Pose-Based EKF SLAM using ICP Scan-Matching on a Kobuki Turtlebot

Joseph Adeola, *Udg*, Moses Ebere, *Udg*, and Preeti Verma, *Udg*

Abstract— This paper presents a Pose-Based EKF SLAM (Simultaneous Localization and Mapping) approach using the Iterative Closest Point (ICP) algorithm. The research focuses on the application of scan-matching techniques on a Kobuki Turtlebot platform. The proposed algorithm combines sensor data from a 2D lidar, IMU, and wheel encoders to perform real-time mapping and localization. Experiments were conducted in both simulation and real-world environments to evaluate the performance and effectiveness of the algorithm. While the results in the real-world experiments did not meet expectations, the simulation experiments demonstrated excellent mapping and localization accuracy. Additionally, a novel pose deletion approach called the MJP approach was proposed to manage the size of the state vector while preserving crucial pose history information.

Index Terms—Pose-Based SLAM, Iterative Closest Point (ICP), Scan-matching, EKF SLAM

1 Introduction

Simultaneous Localization and Mapping (SLAM) plays a vital role in robotics by addressing the problem of estimating a robot's pose while simultaneously constructing a map of its environment. SLAM algorithms have widespread applications in autonomous navigation, robotic exploration, and augmented reality. This research paper focuses on the implementation and evaluation of an Extended Kalman Filter (EKF) SLAM algorithm using the Iterative Closest Point (ICP) method for scan-matching.

The goal of this research is to investigate the effectiveness of the proposed EKF SLAM algorithm in both simulated and real-world environments. The algorithm is implemented on a Kobuki Turtlebot, a popular mobile robot platform equipped with 2D lidar, IMU, and wheel encoders. Simulation experiments provide controlled conditions for evaluating the algorithm's behavior, while real-world experiments offer insights into its practicality and performance in dynamic settings.

To address the challenge of managing the state vector in the SLAM algorithm, a novel approach called the MJP pose deletion algorithm is introduced. This approach aims to remove redundant poses from the state vector while preserving sufficient information for loop closure, thereby enhancing the efficiency of the SLAM algorithm.

The structure of this paper is as follows: Section 2 provides a brief overview of related work in SLAM and highlights the significance of this research. Section 3 outlines the methodology, including the system setup, data acquisition process, and implementation details of the EKF SLAM algorithm and the MJP pose deletion approach. Section 4 presents the experimental setup, encompassing both simulated and real-world experiments. Section 5 presents the obtained results. Finally, Section 6 concludes the paper by summarizing the key findings and discussing potential avenues for future research.

• *Joseph Adeola, Moses Ebere, and Preeti Verma are Erasmus Mundus Master's students in Intelligent Field Robotic Systems at the University of Girona, Spain.*
E-mail: adeolajosepholoruntoba@gmail.com, moseschukaebere@gmail.com, preeti8600verma@gmail.com

Manuscript submitted on June 10, 2023.

2 Related Works

Pose-Based EKF SLAM with ICP scan-matching has undergone extensive research, focusing on algorithms that combine EKF pose estimation and ICP techniques. The algorithm introduced by Mallios et al. [2] utilizes probabilistic scan matching, range scans, and dead-reckoning displacements to solve the underwater SLAM problem using an AUV. Palomer et al. [3] propose a probabilistic surface matching method for bathymetry SLAM, which combines seafloor profiles, dead reckoning, and a flexible two-stage ICP process. Real data demonstrates the effectiveness of this algorithm.

In 2014, an algorithm was presented by Mallios et al. [4] that estimates the local path, corrects distortions, and cross-registers scans using probabilistic scan matching. Wei et al. [5] integrate tightly-coupled SLAM and robust absolute positioning to enhance accuracy by incorporating a spare reference trajectory frame for posture correction. Ferreira et al. [6] tackle the challenge of 3D underwater sonar scan registration by employing a probabilistic scan matching method that processes raw data and estimates individual scan uncertainty. The approach demonstrates superior robustness and accuracy, effectively reducing dead reckoning drift.

2.1 The Map

In Pose-Based EKF SLAM, the map is represented as a scan history, denoted as S_H, which is an array of scans

$$S_H = \begin{bmatrix} {}^{B_1}S_{B_1} & {}^{B_2}S_{B_2} & \dots & {}^{B_k}S_{B_k} \end{bmatrix} \tag{1}$$

Each scan, ${}^{B_k}S_{B_k}$, within the scan history consists of a collection of points:

$${}^{B_k}S_{B_k} = \begin{bmatrix} {}^{B_k}S_1 & {}^{B_k}S_2 & \dots & {}^{B_k}S_N \end{bmatrix}; \forall, {}^{B_k}S_i \in R^2 \tag{2}$$

These points, ${}^{B_k}S_i$, are associated with the viewpoint pose ${}^{B_k}S_{B_k}$, indicating the specific location from which they were gathered.

2.2 SLAM state vector

The SLAM state vector is a central component of the proposed pose-based SLAM algorithm. This vector ${}^{N}x_k$ is modeled as a Gaussian Random Vector of the robot's pose as well as the associated poses from which each scan is acquired in the North East Down, N-frame used in underwater robotics. Since the filer jointly estimates all the view poses, the structure and elements of the SLAM state vector is defined as follows:

$${}^{N}x_k = \begin{bmatrix} {}^{N}x_{B_1} \\ {}^{N}x_{B_2} \\ \vdots \\ {}^{N}x_{B_k} \end{bmatrix}; \quad {}^{N}x_k = \mathcal{N}({}^{N}\hat{x}_k, {}^{N}P_k) \tag{3}$$

where

$${}^{N}\hat{x}_k = \begin{bmatrix} {}^{N}\hat{x}_{B_1} \\ {}^{N}\hat{x}_{B_2} \\ \vdots \\ {}^{N}\hat{x}_{B_k} \end{bmatrix}; \quad {}^{N}P_k = \begin{bmatrix} {}^{N}P_{B_1} & {}^{N}P_{B_1B_2} & \dots & {}^{N}P_{B_1B_k} \\ {}^{N}P_{B_2B_1} & {}^{N}P_{B_2} & \dots & {}^{N}P_{B_2B_k} \\ \vdots & \vdots & \ddots & \vdots \\ {}^{N}P_{B_kB_1} & {}^{N}P_{B_1B_2} & \dots & {}^{N}P_{B_k} \end{bmatrix} \tag{4}$$

Compounding the scan points with their viewpoints is necessary to render the map.

$${}^{N}s_i = {}^{N}x_{B_i} \oplus_i^B S_i \tag{5}$$

to represent them in the N-frame. The details regarding the representation of the robot pose can be found in subsections **??**

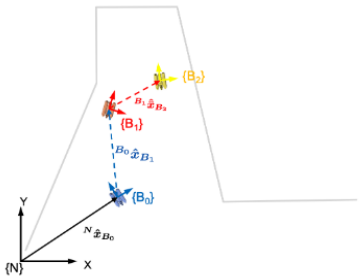


Fig. 1. Dead reckoning trajectory (src: lecture slides - Pere Ridao)

2.2.1 Robot Pose

The robot's pose is represented by its position and yaw orientation in the world $N - frame$. The pose is typically denoted as

$${}^{N}x_{B_k} = \begin{bmatrix} {}^{N}x_k \\ {}^{N}y_k \\ {}^{N}\theta_k \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} {}^{N}\hat{x}_k \\ {}^{N}\hat{y}_k \\ {}^{N}\hat{\theta}_k \end{bmatrix}, {}^{N}P_k \right) \tag{6}$$

where ${}^{N}x_k$ and ${}^{N}y_k$ represent the robot's position, and ${}^{N}\theta_k$ denotes the orientation. Figure 1 shows the different robot poses over time.

2.3 Motion Model

The motion model for the Turtlebot can be represented as a differential drive system, where the robot's position and orientation are updated based on the linear and angular velocities of the robot. The encoder measurements provide information about the robot's angular speed in each wheel, which is used to estimate the robot's linear and angular velocities.

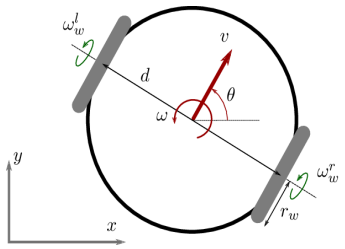


Fig. 2. Differential Drive System

As the robot moves, there's a need to always predict its next state and this is given by the function

$${}^{N}x_{B_k} = f({}^{N}x_{B_{k-1}}, u_k, w_k) \tag{7}$$

where ${}^{N}x_{B_k}$ is the robot's next state, ${}^{N}x_{B_{k-1}}$ the previous state, u_k the control input and w_k, the noise in motion. In this case, the control input is considered to be the left wheel angular velocity ω_w^l and right wheel angular velocities, ω_w^r. Given that the encoder measurements are the primary source of uncertainty in the robot's motion, it is reasonable to assume that the process noise comes from the left and right wheel angular velocities.ie, w_k comprise of $w_l = \sigma_l^2$, the noise in the left wheel and $w_r = \sigma_r^2$, the noise in the right wheel. We assumed $\sigma_l = \sigma_r = 0.01$. Thus for a little change in time Δt, we can derive

$${}^{N}x_{B_k} = {}^{N}x_{B_{k-1}} \oplus (u_k + w_k) \tag{8}$$

$$= \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{r_w \Delta t \cos\theta_{k-1}}{2} & \frac{r_w \Delta t \cos\theta_{k-1}}{2} \\ \frac{r_w \Delta t \sin\theta_{k-1}}{2} & \frac{r_w \Delta t \sin\theta_{k-1}}{2} \\ \frac{r_w \Delta t}{d} & -\frac{r_w \Delta t}{d} \end{bmatrix} \begin{bmatrix} \omega_w^l + w_l \\ \omega_w^r + w_r \end{bmatrix} \tag{9}$$

where

$$w_k \sim \mathcal{N}(0, Q_k); \quad Q_k = \begin{bmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix} = \begin{bmatrix} 0.01^2 & 0 \\ 0 & 0.01^2 \end{bmatrix} \tag{10}$$

During motion, only the robot's pose in the state vector is affected, hence, the motion model $f_p()$ is:

$$f_p({}^N x_{k-1}, u_k, w_k) = \begin{bmatrix} {}^N x_{B_0} \\ \vdots \\ {}^N x_{B_{k-2}} \\ f({}^N x_{B_{k-1}}, u_k, w_k) \end{bmatrix} \tag{11}$$

To apply the Extended Kalman Filter (EKF) to the non-linear motion model, it is imperative to compute the Jacobian of the motion model with respect to the state vector and noise. Matrices F_{1_k} and F_{2_k} in equations 12 & 14 are the Jacobians that are required for the prediction step. F_{1_k} is the Jacobian of the motion model f_p with respect to the previous state and F_{2_k} is the Jacobian of the motion model f_p with respect to the noise. These linearization Jacobians are required for us to convert the non-linear process model to a linear equation.

$$F_{1_k} = \frac{\partial f_p({}^N x_{k-1}, u_k, w_k)}{\partial {}^N x_{k-1}} = \begin{bmatrix} I & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I & 0 \\ 0 & 0 & \dots & 0 & J_{f_x} \end{bmatrix} \tag{12}$$

where

$$J_{f_x} = \frac{\partial f({}^N x_{k-1}, u_k, w_k)}{\partial {}^N x_{B_{k-1}}} = \begin{bmatrix} 1 & 0 & -V\Delta t \sin {}^N\theta_{B_{k-1}} \\ 0 & 1 & V\Delta t \cos {}^N\theta_{B_{k-1}} \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

where V is the linear velocity of the robot. Also,

$$F_{2_k} = \frac{\partial f_p({}^N x_{k-1}, u_k, w_k)}{\partial w_k} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ J_{f_w} \end{bmatrix} \tag{14}$$

where

$$J_{f_w} = \frac{\partial f({}^N x_{B_{k-1}}, u_k, w_k)}{\partial w_k} = \begin{bmatrix} \frac{r_w \Delta t \cos\theta_{k-1}}{2} & \frac{r_w \Delta t \cos\theta_{k-1}}{2} \\ \frac{r_w \Delta t \sin\theta_{k-1}}{2} & \frac{r_w \Delta t \sin\theta_{k-1}}{2} \\ \frac{r_w \Delta t}{d} & -\frac{r_w \Delta t}{d} \end{bmatrix} \tag{15}$$

The derivation of F_{1_k}, J_{f_x}, F_{2_k} and J_{f_w} appendix section.

2.4 Observation Model

We considered two types of observations: displacement observation between two scan poses where lidar scan was taken and IMU sensor observation. The displacement observation is obtained by applying the ICP registration algorithm, which estimates the (x, y, θ) displacement necessary to align two scans in the world frame. These displacement measurements play a critical role in accurately estimating the robot's pose and constructing the map. Additionally, the IMU sensor serves as an absolute sensor, providing observations used to update the robot's heading.

In this particular case, we focused on pose-based SLAM and do not utilize feature observations. Therefore, the ICP displacement measurement, along with its associated observation model outlined in sections 1.3.1 and 1.3.2, is incorporated into the SLAM algorithm. The observation model for the IMU sensor can be found in section 2.4.2 of this paper.

2.4.1 Scan Displacement Observation Model

This observation model is based on estimating the displacement between two viewpoint poses of the robot (i.e., points from which scans were taken) as shown in figure 3

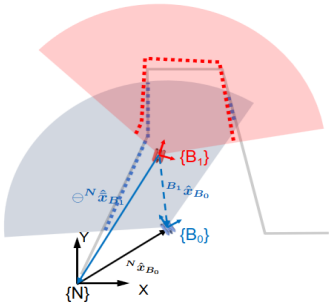


Fig. 3. Two scan viewposes and the displacement between them (src: lecture slides - Pere Ridao)

To estimate the displacement between the current viewpose ${}^N x_{B_i}$ and a previous viewpose ${}^N x_{B_j}$, the observation model is given by given by $h()$:

$$z_k = h({}^N x_{B_i}, {}^N x_{B_j}, v_j) = ((\ominus {}^N x_{B_i}) \oplus {}^N x_{B_j}) + v_j \tag{16}$$

$$h({}^N x_{B_i}, {}^N x_{B_j}, v_j) = \begin{bmatrix} ({}^N x_{B_j} - {}^N x_{B_i}) \cdot \cos({}^N\theta_{B_i}) + ({}^N y_{B_j} - {}^N y_{B_i}) \cdot \sin({}^N\theta_{B_i}) \\ -({}^N x_{B_j} - {}^N x_{B_i}) \cdot \sin({}^N\theta_{B_i}) + ({}^N y_{B_j} - {}^N y_{B_i}) \cdot \cos({}^N\theta_{B_i}) \\ -{}^N\theta_{B_i} + {}^N\theta_{B_j} \end{bmatrix} + \begin{bmatrix} v_{j_x} \\ v_{j_y} \\ v_{j_\theta} \end{bmatrix} \tag{17}$$

where $\ominus$ is the pose inversion operation, $\oplus$ is the pose compounding operation, ${}^N x_{B_i}$ is the robot's current viewpoint pose, ${}^N x_{B_j}$ is the robot's viewpoint pose at the jth

scan which has some overlap with the current scan, and v_j accounts for the observation noise. The operations $\ominus$ and $\oplus$ are defined in appendix 2. The Jacobian of the observation equation with respect to the SLAM state vector is given by:

$$H_k = \frac{\partial h({}^N x_{B_i}, {}^N x_{B_j}, v_j)}{\partial {}^N x_k}|_{{}^N x_k = {}^N \hat{x}_k, v_k = 0} \tag{18}$$

$$= \begin{bmatrix} \frac{\partial h}{\partial x_{B_1}} & \frac{\partial h}{\partial x_{B_2}} & \cdots & \frac{\partial h}{\partial x_{B_k}} \end{bmatrix} \tag{19}$$

The matrix H_k has a dimension $3 \times n$, where n is the number of poses in the state vector. The Jacobian $h()$ with respect to the observation noise is:

$$V_k = \frac{\partial h({}^N x_{B_i}, {}^N x_{B_j}, v_j)}{\partial v_j} = \begin{bmatrix} \frac{\partial h_x}{\partial v_{j_x}} & \frac{\partial h_x}{\partial v_{j_y}} & \frac{\partial h_x}{\partial v_{j_\theta}} \\ \frac{\partial h_y}{\partial v_{j_x}} & \frac{\partial h_y}{\partial v_{j_y}} & \frac{\partial h_y}{\partial v_{j_\theta}} \\ \frac{\partial h_\theta}{\partial v_{j_x}} & \frac{\partial h_\theta}{\partial v_{j_y}} & \frac{\partial h_\theta}{\partial v_{j_\theta}} \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I \tag{21}$$

2.4.2 IMU Observation Model

The IMU mounted on the robot measures the absolute heading of the robot with respect to the north. Thus we used a linear Kalman Filter to model the heading observation equation.

$$z_k = H_k^N x_k + V_k \tag{22}$$

$$z_\theta = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} {}^N\hat{x}_{B_1} \\ {}^N\hat{y}_{B_1} \\ {}^N\hat{\theta}_{B_1} \\ \vdots \\ {}^N\hat{\theta}_{B_k} \end{bmatrix} + V_k \tag{23}$$

Since the IMU sensor measurements are very reliable, we assumed a very low value for the measurement noise V_k and uncertainty in the measurement R_k

$$V_k = \begin{bmatrix} 0.0001 \end{bmatrix}, R_k = \begin{bmatrix} 0.00001 \end{bmatrix} \tag{24}$$

3 METHODOLOGY

The methodology section of this paper outlines the step-by-step process we followed to implement the SLAM mathematical models explained in subsections 2.3. 2.4.1 & 2.4.2 in Python programming language. The primary goal of this section is to provide a detailed description of the approach used to solve the problem at hand, including any data processing, algorithms, and techniques used during the development process as represented in the algorithm workflow design in figure 4.

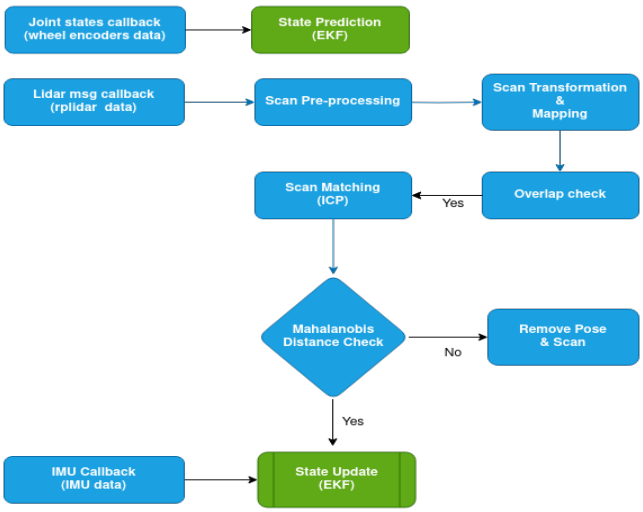


Fig. 4. Algorithm implementation design

3.1 Sensor Fusion

To process the measurements from each sensor, we utilized Robot Operating System (ROS) and implemented parallel processing. Each sensor's measurements were individually processed using a callback function. This allowed us to independently apply the linear Kalman Filter (KF) for the IMU and the Extended Kalman Filter (EKF) for lidar scan-matching.

The subsequent subsections will discuss the steps depicted in Figure 4 in detail.

3.2 Prediction

During robot motion, the angular velocity data of the left and right wheels, obtained from the wheel encoders, is utilized in the EKF prediction step. This step applies the dead-reckoning motion model, as represented by equation 11, to estimate the new pose of the robot. The prediction step not only affects the robot pose (the last element of the state vector) but also impacts the last element of the diagonal and the last row and column of the covariance matrix. These changes influence both the robot's covariance and its cross-covariance with the scan viewpoints stored in the state vector.

3.3 Scan Registration (Scan Preprocessing)

Due to the high rate at which the rplidar sensor gathers environment scans, not every received scan message is used to update the state vector. Instead, a scan is registered only if the robot has traveled a certain distance. Before registration, the scan is converted into point clouds and added to the map of scan history S_H, while the corresponding robot pose at the time of the scan is cloned.

3.4 Scan Transformation

Before checking for overlap, all points in the two scans are transformed into the world frame using the pose at which the scan was taken and the transformation between the lidar and robot frames, as shown in equation 25.

$$ {}^{N}x_F = {}^{N}T_B \cdot {}^{B}T_L \cdot {}^{L}x_F \tag{25} $$

where

$$ {}^{N}T_B = \begin{bmatrix} cos({}^{N}\theta_{B_k}) & -sin({}^{N}\theta_{B_k}) & 0 & {}^{N}x_{B_k} \\ sin({}^{N}\theta_{B_k}) & cos({}^{N}\theta_{B_k}) & 0 & {}^{N}y_{B_k} \\ 0 & 0 & 1 & -0.19 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{26} $$

$$ {}^{B}T_L = \begin{bmatrix} 1 & 0 & 0 & -0.02 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.179 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27} $$

Here, ${}^{N}T_B$ represents the transformation matrix between the robot base frame and the world frame, while ${}^{B}T_L$ represents the transformation matrix between the lidar frame and the robot base frame. Additionally, ${}^{L}x_F = ({}^{S}x_F, {}^{S}y_F, {}^{S}z_F, 1)$ denotes the coordinates of a point cloud feature in the lidar frame. Notably, ${}^{S}z_F = 1$ since lidar measurements are in the (x, y) coordinate system.

3.5 Data Association (Overlap Check)

The data association step involves identifying which stored scans from the scan history S_H, overlap with the most recent scan. We employed two distinct methods for detecting overlaps: the k-nearest neighbor (KNN) approach and the comparison of overlaps through polygon intersection. The details of these two approaches are elaborated in subsections 3.5.1 and 3.5.2, respectively.

3.5.1 K-Nearest Neighbour

The KNN algorithm was used to assess the overlap between two sets of scans in our study. This algorithm allowed us to compare each point in the current scan with its nearest neighbor in the reference scan.

To accomplish this, a distance threshold was defined. Each point in the current scan was then examined to find its nearest neighbor in the reference scan using the KNN algorithm. By measuring the distance between the point and its nearest neighbor, we could determine if there was an overlap between the two sets of scans before passing them to ICP algorithm to estimate the transformation needed to align the two scans. Figure 5 shows two scans obtained using KNN.

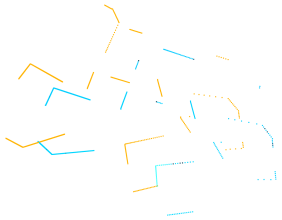


Fig. 5. Two overlapping using KNN approach

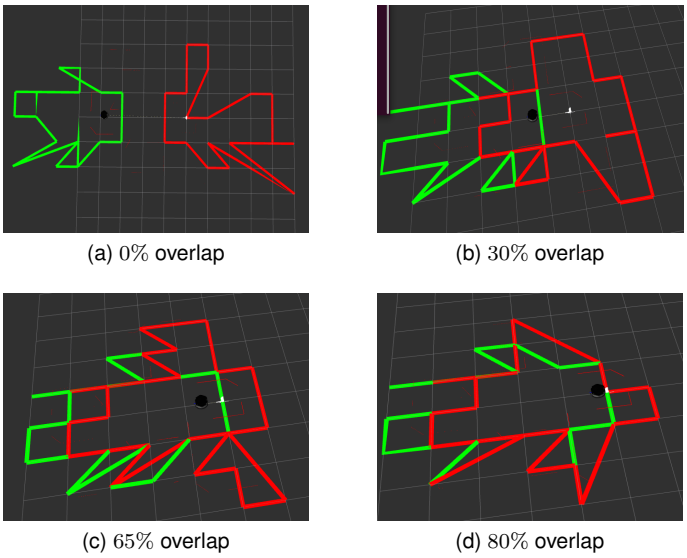
However, it is important to acknowledge a limitation of this approach. The KNN-based overlap detection method relies heavily on the chosen distance threshold. While a fixed threshold can be effective in some scenarios, it may not account for variations in data density or noise levels, potentially leading to false positives or missed overlaps. Hence, the reason why we tried other methods of estimating overlaps.

3.5.2 Polygon overlap percentage estimation

In our implementation, we employed the shapely geometry library in Python to estimate the percentage of overlap between polygons created from the points in each scan. The process involved using the shapely library to construct a polygon encompassing all the points within a given scan. Subsequently, for each scan in the scan history, we compared the polygon of the current scan with the polygon of the reference scan to determine the percentage of overlap. Our implementation considered an overlap percentage threshold of 60%, which indicates that if the intersection area of the two polygons exceeded this threshold, it was assumed that the scans exhibited overlap. This choice was deliberate as we aimed to focus on utilizing the most recent scans in updating the robot's pose while discarding older scans. By setting this threshold, we effectively limited the consideration of overlap to a subset of the scan history. Consequently, the two scans were passed to the iterative closest point (ICP) algorithm for registration.

To provide a visual representation of the estimated overlap percentages, Figure 6 presents several scans alongside their respective calculated overlap percentages using the shapely library.

The advantage of using polygon overlap estimation over K-nearest neighbor (KNN) is that it provides a more direct and geometrically accurate measure of overlap. By constructing polygons from scan points, the overlap estimation considers the spatial extent of the scans, capturing the actual area of overlap. In contrast, KNN relies on a fixed distance threshold, which may not accurately reflect the true overlap between scans.



(a) 0% overlap (b) 30% overlap

(c) 65% overlap (d) 80% overlap

Fig. 6. Comparing the trajectory obtained EKF rplidar only trajectory with ground truth trajectory

3.6 ICP Registration

The ICP (Iterative Closest Point) registration process is a crucial step in the implementation of the scan-matching SLAM algorithm. Its main purpose is to align the most recent scan, referred to as the source point cloud, with the overlapping reference scan, known as the target point cloud, within the map. This alignment is achieved by determining the optimal transformation matrix that effectively transforms the source point cloud to closely match the target point cloud. In our study, we explored two variants of the ICP algorithm: ICP Point-to-Point registration and Point-to-Plane registration. Both of these variants are available in Python's Open3D library.

3.6.1 ICP Point-to-Point Registration

The ICP point-to-point registration focuses on aligning points from the source point cloud to the corresponding points in the target point cloud. This variant minimizes the Euclidean distances between the matched points, resulting in a straightforward alignment based on point-to-point distances. The algorithm can be summarized as follows:

Input:

- Source point cloud: P
- Target point cloud: Q
- Initial transformation matrix: T_{initial}

Procedure:

1) Initialize the transformation matrix $T = T_{\text{initial}}$.
2) Repeat until convergence:
   - Find the corresponding points between P and Q based on the current transformation T.
   - Update the transformation T by minimizing the objective function $E(T) = \sum_{(p,q)\in K} \|p - Tq\|^2$, where K is the set of corresponding point pairs.
3) Return the final transformation matrix T.

3.6.2 ICP Point-to-Plane Registration

The second variant is the ICP point-to-point registration, which not only considers the positions of the points but also their surface normals. Instead of aligning points directly, this variant aligns the source points with the corresponding planes in the target point cloud. The ICP Point-to-Plane registration algorithm can be summarized as follows:

Input:

- Source point cloud: P
- Target point cloud: Q
- Initial transformation matrix: T_{initial}

Procedure:

1) Initialize the transformation matrix $T = T_{\text{initial}}$.
2) Repeat until convergence:
   - Find the corresponding points between P and Q based on the current transformation T.
   - Update the transformation T by minimizing the objective function $E(T) = \sum_{(p,q)\in K} \left((p - Tq) \cdot np\right)^2$, where K is the set of corresponding point pairs and np is the normal of point p.
3) Return the final transformation matrix T.

We conducted tests using the open3D library with and without an initial displacement guess estimate. For the ICP algorithm initial guess, we used the scan displacement observation model in equation 16 to estimate the displacement between the two poses. It is worth noting that the initial displacement guess is an optional parameter in the ICP function provided by the open3D library, allowing flexibility in the registration process. The results obtained from the different variants of the ICP registration are presented in the results section.

3.6.3 Uncertainty in ICP Registration

The ICP algorithm is not without its limitations and can introduce errors during the registration process. One challenge is the difficulty in finding precise correspondences between point clouds, which can result in inaccuracies in the estimated transformation. These uncertainties are akin to the uncertainty in measurements represented by the Rk matrix in an extended Kalman filter.

Due to time constraints, we were unable to implement an algorithm to estimate the uncertainty in the ICP. Consequently, we assumed a fixed value for the

measurement uncertainty, denoted as Rk.

$$R_k = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.02 \end{bmatrix} \tag{28}$$

The chosen Rk value reflects the level of trust placed in the measurements from the ICP. In this case, the heading measurement uncertainty (Rk value) is relatively low compared to other degrees of freedom. This decision was influenced by the presence of an IMU sensor, which provides more reliable heading information. As the IMU sensor publishes data at a higher rate than the rplidar sensor, we opted to place greater confidence in the IMU measurements for updating the robot's heading. Thus, any heading error estimate from the ICP would instantly be corrected by the IMU.
While acknowledging the limitations in our approach, the specific choice of Rk values was based on a pragmatic consideration of the available sensor data and the emphasis placed on the IMU's reliability in capturing the heading information.

3.7 Mahalanobis Distance Check

The EKF-SLAM update step can be performed with the measurements z_k from ICP and hk from the observation equation. However, inaccurate displacement measurements from ICP can negatively impact the accuracy of the estimated robot pose. To mitigate this, we introduced a Mahalanobis distance check as a quality control measure for the displacement estimate obtained from ICP.
In our case, we calculated the Mahalanobis distance between the ICP displacement measurement and the expected displacement distribution based on previous observations and the uncertainty of the measurement.

We have the following equations for data association:

$$v = z_k - h(x_{k|k-1}) \tag{29}$$

$$S = H_k P_{k|k-1} H_k^T + V_k R_k V_k^T \tag{30}$$

$$D^2 = v^T S^{-1} v \tag{31}$$

We take the smallest D value and compare it with the chi-square threshold, denoted as $\chi^2_{\rho,\phi}$. If $D < \chi^2_{\rho,\phi}$, the data association is considered valid. If the Mahalanobis distance exceeds the threshold, it suggests a potential outlier or error in the measurement. In such cases, we disregard the measurement and skip the EKF update step. This check helps ensure that only valid measurements are used in the SLAM algorithm. Equation (29) represents the innovation or measurement residual, Equation (30) calculates the innovation covariance matrix, and Equation (31) computes the Mahalanobis distance squared.

3.8 EKF Update

In the EKF update step, we incorporate valid measurements into the state estimation process to refine the state estimate and reduce uncertainty. This is achieved through the following equations:

$$K = P_{k|k-1} H^T S^{-1} \tag{32}$$

$${}^N x_k = {}^N x_{k|k-1} + Kv \tag{33}$$

$${}^N P_k = (I - K_k H_k) {}^N P_{k|k-1} (I - K_k H_k)^T \tag{34}$$

where K is the Kalman gain, ${}^N x_k$, the updated state estimate and ${}^N P_k$, the updated covariance matrix at time k.

3.9 Pose Decimation

In order to optimize the computational efficiency of our SLAM algorithm, we implemented a subsampling technique to reduce the number of scans considered during the mapping process. This approach involved registering a scan into the map only after a certain distance threshold was reached by the robot. Although the distance threshold condition helped limit the inclusion of scans in the mapping process, it alone was insufficient for bounding the size of the state vector. As longer runs or greater distances were covered, the dimensionality of the state vector would increase significantly. To address this challenge, we developed a novel algorithm called the Multi-Jump Pose (MJP) pose-decimation approach, which effectively constrained the size of the state vector and allowed for the deletion of poses.
The MJP pose-decimation approach bears similarities to the sliding window technique but with notable improvements. In the conventional sliding window approach, old poses are replaced by new poses in the state vector, leading to a loss of pose history as the robot continues its journey. In contrast, our approach, as described in the pseudocode provided in Algorithm 1, ensures a more refined process of deleting poses, thereby maintaining a more comprehensive pose history.

Algorithm 1 MJP Pose-Decimation

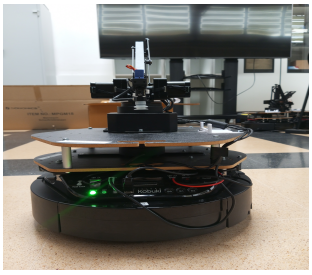
Require: $state\_vector$, max_size
Initialisation:
$index \leftarrow$ index of the second element in $state\_vector$
Loop Process:
while size of $state\_vector > max\_size$ **do**
  if $index =$ index of the last element in $state\_vector$ **then**
    $index \leftarrow$ index of the second element in $state\_vector$
  end if
  Remove element at index $index$ from $state\_vector$
  Remove the associated row and column at index $index$ from the covariance matrix
  $index \leftarrow index + 1$
end while

3.10 Joint Autonomous Task

The joint autonomous task simulation in our project focused on detecting an Aruco marker while the Kobuki Turtlebot2 robot autonomously explored its environment. The primary goal was to enable the robot to estimate the pose of the detected marker and navigate toward it using autonomous control. In addition to this task, we also incorporated the SLAM algorithm to perform simultaneous localization and mapping in the background while the robot carried out exploration, perception, and object-picking tasks. The results obtained from the integration of these components and tasks will be presented in the results section.

4 EXPERIMENT

We conducted experiments using the Kobuki TurtleBot3 robot, which served as the platform for implementing and evaluating the SLAM algorithm. The Kobuki TurtleBot3 is equipped with various components, including a Raspberry Pi microcontroller, a 2D lidar sensor for environment sensing, an IMU for orientation estimation, and wheel encoders for odometry. The robot is shown in figure 7.
To implement the SLAM algorithm, we leveraged the Robot Operating System (ROS) framework, which provided a robust platform for developing and integrating robotic functionalities. The SLAM algorithm was implemented using the Python programming language, enabling seamless communication and control between the different components of the robot.

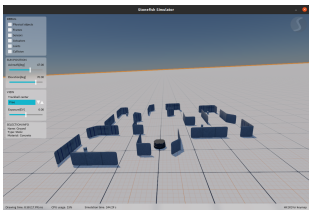


(a)

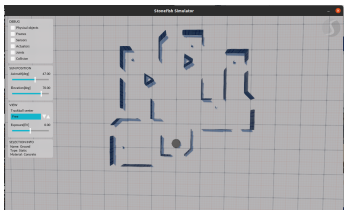
Fig. 7. Kobuki Turtlebot3

4.1 Experimental Procedure

During the experiments, the robot was deployed in both simulated and real-world environments to assess its performance and functionality across different scenarios. The Stonefish simulator was utilized in simulation experiments to replicate the robot's movement, physical environment, and sensor readings. Figure 8a& 8d shows the simulation environment while figure 8c & 8d shows the laboratory environment setup.



(a) Side View



(b) Top View

(c) Open Navigation Environment

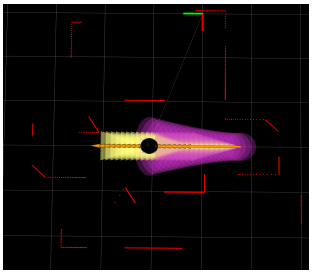
(d) Closed Navigation Environment

Fig. 8. Simulation and Laboratory environment setup

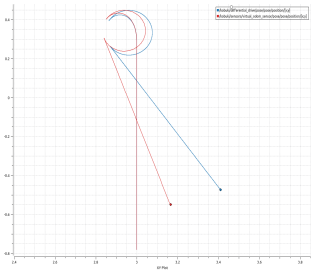
5 RESULTS & DISCUSSION

In this section, we present the findings and outcomes of our study on Pose-Based EKF SLAM using ICP scan-matching on a Kobuki Turtlebot. We provide a concise overview of the experimental results obtained from both simulation and real-world experiments. The presented results highlight the performance, accuracy, and **computational efficiency** of the implemented SLAM algorithm.

5.0.1 Simulation results of prediction only



(a) Increasing uncertainty ellipse

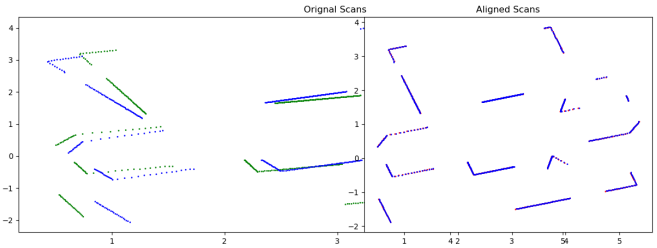


(b) Ground truth trajectory - red; robot's belief trajectory - blue

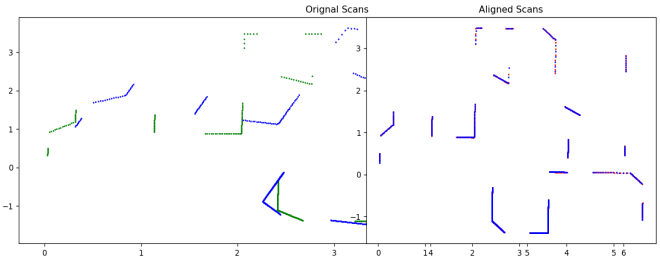
Fig. 9. (a) Prediction step of EKF showing increasing uncertainty ellipse, (b) Ground truth trajectory compared with prediction trajectory after some motion commands.

In figure 9a, the purple ellipse represents the uncertainty of the robot. It can be clearly seen how the uncertainty grows over time without shrinking because no update is being performed and dead-reckoning keeps accumulating errors as the robot travels more distance. While figure9b shows how the robot's belief trajectory () deviates from the ground truth trajectory (red) over some time.

5.0.2 *ICP results in simulation*



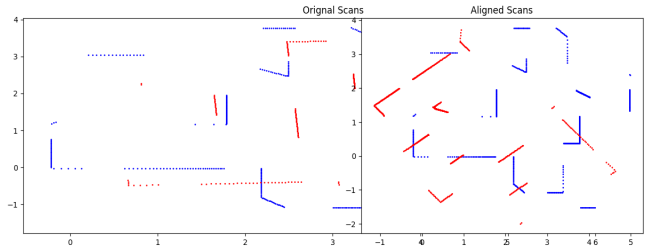
(a) Left - Before alignment; Right:After alignment - with initial guess estimate



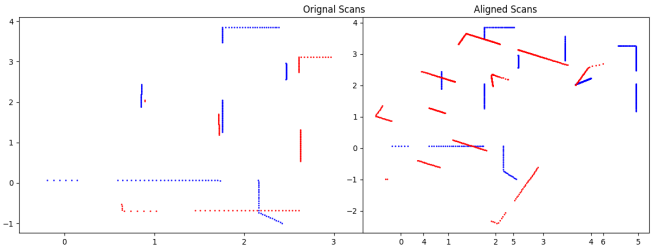
(b) [Left - Before alignment; Right:After alignment - without initial guess

Fig. 10. ICP point-to-point registration without initial guess in simulation

In Figure 10a and 10b, the results of ICP point-to-point registration using the open3D library are displayed. These figures demonstrate the successful alignment of two point clouds, both with and without the use of an initial guess estimate for the transformation between them. In the case of the initial guess estimate, the scan displacement observation model was employed to calculate an estimated displacement. Regardless of the transformation type (rotation, translation, or both) between the point clouds, the algorithm performed effectively. It is important to note that the images may not fully capture all parts of the point cloud due to the axis limits set during the plotting process.



(a) Left - Before alignment; Right:After alignment

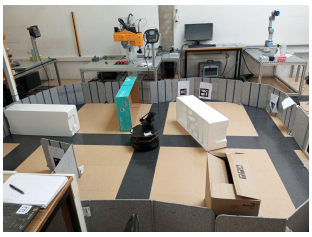


(b) [Left - Before alignment; Right:After alignment

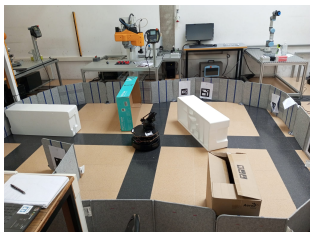
Fig. 11. ICP point-to-plane bad registration with initial guess in simulation

Unlike the ICP point-to-point, the results obtained from the same open3D library for ICP point-to-plane (figure 11a & 11b), were quite bad. We couldn't find the reason for this due to lack of time. Hence the reason why we chose ICP point-to-point without initial guess.

5.0.3 *Real-world scan-matching results*

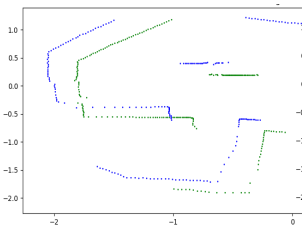


(a) Left - Before alignment;

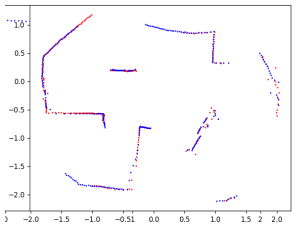


(b) Right:After alignment

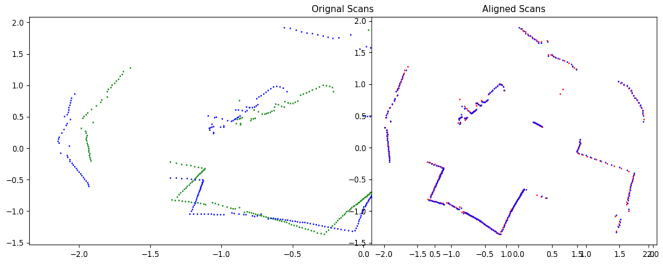
Fig. 12. Real navigation environment setup



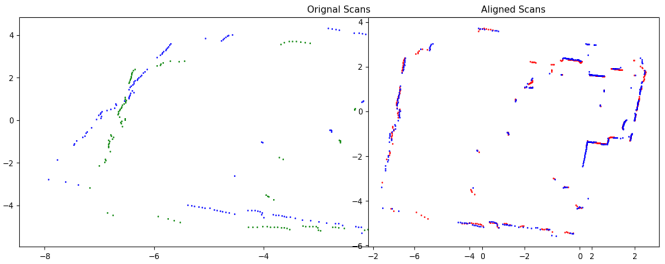
(a) Left - Before alignment; Right:After alignment



(b) Left - Before alignment; Right:After alignment



(c) Left - Before alignment; Right:After alignment

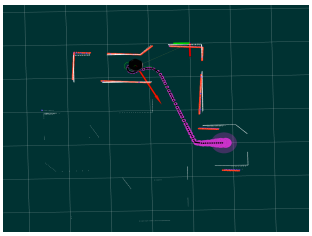


(d) [Left - Before alignment; Right:After alignment

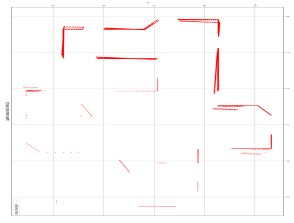
Fig. 13. ICP point-to-point registration without initial guess in a real-world environment

Figure 12a, 12b, 12c & 12d shows some of the results obtained from running the scan-matching test on the robot in a real-world environment

5.0.4 Simulation results of EKF update without IMU

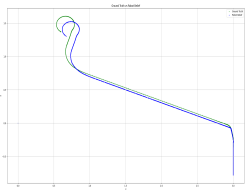


(a) Decreasing Uncertainty - Purple



(b) Map from aligned scans in real-time

Fig. 14.

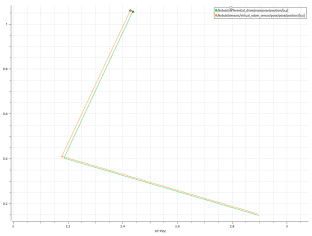


(a) Ground-truth trajectory - Green; Robot's belief trajectory - Blue

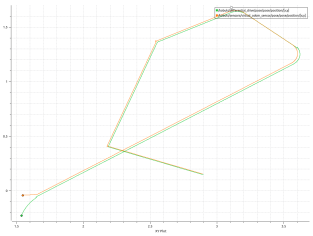
Fig. 15. Comparing the trajectory obtained EKF rplidar only trajectory with ground truth trajectory

During various simulation tests and experiments on the robot, we encountered instances of heading drift when relying solely on the rplidar sensor for pose updates over time. This observation was supported by Figure 14a, which displays the update of the filter and a reduction in uncertainty as time progresses. However, Figure 14b illustrates that the resulting map generated over time is misaligned. Furthermore, Figure 14c depicts a noticeable deviation in the robot's trajectory compared to the ground truth trajectory.

5.0.5 Trajectory of robot's belief and ground truth using rplidar and IMU



(a) Left - Before alignment; Right:After alignment

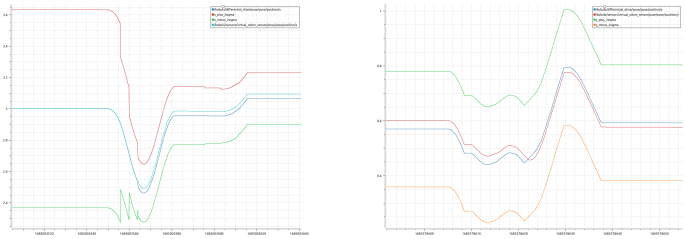


(b) Left - Before alignment; Right:After alignment

Fig. 16. EKF update trajectory of robot's belief and ground truth using IMU and rplidar measurements

Figure 16a & 16b shows the trajectory of the robot's belief in comparison with the ground truth trajectory after the IMU sensor measurement was incorporated into the filter to correct the heading.

5.0.6 Each DOF trajectory Error Estimate



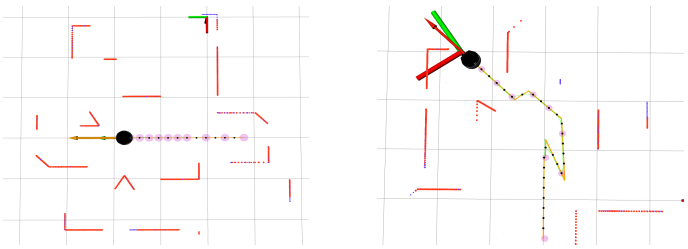
(a) x_belief +2σ(red); x_belief - 2σ(blue)

(b) y_belief +2σ(green); y_belief - 2σ(yellow).

Fig. 17. (a) x DOF belief within 2σ bound of uncertainty, (b) y DOF belief within 2σ bound of uncertainty.

As part of the accuracy test, we examined the 2σ bound of the robot's belief for each degree of freedom. In Figure 17a and 17b, we present the results obtained by comparing the ground truth values with the robot's belief within the 2σ bound threshold. This analysis provides insights into the accuracy of the robot's estimation for each degree of freedom.

5.0.7 MJP Pose-Decimation Results

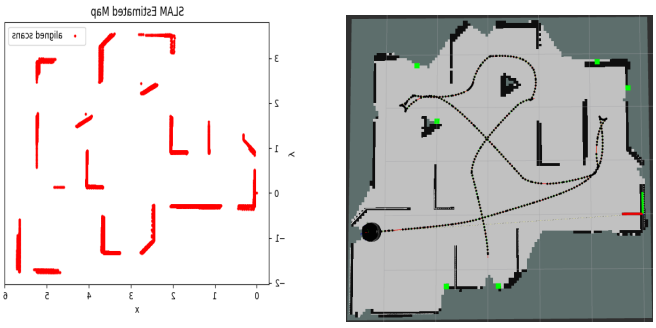


(a) (b)

Fig. 18. (a) History of cloned Poses wi, (b) Ground truth trajectory compared with prediction trajectory after some motion commands .

In Figure 18a and 18b, we illustrate the functionality of the MJP algorithm. Each black marker or dot represents a pose, and the covariance ellipses indicate the uncertainty associated with those poses. As the state vector grows and reaches its maximum capacity, the MJP algorithm selectively deletes poses based on their significance. In the figures, the poses that remain in the state vector are the ones for which covariance ellipses are drawn, while the poses without covariance ellipses have been deleted from the state vector to conserve memory and computational resources. By retaining the poses with associated covariance information, the MJP algorithm ensures that the SLAM system maintains essential historical pose data while effectively managing its size and complexity. In this particular implementation, we were only keeping 11 poses (including the robot's pose) in the state vector.

5.0.8 Comparing Map Obtained from SLAM and Occupancy Grid Map



(a) Map obtained from aligning all scans

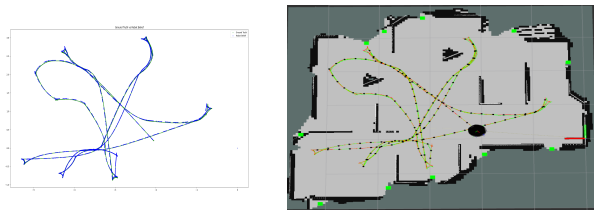
(b) Occupancy grid map

Fig. 19. Map of all aligned scans and occupancy grid map

Figure 19a and 19b illustrate the map generated by plotting the aligned scans during the execution of the Simultaneous Localization and Mapping (SLAM) algorithm. The resulting map is then compared with the occupancy grid map constructed in real-time. This comparison serves to demonstrate the successful integration of the rplidar and IMU sensor measurements within the SLAM framework.

The map presented in Figure 19a (left) exhibits a high degree of alignment when compared to the map depicted in Figure 14b. This alignment signifies the efficacy of the SLAM algorithm in effectively fusing sensor data and producing a coherent representation of the environment.

5.0.9 Joint Autonomous Task Simulation Results



(a) (b)

Fig. 20. (a) x DOF belief within 2σ bound of uncertainty, (b) Ground truth trajectory compared with prediction trajectory after some motion commands .

Figure 20a and 20b depict the results of autonomous exploration in the joint autonomous task, where we combined all hands-on projects simultaneously. During this exploration, the SLAM algorithm ran in the background while the robot navigated through the environment.

In Figure 20a, we observe the ground truth trajectory alongside the robot's belief trajectory, which represents the estimated path based on the SLAM algorithm.

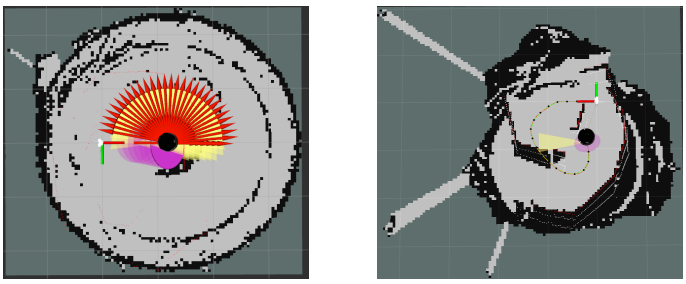
Figure 20b illustrates the map obtained from the exploration. It showcases the cloned poses throughout the entire

exploration period (shown in black) and the ground truth trajectory (depicted in green).

In the subsequent sections, we will delve into the real-time implementation results, which will be discussed in detail in Section 5.1.

5.1 Real-time Implementation results

During the implementation of the SLAM algorithm on the robot, we encountered several notable challenges that had a significant impact on its performance, preventing us from achieving the desired results. These challenges can be summarized as follows:



(a) (b)

Fig. 21. (a) x DOF belief within 2σ bound of uncertainty, (b) Ground truth trajectory compared with prediction trajectory after some motion commands .

- **Drifts:** We encountered significant challenges with drift during the execution of the SLAM algorithm on the robot. The drift was primarily attributed to a combination of factors, including the robot's inherent drift and the low friction of the lab floor. The slippery nature of the floor resulted in the robot experiencing excessive drift, which in turn led to misalignment and inaccuracies in the generated map.
- **Uncertainty Tuning:** The process of fine-tuning the uncertainty parameters in the SLAM algorithm proved challenging. Determining the optimal values for these parameters required extensive experimentation and adjustments to strike a balance between robustness and accuracy which we were not able to achieve.
- **Flipped IMU Axis:** We discovered that the IMU sensor's axis was flipped, which was identified at a relatively late stage of the project. This misalignment of the IMU data negatively affected the estimation of the robot's orientation, further contributing to inaccuracies in the SLAM results.
- **Raspberry Pi Slow Computation Power:** Running the SLAM algorithm on the Raspberry Pi platform posed challenges due to its limited computational power. The processing capabilities of the Raspberry Pi were not sufficient to handle the computational requirements of the SLAM algorithm in real-time. This resulted in slower computations, increased execution time, and impacted the responsiveness of the system.

5.2 Other Challenges

Another challenge was the interference of ROS callback functions due to the high publishing rates of the sensors in simulation. The wheel encoders published messages at a rate of 310 messages per second, the IMU at 10 messages per second, and the lidar at 5 messages per second. To mitigate this issue, we implemented a locking mechanism that allowed only one callback function to execute at a time. This approach effectively managed the interference caused by the high publishing rates, ensuring the integrity of the data processing pipeline.

6 Conclusion

7 Future Works

One avenue for future research is for us to explore the utilization of deep learning models to address two specific challenges: detecting overlaps between LiDAR scans and identifying loop closures. By training a deep neural network, we can collectively learn a similarity metric between scans, enabling us to quantify the degree of overlap between two scans and also output a probability for potential loop closure candidates.

Appendix

1) **Link for simulation implementation and Joint Autonomous Task**: All Simulation and Real-world Implementation Videos for Hands-On Planning and the Joint Autonomous Task

2) Link to model derivations, project follow-up updates, and Gantt chart

Acknowledgments

The authors would like to thank our supervisors Prof. Ridao Rodriguez, Pedro and Pi Roig, Roger for their immense support during the course of this research project.

References

[1] P. Ridao, Probabilistic Robot Localization & Mapping, Unpublished manuscript.

[2] A. Mallios, P. Ridao, D. Ribas, F. Maurelli, and Y. Petillot, "EKF-SLAM for AUV navigation under probabilistic sonar scan-matching," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4404–4411, 2010, organization: IEEE.

[3] A. Palomer, P. Ridao, D. Ribas, A. Mallios, N. Gracias, and G. Vallicrosa, "Bathymetry-based SLAM with difference of normals point-cloud subsampling and probabilistic ICP registration," in *2013 MTS/IEEE OCEANS - Bergen*, pp. 1-8, 2013, doi: 10.1109/OCEANS-Bergen.2013.6608091.

[4] A. Mallios, P. Ridao, D. Ribas, and E. Hernández, "Scan matching SLAM in underwater environments," *Autonomous Robots*, vol. 36, pp. 181–198, 2014, publisher: Springer.

[5] W. Wei, M. Ghafarian, B. Shirinzadeh, A. Al-Jodah, and R. Nowell, "Posture and Map Restoration in SLAM Using Trajectory Information," *Processes*, vol. 10, no. 8, p. 1433, 2022, publisher: MDPI.

[6] A. Ferreira, J. Almeida, A. Martins, A. Matos, and E. Silva, "3DupIC: An Underwater Scan Matching Method for Three-Dimensional Sonar Registration," *Sensors*, vol. 22, no. 10, p. 3631, 2022, publisher: MDPI.