LiDAR-Based SLAM in a 2D Simulated Environment

Jurica Maltar*, Domagoj Ševerdija*
* J. J. Strossmayer University of Osijek, School of Applied Mathematics and Informatics, Osijek, Croatia
{jmaltar, dseverdi}@mathos.hr

***Abstract*—One of the best-known problems in robotics is simultaneous localization and mapping, where the robot tries to build the map of a place it navigates while simultaneously determining its configuration in that space. Different sensor modalities, e.g., LiDAR sensor(s), visual sensor(s), or their combination, can tackle this problem. Configurations of the robot over time are estimated using standard estimation techniques such as the extended Kalman filter or factor graphs. Unfortunately, one of the main issues during the robot's movement is the drift, which occurs due to imperfect sensors and actuators. This can be diminished by deploying a loop closing algorithm that detects whether an already visited place has been observed. That being said, a SLAM implementation itself has a steep learning curve. Therefore, this paper provides an extensive overview of all the steps needed to implement such a system. Also, we provide the quantitative evaluation of our approach, where we deploy our state-of-the-art loop closing algorithm that diminishes drift. We showed how our loop closing approach improves the absolute trajectory error measure compared to the standard loop closing approach.**

***Keywords*—SLAM, LiDAR, factor graphs, loop closing, ICP**

I. INTRODUCTION

In the last few years, we have witnessed the ubiquity of autonomous vehicles, whether the most modern electric cars, robot vacuums, or industrial mobile robots. Such vehicles are capable of discerning where in an environment they are and progressively building a map of that environment. This is a well-known robotics problem called *simultaneous localization and mapping* (abbr. SLAM) [1]. In SLAM, a well-built map implies good localization results, and vice versa; good localization eventually yields a better map.

In its essence, SLAM is an estimation problem; therefore, the standard estimation theory methods can be used to tackle it, e.g., the extended Kalman filter (abbr. EKF) as used in [2]. As mentioned by the same authors, filter-based SLAM approaches are, in the last few years, outperformed by factor graph-based approaches such as ORB-SLAM [3] and LSD-SLAM [4]. The estimation component of a SLAM system that estimates the robot's configurations over time is also called SLAM *back-end*. Conversely, SLAM *front-end* deals with sensor processing, i.e., converting the robot's measurements over time to an appropriate form suitable for a back-end.

Speaking of sensors, two main sensor modalities are *range* sensors and *visual* sensors. From both types of measurements, point clouds and images, it is possible to extract relative transformations between measurements and 3D points of an environment. LiDAR, which stands for *light detection and ranging*, is a range sensor that is used by many approaches [5], [6]. The key principle in these approaches is to find relative transformations between LiDAR scans by using *the iterative closest point* (abbr. ICP) algorithm [7] or its variants. Having (a) calibrated visual sensor(s), it is also possible to retrieve relative transformations as well as 3D points of an environment utilizing projective geometry [8]. Among approaches that use visual sensors are [9], [10], while some approaches use both of these sensor modalities, such as [11], [12]. Furthermore, it is possible to utilize another sensor modality – *inertial measurement unit* (abbr. IMU) that gives us the acceleration and the rotation rate with respect to an inertial frame, and ultimately, to improve estimation even more [13].

Because multiple building blocks in a SLAM system should be considered and carefully thought out, it is challenging to implement such a system on its own. In the upcoming chapters, we will first examine factor graphs, as they are extensively used in state estimation for robotics. Then, we will provide a thorough overview of our SLAM and the experimental evaluation demonstrating how our loop closing algorithm improves *absolute trajectory error* (abbr. ATE) results compared to the naive loop closing detection. The paper's main contribution is the integration of our loop closing algorithm NOSeqSLAM, previously used only for the visual place recognition problem [14], in the context of SLAM.

II. FACTOR GRAPHS

Factor graphs belong to the category of bipartite graphs, so let us first define bipartite graphs. A bipartite graph is an undirected graph $(\mathcal{V}, \mathcal{E})$ in which nodes $\mathcal{V}$ can be partitioned into disjoint $\mathcal{V}_1$ and $\mathcal{V}_2$ such that an edge $\{u, v\} \in \mathcal{E}$ implies either $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ or $u \in \mathcal{V}_2$ and $v \in \mathcal{V}_1$ [15]. With a bipartite factor graph

$$F = (\mathcal{U} \cup \mathcal{V}, \mathcal{E}), \tag{1}$$

we estimate a stochastic process that consists of multiple states and measurements. $\mathcal{U}$ and $\mathcal{V}$ form a partition of nodes, while specifically, nodes from $\mathcal{U}$ are called *factors* and nodes from $\mathcal{V}$ are called *variables*. According to the definition for a bipartite graph in the context of factor graphs, this means that an edge $e_{i,l} \in \mathcal{E}$ connects a factor $\phi_i \in \mathcal{U}$ and a variable $x_l \in \mathcal{V}$. Also, in the context of factor graphs, such an edge $e_{i,l}$ indicates that x_l *factorizes* ϕ_i – hence the name "factor graphs". All variables that factorize a factor ϕ_i, thus all variables $x_k \in \mathcal{V}$ such that $e_{i,k}$ exists, are denoted as $X_l = \{x_k\} \subseteq \mathcal{V}$.

Using a factor graph F, a stochastic process can be factorized as

$$\phi(\mathcal{V}) = \prod \phi_i(X_i). \tag{2}$$

Optimizing (2) with respect to $\mathcal{V}$, an optimal set of variables $\mathcal{V}^*$ will be obtained:

$$\begin{aligned} \mathcal{V}^* &= \arg\max_{\mathcal{V}} \prod_i \phi_i(X_i) && (3)\\ &= \arg\min_{\mathcal{V}} \sum_i \|h_i(X_i) - z_i\|^2_{\Sigma_i}\,, && (4) \end{aligned}$$

This is *a nonlinear least-squares problem* that can be optimized with techniques such as the Gauss-Newton algorithm [16] and the Levenberg-Marquardt algorithm [17]. Upon the finalization of the optimization process (3), an optimal set of variables $\mathcal{V}^*$ represents the optimal states of a process being estimated.

The simplest process considered for optimization with factor graphs is called *tracking*, where we estimate states X given measurements Z. A factor graph that models tracking is shown in Fig. 1a. Then, different variable types can be introduced, i.e., we can partition variables into two subsets $\{x_i\}$ and $\{y_i\}$. Such a system is called *the switching system* and is depicted in Fig. 1b. Specific to factor graphs in SLAM, i.e., in the estimation of the robot's states, is that both variables and factors are elements of *the special Euclidean group* $SE(n)$ which represents all rigid motions in the n-dimensional space where $n \in \{2, 3\}$. This is meaningful for variables because we want to estimate the robot's states over time. In addition, a relative transformation between two states is also a rigid motion. Therefore, this is meaningful for factors, too.

Such a variant of SLAM, the simplest one, where only rigid motions are considered, is called *pose graph optimization* (abbr. PGO). Technically, in PGO, we distinguish how factors are added to a graph. If they are added utilizing an odometry procedure between two successive poses $T_i \in SE(n)$ and $T_{i+1} \in SE(n)$, we are talking about *odometry factors* $\phi_{\text{l.c.}}$. If they are added by means of a loop closure detection between non-successive poses $T_i \in SE(n)$ and $T_j \in SE(n)$, we are talking about *loop closing factors* $\phi_{\text{l.c}}$. The corresponding factor graph for PGO is shown in Fig. 1c. Furthermore, we can upgrade PGO and introduce another type of variables – *landmarks*. Then, with the third type of factors – *bearing-range* factors $\phi_{\text{b.r.}}$ – we model the relation between the robot's poses and landmarks in an environment as shown in Fig. 1d. It is easy to incorporate all these functionalities in a robotic application by using the versatile factor graphs library GTSAM [18].

III. SIMULATED 2D ENVIRONMENT

The environment in which the robot maneuvers is a 2D $3m \times 3m$ square room. Inside it, we place multiple poly-

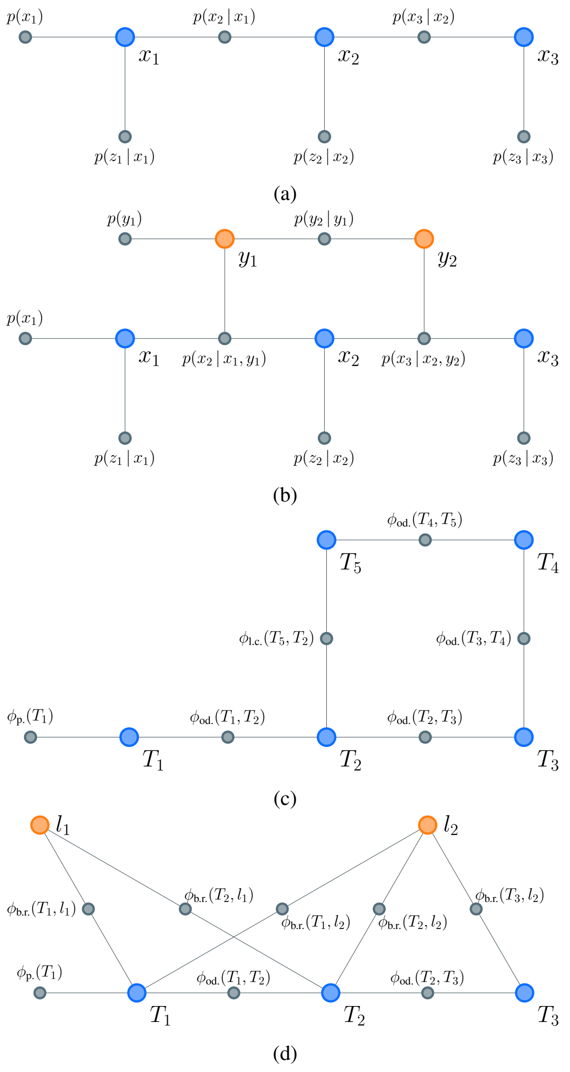
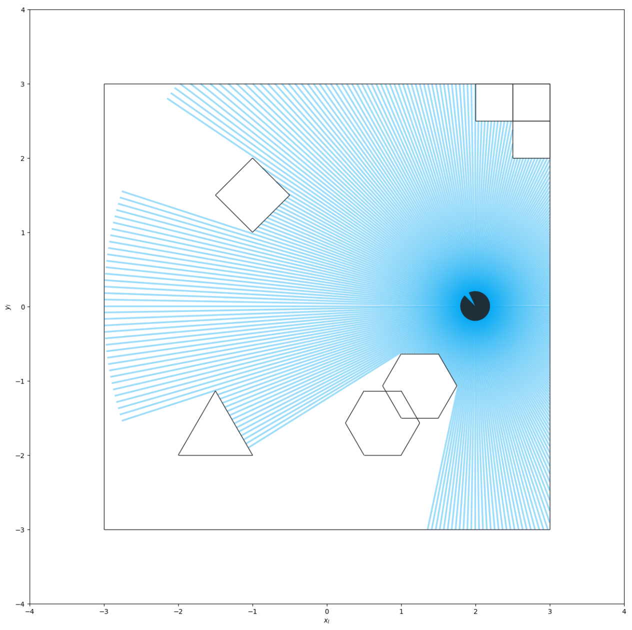


Fig. 1: Factor graphs are used for (a) tracking, (b) switching systems, (c) pose graph estimation, and (d) simultaneous localization and mapping.

gons so that each location in the environment is distinctive, which is an essential requirement for the localization task. Technically, both the room and the inner "furniture" that describes that room are defined as sequences of linear segments (gray lines in Fig. 2). We put all these segments in a single $(n_1, 2, 2)$-dimensional tensor.

The robot's sensor is also defined with linear segments. Starting from the center of the sensor's frame, we distribute the sensor's beams radially every β°_Δ from $\beta^\circ_{\text{start}}$ to β°_{end} (blue lines in Fig. 2). Similarly to the environment's segments, we put the sensor's segments into a single $(n_2, 2, 2)$-dimensional tensor. For simplicity, the sensor's coordinate frame is aligned with the robot's coordinate frame. Otherwise, we could define a relative transformation between the sensor and the robot, although it would not make any difference in the evaluation.

Following the line-line intersection formula, given two



(a)

Fig. 2: The 2D simulated environment. The room, its inner shapes (gray lines), and the sensor's beams are defined as linear segments. Using the line-line intersection formula, we can find their intersection and simulate range measurements.

linear segments L_1 and L_2

$$L_1 = (x_1, y_1) + t(x_2 - x_1, y_2 - y_1),\ t \in [0,1], \tag{5}$$

$$L_2 = (x_3, y_3) + u(x_4 - x_3, y_4 - y_3),\ u \in [0,1], \tag{6}$$

we find their intersection by calculating

$$t^* = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{7}$$

and

$$u^* = \frac{(x_1 - x_3)(y_1 - y_2) - (y_1 - y_3)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}. \tag{8}$$

If $0 \leq t^* \leq 1$ and $0 \leq u^* \leq 1$, then L_1 and L_2 intersect at

$$\begin{aligned}(x^*, y^*) &= (x_1 + t^*(x_2 - x_1), y_1 + t^*(y_2 - y_1)) \\ &= (x_3 + u^*(x_4 - x_3), y_3 + u^*(y_4 - y_3)).\end{aligned} \tag{9}$$

Intersections are calculated at each time step between the environment's and sensors' segments. This way, we can obtain the Euclidean distance from the start of each beam to a line of the environment it points to and, this way, simulate a range sensor. An important implementation detail is that, as we put both sets of segments into tensors, we can calculate all these intersections via matrix operations in a vectorized way[1].

The robot is a differential mobile robot drive with implemented forward and inverse kinematics models as defined in [20]. The wheel radius is $r = 5cm$, and the wheelbase length is $l = 10cm$. Although the relative transformation between two successive poses in two timesteps can be obtained via ICP, a more robust variant is to provide an ICP algorithm with an estimation of what this transformation could be. Such an estimation can be obtained with wheel encoders. In real-world experiments, encoders are attached to DC motors and provide us with the rate of change between two successive angles measured, i.e., with angular velocities. Each encoder produces a specified number of discrete ticks in a single revolution, i.e., the number of cycles per revolution cpr.

We simulate the encoders' readings as follows. Say that inverse kinematics gives us angular velocities $\dot{\varphi}_{\text{left}}$ and $\dot{\varphi}_{\text{right}}$ that should be applied to motors in order to attain a rate of change in the robot's position and orientation $\dot{\xi}_I$ expressed in the global coordinate frame I. For clarity, let us examine a single angular velocity $\dot{\varphi}$ no matter what wheel it is applied to. The difference between the current and previous number of the encoder's cycles is

$$\Delta c = c_{i+1} - c_i = \left\lfloor \frac{\dot{\varphi} \cdot \Delta t \cdot cpr}{2\pi} \right\rfloor \tag{10}$$

where Δt is the time between two iterations. Conversely, a measured-by-a-wheel-encoder angular velocity will be

$$\dot{\tilde{\varphi}} = \frac{\Delta c \cdot 2\pi}{\Delta t \cdot cpr} \frac{rad}{s}. \tag{11}$$

To make the simulation more realistic, we can additionally add a normally distributed noise $\epsilon \sim \mathcal{N}(\mu, \sigma)$ to (11). We evaluate (10) and (11) for both the left and the right wheel and this way obtain $\dot{\tilde{\varphi}}_{\text{left}}$ and $\dot{\tilde{\varphi}}_{\text{right}}$. Then, these values are mapped via forward kinematics into $\dot{\tilde{\xi}}_{i+1}$ that is further used to obtain the robot's pose in the $(i+1)$-th iteration as

$${}^I\tilde{\xi}_{i+1} = {}^I\tilde{\xi}_i + \dot{\tilde{\xi}}_{i+1} \cdot \Delta t. \tag{12}$$

The relative transformation between ${}^I\tilde{\xi}_i$ and ${}^I\tilde{\xi}_{i+1}$, i.e., ${}^i\tilde{T}_{i+1} \in SE(2)$, is used as an initial guess for ICP odometry between measurement scans obtained in the i-th and $(i+1)$-th iterations. Then, an ICP-obtained relative pose ${}^iT_{i+1} \in SE(2)$ is added as a factor $\phi_{\text{od.}}$ between nodes T_i and T_{i+1} into the factor graph.

After the world and the robot with its sensors are defined, the next building block is the path planning subsystem, which should navigate the robot toward a goal while simultaneously avoiding obstacles. This system is based on the paper [21] where we represent the robot's behavior as *a finite-state automaton*. According to its current state and the most recent measurement scan, the robot, as an automaton, can have the states: "move towards a goal", "avoid obstacles", and, "stand still". If the robot moves toward a goal and it is measured that an obstacle is near, i.e., if a sensor's beam measures a substantially small range, then the state shifts to the obstacle avoidance mode. While in the obstacle avoidance state, the robot is expected to move away from an obstacle by a safe amount and start to move toward a goal. If it is detected that the robot reached a goal or unhappily hit a wall, then it will stand still.

A goal's position ${}^Ig$ in the global frame should be defined. Then, we can construct a vector from the robot's

[1]Specifically, using the Numpy library [19].

frame origin to a goal. Trying to attain this orientation will eventually bring the robot to a goal. This automaton can also be generalized for multiple goals, as we did, where – while not all goals are reached, the robot should move toward the next goal. Multiple goals are defined for the purpose of the loop closing detection. Goals are: $(-2.5, -2.5)$, $(2.5, -2.5)$, $(2, 2)$, $(-2.5, 2.5)$, once again $(-2.5, -2.5)$, once again $(2.5, -2.5)$ and once again $(2.0, 2.0)$ Similarly, obstacle avoidance is achieved in a way that multiple vectors, *counterbalanced* with respect to their corresponding beams, are created, then aggregated together in a single vector, which orientation the robot will try to achieve, and this way avoid obstacles.

As the robot moves, it is inherent that the drift will be accumulated. However, once when the robot arrives in the same place, thus two times in the same goal, and it is detected by the loop closing detection method that it arrived there once again, a new drift-free relative transformation between these measurements can be taken, and added as a loop closing factor. According to this, the ICP subsystem, implemented with the open-source libpointmatcher library [22], has two roles. First, it will give us a relative transformation ${}^{i}T_{i+1} \in SE(2)$ between measurement scans $m_i \in \mathbb{R}^n$ and $m_{i+1} \in \mathbb{R}^n$. Then in the factor graph, we use it as an odometry factor $\phi_{\text{od.}}(T_i, T_{i+1})$. Also, once when in a j-th iteration the robot arrives in a place that has been seen in an i-th iteration, i.e., the loop closing subsystem detects such a place, the ICP will give us a relative transformation ${}^{j}T_i \in SE(2)$ between $m_j \in \mathbb{R}^n$ and $m_i \in \mathbb{R}^n$ being used as a loop closing factor $\phi_{\text{l.c.}}(T_j, T_i)$.

The final ingredient of our system is the loop closing detection method. It is based on our previous work NOSeqSLAM [14] where places are matched according to their visual images. The scenario it has been used for – *visual place recognition* – is another research topic on its own, and we had to modify it for the purpose of SLAM. In visual place recognition, we have two image streams of the same route, namely a query dataset $\mathcal{Q}$ and a reference dataset $\mathcal{R}$. Then, each image $I_{q_i} \in \mathcal{Q}$ is compared against each image $I_{d_j} \in \mathcal{R}$. The goal is to match images properly according to hand-labeled ground truth data $\mathcal{GT}$, i.e. to find matches $(I_{q_i}, I_{d^*_{q_i}}) \in \mathcal{Q} \times \mathcal{R}, \forall I_{q_i} \in \mathcal{Q}$ so that $I_{d^*_{q_i}}$ belongs to a ground-truth set $\mathcal{GT}_{q_i} \subseteq \mathcal{R}$. In SLAM, we have a single stream of measurements,[2] Be it a stream of LiDAR scans or visual images. LiDAR scans are, by default, n-dimensional vectors. Global descriptor techniques such as *histogram of oriented gradients* [23] or *feature maps extraction* with *deep convolutional neural networks* can map grayscale/RGB images into n-dimensional vectors. Therefore, we can denote all measurements in a stream, be it from a visual sensor or a LiDAR, as

$$\mathcal{M} = \{m_1, \ldots m_i, \ldots, m_{|\mathcal{M}|}\} \subset \mathbb{R}^n. \tag{13}$$

[2]Unless multiple sensor modalities are used. However, the recording of all these streams will take place at the same time. Therefore, we will not have two different streams of the same route recorded at different times by a single sensor.

TABLE I: Positional and rotational absolute trajectory errors defined with (14) and (15) for different setups in the simulated environment.

Setup	$\varepsilon_{\text{pos.}}$ [m]	$\varepsilon_{\text{rot.}}$ [°]
ICP odometry + NOSeqSLAM	**0.02836**	**0.59485**
ICP odometry + Naive	0.07611	1.40690
ICP odometry (w/o loop closing)	0.25755	5.01363
Wheel odometry (w/o loop closing)	0.58594	12.19906

The idea behind NOSeqSLAM is to measure the cosine similarity between a measurement m_j together with its temporal predecessors $\{m_{j-1}, \ldots, m_{j-d_s+1}\}$ and a measurement m_i together with its temporal predecessors where $i << j$ by utilizing *directed acyclic graphs*. For more information, confer [14]. Ultimately, the loop closing detection subsystem will detect if, given a most recent measurement m_j there are loop closing candidates. Say that m_i^* is a measurement with the highest NOSeqSLAM measure for m_j. Then, as already mentioned, the ICP will calculate the relative transformation between these two measurements and a loop closing factor will be added. Naively, we can perform loop closing detection where m_j is compared solely with m_i without incorporating their temporal neighbors. In the upcoming section, we will see that NOSeqSLAM outperforms such a naive loop closing detection.

IV. Experimental Results

In this section, we will present the quantitative and qualitative results of our evaluation. Quantitative results are expressed with *the absolute trajectory error* (abbr. ATE) [24] and with positional and orientational error plots. A trajectory refers to all estimated states of the robot. Let $T_i \in SE(2), \forall i \in \{1, \ldots, N\}$ denote the ground truth robot poses and let $T_i', \forall i \in \{1, \ldots, N\}$ denote the corresponding estimated poses. Then, the positional trajectory error is defined as

$$\varepsilon_{\text{pos.}} = \left(\frac{1}{N} \sum_{i=1}^{N} \|p_i - p_{i'}\|^2 \right)^{\frac{1}{2}}, \tag{14}$$

while the orientational trajectory error is defined as

$$\varepsilon_{\text{rot.}} = \left(\frac{1}{N} \sum_{i=1}^{N} \|\sphericalangle \left(R_i R_{i'}^T \right) \|^2 \right)^{\frac{1}{2}}. \tag{15}$$

$(R_i, p_i) \in SO(2) \times \mathbb{R}^2$ and $(R_i', p_i') \in SO(2) \times \mathbb{R}^2$ are the corresponding rotation matrix and position vector for T_i and T_i', respectively. $\sphericalangle(\cdot)$ denote the angle-axis angle extraction [25].

A. Simulated Environment

Errors (14) and (15) were evaluated for different estimation setups, and the results are represented in Table I. First, we run the robot in the simulated environment with the ICP and loop closing detection subsystems turned off. Therefore, the estimated trajectory in such a setup is solely based on (12). In the next experiment, we ran the robot with the ICP turned on but without loop closing detection. Then, we turned on the naive loop closing detection, and in

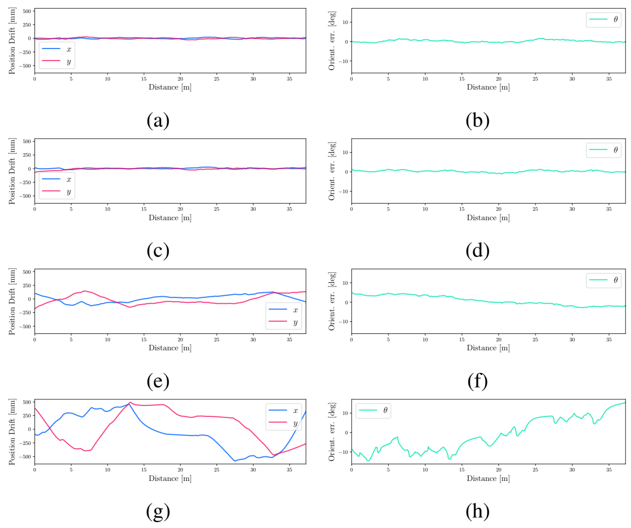


Fig. 3: Positional errors (left) and orientational errors (right) for (a, b) ICP odometry + NOSeqSLAM, (c, d) ICP odometry + Naive loop closing detection, (e, f) ICP odometry without loop closing detection and (g, h) wheel odometry only setups. Plots created with [24].

the final run, instead of the naive loop closing detection, NOSeqSLAM was utilized. As expected, the worst run, i.e., the highest ATE, came from the wheel-odometry-only setup, and vice versa; the lowest ATE came from the ICP odometry setup with ours NOSeqSLAM. From the results, we can conclude that the wheel odometry performs subpar compared to the ICP odometry, which is reasonable – a wheel odometry measurement is passed as an initial guess to ICP. Also, it is visible that our NOSeqSLAM loop closing detection outperforms the naive loop closing detection. The corresponding plots that show how an error differs with respect to the length of a traversal are shown in Fig. 3.

Qualitatively, the better an estimation of trajectory is, the better a map that was built, and, as already mentioned, the smaller the ATE, i.e., estimated and ground truth trajectories are alike. This is visible in Fig. 4. Blue dots were measured as seen by the estimated orange robot. The worst map and the estimated robot's pose are for the wheel odometry-only run (Fig. 4d). Then, the map and the estimated pose are, to an extent, better with the ICP subsystem turned on (Fig. 4c). The significant difference between a built map and an estimated pose is noticed when the loop closing detection kicks in. Once again, because NOSeqSLAM considers temporal neighbors, it is expected that it will perform better (Fig. 4a) than the naive matching (Fig. 4b).

B. Real-World Dataset

After finishing experiments in the last subsection, we were curious about how our system would perform on a real-world dataset. We picked the Intel Research Lab dataset [26]. Unfortunately, this dataset lacks a ground-truth trajectory. However, the raw odometry-only trajectory and the corrected trajectory obtained with the FastSLAM, a SLAM system presented in the same paper, are provided.

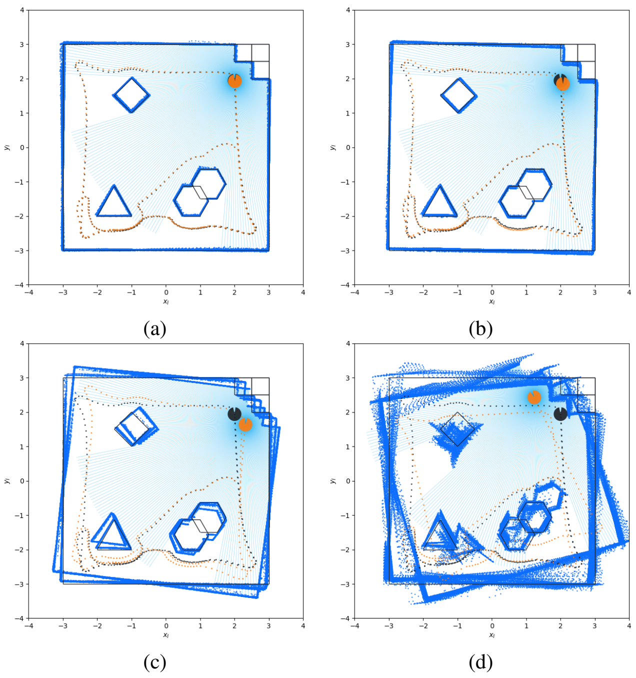


Fig. 4: Visualization of maps that were built (blue dots) and an estimated robot's pose (orange robot) with respect to the ground truth robot's pose (gray robot) and the ground truth world (gray lines) for different setups.

TABLE II: Positional and rotational absolute trajectory errors defined with (14) and (15) for different setups in the Intel Research Lab dataset with respect to the trajectory obtained with FastSLAM [26].

Setup	$\varepsilon_{\text{pos.}}$ [m]	$\varepsilon_{\text{rot.}}$ [°]
ICP odometry + NOSeqSLAM	**0.33030**	**1.17689**
ICP odometry (w/o loop closing)	0.80209	2.25328
Wheel odometry (w/o loop closing)	11.49560	94.49847

Qualitative performance is visible in Fig. 5. A map built with the wheel-odometry-only system (Fig. 5d) is bad and does not reflect the shape of the lab. By turning on the ICP module of our system, a more coherent map is obtained (Fig. 5c), and the lab's shape begins to appear. Once NOSeqSLAM is deployed, the shape is even less disheveled (Fig. 5b) – almost like the map obtained with FastSLAM (Fig. 5a).

Because a ground-truth trajectory is non-existent, we declared the FastSLAM trajectory as the ground-truth trajectory, and we evaluated (14) and (15) for different setups like before. These results are presented in the Table II. As expected, the ICP + loop closing system outperforms the loop closing-only system, while the wheel odometry-only system performs subpar with respect to these two.

V. CONCLUSION

In this paper, we have presented how to build a range sensor-based SLAM system. First, we looked at how to build an environment where the robot operates. Then, we simulated the robot itself by implementing its forward and inverse kinematics models along with noisy encoder readings. Onwards, the path planning was established by considering the robot a finite-state automaton. Rigid

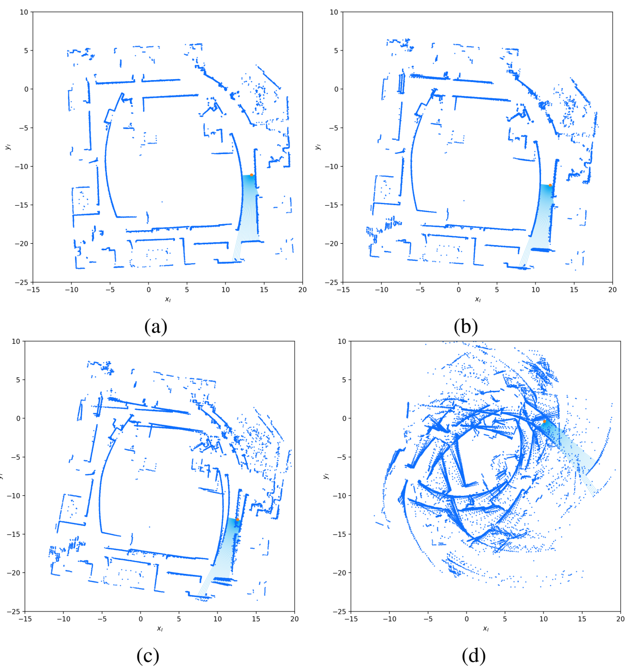


Fig. 5: Visualization of maps that were built (blue dots) and an estimated robot's pose (orange robot) on the Intel Research Lab dataset for different setups.

motions between the robot's movements were obtained through either wheel odometry or ICP odometry. Finally, loop closures were detected either naively or with our NOSeqSLAM method. With the experimental evaluation in the simulated environment, it was shown how the ICP odometry in combination with the NOSeqSLAM loop closing detection outperforms both quantitatively and quantitatively other setups. Also, it was shown how our system performs on par with a real-world dataset compared to another SLAM system – FastSLAM.

REFERENCES

[1] S. Thrun and J. J. Leonard, *Simultaneous Localization and Mapping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 871–889. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_38

[2] K. Lenac, J. Cesic, I. Markovic, I. Cvisic, and I. Petrovic, "Revival of filtering based SLAM? Exactly sparse delayed state filter on Lie groups," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-September, pp. 1012–1018, 2017.

[3] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] J. Engel, J. Sturm, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1449–1456, 2013.

[5] E. Mendes, P. Koch, and S. Lacroix, "Icp-based pose-graph slam," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2016, pp. 195–200.

[6] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "Ct-icp: Real-time elastic lidar odometry with loop closure," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5580–5586.

[7] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992, range Image Understanding. [Online]. Available: https://www.sciencedirect.com/science/article/pii/026288569290066C

[8] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[9] X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: Direct Sparse Odometry with Loop Closure," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2198–2204, 2018.

[10] I. Cvisic, I. Markovic, and I. Petrovic, "SOFT2: Stereo Visual Odometry for Road Vehicles Based on a Point-to-Epipolar-Line Metric," *IEEE Transactions on Robotics*, 2022.

[11] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, no. June, pp. 2174–2181, 2015.

[12] Y. An, J. Shi, D. Gu, and Q. Liu, "Visual-LiDAR SLAM Based on Unsupervised Multi-channel Deep Neural Networks," *Cognitive Computation*, vol. 14, no. 4, pp. 1496–1508, 2022.

[13] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4290–4297.

[14] J. Maltar, I. Marković, and I. Petrović, "Visual place recognition using directed acyclic graph association measures and mutual information-based feature selection," *Robotics and Autonomous Systems*, vol. 132, p. 103598, 2020.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[16] W. R. Esposito and C. A. Floudas, *Gauss–Newton method: Least squares, relation to Newton's method.* Boston, MA: Springer US, 2009, pp. 1129–1134. [Online]. Available: https://doi.org/10.1007/978-0-387-74759-0_197

[17] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, G. A. Watson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116.

[18] F. Dellaert, R. Roberts, V. Agrawal, A. Cunningham, C. Beall, D.-N. Ta, F. Jiang, lucacarlone, nikai, J. L. Blanco-Claraco, S. Williams, ydjian, J. Lambert, A. Melim, Z. Lv, A. Krishnan, J. Dong, G. Chen, K. Chande, balderdash devil, DiffDecisionTrees, S. An, mpaluri, E. P. Mendes, M. Bosse, A. Patel, A. Baid, P. Furgale, matthewbroadwaynavenio, and roderick koehle, "borglab/gtsam," May 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5794541

[19] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[20] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. The MIT Press, 2011.

[21] M. Egerstedt, "Control of Autonomous Mobile Robots," *Handbook of Networked and Embedded Control Systems*, pp. 767–778, 2005.

[22] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP Variants on Real-World Data Sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, Feb. 2013.

[23] R. K. McConnell, "Method of and apparatus for pattern recognition," Patent US4 567 610A, 1982.

[24] Z. Zhang and D. Scaramuzza, "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry," *IEEE International Conference on Intelligent Robots and Systems*, pp. 7244–7251, 2018.

[25] K. J. Waldron and J. Schmiedeler, *Kinematics*. Cham: Springer International Publishing, 2016, pp. 11–36. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_2

[26] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 1, 2003, pp. 206–211 vol.1.