RL-PGO: Reinforcement Learning-Based Planar Pose-Graph Optimization

Nikolaos Kourtzanidis and Sajad Saeedi, Member, IEEE

Abstract—In this letter, we present to the best of our knowledge, the first deep reinforcement learning (DRL) based 2D pose-graph optimization (PGO). We demonstrate that the pose-graph optimization problem can be modeled as a partially observable Markov Decision Process. The proposed agent outperforms state-of-the-art solver g2o on challenging instances where traditional nonlinear least-squares techniques may fail or converge to unsatisfactory solutions. Experimental results indicate that iterative-based solvers bootstrapped with the proposed approach allow for significantly higher quality estimations.

Index Terms—Machine learning, optimization, robotics.

I. INTRODUCTION

POSE-GRAPH optimization (PGO), with applications in structure from motion (SfM) and simultaneous localization and mapping (SLAM), is a process where relative pose measurements – typically obtained from cameras, laser ranger, IMUs, or wheel odometry – are optimized to remove processing noise and provide an accurate estimate of a robot’s poses [1]. Factor graphs [2] are often used to perform the optimization probabilistically. There exist instances of PGO problems which consist of highly non-convex cost functions, attributed to system nonlinearity, high levels of noise corruption, large inter-nodal distance spacing, and incorrect data associations from the front end [3]. This further results in cost functions with large valleys [4], which may cause classical gradient-based approaches to fail, as depicted in Fig. 1-(c).

This letter presents a novel reinforcement learning (RL)-based method that can perform exceptionally well on graphs with poor initial guesses (See Fig. 1-(b,d)). In certain situations, obtaining ground truth trajectory labels in the supervised SLAM algorithms may be laborious and expensive. RL algorithms can circumvent this problem by interacting with their environment. The contributions of this letter are: (I) The first deep RL model to learn a policy that predicts the optimal orientation retraction [5] from pose-graph observations for PGO. It is composed of a neural graph encoder [6] and a recurrent-based soft actor-critic (SAC) [7] agent. (II) A planar pose-graph environmental framework based on [8] and GUI for visualization during evaluation, publicly available at https://sites.google.com/view/rl-pgo, and (III) Extensive experiments on simulated and real-world benchmarks illustrate the accuracy and generalizability of the proposed agent.

In the rest of this letter, Section II provides the related works. Section III presents the problem formally. Section IV presents our method. Sections V and VI present the results and conclusions.

II. RELATED WORK

Classical solvers, such as g2o [9], Ceres [10], and GTSAM [11], are commonly used for nonlinear optimization of problems represented as graphs. These solvers employ second-order methods that iteratively linearize the nonlinear least-squares objective function and solve the corresponding least-squares equations until convergence [2]. Initialization strategies can enhance their convergence, as shown in [12], [13], [14], [15], [16]. However, their convergence to global optima is not guaranteed, particularly when the problem exhibits non-convexities [3]. To address this, direct and indirect linear solvers, including preconditioned conjugate gradient (PCG), SuiteSparseQR [17], and CHOLMOD [18],...
have gained interest due to reduced computational complexity compared to iterative methods relying on matrix computations during linearization. Semidefinite relaxation (SDR) technique is also applicable to various non-convex quadratically constrained quadratic programmings (QCQPs), including pose SLAM and Landmark SLAM. Notable such contributions include [6], [19], [20], [21], [22], [23].

There has also been a plethora of works involving Graph Neural Networks for pose-graph problems, such as rotation problems [24], [25], [26], [27]. Further, several recent papers incorporate deep neural networks into state estimation and problems [24], [25], [26], [27].

There has also been a plethora of works involving Graph Neural Networks for pose-graph problems, such as rotation problems [24], [25], [26], [27]. Further, several recent papers incorporate deep neural networks into state estimation and SLAM pipelines [28], [29]. Most of the methods require ground truth labels in supervised/semi-supervised manner. RL methods revolve around the concept of learning from interaction [30].

III. PROBLEM STATEMENT

Here the problem formulation for planar pose-graph optimization (PGO) is explained. Assume there are $n$ robot poses with $m$ relative pose measurements, $m \geq n$. The measurements describe the relative translation and rotation between poses. These are typically computed from sensors such as wheel encoders, cameras, or inertial measurement units and are noisy. The goal of PGO is to return the optimal translation and orientation estimate of poses which best fit the measurements. This is done by building a cost function and minimizing it. Assume $x \in \mathbb{SE}(2)$ is the set of poses. Each pose, $x_i = [t_i, R_i]^T$, $i \in \mathbb{R}^2$ and $R_i \in \mathbb{SO}(2), \forall i \in 1...n$ denote the absolute translation and orientation in a global frame. The measurements between poses $i$ and $j$ are $\bar{t}_{ij} \in \mathbb{R}^2$ and $\bar{R}_{ij} \in \mathbb{SO}(2)$, and are impacted by independent Gaussian noise $\Sigma_{ij} = \text{diag}(\sigma_t, \sigma_R)$. We form a directed graph $G$, where nodes $i, j$ are poses and edge $(i,j) \in G$ is the measurement. PGO minimizes the following cost function by finding the optimal $t_i^*, R_i, i = 1...n$ [3] :

$$\sum_{(i,j) \in \mathcal{E}} \left\| R_i (t_j - t_i) - \bar{t}_{ij} \right\|_{\sigma_t}^2 + \left\| (R_j - R_i) - \bar{R}_{ij} \right\|_{\sigma_R}^2,$$

where $\| \cdot \|_2$ is the Mahalanobis distance. A pose-graph has 5 parameters: 1) the number of poses $n$, 2) orientation measurement uncertainty $\sigma_R$, 3) translation measurement uncertainty $\sigma_t$, 4) inter-nodal distance spacing $d$, and 5) probability of loop closures $lc$ [3]. Estimating rotations first allows for greater convergence guarantees and closed-form solutions [14].

IV. PROPOSED APPROACH

Fig. 2 shows the structure of the proposed method. First, given the input graph in the ‘observation’ block, the orientation residual is passed into the encoder network. The ‘encoder’ then returns a highly expressive low dimensional input representation of the state. Once the state is then provided to the ‘recurrent-based soft actor-critic (SAC)’ policy’ network, the optimal retractions on neighbouring poses are applied. The final orientation estimate returned by the agent is used in a least-squares translation estimation, based on [3]. Next, we present A) the Markov decision process (MDP) framework, B) observations encoding, and C) the policy network.

A. Markov Decision Process (MDP) Framework

To solve the 2D PGO problem with RL, we first define an MDP, characterized by the following components:

1) Objective: The objective is to optimize 2D pose-graphs using an RL agent. This agent aims to make optimal decisions to minimize orientation residuals and improve the initial quality of pose estimates followed by a least squares translation solver.

2) State: The state space consists of two parts: i) The encoded history of orientation residuals at each time step (discussed in Section IV-B), and ii) The angular difference between the observed and measured orientations of the current edge:

$$\log\left(\tilde{R}_{ij}^T R_{ij}\right),$$

where $\tilde{R}_{ij} \equiv R_{ij}^T, (i,j) \in \mathcal{E}$ and Log(.) represents the logarithmic mapping of the matrix Lie group [5].

3) Action: The agent’s actions are defined as retraction operations on neighbouring poses along an edge. These actions are parameterized by a vector $a_i$ that influences the retraction. For each step-by-step transition along the edges, the output from the policy is passed through the $\tanh$ activation layer and then multiplied by a user-defined action range factor. The resultant vector is used to apply a retraction on the neighbouring poses connected by the edge, in which the agent resides at that particular instance in time: $R_i \oplus a_i[l] \oplus R_j \oplus a_j[1]$. Here $\oplus$ is the retraction operation [5]. In one cycle, each pose is perturbed twice, assuming no loop closures. At the end of an episode, the final pose translation estimates are recovered via linear least-squares [3].

4) Reward: At each time step, the agent receives a reward calculated based on the difference between the predicted and actual orientation. The reward function encourages the agent to reduce orientation residuals between poses. The reward is:

$$\text{Reward} = \begin{cases} 
\frac{100}{1+OC} + 25 & \text{if } OC \text{ decreases by a factor of } 10 \\
\frac{100}{OC} & \text{otherwise.} 
\end{cases}$$

Fig. 2. Agent (red), cycling through every edge in the pose-graph and applying optimal retractions to the orientation of neighbouring poses.
OC, orientation cost, the squared sum of chordal distances, is
\[ OC = \sqrt{\sum_{i,j} (\hat{R}_{ij} - R_{ij})^2}. \]  
(4)
\[ \| \cdot \|_F \text{ is the Frobenius norm. The term } +25 \text{ helps increase } \]
the reward difference when OC falls less than 0.01, while decreasing by a factor of 10, leading to optimal solutions.

5) Termination of Episode Criteria: As shown in Fig. 2, the agent’s location along the graph is highlighted in red. A transition from one edge to another is a ‘step’. A cycle is when an agent traverses every edge in the graph once. This value determines the number of times the agent traverses across each edge, and serves as an episode termination criterion, e.g., an episode can terminate after a user-defined number of cycles.

B. Graph Encoder Architecture
For the graph encoder, we adopt the architecture proposed in [6]. The nodes of each graph input store a cost feature and the absolute orientation \( R_i \) of the node itself. We utilize an augmented message passing function which is dependent on orientation components only, as opposed to both translation and orientation as depicted in Eq. (5). Once the input pose-graph observation passes forward, a two-step process occurs. The first step is the message passing step, which involves computing the Frobenius norm of the absolute orientations shared by each edge, where \( \beta \) is a learnable weight:
\[ msg_{i,j} = \beta \times \| R_i R_{ij} - R_j \|_F. \]  
(5)
Proceeding the computation of messages for each of the connected nodes, an aggregate sum is then stored in the cost feature associated with the corresponding node itself,
\[ cost_j = \sum_{i} msg_{i,j}. \]  
(6)
The mean of all cost features from graph observation at time \( t \) is then passed through a linear layer and concatenated with the angular difference between the observed and measured orientations, corresponding to the agent’s edge location computed from Eq. (2). The output dimension of the linear layer is user-defined, here set to 20 floats. This resultant state \( s_t \) is passed into the recurrent SAC, for an action to be applied.

C. Recurrent SAC
Soft actor-critic (SAC) [31], [32] is an RL algorithm, learning an optimal policy \( \pi \), with an augmented objective:
\[ \mathbb{E}_{\pi} \left[ \sum_{t} \gamma^t [R(s_t, a_t) + \alpha H(\cdot | s_t)] \right]. \]  
(7)
where \( \gamma, a_t, s_t, \pi \) represent the discount factor, action, state, and the policy respectively. The temperature parameter \( \alpha \) dictates the relative importance between the entropy \( H \) and reward \( R \). Thus, the agent is encouraged to explore the state space and unseen trajectories, which can speed up learning and prevent suboptimal convergence. Our implementation is based on [7]. After observing the current state and history of previous state-action pairs, the recurrent SAC policy network applies the optimal retractions to each of the poses shared by the edge in which the agent is located at for every time step.
The recurrent SAC algorithm utilizes two networks, the Q-function and the policy. The Q-function \( Q_{\phi} \) is parameterized by \( \phi \), and the policy \( \pi_{\theta} \) by \( \theta \) which is modeled as a Gaussian with mean \( \mu_s \) and covariance as depicted in Fig. 2. To mitigate bias, two action-value functions \( Q_{\phi_1} \) and \( Q_{\phi_2} \) are trained. The corresponding target action-value functions are shown with \( \bar{Q}_{\phi_1} \) and \( \bar{Q}_{\phi_2} \). The 2D action vector is \( a_t = \pi_\theta (\epsilon_t; s_t, a_{t-1}, z_t) \), where \( \epsilon_t \) is an input noise vector sampled from a standard normal distribution, and \( z_t \) represents the internal (hidden) state. Both policy and Q value architectures follow the structure in [33]. The graph encoder \( EN_{\beta} \) is parameterized by \( \beta \), trained end-to-end with the policy network. The recurrent branch further allows the agent to make informed optimal decisions on the next actions from the previous state and action pairs, by the internal state representation [33] \( z_t = z(h_t) \) modeled by a Long Short-Term Memory (LSTM). \( h_t = [a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}, \ldots ] \) is the history of state/actions.

Further details of the previous works in which this proposed agent was inspired from can be referred to in [33] and [34]. The final algorithm is listed in Algorithm 1 and the gradient update procedure is summarized as \( (i = 1 \ldots N) \) is number of episodes, and \( t = 1 \ldots T \) is the number of steps in each episode. \( N \) and \( T \) are the total number of episodes and steps, respectively. \( G \) is the pose-graph. \( D \) is the replay buffer.

1) Sample states, actions, rewards, and hidden states from the replay buffer for each episode in the minibatch:
\[ s_{ti}, a_{ti}, a_{ti-1}, r_{ti} \text{ for } t = 1 \ldots T, \text{ and } z_{t0}, \ldots, z_{T0} \text{ for each episode trajectory in the minibatch.} \]
2) Evaluate current actions, \( \hat{a}_{ti} \), using policy \( \pi_\theta \):
\[ \log \pi_\theta (\hat{a}_{ti} | s_{ti}, a_{ti-1}^{t-1}, z_{t0}^{t-1}) \]
3) Update temperature parameter \( \alpha \) with optimization and inverse log operation (\( \bar{H} \) is target entropy), minimizing:
\[ L(\alpha) = \frac{\log(\alpha)}{\bar{H}} \sum_i \sum_t [\log \pi_\theta (\hat{a}_{ti} | s_{ti}, a_{ti-1}^{t-1}, z_{t0}^{t-1}) + \bar{H}] \]
4) To mitigate bias, infer action-value functions:
\[ Q_{\phi_1}(s_t, a_t, d_{t-1}, \sigma_{t-1}) \]
\[ Q_{\phi_2}(s_t, a_t, d_{t-1}, \sigma_{t-1}) \]
\[ Q_{\phi_3}(s_{t+1}, a_{t+1}, d_{t+1}, \sigma_{t+1}) \]
\[ Q_{\phi_4}(s_{t+1}, a_{t+1}, d_{t+1}, \sigma_{t+1}) \]
\[ Q_{\phi_5}(s_{t+1}, a_{t+1}, d_{t+1}, \sigma_{t+1}) \]

5) Compute estimated target values:
\[ y_t = r_t + \gamma Q_{\phi_{\text{min}}} - \alpha \log \pi_{\theta}(a_{t+1} | s_{t+1}, d_{t+1}, \phi_{\text{true}}) \]

6) Minimize the network loss function for Q-functions:
\[ L(Q_{\phi}) = \frac{1}{2} \{ \text{MSE}(Q_{\phi}, y_t) + \text{MSE}(Q_{\phi}, y_t) \} \]

7) Recompute Q-function values using all currently evaluated actions \( \phi \) as input.

8) Compute policy network loss function:
\[ L(\pi_{\theta}) = \frac{1}{NT} \sum_i \sum_t \left[ \alpha \log \pi_{\theta}(a_t | s_t, d_{t-1}, \phi_{\text{true}}) - Q_{\phi_{\text{min}}} \right] \]

9) Define the unbiased policy objective loss function:
\[ L(\pi_{\theta})_{\text{unbiased}} = L(\pi_{\theta}) + Q_{\phi_{\text{min}}} \]

10) Formulate the joint objective loss function to update network parameters (w is the actor-critic joint loss weight):
\[ L_{\text{joint}} = L(Q_{\phi}) + wL(\pi_{\theta})_{\text{unbiased}} \]

11) Update target Q-function:
\[ \tilde{\phi}_t = \tau \phi_t + (1 - \tau) \phi_t \]

V. EXPERIMENTAL RESULTS

In this section, the proposed method is evaluated and compared against the gradient-based g2o [9] Gauss-Newton, Levenberg-Marquardt iterative solvers, as well as the globally certifiably SE-Sync [20] are provided. As proven in [3], the non-convexity of a given problem instance is related to the ratio of orientation to translation uncertainty \( \frac{\sigma_R}{\sigma_T} \) and the squared sum of all measurement distances. In Section V-B and V-C, we conduct analytical tests similarly done and inspired by [3], where comparisons were made against Gauss Newton 100 iterations (GN100, i.e., upon convergence) in their evaluations. The quality of estimation provided by our approach is assessed under the influence of the adjustable environmental parameters and further illustrates the effectiveness of the proposed approach under challenging scenarios. A head-to-head comparison with the Levenberg-Marquardt solver is also provided on standard real-world and synthetic benchmarks in Section V-D. Results for SE-Sync with uniformly random initialization are also provided in each test case, and the additional optimization parameters were left as the recommended default values provided by the package.

We train five separate agents, and for all evaluations depict results for the agent which performed best in the testing environment. Each of the agents is trained on small pose-graph environments of size \( n = 20 \) randomly sampled from their assigned environmental Gaussian noise distributions every episode. The agents are then evaluated on synthetic and real-world graphs, much larger in size and unseen during training to demonstrate generalizability. The five training environment details are indicated in Table I.

A. Implementation and Training Details

For the policy and Q-value network, all related layers consist of 512 fully connected units followed by 512 LSTM units in the recurrent branch. Kaiming Initialization [35] was employed on both Q-function and policy network weights. Rectified non-linearity (ReLU) were used for all hidden layers. Adam optimizer with a learning rate of 3.0E−04 was the chosen method for gradient update, the size of the minibatch was set to \( N = 128 \), the actor-critic joint loss weight parameter \( w \) was set to 1, the expected target entropy was chosen as \( -2 \), and the discount factor \( \gamma = 1.0 \). All Q-function and policy networks are updated after every episode with target networks updated by an exponential moving average, with \( \tau = 1.0E−02 \). The networks are trained on a single Nvidia GeForce RTX 3090 GPU with 24GB memory. For best performance, the final policy was set to sample deterministic actions \( \epsilon = 0 \). The trainings were set to terminate after 1000 episodes as it was observed that there was no further increase in performance, which was verified by conducting several experiments and observing the cumulative rewards. In common reinforcement learning training pipelines, a sliding window average of the previous 100 rewards can be compared such that if the difference is less than a predefined threshold, the agent’s performance is said to have converged, indicated by a plateau in the cumulative reward plot.

B. Effect on Measurement Uncertainty Ratio

In this first analytical case study, comparisons are performed against SE-Sync, and Gauss Newton evaluated to perform 100 iterations (GN100, i.e., upon convergence) which can be interpreted as applying 100 pose retractions or actions to each pose in the graph. We utilize the agent trained in environment number 2 where the number of evaluation cycles was set to 8, and therefore performs far fewer retractions per pose than GN100. This analysis involves evaluation of the performances on a test pose-graph environment with decreasing translation uncertainty, while all other parameters are kept fixed. In this case, \( n = 300 \), \( \sigma_R = 0.3 \)rad, \( d = 3 \)m, \( lc = 0.5 \), while \( \sigma_I \) ranges from \([0.2, 0.1, 0.05, 0.03, 0.01]\)m. Objective cost function values and total elapsed optimization times are presented in Table II. It is observed that as the ratio of orientation to translation uncertainty increases, estimates produced by
GN100 are extremely inaccurate as the non-convexities of the pose-graph optimization problem also increase. Standalone RL was shown to outperform GN100 in situations where $\sigma_t = 0.05m, 0.03m, \text{ and } 0.01m$. It is also noticed when utilizing the RL estimate as an initial guess, the ground truth trajectory and hence, global minimum, is returned in almost all cases in far fewer retractions per pose. Regardless of GN being set to perform 50 iterations, once bootstrapped by the RL estimate, graph instances $\sigma_t = 0.2m, 0.1m, 0.05m, \text{ and } 0.03m$ converged in less than 11 seconds. SE-Sync performs superior, achieving global optimal trajectories under 0.16 seconds.

C. Effect on Inter-Nodal Distance

In this section, we conduct our evaluations on test pose-graph environments synthetically generated with various inter-nodal distance spacing $d$, while all other parameters are kept fixed. The environment parameters are, $n = 300$, $\sigma_R = 0.1\text{rad}$, $\sigma_t = 0.01m$, $lc = 0.5$, while $d$ ranges from $[1, 3, 5, 8, 10]\text{m}$. Agent trained on environment 5 was utilized for evaluations, and the final metrics are provided in Table III.

The standalone RL outperforms GN100 on instances with larger inter-nodal distance spacing. These results also conform to the analysis performed in [3]. GN10 bootstrapped by the RL estimate, graph instances $\sigma_t = 0.2m, 0.1m, 0.05m, \text{ and } 0.03m$ converged in less than 11 seconds. SE-Sync performs superior, achieving global optimal trajectories under 0.1 seconds.

D. Standard Benchmark Datasets

To assess the efficacy of the proposed approach, we evaluate performance on standard real-world and synthetic benchmarks provided by [36] and [3], never seen by our agent. Comparisons are made with SE-Sync, and g2o’s LM solver fixed at 30 (LM30), and 100 (LM100) iterations, with stopping criteria based on reaching a relative error decrease or number of iterations. Additionally, we provide the objective cost function values and optimization time required for the LM30 estimate initially bootstrapped by our approach. The datasets include Manhattan world M3500 variants A, B, and C, which by default have standard deviations of 0.1rad, 0.2rad, and 0.3rad added to the relative orientation measurements of the original. Further, we evaluate on City10k [37], as well as Intel and MIT. In this assessment, we double the number of cycles the agent is set to perform during test time, except for the evaluation on City10k. Increasing the number of cycles for evaluations was found to further improve the quality of the RL estimate due to a larger number of pose refinements provided by the agent. Allowing the agent to perform more cycles; however, comes at the cost of longer optimization times required. This is apparent for graphs much larger in size such as City10K.

**Table IV**

| Dataset (graphs/edges) | Metric | RL | LM30 | LM100 | RL+LM30 | SE-Sync |
|------------------------|--------|----|------|-------|---------|---------|
| M3500 (3500/5455) | $F(\phi)$ time [s] | 3.87E+02 | 3.88E+02 | 3.88E+02 | 3.88E+02 | 3.88E+02 |
| M3500A (3500/5455) | $F(\phi)$ time [s] | 7.35E+02 | 7.35E+02 | 7.35E+02 | 7.35E+02 | 7.35E+02 |
| M3500B (3500/5455) | $F(\phi)$ time [s] | 7.14E+04 | 7.14E+04 | 7.14E+04 | 7.14E+04 | 7.14E+04 |
| City10k (10K/26887) | $F(\phi)$ time [s] | 4.50E+04 | 4.53E+04 | 4.51E+04 | 4.52E+04 | 4.52E+04 |
| Intel (1224/1455) | $F(\phi)$ time [s] | 9.56E+04 | 9.56E+04 | 9.56E+04 | 9.56E+04 | 9.56E+04 |
| MIT (808/827) | $F(\phi)$ time [s] | 3.91E+03 | 3.91E+03 | 3.91E+03 | 3.91E+03 | 3.91E+03 |

In City10K, the number of edges and the total sum of squared distance measurements are much larger than other

**Fig. 3.** City10K dataset, (left to right) Standalone RL best estimate from 10 evaluations, LM100 estimate, LM30 estimate bootstrapped by the RL. The standalone RL has never seen these graphs in training.
datasets. Although RL standalone was able to outperform LM30 in City10K and MIT as well, by allowing more iterations, LM100 was eventually able to attain accurate estimations. Nonetheless, when the RL estimate is utilized as an initial guess, LM30 was capable of achieving even higher quality estimations (see Fig. 3). This is attributed to the fact that MIT has the fewest number of edges and loop closures. Further ablation studies, justification for the inclusion of LSTM, experiments, and comparative visualization figures are available at https://sites.google.com/view/rl-pgo.

In Summary, when the level of noise is high, RL performs better than GN and LM. This shows the usefulness of RL for a class of applications involving high levels of noise. SE-Sync is better than GN and LM. This shows the usefulness of RL for this letter shows a new possibility for learning algorithms to be integrated with classical ones.

VI. CONCLUSION AND FUTURE WORKS

Our RL-based method demonstrates remarkable performance improvements over classical non-linear optimizers, such as Gauss-Newton and Levenberg-Marquardt, in challenging and non-standard scenarios. The results underscore the potential of our approach in addressing complex problems. We acknowledge that SE-Sync, with its convex relaxation approach, consistently outperforms our method in terms of convergence speed, solution quality, and robustness. Our study provides valuable insights into the strengths and limitations of RL methods in the context of pose-graph optimization.

While our RL-based approach promises and excels in specific scenarios, further refinement and exploration are warranted. Future work could focus on combining the strengths of both approaches or identifying specific problem domains where our method may offer unique advantages.

REFERENCES

[1] X. Gao, T. Zhang, Y. Liu, and Q. Yan, 14 Lectures on Visual SLAM: From Theory to Practice. Beijing, China: Publ. House Electron. Ind., 2017.
[2] F. Dellaert and M. Kaess, Factor Graphs for Robot Perception. Hanover, MA, USA: Now Publ. Inc., 2017.
[3] L. Carlone, R. Aragues, J. Castellanos, and B. Bona, “A fast and accurate approximation for planar pose graph optimization,” Int. J. Robot. Res., vol. 33, no. 7, pp. 965–987, 2014.
[4] E. Olson, J. J. Leonard, and S. J. Teller, “Fast iterative alignment of pose graphs with poor initial estimates,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2006, pp. 2262–2269.
[5] J. Deray and J. Solà, “Manif: A micro lie theory library for state estimation in robotics applications,” J. Open Source Softw., vol. 5, no. 46, p. 1371, 2020.
[6] R. Azzam, F. H. Kong, T. Taha, and Y. Zwei, “Pose-graph neural network classifier for global optimality prediction in 2D SLAM,” IEEE Access, vol. 9, pp. 80466–80477, 2021.
[7] Z. Ding, “Popular RL algorithms,” GitHub.com. 2019. [Online]. Available: https://github.com/quantumiracle/Popular-RL-Algorithms.
[8] J. Dong and Z. Lv, “miniSAM: A flexible factor graph non-linear least squares optimization framework,” 2019, arXiv:1909.00993.
[9] R. Kümmeler, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2O: A general framework for graph optimization,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2011, pp. 3607–3613.
[10] S. Agarwal and K. Mierle, “Ceres solver,” 2023. [Online]. Available: http://ceres-solver.org.
[11] F. Dellaert, “Factor graphs and GTSAM: A hands-on introduction,” Georgia Inst. Tech., Atlanta, GA, USA, Rep. TR-TR-TRIM-CP&R-2012-002, 2012.