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

Muhammad Faran Akram[1], Solomon Chibuzo Nwafor[2], and Enis Hidri[3]

Computer Vision and Robotics Institute, University of Girona, Spain

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. 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.

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

1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) allows a mobile robot to build a map of an unknown environment while estimating its own position. This is critical for autonomous systems operating without external localization infrastructure.

This work implements a pose-based Extended Kalman Filter (EKF) SLAM framework that uses Iterative Closest Point (ICP) scan-matching. The system fuses sensor data from a Realsense depth camera, an Inertial Measurement Unit (IMU), and wheel encoders on a Kobuki Turtlebot. The Realsense camera provides RGB-D frames, which are converted into 2D point clouds using octomap server.

Unlike landmark-based SLAM, this pose-based method maintains only a history of robot poses in the state vector. This reduces computational load and avoids explicit feature tracking. To align incoming scans, the ICP algorithm is applied to the 2D point clouds. Dead reckoning provides an initial transformation estimate, and the aligned scan is used to correct the pose estimate in the EKF update.

IMU yaw readings and differential drive encoder data are used for motion prediction. To manage the size of the state vector over long trajectories, a pose deletion scheme called the Motion-based Jump Pose (MJP) method is introduced. It selectively retains key poses based on motion significance.

This paper presents the complete system architecture, the scan alignment and filtering pipeline, and an evaluation of SLAM performance across multiple environments.

2 METHODOLOGY

2.1 Pose-Based EKF Formulation

The SLAM system maintains a pose-based state vector, where each state corresponds to a discrete robot pose. Unlike landmark-based EKF SLAM, no explicit map of features is stored. The state vector at time k is:

$$\mathbf{x}_k = \begin{bmatrix}\mathbf{x}_0^\top & \mathbf{x}_1^\top & \cdots & \mathbf{x}_k^\top\end{bmatrix}^\top, \quad \text{with } \mathbf{x}_i = [x_i, y_i, \theta_i]^\top$$

Each pose is a 2D position and orientation tuple. The covariance matrix $\mathbf{P}_k$ captures uncertainty between all past poses. As the robot moves, new poses are appended to the state.

- **Prediction**: Uses dead reckoning and IMU data to estimate the next pose.
- **Update**: Uses ICP alignment to refine the pose estimate and correct accumulated error.

The nonlinear motion model is integrated using a first-order approximation, and Jacobians are computed analytically. The EKF propagates both the state and its uncertainty:

$$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$$
$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^\top + \mathbf{Q}_k$$

where $\mathbf{F}_k$ is the Jacobian of the motion model and $\mathbf{Q}_k$ is the process noise.

This structure avoids the cost of managing a full map and focuses computation on refining the pose chain, making it suitable for dense scan environments.

2.2 Dead Reckoning Prediction

Dead reckoning provides a motion prior by integrating control inputs over time. In this system, it combines wheel encoder data from the Kobuki base with IMU yaw measurements.

The differential drive kinematics model is used:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega$$

where v and ω are the linear and angular velocities computed from encoder ticks.

These velocities are integrated over a short time interval Δt to compute the predicted pose:

Authors Emails: [1]u1999088@campus.udg.edu, [2]u1999124@campus.udg.edu, [3]u1999254@campus.udg.edu

$$x_k = x_{k-1} + v\cos\theta_{k-1}\cdot\Delta t$$
$$y_k = y_{k-1} + v\sin\theta_{k-1}\cdot\Delta t$$
$$\theta_k = \theta_{k-1} + \omega\cdot\Delta t$$

The prediction step also propagates the state covariance using the Jacobian of the motion model. Let $\mathbf{F}_k$ be the Jacobian with respect to the state and $\mathbf{G}_k$ with respect to the noise. Then:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^\top + \mathbf{G}_k\mathbf{Q}_k\mathbf{G}_k^\top$$

This predicted pose and covariance are used as the initial estimate for the ICP scan alignment. The effectiveness of dead reckoning depends on accurate time synchronization, correct wheel radius calibration, and clean IMU input.

2.3 IMU Integration for Yaw Correction

The Realsense IMU provides angular velocity measurements, which are integrated to compute yaw orientation. While wheel encoders estimate heading based on differential velocities, they suffer from drift during slippage or uneven terrain.

To correct this, we use the IMU yaw independently and fuse it with the encoder-based prediction. The angular velocity ω_z around the vertical axis is integrated as:

$$\theta_{\text{imu},k} = \theta_{\text{imu},k-1} + \omega_z\cdot\Delta t$$

This raw IMU yaw is used in the SLAM system in two stages: - For correcting the predicted yaw before ICP. - For an additional update in the EKF via a pseudo-measurement.

The IMU update is treated as a direct measurement of the current orientation. The measurement model is:

$$z_k = h(\mathbf{x}_k) + v_k = \theta_k + v_k$$

where $v_k \sim \mathcal{N}(0, R_\theta)$ is measurement noise. The standard EKF update equations apply:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + R_\theta)^{-1}$$
$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k(z_k - h(\mathbf{x}_{k|k-1}))$$
$$\mathbf{P}_k = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

Here, $\mathbf{H}_k$ is a Jacobian that selects the yaw component from the state. This correction reduces heading drift and stabilizes the ICP initialization, especially in regions with low scan structure.

The IMU update is performed at a fixed rate independent of the scan frequency, ensuring high-frequency orientation correction.

2.4 ICP Scan Alignment using Realsense

After the prediction step, a new point cloud is received from the Realsense-based OctoMap server. The scan is transformed into the robot's `base_footprint` frame and preprocessed into a 2D slice.

ICP alignment is used to estimate the relative motion between the current scan and the previous one. The transformation computed by ICP corrects the predicted pose.

We use Open3D's point-to-plane registration method. Given the source point cloud $\mathscr{P}_k$ and target point cloud $\mathscr{P}_{k-1}$, the ICP algorithm estimates the rigid-body transformation $T_{k,k-1}$ that minimizes the normal-projected residuals:

$$T_{k,k-1} = \arg\min_T \sum_{i=1}^{N} \left[(T\mathbf{p}_i - \mathbf{q}_i)\cdot\mathbf{n}_i\right]^2$$

where $\mathbf{p}_i \in \mathscr{P}_k$, $\mathbf{q}_i \in \mathscr{P}_{k-1}$, and $\mathbf{n}_i$ is the normal at $\mathbf{q}_i$.

The ICP is initialized using the dead reckoning estimate $\hat{T}^{\text{DR}}_{k,k-1}$. This helps guide convergence and improves reliability when scan overlap is small.

After alignment, the final transform $T^{\text{ICP}}_{k,k-1}$ is extracted and used as a measurement in the EKF update. The fitness and RMSE scores are logged and monitored. Scans that fail to meet a minimum fitness threshold are rejected to prevent corrupt updates.

This alignment step is the primary observation in the SLAM loop and corrects accumulated pose drift from the prediction step.

2.5 State Update in the EKF

The ICP alignment output provides a relative pose measurement between the current scan and the previous scan, expressed in the robot frame. This measurement is used to correct the predicted pose in the EKF.

Let the estimated transform from ICP be:

$$\mathbf{z}_k = \begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix}$$

This represents the observed motion between two consecutive poses $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$. The expected observation based on the predicted state is:

$$\hat{\mathbf{z}}_k = h(\mathbf{x}_{k-1}, \mathbf{x}_k) = \begin{bmatrix}x_k - x_{k-1}\\ y_k - y_{k-1}\\ \theta_k - \theta_{k-1}\end{bmatrix}$$

The innovation is:

$$\mathbf{r}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k$$

A linearized update is applied using the standard EKF correction step:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}$$
$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k\mathbf{r}_k$$
$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

Here, $\mathbf{H}_k$ is the Jacobian of the observation function $h(\cdot)$, and $\mathbf{R}_k$ is the observation noise covariance. The noise level is empirically tuned based on ICP registration RMSE.

The update affects only the latest pose and its correlation with prior poses. This corrects drift from dead reckoning and fuses spatial observations from the scan alignment into the trajectory estimate.

Each EKF update also resets the dead reckoning system to match the corrected pose, ensuring consistent prediction in the next cycle.

2.6 Pose Management using MJP

To prevent unbounded growth of the state vector, the system employs a selective update mechanism based on statistical significance. Rather than appending every predicted pose, we use a Mahalanobis distance check to decide whether the new pose contributes sufficient new information.

Given the ICP-derived relative pose measurement $\mathbf{z}_k$ and its expected value $\hat{\mathbf{z}}_k$, we compute the innovation:

$$\mathbf{r}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k$$

The innovation covariance is:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k$$

The Mahalanobis distance is then:

$$d_k^2 = \mathbf{r}_k^\top \mathbf{S}_k^{-1} \mathbf{r}_k$$

This value is compared against a chi-square threshold $\chi^2_{\alpha,dof}$, where α is the confidence level and dof is the dimension of the measurement. If $d_k^2 \leq \chi^2_{\alpha,3}$, the update is rejected as statistically insignificant. Otherwise, the update is accepted and added to the state.

This method ensures that only updates with statistically meaningful differences are incorporated into the trajectory. It suppresses redundant poses that are within expected noise bounds and preserves only those that indicate substantial change.

By using the Mahalanobis distance criterion, the system achieves adaptive pose management without relying on fixed thresholds in Euclidean space. This improves consistency and robustness in environments with variable scan quality or motion dynamics.

3 IMPLEMENTATION

3.1 ROS Node Architecture

The SLAM system is implemented using the Robot Operating System (ROS). All components are structured into modular nodes to ensure clean separation of functionality.

The central node, `slam_node`, manages prediction, update, and transform broadcasting. It subscribes to three primary topics: `/camera/depth/image_raw`, `/imu/data`, and `/odom`. These provide depth images, IMU yaw data, and encoder-based odometry, respectively.

A separate ICP module runs inside the SLAM core class and aligns consecutive point clouds using the estimated motion from dead reckoning as the initial guess.

A timer-driven callback publishes the SLAM pose, trajectory, and global map. Another timer handles IMU updates independently, allowing asynchronous fusion of orientation data.

All TFs are managed internally and published through `tf2_ros`, including `odom` to `base_footprint` and `map` to `odom`.

3.2 Point Cloud Extraction from Depth Frames

Point clouds are obtained from the OctoMap server, which processes RGB-D input from the Realsense camera and publishes voxel-based 3D data.

The point cloud is initially expressed in the `odom` frame. A transformation is applied using TF to convert the cloud into the `base_footprint` frame. This ensures the scan is robot-centric and ready for direct comparison with previous scans.

To reduce dimensionality and noise, only points within a narrow horizontal band are retained. These are treated as a 2D slice of the 3D structure.

3.3 ICP Alignment Parameters and Filtering

Iterative Closest Point (ICP) is a scan-matching algorithm used to estimate the rigid-body transformation between two point clouds. It minimizes the alignment error by iteratively refining the transformation that best aligns a source cloud to a target cloud.

The input to ICP consists of two point clouds: a new scan (source) and a previously stored reference scan (target). Given an initial transformation estimate T_0, ICP computes the optimal transformation T^* that minimizes the distance between corresponding points.

Point-to-Point ICP

Point-to-point ICP minimizes the Euclidean distance between corresponding points. The objective is:

$$T^* = \arg\min_{R,t} \sum_{i=1}^{N} \|R\mathbf{p}_i + t - \mathbf{q}_i\|^2$$

where $\mathbf{p}_i \in \mathbb{R}^3$ are points from the source cloud, $\mathbf{q}_i \in \mathbb{R}^3$ are their corresponding points in the target cloud, $R \in SO(3)$ is a rotation matrix, and $t \in \mathbb{R}^3$ is a translation vector.

This method works well for small, noise-free clouds but is sensitive to surface curvature and initial misalignment.

Point-to-Plane ICP

Point-to-plane ICP improves alignment for structured surfaces by minimizing the distance along the surface normal direction:

$$T^* = \arg\min_{R,t} \sum_{i=1}^{N} \left[(R\mathbf{p}_i + t - \mathbf{q}_i) \cdot \mathbf{n}_i\right]^2$$

where $\mathbf{n}_i$ is the normal vector at the corresponding point $\mathbf{q}_i$. This constraint projects the residual onto the normal direction, resulting in faster convergence on planar surfaces.

In our implementation, Open3D's point-to-plane registration method is used. The initial guess comes from dead reckoning, and a voxel downsampling filter is applied to both clouds to ensure uniform resolution in case of point to plane ICP.

4 EXPERIMENTS AND RESULTS

To evaluate the performance of the proposed Pose-Based EKF SLAM system, we conducted a series of experiments in both simulated enviroment. These experiments aimed to assess the accuracy of localization, the quality of the generated maps, and the robustness of the scan-matching and state estimation pipeline under varying conditions. The system was configured to use encoder and IMU-based dead reckoning for prediction, and 2D point cloud alignment using ICP for correction. Both qualitative and quantitative results are reported.

4.1 Simulation Environment Setup

To better understand the following plot, it is important to clarify the meaning of the different colors used in the colored point clouds. Note that all point clouds are transformed with respect to the fixed frame for visualization purposes only.

Point Cloud Colors:

- White: Stitched point cloud
- Red: Latest point cloud in the buffer
- Blue: Previous point cloud in the buffer

- Green: Matched point cloud after ICP

Covariance Visualization Colors:

- Red: Covariance after taking a scan
- Sky Blue: Predicted covariance
- Green: Covariance after IMU update
- Yellow: Updated covariance
- Purple: Covariance after loop closure

4.2 Qualitative Results: Maps and Trajectories

As shown in Figure 1, the robot was teleoperated to follow a straight trajectory. However, due to sensor noise and motion uncertainty, the estimated path slightly drifts to the left. Despite this drift, the localization update steps (e.g., IMU and ICP corrections) continuously help to reestablish an accurate robot position.

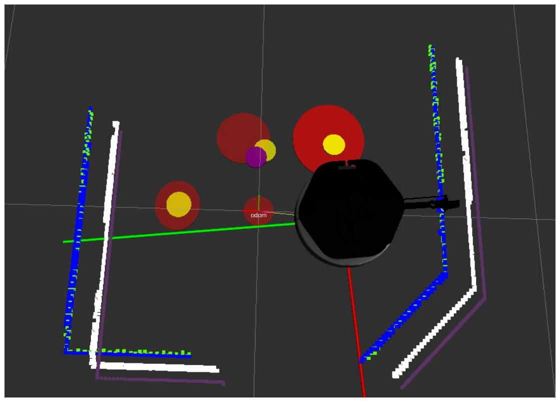

Figure 1: Localization in the simulator

Since SLAM is performed during localization, a virtual representation of the observed environment is simultaneously built. This allows the system to refine both the trajectory and the map based on sensor updates, as it can be appreciated in this final plot of the map:

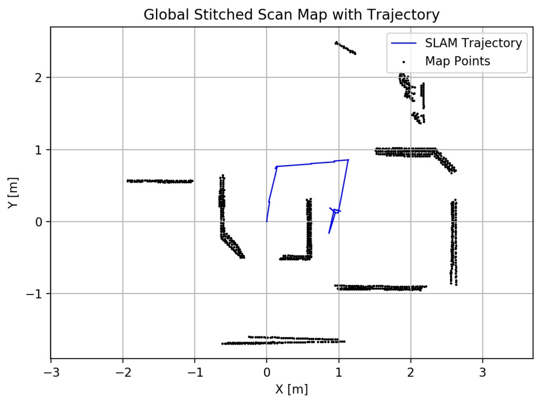

Figure 2: Final representation of the map. Note the fact that the map is mirrored due the matplotlib library.

4.3 Quantitative Evaluation

Figure 3 shows the evolution of the localization error over time for the x, y, and yaw components. As observed, the errors remain within the estimated covariances, indicating consistency in the state estimation. Notably, the covariance shrinks after each IMU update, especially visible in the yaw component. In the x and y directions, the most significant improvements in accuracy occur after the ICP corrections.

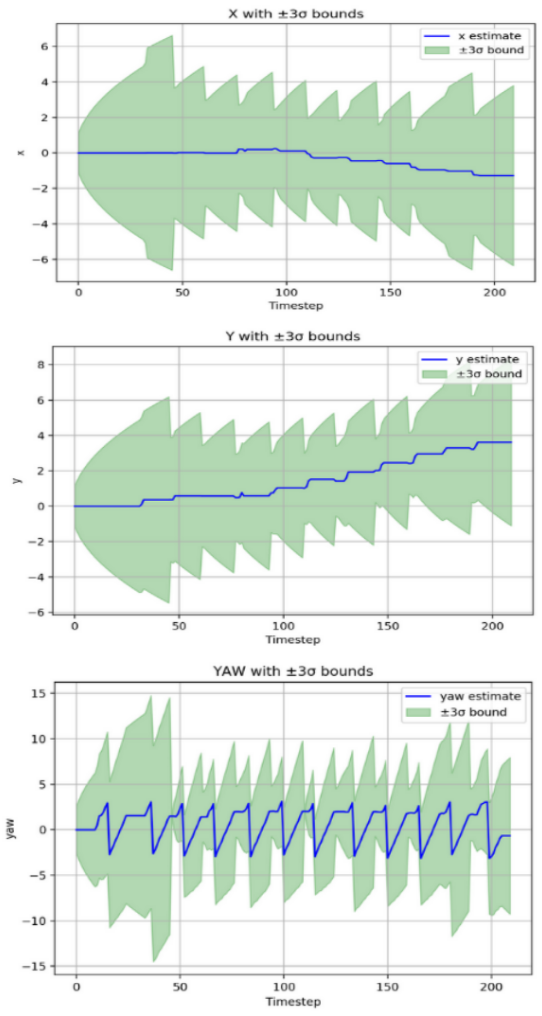

Figure 3: Plot of the errors: x, y, yaw

4.4 Failure Cases and Observations

5 CONCLUSION

The system implemented a pose-based EKF SLAM using Realsense-derived point clouds, encoder odometry, and IMU yaw measurements. Dead reckoning predicted motion using differential drive kinematics. IMU yaw was fused as a pseudo-measurement to correct heading. ICP scan-matching, initialized with predicted motion, provided relative pose corrections. These were used in EKF updates to refine the robot trajectory and reduce accumulated drift.

A Mahalanobis-based pose filtering method managed the growth of the state vector. It rejected updates that did not carry statistically significant information. In simulation, the estimated trajectory remained consistent and aligned with scan corrections. Covariance reduced after each update step, and the final map showed coherent structure. The system maintained a stable pose chain and produced accurate localization without explicit landmarks.

References

[1] R. Spica et al., “Aerial grasping of a moving target with a quadrotor UAV,” IROS, 2012.

[2] X. Ding et al., “A review of aerial manipulation,” Chinese J. Aeronaut., vol. 32, 2019.

[3] R. Jiao et al., “Control of quadrotor equipped with a two DOF robotic arm,” ICARM, 2018.

[4] M. Misin and V. Puig, “LPV MPC control of an autonomous aerial vehicle,” MED, 2020.

[5] S. Kannan et al., “Control of aerial manipulation vehicle in operational space,” ECAI, 2016.