicts the robot's trajectory based on encoder-derived linear velocity and IMU-measured yaw rate, while ICP refines the trajectory by aligning consecutive LiDAR scans.

This project focuses on the implementation of ICP-based LiDAR scan matching and motion model predictions to estimate the robot's trajectory. While the full SLAM pipeline, including occupancy grid mapping and pose graph optimization, was not completed, the foundational work of trajectory estimation and scan matching provides a framework for future extensions. The project leverages sensor data from the robot, including encoder counts, IMU measurements, and LiDAR scans, to validate the localization and odometry refinement process.

This report outlines the problem formulation, technical approach, and results of the project. It includes a detailed description of the motion model, ICP-based scan matching, and the process of aligning LiDAR scans to refine the robot's trajectory. The results section presents trajectory plots and ICP alignment visuals, demonstrating the effectiveness of the proposed approach.

II. Problem Formulation

A. ICP for Localization and Odometry from Point Clouds

The problem of localization and odometry refinement involves estimating the trajectory of a robot while simultaneously improving the accuracy of its pose estimates using sensor data. In this project, we focus on a differential-drive robot equipped with encoders, an Inertial Measurement Unit (IMU), and a 2-D LiDAR scanner. The goal is to estimate the robot's pose (position and orientation) over time using odometry from the encoders and IMU, and to refine this estimate by aligning LiDAR scans using the Iterative Closest Point (ICP) algorithm. The problem is formulated as follows:

The goal can be seen as:

$$T^*_{0:T} = \arg\min_{T_{0:T}} \sum_{t=0}^{T} \|T_t - T^*_t\|^2,$$

where:

Department of Electrical and Computer Engineering, University of California, San Diego

- T_t represents the estimated transformation (pose) of the robot at time t,
- T_t^* represents the ground-truth,

The objective function penalizes deviations between the estimated and true poses while ensuring that the transformations remain physically plausible (e.g., valid rotations and translations).

B. Motion Model for Differential-Drive Robot

The robot's motion is modeled using a differential-drive kinematic model. The state of the robot at time t is represented as $x_t = (p_t, \theta_t)$, where:

- $p_t = (x_t, y_t) \in \mathbb{R}^2$ is the robot's position in the world frame,
- $\theta_t \in (-\pi, \pi]$ is the robot's orientation (yaw angle) in the world frame.

The control input $u_t = (v_t, \omega_t)$ consists of:

- $v_t \in \mathbb{R}$: linear velocity (derived from encoder measurements),
- $\omega_t \in \mathbb{R}$: angular velocity (yaw rate, derived from IMU measurements).

The continuous-time kinematic model is given by:

$$\dot{x}_t = f(x_t, u_t) = \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}.$$

For discrete-time implementation, the state update can be approximated using Euler discretization over a time interval τ_t:

$$x_{t+1} = f_d(x_t, u_t) = x_t + \tau_t \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}.$$

Alternatively, the state update can be computed using **exact integration** over a time interval τ_t. The exact integration is equivalent to the discrete-time pose kinematics:

$$\begin{bmatrix} R(\theta_{t+1}) & p_{t+1} \\ 0^\top & 1 \end{bmatrix} = \begin{bmatrix} R(\theta_t) & p_t \\ 0^\top & 1 \end{bmatrix} \exp\left(\tau_t \begin{bmatrix} 0 & -\omega_t & v_t \\ \omega_t & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right),$$

where:

- $R(\theta_t)$ is the 2D rotation matrix corresponding to the orientation θ_t,
- $\exp(\cdot)$ is the matrix exponential, which computes the transformation matrix for the given twist $(v_t, 0, \omega_t)^\top$.

This formulation provides a more accurate representation of the robot's motion by accounting for the continuous curvature of its path during the time interval τ_t.

C. LiDAR Scan Matching Using ICP

To address the drift in odometry, LiDAR scan matching is employed using the Iterative Closest Point (ICP) algorithm. Given two consecutive LiDAR scans S_t and S_{t+1}, the goal is to estimate the relative transformation ${}^tT_{t+1}$ that aligns S_{t+1} with S_t. This transformation is computed by minimizing the error between corresponding points in the two scans:

$${}^tT_{t+1} = \arg\min_T \sum_{i=1}^{N} \|p_i - T \cdot q_i\|^2,$$

where:

- $p_i \in S_t$ and $q_i \in S_{t+1}$ are corresponding points,
- T is the transformation matrix representing rotation and translation.

The ICP algorithm iteratively refines the transformation by alternating between finding point correspondences and solving for the optimal transformation.

D. Objective

The objective is to generate an accurate trajectory estimate $T_{0:T}$ that minimizes the cumulative error in the robot's pose over time. The quality of the trajectory depends on the accuracy of the odometry estimates and the effectiveness of the ICP algorithm in refining these estimates

III. Preliminaries

A. Transformation Matrices

Transformations between coordinate frames are represented using homogeneous transformation matrices in $SE(2)$ or $SE(3)$. For example, the transformation from frame A to frame B is:

$$T_{A\to B} = \begin{bmatrix} R_{A\to B} & t_{A\to B} \\ 0^\top & 1 \end{bmatrix},$$

where $R_{A\to B}$ is the rotation matrix and $t_{A\to B}$ is the translation vector.

IV. Technical Approach

A. Velocity Computation from Encoders

The robot's linear velocity v_t is derived from the encoder counts. The encoders measure the rotation of the wheels, which is converted into linear displacement using the wheel geometry. The process involves the following steps:

1) Encoder Counts to Displacement: The encoder counts for the right and left wheels are averaged to reduce noise:

$$\text{RIGHT\_COUNT} = \frac{\text{counts}_{\text{right\_front}} + \text{counts}_{\text{right\_rear}}}{2},$$

$$\text{LEFT\_COUNT} = \frac{\text{counts}_{\text{left\_front}} + \text{counts}_{\text{left\_rear}}}{2}.$$

The linear displacement of each wheel is computed using the wheel radius r and the encoder resolution:

$$\text{DISPLACEMENT} = \text{counts} \times \frac{2\pi r}{\text{resolution}},$$

where:

- $r = 0.127\,\text{m}$ is the wheel radius,
- resolution $= 360$ is the number of encoder ticks per revolution.

2) Linear Velocity Calculation: The linear velocity of each wheel is computed as:

$$v_{\text{right}} = \frac{\text{DISPLACEMENT}_{\text{right}}}{\Delta t},$$

$$v_{\text{left}} = \frac{\text{DISPLACEMENT}_{\text{left}}}{\Delta t},$$

where Δt is the time difference between consecutive encoder readings.

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

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

B. Angular Velocity from IMU

The robot's angular velocity ω_t (yaw rate) is obtained from the IMU. The IMU provides angular velocity measurements in the body frame, which are used directly in the motion model.

1) Data Synchronization: The encoder and IMU data are not synchronized, as they are sampled at different rates. To synchronize the data, the following steps are performed:

1. Interpolation: The IMU angular velocity data is interpolated to match the timestamps of the encoder data. Linear interpolation is used:

$$\omega_t = \text{interp}(t, t_{\text{imu}}, \omega_{\text{imu}}),$$

where t_{imu} and ω_{imu} are the IMU timestamps and angular velocity measurements, respectively.

2. Timestamp Alignment: The LiDAR data is aligned with the encoder and IMU data by finding the closest encoder timestamp for each LiDAR scan:

$$\text{index} = \arg\min_i |t_{\text{lidar}} - t_{\text{encoder}}[i]|.$$

C. Iterative Closest Point (ICP) Algorithm

The ICP algorithm is used to refine the robot's trajectory by aligning consecutive LiDAR scans. Given two point clouds S_t and S_{t+1}, the goal is to estimate the relative transformation ${}^tT_{t+1}$ that minimizes the alignment error:

$${}^tT_{t+1} = \arg\min_T \sum_{i=1}^{N} \|p_i - T \cdot q_i\|^2,$$

where:

- $p_i \in S_t$ and $q_i \in S_{t+1}$ are corresponding points,
- T is the transformation matrix representing rotation and translation.

The ICP algorithm iteratively refines the transformation by alternating between finding point correspondences and solving for the optimal transformation. The process involves the following steps:

1) Nearest Neighbor Search: For each point in the source cloud S_t, the closest point in the target cloud S_{t+1} is found using a nearest neighbor search. This step establishes correspondences between the two point clouds.

2) Kabsch Algorithm: The optimal rotation R and translation p that align the source cloud to the target cloud are computed using the Kabsch algorithm. The Kabsch algorithm solves the following optimization problem:

$$R, p = \arg\min_{R,p} \sum_{i=1}^{N} \|(R \cdot p_i + p) - q_i\|^2.$$

3) Derivation of the Kabsch Algorithm: The Kabsch algorithm minimizes the sum of squared errors between the source and target point clouds. Let $P = \{p_i\}$ and $Q = \{q_i\}$ be the source and target point clouds, respectively. The centroids of the point clouds are computed as:

$$\bar{p} = \frac{1}{N}\sum_{i=1}^{N} p_i, \quad \bar{q} = \frac{1}{N}\sum_{i=1}^{N} q_i.$$

The centered point clouds are defined as:

$$p_i' = p_i - \bar{p}, \quad q_i' = q_i - \bar{q}.$$

The cross-covariance matrix H is computed as:

$$H = \sum_{i=1}^{N} p_i' q_i'^{\top}.$$

The singular value decomposition (SVD) of H is performed:

$$H = U\Sigma V^{\top},$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values.

The optimal rotation matrix R is given by:

$$R = UV^{\top}.$$

The optimal translation vector p is computed as:

$$p = \bar{q} - R\bar{p}.$$

4) Transformation Update: The computed transformation is applied to the source cloud, and the process is repeated until convergence. The transformation T is updated as:

$$T = \begin{bmatrix} R & p \\ 0^{\top} & 1 \end{bmatrix}.$$

D. Scan Matching

The robot's trajectory is refined by aligning consecutive LiDAR scans using the ICP algorithm. The relative transformation between scans is computed and used to update the robot's pose. The process involves:

1) Initial Guess: The motion model provides an initial guess for the relative transformation between scans. This initial guess is used to initialize the ICP algorithm.

2) ICP Refinement: The ICP algorithm refines the transformation by minimizing the alignment error between the source and target point clouds. The refined transformation is used to update the robot's pose.

3) Trajectory Update: The robot's trajectory is updated by composing the relative transformations. The updated trajectory is used to construct a map of the environment.

E. Implementation Details

The implementation consists of the following key components:

1) Driver Class: The `Driver` class handles sensor data processing, including encoder counts, IMU measurements, and LiDAR scans. It computes the robot's linear and angular velocities and prepares the LiDAR data for scan matching.

2) Motion Model: The motion model computes the robot's pose using the differential-drive kinematic model. It integrates the linear and angular velocities to predict the robot's trajectory.

3) ICP Algorithm: The ICP algorithm aligns LiDAR scans using the Kabsch algorithm for rotation and translation estimation. It iteratively refines the transformation between consecutive scans.

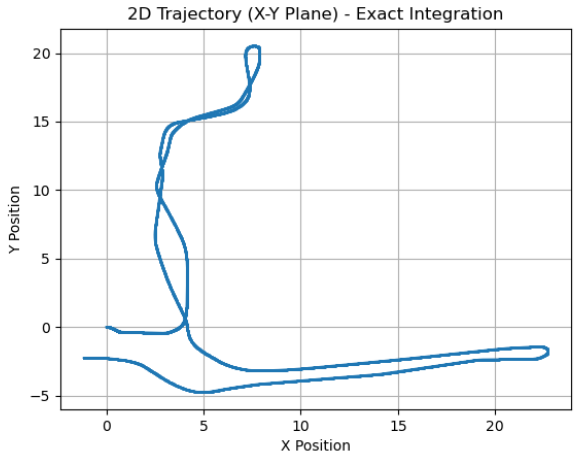
4) Scan Matching: The scan matching process refines the robot's trajectory by iteratively applying ICP to consecutive LiDAR scans. The refined trajectory is used to construct a map of the environment.

V. Results

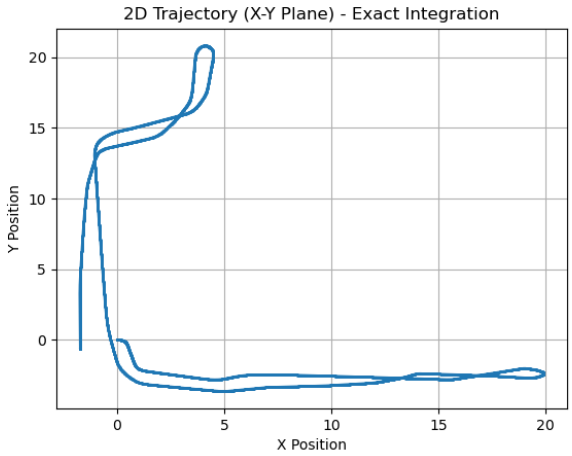
A. Motion Model (Initial Trajectory)

The initial trajectory of the robot was estimated using the differential-drive kinematic model, which integrates encoder and IMU data. The motion model computes the robot's pose (position and orientation) over time based on the linear velocity v_t (derived from encoder counts) and angular velocity ω_t (derived from IMU measurements).

1) Dataset 20: The motion model was applied to Dataset 20 to estimate the robot's trajectory. The figures below show the estimated trajectory,. The results demonstrate that the motion model provides a reasonable initial estimate of the robot's trajectory, but drift is observed over time due to cumulative errors in the encoder and IMU measurements.



2) Dataset 21: The motion model was also applied to Dataset 21. The figures below show the estimated trajectory for this dataset. Similar to Dataset 20, the motion model provides a good initial estimate, but the trajectory exhibits significant drift in regions with rapid motion or sensor noise.



B. ICP Testing (Drill and Liquid Container)

The ICP algorithm was tested on depth images of a drill and a liquid container to evaluate its ability to align point clouds. The goal was to estimate the relative transformation between two point clouds and refine the alignment iteratively.

1) Drill Alignment: The ICP algorithm was applied to align point clouds of a drill captured from different viewing angles. The figures below show the alignment process at different iterations. The algorithm successfully aligned the point clouds, reducing the alignment error and producing a visually coherent result.

2) Liquid Container Alignment: The ICP algorithm was also tested on point clouds of a liquid container. The figures below show the alignment process at different iterations. The algorithm successfully aligned the point clouds, demonstrating its robustness to different object shapes and sizes.

C. Scan Matching with LiDAR Using ICP (Optimized Trajectory)

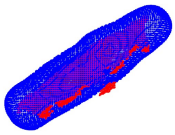
The ICP algorithm was used to refine the robot's trajectory by aligning consecutive LiDAR scans. The initial trajectory, estimated using the motion model, was used as an initial guess for the ICP algorithm.

1) Dataset 20: The figures below show the optimized trajectory for Dataset 20. The ICP algorithm reduced the drift in the initial trajectory, particularly in regions with rapid motion or sensor noise. It seems though that it added more drift as a result of a badly written ICP algorithm.

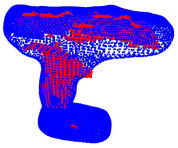
2) Dataset 21: Could not get this optimization to work due to indexing errors.

D. Discussion

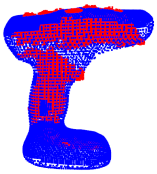
The results introduce the effectiveness of the motion model and ICP algorithm in estimating and refining the robot's trajectory. The motion model provides a good initial estimate, but drift is observed over time due to cumulative errors in the encoder and IMU measurements. The ICP algorithm although implemented incorrectly, conceptually reduces this drift, producing a more accurate and consistent trajectory.



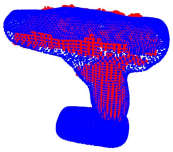
(a) Top View



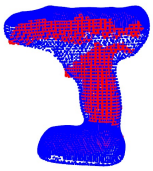
(b) Right 45° View



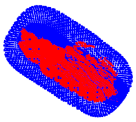
(c) Right Side View



(d) Left 45° View

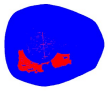


(e) Left Side View

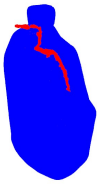


(f) Bottom View

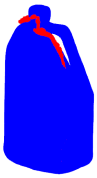
Fig. 1: ICP Alignment of Drill Point Clouds



(a) Top View

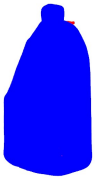


(b) Right 45° View

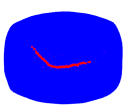


(c) Right Side View

(d) Left 45° View



(e) Left Side View



(f) Bottom View

Fig. 2: ICP Alignment of Liquid Container Point Clouds

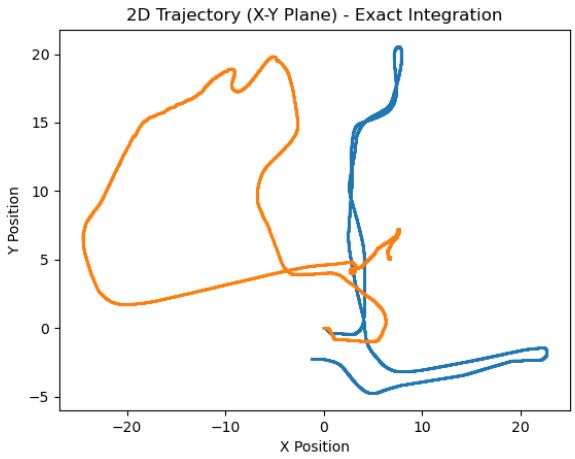


Fig. 3: Dataset 20: Optimized Trajectory Using ICP

However, some limitations were observed. The ICP algorithm occasionally struggled to align point clouds in regions with sparse or noisy data. Future work could explore the use of additional sensors or advanced filtering techniques to improve the robustness of the algorithm in such scenarios.

E. Conclusion

The motion model and ICP algorithm provide a framework for estimating and refining the robot's trajectory using encoder, IMU, and LiDAR data. The project demonstrate the algorithm's ability to achieve accurate and efficient trajectory estimation, even in the absence of ground truth data.

ACKNOWLEDGMENT

I would like to thank the University of California, San Diego (UCSD) Electrical and Computer Engineering (ECE) Department and the teaching staff of **ECE 276A: Sensing & Estimation in Robotics** for their guidance and support.

Instructor: Nikolay Atanasov (natanasov@ucsd.edu)
Teaching Assistants: Yinzhuang Yi (yiyi@ucsd.edu) Jason Stanley (jtstanle@ucsd.edu) Jay Paek (jpaek@ucsd.edu)

REFERENCES

[1] IEEE, "IEEE conference templates," [Online]. Available: https://www.ieee.org/conferences_events/conferences/publishing/templates.html.
[2] N. Atanasov, "ECE 276A Course Introduction," [Online]. Available: https://natanaso.github.io/ece276a/ref/ECE276A_1_Introduction.pdf.
[3] N. Atanasov, "Motion and Observation Models," [Online]. Available: https://natanaso.github.io/ece276a/ref/ECE276A_4_MotionAndObservationModels.pdf.
[4] N. Atanasov, "Factor Graph SLAM," [Online]. Available: https://natanaso.github.io/ece276a/ref/ECE276A_5_FactorGraphSLAM.pdf.
[5] N. Atanasov, "Localization and Odometry," [Online]. Available: https://natanaso.github.io/ece276a/ref/ECE276A_6_LocalizationOdometry.pdf.
[6] M. Brett, "transforms3d: Python library for 3D transformations," [Online]. Available: https://matthew-brett.github.io/transforms3d/..
[7] GTSAM, "Georgia Tech Smoothing and Mapping (GTSAM) Library," [Online]. Available: https://gtsam.org/.
[8] Open3D, "Open3D: A Modern Library for 3D Data Processing," [Online]. Available: http://www.open3d.org/.
[9] NumPy, "NumPy: Fundamental package for scientific computing with Python," [Online]. Available: https://numpy.org/.