Robust Graph SLAM for Autonomous Navigation using ICP-Based LIDAR Scan Matching and EKF-IMU preintegration

Atharva Anil Patwe
Intelligent Field Robotics Systems
Universitat de Girona
Girona, Spain
patweatharva@gmail.com

Loc Thanh Pham
Intelligent Field Robotics Systems
Universitat de Girona
Girona, Spain
u1992429@campus.udg.edu

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a critical challenge in robotics, where a robot concurrently builds or updates a map of an unknown environment and tracks its position within that environment. This work aims to implement a robust Simultaneous Localization and Mapping algorithm on a turtlebot platform using onboard sensors - 2D Lidar, Wheel Encoders, and Magnetometer compass. This study explores the implementation of online SLAM and full SLAM using factor graphs, specifically leveraging the Georgia Tech Smoothing and Mapping (GTSAM) library on a Turtlebot robot platform with the Robot Operating System (ROS). We describe the construction of the factor graph for SLAM, incorporating 2D Lidar scans and odometry data, and validate our approach through simulations and experiments in real world. Our study also incorporates the Closed form ICP covariance estimation method proposed by an existing work [4]. The Graph SLAM output, both with and without the proposed method, was recorded during real-world testing. Additionally, the impact of the keyframe triggering parameter was evaluated.**

***Index Terms*—Simultaneous Localization and Mapping, ICP, Scan Matching, Lidar, Factor Graphs, Graph SLAM, GTSAM, Pose Graph Optimization, ROS, , Stonefish, Turtlebot**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in the field of robotics, where a robot constructs or updates a map of an unknown environment while simultaneously keeping track of its own location within that environment. It is carried out by observing changes in the environment using various sensors including 2D/3D Lidars, Wheel Encoders, IMU, Cameras etc. SLAM encompasses a wide range of applications, from autonomous vehicles to household robots, and relies heavily on the integration of various sensor inputs to achieve accurate and reliable mapping and localization.

In Online SLAM, the whole map is tracked along with the current estimate of state continuously throughout the operation. This dual task of mapping and localization is crucial in autonomous navigation task in complex and dynamic environment. Online SLAM can be implemented using methods like Kalman Filter, Extended Kalman Filter, Unscented Kalman Filter, Particle Filter. However, In full SLAM, The whole problem can be relinearized during operation, minimizing the linearization error accumulation. Linearized problem can then be solved by Least squares optimization (Gauss Newton method) or Gaussian Belief Propagation (message passing).

Among the many approaches to SLAM, factor graphs have emerged as a powerful and flexible framework for representing and solving the SLAM problem. A factor graph is a bipartite graph that consists of variable nodes, which represent the unknown quantities such as robot poses and landmark positions, and factor nodes, which represent constraints or measurements between these variables. This representation allows for a clear and intuitive formulation of the SLAM problem, where the objective is to find the configuration of variables that best satisfies the given constraints.

Factor graphs uses the structure of the problem to enable efficient optimization, typically through methods such as nonlinear least squares. One of the prominent tools for implementing factor graph-based SLAM is the Georgia Tech Smoothing and Mapping (GTSAM) library. GTSAM provides a flexible and efficient framework for defining factor graphs and performing the necessary optimizations to solve SLAM problems. Thus, this work utilizes GTSAM for constructing the SLAM problem and implementation on the turtlebot robot platform using Robot Operating System(ROS).

This paper follows following structure: Section II elaborates on the methodology theory used for this work followed by the Implementation details explained in the section III. In this section, ROS nodes developed for the implementation are discussed in detail. The validation of the algorithms are carried out in the section IV where simulation results are reported followed by the experimental validation on turtlebot 2 platform.

II. Methodology

A factor graph consists of two main components: variables and factors. Variables represent the unknown quantities in the system that need to be estimated, such as robot poses or landmark positions. Factors encode the probabilistic relationships or constraints between these variables (See Fig. 1),

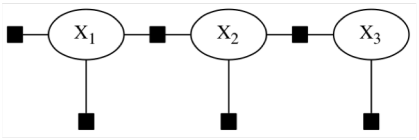


Fig. 1: An HMM with observed measurements, unrolled over time, represented as a factor graph.

derived from sensor measurements or other observations. As stated earlier, this work utilizes following sensors 2D Lidar, Magnetometer compass and Wheel encoders. The 2D Lidar is used to obtain the scans of the environment from the different poses and Compass and wheel encoders are used to obtain the odometry readings.

In this work, factor graph is implemented for the SLAM problem of turtlebot which is a three degrees of freedom *Pose2D* state vector $\mathbf{X}$ represented as follows:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

where x and y are Cartesian coordinates and θ is the yaw or heading of the robot.

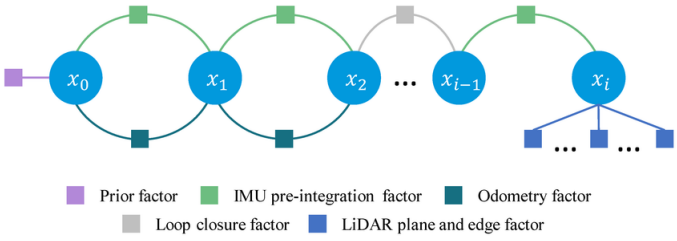


Fig. 2: Factor Graph Structure With Encoder/IMU Preintegration and Lidar Scan Matching Factors

The 2D Pose Factor Graph was constructed by adding Variables or Keyframes representing the 2D poses of the robot as the robot moves in the environment (See Fig. 2). Subsequently, the scan matching pose estimations and odometry preintegration factors are included in between the keyframes of the graph providing the necessary constraints and measurements to accurately estimate the robot's trajectory. A Prior pose with a noise model is added to initialize the graph. Initial estimates for the keyframe poses are further added to the graph while the robot is moving by composing the more accurate measurement (usually Scan Matching) from both of the factors to the last keyframe pose using lie algebra of GTSAM library.

The factors of the graph are estimated with the help of two separate navigators - Scan Matching Navigator and EKF Odometry. The scans obtained from the lidar are converted into the robot base frame after and registered with all the scan saved earlier using transformations obtained from matching. The pointcloud matching is carried out by ICP algorithm giving the transformation between the scans which are then added to the graph as Pose 2D constraints with some uncertainty.On the other hand, the odometry factors are estimated using an Extended Kalman Filter (EKF). The EKF predicts the robot's pose based on encoder readings and updates this prediction with compass measurements. This approach corrects the odometry and reduces drift compared to pure dead reckoning with encoders alone. The implementation details are given in the subsequent section III.

The sampling frequency of the wheel encoder and compass is relatively high, in the range of 100-200 Hz, allowing the EKF Odometry Navigator to update odometry at a high rate. The Lidar, however, operates at a much lower frequency. To obtain meaningful scan matching results, the robot needs to have some displacement between subsequent Lidar scans. Therefore, new poses are added at the frequency of Lidar matches. Between two Lidar matches, the odometry data is preintegrated and added once along with the scan matching factor. As mentioned earlier, the initial estimates (or initial guess) for the optimization are calculated by composing the more accurate factor with the previous pose.

The uncertainty of the odometry is calculated by the EKF by tuning the uncertainties of the encoders and compass. The final covariance matrix of the odometry is used to estimate the Gaussian noise model for that factor. On the other hand, since ICP does not have a built-in covariance estimation procedure, the uncertainty of the scan matching factor is estimated using a method derived from the work of A. Censi as proposed in [3]. Further work presented in [4] extends this covariance estimation method to 3D point clouds, as opposed to the 2D estimation in [3]. After adding all the factors with their uncertainties and variables, the graph is optimized. For Online applications, sparse nonlinear incremental optimization (iSAM2) is used which achieves improvements in efficiency through incremental variable re-ordering and fluid relinearization, eliminating the need for periodic batch steps as proposed in [5]. For the full SLAM problem, more accurate solvers, such as the Levenberg-Marquardt solver, are used. One drawback of using iSAM2 in GTSAM is that, as an incremental solver, it cannot estimate the uncertainties of the robot's past poses. The optimized poses are then used to register all the scans again to build the world map.

III. Implementation

While implementing the mentioned methodology, three ROS nodes were developed. - Scan Handler, Odometry Handler, and GraphSLAM Handler. The overall architecture is depicted in the figure 3.

A. Scan Handler

The Scan Handler node subscribes to the Lidar topic published by the Lidar driver package. It also subscribes to the odometry topic to access the odometry messages published by the EKF Odometry Handler. The Scan Handling pipeline is depicted in Algorithm 1. Using odometry trigger and time

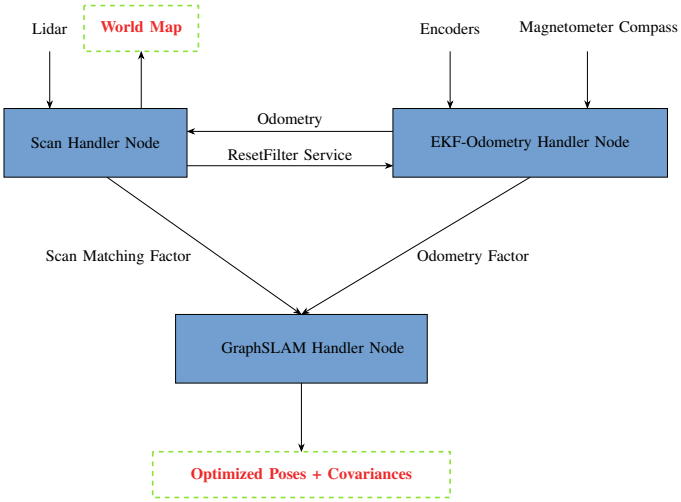


Fig. 3: Graph SLAM ROS Package Architecture

trigger, scans are taken at specific intervals based on the robot's displacement and elapsed time. Once a Lidar message is received, it is converted into a point cloud in the robot's base frame, with skewness removed through interpolation between the scan's start and end times. The base frame point cloud is then converted into a PCL (Point Cloud Library) data type for easy manipulation.

The current odometry data is saved to create a keyframe in the graph. After saving the keyframe, the odometry is reset using the resetfilter service, which resets both the odometry and covariance to zero for the new keyframe. Thus, the odometry operates from keyframe to keyframe.

The saved point cloud is then processed to remove noise. Since the Lidar is 2D, a planar model is fitted, and outliers are removed. A hypothesis is then formed, containing potential candidates that may match well with the current scan. This is based on the distance between the current frame and a particular scan. The hypothesis is passed for registration to estimate the transformation between them. In this work Iterative Closest Point(ICP) algorithm from PointCloud Library is used for matching. ICP uses reciprocal correspondences to estimate the transformation between source and target pointcloud. The current odometry keyframe is given as initial guess for registration with the last scan and the initial guess with any keyframe of graph given in figure 2 is calculated as follows:

$$\mathbf{T^i_{current}} = (\mathbf{T^w_i})^{-1} \cdot \mathbf{T^w_{current}}$$

where $\mathbf{T^i_{current}}$ is the transformation matrix from the i-th keyframe to the current keyframe which is the initial guess for the ICP, $\mathbf{T^i_w}$ is the transformation matrix from the i-th keyframe to the world, which is the inverse of $\mathbf{T^w_i}$ that was already saved when i^{th} keyframe was created, and $\mathbf{T^w_{current}}$ is the transformation matrix from the world to the current frame which is calculated by composing odometry measurement from the last keyframe to the last keyframe pose.

The Closed form ICP covariance estimation method is developed which takes the reciprocal correspondences and ICP output transformation and estimates the covariance of the registration process as shown in [4]. A Pose 2D transformation is extracted from the Transformation matrix returned by the ICP algorithm. The same process is repeated for all the scans present in the hypothesis.

The Scan Handler Node also handles another crucial aspect of the SLAM problem: mapping. This node subscribes to the optimized poses topic published by the Graph SLAM node and reconstructs the map by transforming the point clouds from each pose into the world frame. The resulting world point cloud is then published. This point cloud is subsequently subscribed to by the octomap server to generate an occupancy grid, as illustrated in Figure 8.

Algorithm 1 Scan Registration Pipeline Pseudo code

1: **Start** (Receive LiDAR message)
2: Convert to Pointcloud in LiDAR Frame
3: Remove Skew
4: Convert to Robo Base Frame
5: Convert PointCloud2 to PCL format
6: Save Current Keyframe pose using Odometry
7: Reset EKF Odometry Filter
8: Remove noise from Scan(Fitting planar model)
9: Store Pointcloud
10: Estimate matching hypothesis vector
11: Registration using ICP with initial guess
12: Estimate Covariance from matching
13: Register Transformations from world to current keyframe using registration transformations
14: Publish transformation and Covariances

B. Odometry Handler

The Odometry Node subscribes to wheel encoder readings and magnetometer compass measurements. This node implements an Extended Kalman Filter (EKF), which predicts the robot's motion based on encoder readings and corrects the prediction using compass measurements. Initially, the pose and covariance are set to zero, and a map frame is established when the EKF node starts. Once the filter is initialized, the Localize method is called within the callback function whenever new encoder readings are received. The pseudo code for the Localize method is given in Algorithm 2. GetInput method extracts the displacement carried out by the robot from the received encoder readings.

Algorithm 2 Localize

Require: Previous robot mean state x_{k-1}, Previous covariance of robot state P_{k-1}
Ensure: New Robot mean state x_k, New Covariance P_k
1: $(u_k, Q_k) \leftarrow$ GetInput()
2: $(\bar{x}_k, \bar{P}_k) \leftarrow$ Prediction$(u_k, Q_k, x_{k-1}, P_{k-1})$
3: $(z_k, R_k, H_k, V_k) \leftarrow$ GetMeasurements()
4: **if** z_k **then**
5: $(x_k, P_k) \leftarrow$ Update$(z_k, R_k, \bar{x}_k, \bar{P}_k, H_k, V_k)$
6: **return** (x_k, P_k)
7: **else**
8: **return** $(\bar{x}_k, \bar{P}_k)$
9: **end if**

1) Method- GetInput(): The *GetInput* method returns the displacement carried out by the robot since the last time step using the current encoder reading message in the following manner:

$$u_k, Q_k = \text{ReadEncoders()} \tag{1}$$
$$d_l = u_k[0] \cdot dt \cdot \textit{wheel Radius} \tag{2}$$
$$d_r = u_k[1] \cdot dt \cdot \textit{wheel Radius} \tag{3}$$
$$d = \frac{d_l + d_r}{2} \tag{4}$$
$$\delta_\theta = \frac{d_r - d_l}{\textit{wheel Base}} \tag{5}$$
$$u_k = \textit{Pose2D}\left(\begin{bmatrix} d[0] \\ 0 \\ \delta_\theta[0] \end{bmatrix}\right) \tag{6}$$

u_k represents the displacement, Q_k represents the covariance matrix for the encoder readings, d_l represents the distance traveled by the left wheel, d_r represents the distance traveled by the right wheel, d represents the average distance traveled, δ_θ represents the change in orientation, *wheel Radius* represents the radius of the wheel, wheel Base represents the distance between the two wheels, and Pose2D(***x***) represents a 2D pose with coordinates ***x***.

2) Method- Prediction(): If new encoder readings are obtained, the prediction method is called. The prediction method applies the motion model and calculates the new mean and covariance of the state by composing/compounding the previous mean with the input displacement. In the implementation, the motion model using compunding is applied by the function f. The compounding in the EKF filter is carried out using method *oplus()*.

The method ***oplus()*** implements the pose compounding of two poses. It receives two poses, ${}^AX_B$ and ${}^BX_C$, as input and returns the pose, ${}^AX_C$. Pose compounding operations will be represented by $\oplus$. The derivation steps are shown below:

$${}^AX_B = \begin{bmatrix} {}^Ax_B \\ {}^Ay_B \\ {}^A\psi_B \end{bmatrix} = \begin{bmatrix} \cos({}^A\psi_B) & -\sin({}^A\psi_B) & {}^Ax_B \\ \sin({}^A\psi_B) & \cos({}^A\psi_B) & {}^Ay_B \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^BX_C = \begin{bmatrix} {}^Bx_C \\ {}^By_C \\ {}^B\psi_C \end{bmatrix} = \begin{bmatrix} \cos({}^B\psi_C) & -\sin({}^B\psi_C) & {}^Bx_C \\ \sin({}^B\psi_C) & \cos({}^B\psi_C) & {}^By_C \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^AX_C = {}^AX_B \times {}^BX_C = \begin{bmatrix} \cos({}^A\psi_B) & -\sin({}^A\psi_B) & {}^Ax_B \\ \sin({}^A\psi_B) & \cos({}^A\psi_B) & {}^Ay_B \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos({}^B\psi_C) & -\sin({}^B\psi_C) & {}^Bx_C \\ \sin({}^B\psi_C) & \cos({}^B\psi_C) & {}^By_C \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^AX_C = {}^AX_B \oplus {}^BX_C = \begin{bmatrix} {}^Ax_C \\ {}^Ay_C \\ {}^A\psi_C \end{bmatrix} = \begin{bmatrix} {}^Ax_B + {}^Bx_C\cos({}^A\psi_B) - {}^By_C\sin({}^A\psi_B) \\ {}^Ay_B + {}^Bx_C\sin({}^A\psi_B) + {}^By_C\cos({}^A\psi_B) \\ {}^A\psi_B + {}^B\psi_C \end{bmatrix}$$

Given Gaussian random vectors X and Y with means $\hat{X}$ and $\hat{Y}$, and covariances P_X and P_Y, the vectorial expression for $z = AX + BY$ along with calculations for its mean and covariance can be represented as follows:

$$z = AX + BY \tag{7}$$
$$\text{Mean of } z = A\hat{X} + B\hat{Y} \tag{8}$$
$$\text{Covariance of } z = AP_XA^\top + BP_YB^\top + AP_{XY}B^\top + BP_XA^\top \tag{9}$$

In our case the, the vectors are independent, and hence, the cross covariance becomes zero. Thus the formula for covariance new predicted state becomes

The equation 9 can be represented in terms of $\bar{P}_k$ as:

$$\bar{P}k = ((A_k \cdot P_{k-1} \cdot (A_k^\top)) + (W_k \cdot Q_k \cdot (W_k^\top)))$$

where,

A_k is Jacobian of compounding w.r.t. state $(x_{k-1}) = \left.\dfrac{\partial f}{\partial x_{k-1}}\right|_{\hat{x_{k-1}}, \hat{w}=0}$

W_k is Jacobian of compounding w.r.t. noise $(w_k) = \left.\dfrac{\partial f}{\partial w_k}\right|_{\hat{x_{k-1}}, \hat{w}=0}$

The prediction method then returns the new predicted mean $\hat{\bar{x}_k}$ and covariance $\hat{\bar{P}_k}$.

3) Method- Update(): The predicted measurement $h(x_{k|k-1})$ from the predicted state $x_{k|k-1}$ is obtained using the observation model.The innovation or residual term $\tilde{z}_k$ represents the difference between the actual measurement z_k and the predicted measurement:

$$\tilde{z}_k = z_k - h(x_{k|k-1})$$

The covariance of the innovation S_k is the combination of the predicted measurement covariance $H_kP_{k|k-1}H_k^\top$ and the measurement noise covariance R_k:

$$S_k = H_kP_{k|k-1}H_k^\top + R_k$$

The Kalman gain K_k is the matrix that weighs the predicted state and the measurement, adjusting the state estimation:

$$K_k = P_{k|k-1} H_k^\top S_k^{-1}$$

The derivation involves finding the optimal weighting factor (the gain) that minimizes the expected estimation error covariance, blending the predicted state with the measurement in a way that accounts for uncertainties in both the predicted state and the measurement.

The Kalman gain effectively balances the trust given to the prediction (through $P_{k|k-1}$) and the trust given to the measurement (through S_k^{-1}) to update the state estimate optimally. The predicted mean state is then updated using this Kalman gain to obtain the new mean state. This is achieved by multiplying the Kalman gain by innovation factor and added to the predicted state.

$$x_k = x_{k|k-1} + K_k \left(z_k - h(x_{k|k-1}) \right)$$

Finally, The covariance is updated by the correction factor.

$$P_k = (I - K_k H_k) P_{k|k-1}$$

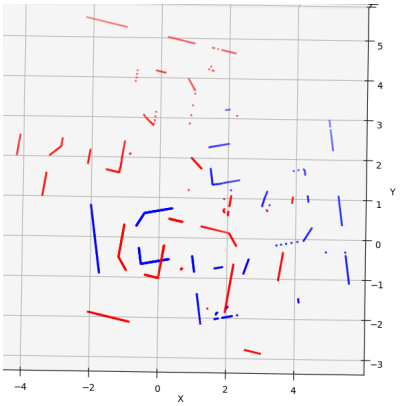
Similarly, if a new heading is obtained, the update is carried out using standard Kalman filter equations. Finally, the odometry readings are published. Additionally, the node hosts a reset odometry service server, allowing for the reset of odometry to zero upon receiving a request from the Scan Handler Node.

C. GraphSLAM Handler

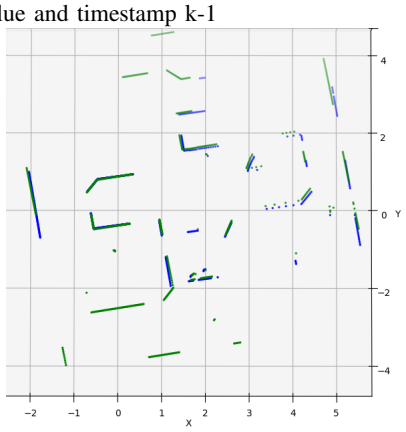
This node is responsible for the construction of the Nonlinear Factor Graph, an important component in solving the SLAM problem utilizing Odometry and Scan Registration. Initially, it reads and initializes all parameters required for the optimization process. Within the callback method for Scan registrations, Pose 2D transformation factors are extracted from the messages and incorporated into the graph. As previously noted, initial estimates for the optimization process are established by composing the more precise factor with the last known robot pose. Subsequently, the graph undergoes optimization following each Scan Matching factor. Once optimized, the updated keyframe poses are then published to build the map.

IV. Results and Discussion

The Graph SLAM Package was developed in ROS Noetic on an Ubuntu 20.04 machine. It was initially tested in the Stonefish simulator before being deployed onto the Turtlebot2 platform for further experimental validation.



(a) Source Pointcloud in red at timestamp k and Target Pointcloud in blue and timestamp k-1



(b) Aligned Pointcloud in green of timestamp k and Target Pointcloud in blue at time k-1

Fig. 4: Point Cloud Comparisons before and after ICP: (a) Source and Target Point Clouds, (b) Aligned and Target Point Clouds.

A. ICP Alignment

The ICP registration methodology in the Scan Matching Navigator was thoroughly checked to avoid the issue of ICP local minima. Firstly, the initial guess was carefully calculated using the method described earlier in Section III-A. To improve accuracy and robustness, reciprocal correspondences were used during ICP registration. Additionally, stopping criteria were carefully tuned to maximize efficiency while maintaining the accuracy of the registration.

The point clouds shown in red in Figure 4a represent the scan taken at time k, when the robot moved from the keyframe at timestamp $k-1$, represented by the blue scan. The transformation between these two timestamps corresponds to the transformation of the robot's base frame. The initial guess between these two point clouds is simply the odometry reading at time k, as the odometry is relative to the last keyframe, which is blue in this case. After performing ICP registration, it is clearly observed in Figure 4b that the green point cloud, which has been transformed by the transformation returned by the ICP, aligns well with the target.

B. Graph SLAM Simulation Results

The developed ROS package was initially tested in the Stonefish simulator. Figure 5 shows the SLAM results obtained using only the dead reckoning method. It is evident that the estimated robot pose (indicated by the red arrow) drifts from the ground truth as the robot traverses the map. Consequently, this drift causes the map created by composing the keyframe poses to also drift which is depicted in figure 6, rendering the octomap inaccurate. On the other hand, when the odometry is combined with the Scan Registration Navigator and added to the graph, the point cloud becomes more consistent, as evident in Figure 7. This improvement renders the occupancy grid more accurate, as shown in Figure 8.

The comparison between ground truth of robot pose and Pose graph optimization output for the test given in figure 8 is shown in figure 9. In this test, it is observed that the Graph SLAM is able to follow

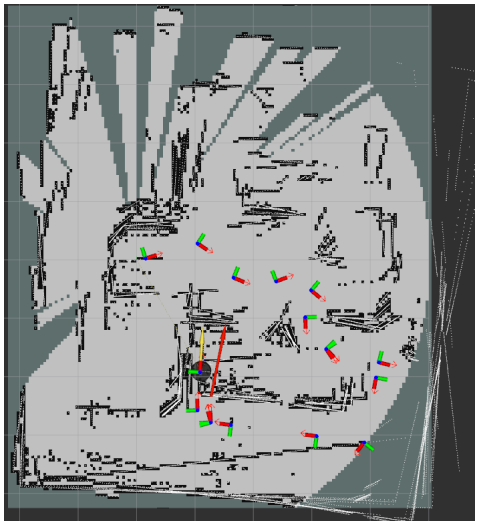


Fig. 5: Simulation: World Point Cloud Constructed Using Dead Reckoning Only(Odometry). Red and Green Axes Indicate Factor Graph Keyframes, Red Arrow Represents Robot Pose Calculated by Graph SLAM, and Yellow Arrow Indicates Ground Truth

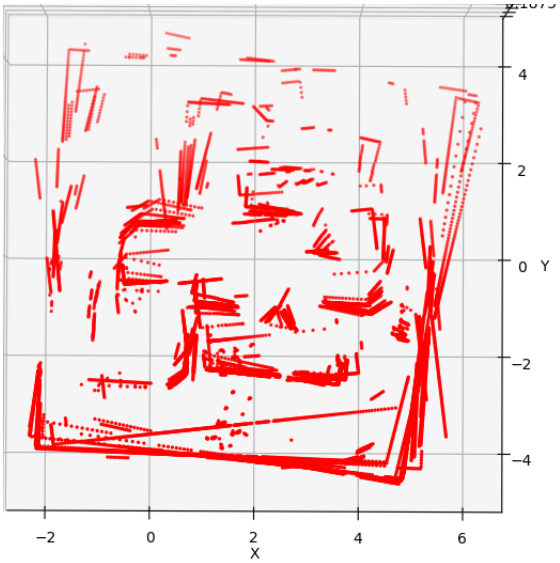


Fig. 6: Simulation: World Point Cloud Constructed Using Dead Reckoning Only (Odometry)

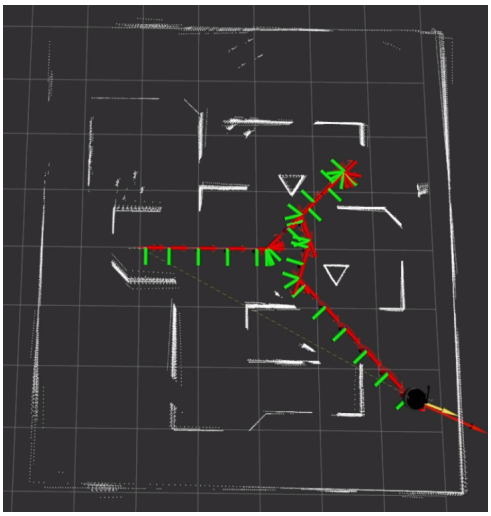


Fig. 7: Simulation: World Point Cloud Constructed Using Pose Graph Optimization via Graph SLAM. Red and Green Axes Indicate Factor Graph Keyframes, Red Arrow Represents Robot Pose Calculated by Graph SLAM, and Yellow Arrow Indicates Ground Truth

C. Graph SLAM Experimental Validation and Results

1) Loop Closure: For experimental validation of the loop closure condition and its effects, a map depicted in Figure 11 was created. In SLAM, loop closure refers to the process of identifying and closing loops in the robot's trajectory as it explores an environment. When a robot revisits a previously visited location, loop closure algorithms detect this revisitation point and reconcile the new data with the existing map, effectively 'closing' the loop. When the robot revisits a keyframe, the uncertainties associated with the keyframes in the graph naturally decrease as the drift is corrected. To demonstrate this effect, the robot was traversed along a circular track in the map. This behavior is evident in Figures 12 and 13, where as the robot progresses through the map, the uncertainty ellipsoid (depicted in red) gradually expands until it approaches the starting pose. Upon reaching close to the previous keyframe, a new factor is introduced between the current keyframe and the previous keyframes. During the optimization process, this factor facilitates the correction of drift, resulting in a more confident pose graph. This correction is visually indicated by the reduction in the size of the ellipsoids, signifying improved certainty in the robot's localization and mapping.

In this test, two conditions were evaluated: one without the closed-form ICP covariance estimation method and one with it. When there is no estimation of the ICP registration

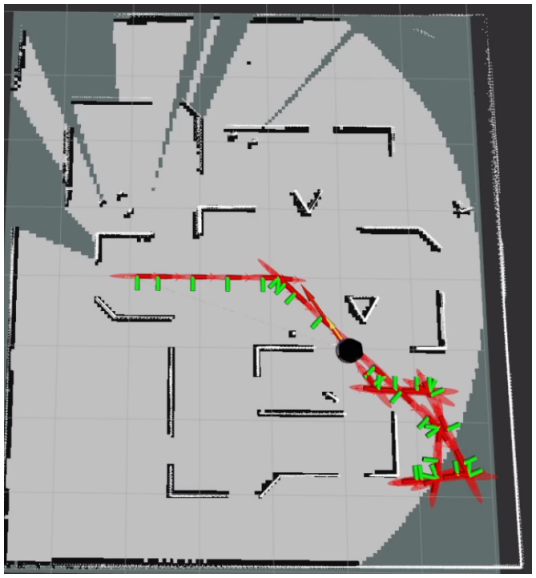


Fig. 8: Simulation 2: World Point Cloud Constructed Using Pose Graph Optimization via Graph SLAM. Red and Green Axes Indicate Factor Graph Keyframes, Red Arrow Represents Robot Pose Calculated by Graph SLAM, and Yellow Arrow Indicates Ground Truth, Occupancy Grid Map constructed from World Pointcloud

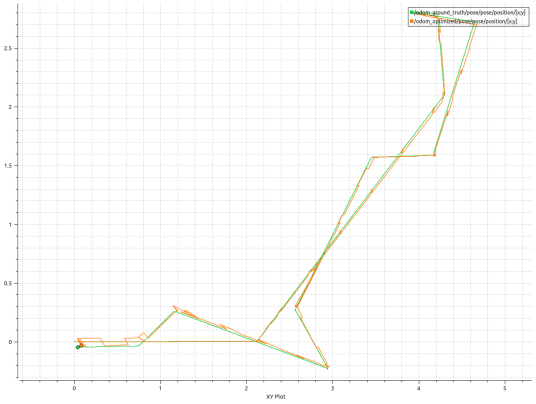


Fig. 9: Simulation 2: Ground truth (green) Vs. Graph SLAM output (orange) for the test shown in Figure 8.

covariance, a constant noise model with a standard deviation of (0.1,0.1,0.08) is inserted into the graph. This condition assigns equal uncertainty to all the registration factors, thus when optimizing the graph, the optimizer gives equal weights to all the factors, making it prone to drifting if the scan matching is inaccurate. This effect is clearly visible in Figure 12, where the environment wall exhibits noticeable drift, resulting in an unnaturally broad wall. When employing the closed-form ICP covariance method, the observed drift in the point clouds is significantly reduced, leading to enhanced accuracy in localization and mapping as evident from the figure 13. In this test, the odometry trigger was set to 0.1 meters, and the time trigger was set to 1 second. Consequently, a large number

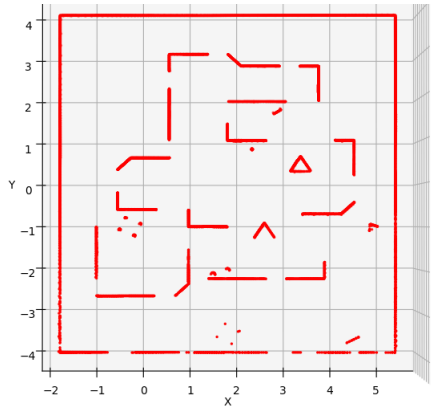


Fig. 10: Simulation 2: World Map Pointcloud Obtained from Graph SLAM in Stonefish Simulator

of keyframes were generated, leading to accumulation of point clouds.

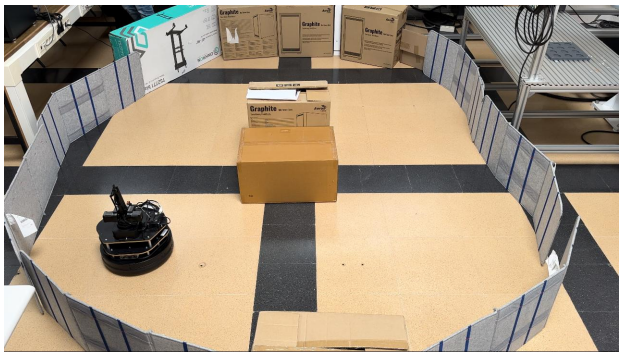


Fig. 11: Experimental Validation Environment: Loop Closure

2) Mapping: After testing the loop closure condition, we evaluated Graph SLAM for mapping in an environment with obstacles, as shown in Figure 14. The robot was tele-operated to navigate through this environment. During this evaluation, we tested the effect of the odometry trigger parameter with two different keyframe trigger values: 0.5 meters and 1.0 meter. The results indicated that Graph SLAM with a keyframe trigger of 0.5 meters produced more accurate mapping and localization, as illustrated in Figure 15.

This improved performance is due to the initial guess used in the ICP registration process, which relies on odometry data. When the keyframe trigger is set to 0.5 meters, the odometry data accumulates less noise, resulting in better ICP registration and more reliable odometry factors in the Factor Graph. Conversely, with a 1.0 meter trigger, the odometry accumulates more noise, which negatively impacts both the ICP registration process and the reliability of the odometry factors in the Factor Graph.

V. Conclusion

In this study, we successfully implemented a factor graph-based SLAM system on a Turtlebot platform using the GT-

Fig. 12: Experimental Validation 1: Turtlebot2 Graph SLAM Results without Closed form ICP Covariance Estimation Method

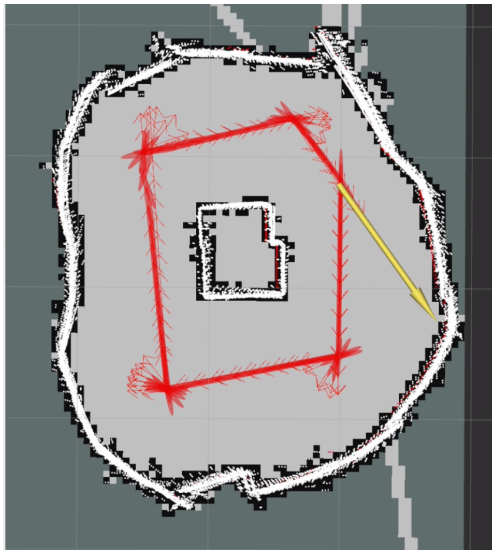


Fig. 13: Experimental Validation 2: Turtlebot2 Graph SLAM Results with ICP Covariance Estimation Method

SAM library and ROS. Our approach integrated data from 2D Lidar scans, wheel encoders, and a magnetometer compass to achieve accurate mapping and localization. The Scan Handler node efficiently processed Lidar data and performed ICP-based scan matching, while the Odometry Handler node utilized an Extended Kalman Filter to enhance odometry accuracy. The GraphSLAM Handler node constructed and optimized the factor graph, significantly improving the consistency and accuracy of the robot's pose estimation compared to dead reckoning alone.

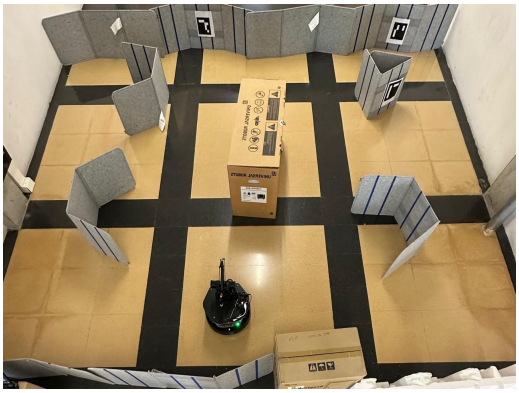


Fig. 14: Experimental Validation 3: Testing Environemt

Fig. 15: Experimental Validation 3: Turtlebot2 Graph SLAM results with Keyframe Odometry trigger of 0.5m

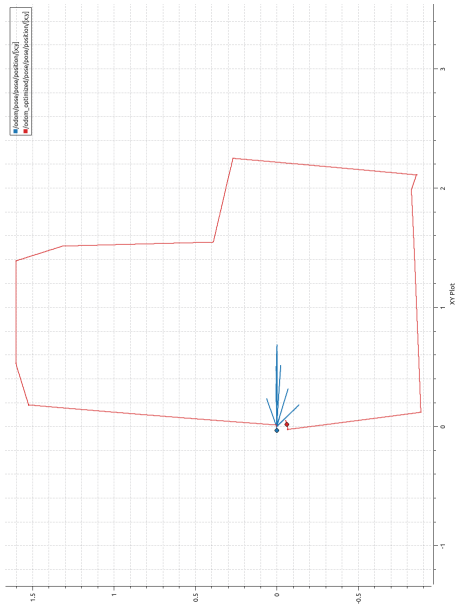


Fig. 16: Experimental Validation 3: Turtlebot2 Graph SLAM Trajectory Plot with Keyframe Odometry trigger of 0.5m

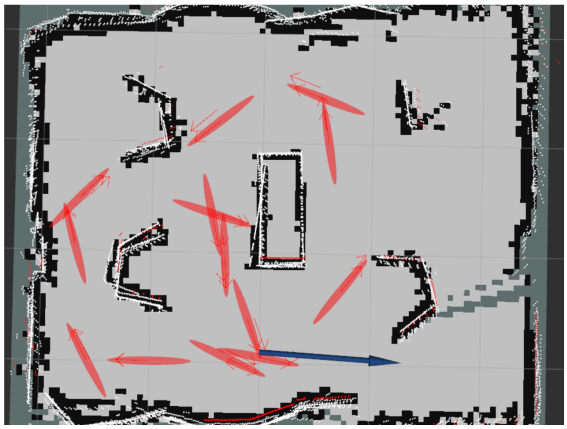


Fig. 17: Experimental Validation 3: Turtlebot2 Graph SLAM results with Keyframe Odometry trigger of 1.0m

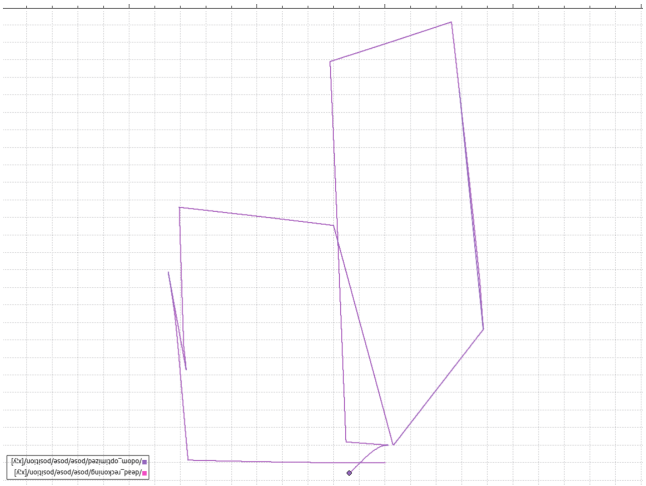


Fig. 18: Experimental Validation 3: Turtlebot2 Graph SLAM Trajectory Plot with Keyframe Odometry trigger of 1.0m

REFERENCES

[1] Dellaert F, Kaess M. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. The International Journal of Robotics Research. 2006;25(12):1181-1203. doi:10.1177/0278364906072768

[2] H. . -A. Loeliger, "An introduction to factor graphs," in IEEE Signal Processing Magazine, vol. 21, no. 1, pp. 28-41, Jan. 2004, doi: 10.1109/MSP.2004.1267047.

[3] A. Censi, "An accurate closed-form estimate of ICP's covariance," Proceedings 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 2007, pp. 3167-3172, doi: 10.1109/ROBOT.2007.363961.

[4] S. M. Prakhya, L. Bingbing, Y. Rui and W. Lin, "A closed-form estimate of 3D ICP covariance," 2015 14th IAPR International Conference on Machine Vision Applications (MVA), Tokyo, Japan, 2015, pp. 526-529, doi: 10.1109/MVA.2015.7153246.

[5] Kaess, Michael Johannsson, Hordur Roberts, Richard Ila, Viorela Leonard, John Dellaert, Frank. (2012). iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. International Journal of Robotic Research - IJRR. 31. 216-235. 10.1177/0278364911430419.