The Implementation of Optimization-based Localization

Pudit Tempattarachoke, Sijie Cui

***Abstract*—State estimation in simultaneous localization and mapping (SLAM) plays a pivotal role in the functionality of autonomous systems. It involves managing multiple states effectively within a complex framework. Central to this process is the utilization of a graph structure, wherein nodes depict robot poses and edges signify constraints between these poses. These constraints are derived from environmental observations or the robot's movements.**

Once the graph is established, the map can be inferred by determining the node configuration that best aligns with the measurements represented by the edges. To accomplish this, optimization methods are employed to minimize the loss function associated with the graph. Additionally, acceleration algorithms can be leveraged to enhance computational efficiency.

In our study, we explore various optimization-based localization techniques to delve into the system intricacies, conducting comparisons to analyze their respective robustness. Through practical implementation and rigorous analysis, we aim to gain deeper insights into the performance and efficacy of these methods in real-world scenarios.

***Index Terms*—SLAM, pose graph, optimization, approximation matrix, non-linear system**

I. Introduction

PERCEIVING the environment surrounding a robot is an essential aspect in robotics and automation. It enables the robots to interact with the environment without human operators. Common sensors such as Light Detection and Ranging (LiDAR), camera, wheel odometry and Inertial Measurement Units (IMUs) are used in perception. However, these sensors introduce uncertainties and drift that will degrade the performance. Therefore, some probabilistic methods are proposed to minimize the negative effect and calibrate the measurements to their true value.

Simultaneous Localization and Mapping (SLAM) is proposed as a cornerstone in environment perception. Compared to traditional probabilistic methods, SLAM uses the global data and states to do the optimization rather than only focus on pose to pose graph optimization. However, the inherently highly non-linearity and noise initial state estimates make SLAM a challenge. To deal with the poor initial state, methods such as Extended Kalman Filter (EKF) [1] are proposed, but their high computational complexity makes this hard to be applied to time sensitive cases.

A non-linear map optimization algorithm is implemented in this project and this project's object is to optimize the global map with corrupted input map as the input. Besides, Stochastic Gradient Descent method (SGD) [2] is used in this project followed Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates proposed by Olson [3]. Moreover, the concept of solving pose graph optimization [4] by directly constructing a linear system is also implemented. After that, both optimiziation-based methods will be evaluated based on 2D Pose Graph Optimization dataset in g2o format to compare their accuracy and robustness. Our code is online at https://github.com/Pudit/optimization-based-slam/.

II. Preliminaries

The states of a robot together with the constraints and measurements can be represented as a pose graph as shown in Fig. 1. For the pose graph problem, x_i represents the robot pose at i, while c_i and z_i are the loop closing constraint and measurement respectively. The prior node p is also added as the reference for the entire graph. The relationship of the measurement between node i-1 and i can be represented as:

$$z_i = x_i - x_{i-1} \tag{1}$$

Moreover, we can represent this pose graph problem as a probabilistics graphical model and model each conditional probability is gaussian distribution $\sim \mathcal{N}(e_{ij}, \Omega_{ij}^{-1})$. e_{ij} is the error between estimation $\hat{z}(x_{ij})$ and measurement z_{ij} between node i and j, which can be written as $\hat{z}(x_{ij}) - z_{ij}$. Meanwhile, Ω_{ij} is the information matrix among those two nodes. Due to the sufficient statistics, this pose graph problem can be written as graph optimization as demonstrated in Eq. 2 over the set S of constraints and connected poses.

$$X^* = \arg\max_X P(X) = \arg\min_X \sum_{i,j \in S} e_{ij}^T \Omega_{ij} e_{ij} \tag{2}$$

Due to the non-linear measurement error, the iterative method is used, and thus the measurement error is linearized. Eq. 3 represents the first-order taylor's series for the linearization of the measurement error between node i and j.

$$e_{ij}(x + \Delta x) \approx e_{ij}(x) + J_{ij}\Delta x \tag{3}$$

where J_{ij} is the jacobian matrix of the measurement error between node i and j.
Therefore, we can rewrite our loss function as:

$$f_{ij}(\Delta x) = (J_{ij}\Delta x + e_{ij})^T \Omega_{ij} (J_{ij}\Delta x + e_{ij}) \tag{4}$$

The minimization problem of the sum over S of loss function in Eq.4 can be formed as the linear system by stacking the loss function as shown in Eq. 5.

$$(J^T \Omega J)\Delta x = -J^T \Omega e \tag{5}$$

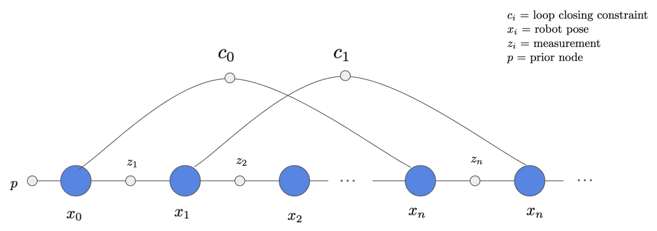

Fig. 1. Pose Graph Problem

III. Methodology

To obtain the incremental Δx, we need to solve the Eq. 5. However, directly solving the equation by taking the inverse of the LHS term ($J^T \Omega J$) is extremely expensive, leading to computational issues. Therefore, in order to improve the computational efficiency, the concept of matrix approximation and sparsity are utilized. We will follow modified stochastic gradient descent method [3] in section III-A and pose graph optimization by [4] in section III-B.

A. Modified Stochastic Gradient Descent (MSGD)

One way to improve the performance is to approximate the matrix $J^T \Omega J$. From [5], the concept of Jacobi Preconditioning is used to approximate that matrix as shown in Eq. 6.

$$M \approx diag(J^T \Omega J) \tag{6}$$

In order to compute the jacobian efficiency, the incremental pose is introduced. The incremental change in robot pose between two consecutive poses, or the difference between them, makes up the state space. If (x_i, y_i, θ_i) represent the global robot poses, then the following are the state vectors for relative and incremental pose:

$$\underbrace{\begin{bmatrix} \cos(\theta_0)(x_1 - x_0) + \sin(\theta_0)(y_1 - y_0) \\ -\sin(\theta_0)(x_1 - x_0) + \cos(\theta_0)(y_1 - y_0) \\ \theta_1 - \theta_0 \\ \cos(\theta_1)(x_2 - x_1) + \sin(\theta_1)(y_2 - y_1) \\ -\sin(\theta_1)(x_2 - x_1) + \cos(\theta_1)(y_2 - y_1) \\ \theta_2 - \theta_1 \\ \cos(\theta_2)(x_3 - x_2) + \sin(\theta_2)(y_3 - y_2) \\ -\sin(\theta_2)(x_3 - x_2) + \cos(\theta_2)(y_3 - y_2) \\ \theta_3 - \theta_2 \\ \vdots \end{bmatrix}}_{X_{\text{ref}}} \underbrace{\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ x_1 - x_0 \\ y_1 - y_0 \\ \theta_1 - \theta_0 \\ x_2 - x_1 \\ y_2 - y_1 \\ \theta_2 - \theta_1 \\ \vdots \end{bmatrix}}_{X_{\text{incr}}} \tag{7}$$

With the incremental pose and the exploitation of the sparsity structure, the jacobian can be computed cheaper in order to update the approximation matrix. The jacobian of the incremental pose between node 2 and n for 2D case is equal to:

$$J_{2,n} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 1 \end{bmatrix}$$

One way to solve the Eq. 5 is to apply stochastic gradient descent (SGD), the famous first-order optimization method. For the pose graph optimization, instead of randomly shuffling data, we modify this method by updating the pose for every constraint. Hence, the cost (f_i) and gradient (∇f_i) for each constraint can be computed as follows:

$$f_i = e_i^T \Omega e_i \tag{8}$$

$$\begin{aligned} \nabla f_i &= \frac{\partial f_i}{\partial \Delta x} \\ &= \frac{\partial f_i}{\partial e_i} \frac{\partial e_i}{\partial \Delta x} \\ &= 2 e_i^T \Omega J_i \end{aligned} \tag{9}$$

Therefore, the update step of the modified stochastic gradient descent with the vanishing step (α) is equal to:

$$x \leftarrow x + 2\alpha M^{-1} J_i \Omega e_i \tag{10}$$

where $\alpha = \frac{1}{\gamma k}$. The parameter k is the current iteration step, while γ is defined as the smallest covariance of the measurements as shown in Eq. 11.

$$\gamma = \min(diag(\Omega_{ij})) \tag{11}$$

The modified stochastic gradient descent is shown in algorithm 1, which contains two main steps: update approximation matrix M and compute modified stochastic gradient descent.

B. Pose Graph Optimization (PGO)

In contrast to the modified stochastic gradient descent method, PGO does not require to approximate the matrix, but directly compute and exploit the sparsity of the relationship between nodes and constraints. Furthermore, this method is iteratively solving the incremental pose (Δx). Instead of using the incremental pose as stated in MSGD, the way of computing error is defined in the Eq. 12, where X_i and Z_ij are homogeneous matrices of pose at node i and measurement between node i and j respectively. The vee operator is a mapping function $\cdot^\vee : g \in \mathbb{R}^{n \times n} \to \mathbb{R}^n$.

$$e_{ij}(x_i, x_j) = (Z_{ij}(X_i^{-1} X_j))^\vee \tag{12}$$

In terms of the jacobian, even though it has the similar structure as MSGD, the elements at node i and j are different as follows:

$$J_{ij} = \begin{bmatrix} 0 & \dots & 0 & A_{ij} & 0 & \dots & 0 & B_{ij} & 0 \end{bmatrix}$$

where $A_{ij} = \frac{\partial e_{ij}(x)}{\partial x_i}$ and $B_{ij} = \frac{\partial e_{ij}(x)}{\partial x_j}$.
To clarify, the jacobian of error function for 2D case can be explained as:

$$A_{ij} = \begin{bmatrix} -R_{ij}^T R_i^T & R_{ij}^T \frac{\partial R_i^T}{\partial \theta_i}(t_j - t_i) \\ 0 & 1 \end{bmatrix}, B_{ij} = \begin{bmatrix} R_{ij}^T R_i^T & 0 \\ 0 & 1 \end{bmatrix}$$

where R_i represents rotation matrix at node i, while t_i is a pose at node i.

Algorithm 1 MSGD Optimization

```
1: $k = 0$
2: while $\neg$ converged do
3:     $k++$
4:     $\gamma = [\infty\ \infty\ \infty]^T$
5:
6:     {Update approximation $M = J^T \Omega J$}
7:     $M = zeros(numstates, 3)$
8:     for all $(a, b, t_{ab}, \Omega_{ab})$ in constraints do
9:         $R = \mathbf{Rot}(P_a)$
10:        $W = (R\Omega_{ab}^{-1}R^T)^{-1}$
11:        for all $i \in [a+1, b]$ do
12:            $M_{i,:3} = M_{i,:3} + \mathbf{diag}(W)$
13:            $\gamma_y = \min(\gamma_y, \mathbf{diag}(W))$
14:        end for
15:    end for
16:
17:    {Modified Stochastic Gradient Descent}
18:    for all $\{a, b, t_{ab}, \Omega_{ab}\}$ in Constraints do
19:        $R = \mathbf{Rot}(P_a)$
20:        $P_b' = P_a T_{ab}$
21:        $r = p_b' - p_b$
22:        $r_3 = mod2pi(r_3)$
23:        $d_{1:3} = 2(R^T \Omega_{ab}^{-1} R)^{-1} r$
24:    end for
25:
26:    {Update $x$, $y$, and $\theta$}
27:    $\alpha = 1/(\gamma_j k)$
28:    $totalweight = \sum_{i \in [a+1,b]} M_{i,j}$
29:    $\beta = (b-a) dL_j \alpha$
30:    for all $j \in [1, 3]$ do
31:        if $|\beta| > |r_j|$ then
32:            $\beta[j] = r_j$
33:        end if
34:        $dpose = 0$
35:        for all $i \in [a+1, N]$ do
36:            if $i \in [a+1, b]$ then
37:                $dpose = dpose + \beta[j]/M_{i,j}/totalweight[j]$
38:            end if
39:            $p_{i,j} = p_{i,j} + dpose$
40:        end for
41:    end for
42: end while
```

To simplify Eq. 5, let $H_{ij} = J_{ij}^T \Omega_{ij} J_{ij}$ and $b_{ij} = J_{ij}^T \Omega_{ij} e_{ij}$, we can explain these two terms as:

$$H_{ij} = \begin{bmatrix} \ddots & & & & \\ & A_{ij}^T \Omega_{ij} A_{ij} & \dots & A_{ij}^T \Omega_{ij} B_{ij} & \\ & \vdots & \ddots & \vdots & \\ & B_{ij}^T \Omega_{ij} A_{ij} & \dots & B_{ij}^T \Omega_{ij} B_{ij} & \\ & & & & \ddots \end{bmatrix}$$

$$b_{ij} = \begin{bmatrix} \vdots \\ A_{ij}^T \Omega_{ij} e_{ij} \\ \vdots \\ B_{ij}^T \Omega_{ij} e_{ij} \\ \vdots \end{bmatrix}$$

Algorithm 2 is the summary of an iterative optimization method for PGO.

Algorithm 2 Pose Graph Optimization framework

```
1: while $\neg$ converged do
2:     b ← 0, H ← 0
3:     {For each constraint}
4:     for all $(e_{ij}, \Omega_{ij})$ do
5:         $A_{ij} = \frac{\partial e_{ij}(x)}{\partial x_i}, B_{ij} = \frac{\partial e_{ij}(x)}{\partial x_j}$
6:
7:         {Construct Linear System}
8:         $H_{[ii]}$ += $A_{ij}^T \Omega_{ij} A_{ij}$, $H_{[ij]}$ += $A_{ij}^T \Omega_{ij} B_{ij}$
9:         $H_{[ji]}$ += $B_{ij}^T \Omega_{ij} A_{ij}$, $H_{[jj]}$ += $B_{ij}^T \Omega_{ij} B_{ij}$
10:        $b_{[i]}$ += $A_{ij}^T \Omega_{ij} e_{ij}$, $b_{[j]}$ += $B_{ij}^T \Omega_{ij} e_{ij}$
11:    end for
12:
13:    {Fixed the first node}
14:    $H_{[11]}$ += $I$
15:
16:    {Solve the linear system ($H\Delta x = -b$)}
17:    $\Delta x \leftarrow solve(H\Delta x, -b)$
18:    $x \leftarrow x + \Delta x$
19:
20:    {Release the first node at (0, 0), and transform the rest}
21:    release_graph(x)
22: end while
```

In order to solve the linear system, we modified three different solvers: 1. Gauss-Newton [6], 2. Levenberg-Marquardt [7], and 3. Powell's dog leg method [8]. Since the matrix H is positive definite, meaning it is a square matrix, we do not need to multiply by its transpose. Algorithm 3, 4, and 5 describe the Gauss-Newton, Levenberg-Marquardt, and Powell's dog leg method to solve the linear system in PGO respectively. Gauss-Newton is the core method for the other two. The main feature is that Levenberg-Marquardt has more than Gauss-Newton is adding an augmented matrix with adaptive λ in order to deal with ill-conditioned problems. This is because the Gauss-Newton step is efficient for well-conditioned problems, and adding the augmented matrix can also tackle non-invertible issues. Powell's dogleg algorithm is another method used for the pose graph optimization, which is the combination of the advantages of both steepest descent and Newton's methods. The key idea behind this algorithm is to verify and select the direction for the update step as shown in Fig. 2. The direction and value of a Powell's step (h_{dl}) will move forward, which depend on the truth region (Δ) – the radian of the circle in Fig. 2. In the powell's dog leg algorithm, for each iteration, both steepest descent and Newton's step are computed. As

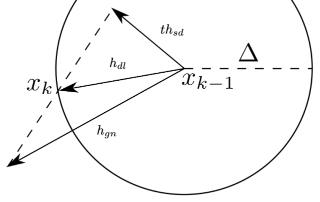

Fig. 2. The truth region of Powell's dogleg algorithm

the combination of steepest descent (h_{sd}) and Gauss-Newton (h_{dn}), the direction and step size of powell's method is captured and scaled based on the trust region following the algorithm 5. Moreover, the size of the truth region is adapted based on the weight between the design step it takes. For the powell's algorithm, φ is the gain ratio, which is determined when the truth region should change. Meanwhile, $F(x)$ is the loss function of pose graph, and $L(x)$ is least-square error $||Hx+b||_2^2$.

Algorithm 3 Pose Graph Optimization framework

```
1: while ¬ converged do
2:     $\Delta \leftarrow$ solve $H\Delta = -b$
3:     $X^{t+1} = X^t + \Delta$
4:     $t \leftarrow t+1$
5: end while
```

Algorithm 4 The Levenberg-Marquardt algorithm

```
 1: $\lambda \leftarrow 10^{-4}$
 2: $t \leftarrow 0$
 3: while ¬ converged do
 4:     $A, b \leftarrow$ linearize $g(X)$ at $X^t$
 5:     $\Delta \leftarrow$ solve $(H + \lambda \cdot \text{diag}(H))\Delta = -b$
        {$\lambda$ is a damping factor here}
 6:     if $g(X^t + \Delta) < g(X^t)$ then
 7:         $X^{t+1} = X^t + \Delta$
 8:         $\lambda \leftarrow \lambda/10$
 9:     else
10:         $X^{t+1} = X^t$
11:         $\lambda \leftarrow \lambda \cdot 10$
12:     end if
13:     $t \leftarrow t+1$
14: end while
```

IV. Experiments and Results

A. Dataset

To evaluate the performance of the algorithms, M3500 (manhattan with 3,500 nodes) and intel dataset are used.

1) M3500 Dataset: this dataset consists of 3,500 robot positions (x, y, θ) and 5,598 constraints (measurement data) as shown in Fig. 3. This dataset was created by Olson et al. [3]. Meanwhile, Fig. 4 is the ground truth of the M3500 dataset.

Algorithm 5 The Powell's Dogleg algorithm

```
 1: $k \leftarrow 0$;   $x \leftarrow x_0$;   $\Delta \leftarrow \Delta_0$;   $g \leftarrow A^T b$
 2: while ¬ converged do
 3:     $k \leftarrow k+1$
 4:
 5:     {calculated by $h_{sd} = -g\frac{||g||^2}{||Ag||^2}$}
 6:     Compute $h_{sd}$
 7:
 8:     {solved by $Hh_{gn} = -b$}
 9:     Compute $h_{gn}$
10:
11:     {based on $h_{sd}$ and $h_{gn}$}
12:     Obtain $h_{dl}$ from algo. 6
13:     $X^{t+1} \leftarrow X^t + h_{dl}$
14:     $\varphi \leftarrow (F(x^t) - F(x^{t+1}))/(L(0) - L(h_{dl}))$
15:     if $\varphi > 0$ then
16:         $X^t \leftarrow x^{t+1}$
17:         if $\varphi \geqslant 0.75$ then
18:             $\Delta \leftarrow \max\{\Delta, 3 \cdot \|h_{dl}\|\}$
19:         else if $\varphi < 0.25$ then
20:             $\Delta \leftarrow \Delta/3$
21:         end if
22:     end if
23: end while
```

Algorithm 6 h_{dl} determination
based on h_{sd} and h_{gn}

```
1: if $\|h_{gn}\| \leqslant \Delta$ then
2:     $h_{d1} \leftarrow h_{gn}$
3: else if $|h_{sd}| \geqslant \Delta$ then
4:     $h_{d1} \leftarrow \left(\frac{\Delta}{\|h_{sd}\|}\right) h_{sd}$
5: else
6:     $h_{d1} \leftarrow h_{sd} + \beta(h_{gn} - h_{sd})$
7:     solve $\beta$ by $\|h_{d1}\| = \Delta$
8: end if
```

2) Intel Dataset: this dataset was created in Seattle under the Intel Research Lab [9], consisting of 943 poses and 1837 constraints as demonstrated in Fig. 5. The ground truth of this dataset is shown in Fig. 6.

B. Results

The results after processing optimization-based methods for M3500 based on MGSD after 200 iterations and 10-iteration of PGO are illustrated in the Fig. 7 and 8 respectively. Since the results from the three different solvers of PGO are almost similar and take similar iterations before reaching the local optimum, we will visualize only the result from the Gauss-Newton approach. Meanwhile, Fig. 9 and 10 are the result after 900-iteration of MSGD and only 10-iteration of PGO.

C. Analysis

Based on the experiment on M3500 and INTEL dataset, the performance of MSGD is significantly worse than PGO in terms of both convergence rate and accuracy. The main

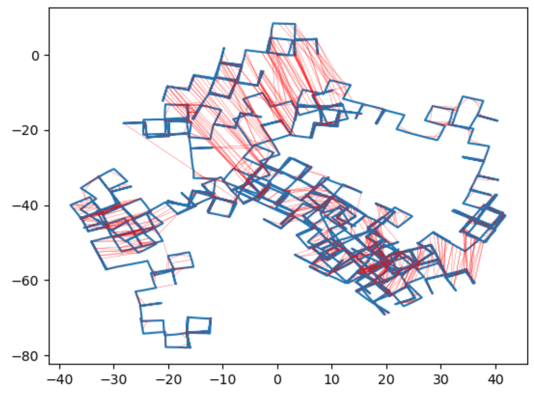

Fig. 3. The visualization of M3500 dataset with constraints (red lines) associated with the robot poses (blue lines)

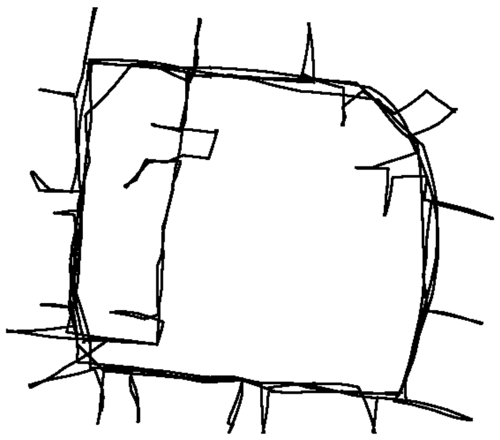

Fig. 6. The ground truth of Intel dataset

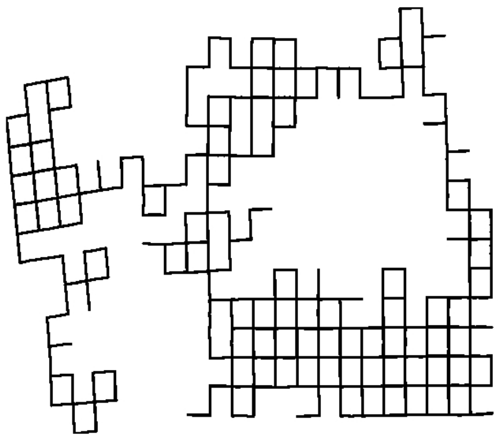

Fig. 4. The ground truth of M3500 dataset

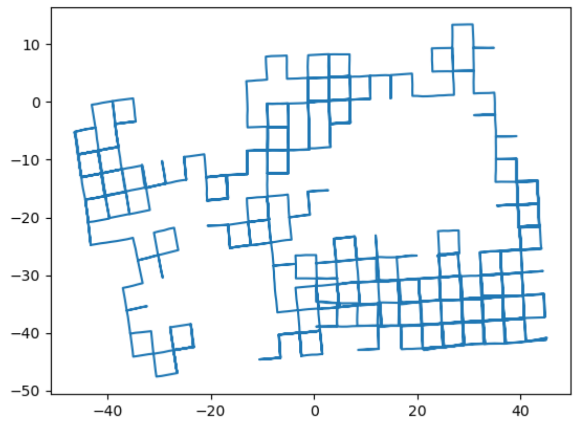

Fig. 7. The result of M3500 after 200-iteration optimization based on MSGD.

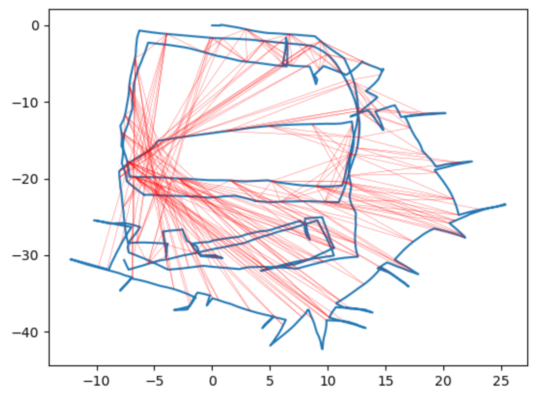

Fig. 5. The visualization of Intel dataset with constraints (red lines) associated with the robot poses (blue dots)

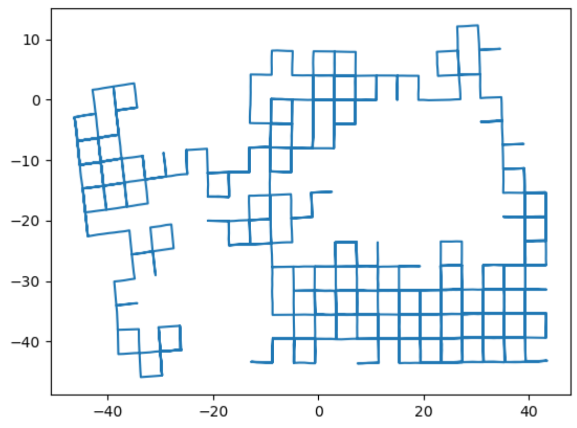

Fig. 8. The result of M3500 after 10-iteration optimization based on PGO, using Gauss-Newton (all of three solvers produce the same result).

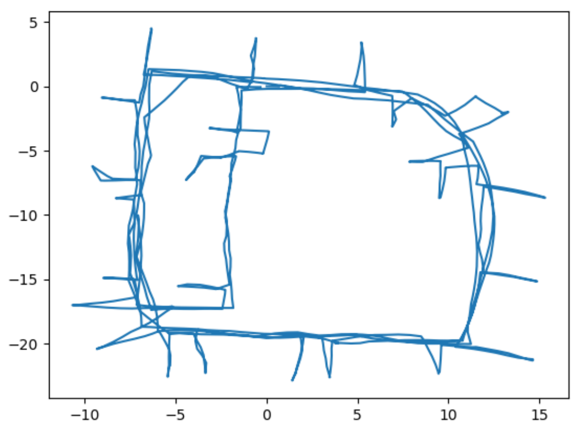

Fig. 9. The result of M3500 after 900-iteration optimization based on MSGD.

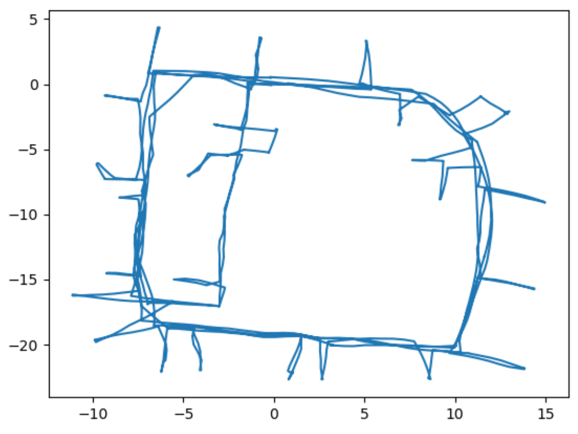

Fig. 10. The result of M3500 after 10-iteration optimization based on PGO, using Gauss-Newton (all of three solvers produce the same result).

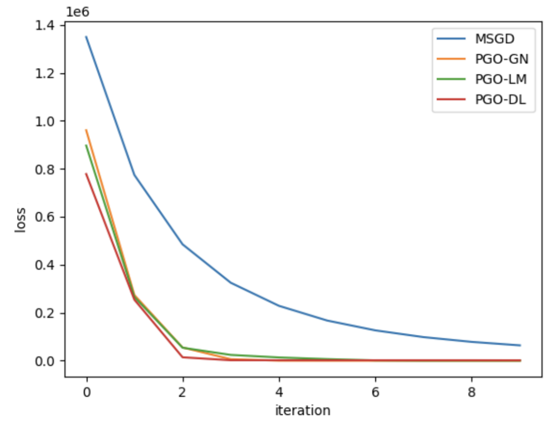

Fig. 11. The comparison of loss value from MSGD (blue) and PGO with three different solvers: Gauss-Newton (orange), Levenberg Marquardt (green), and Powell's dog leg (green)

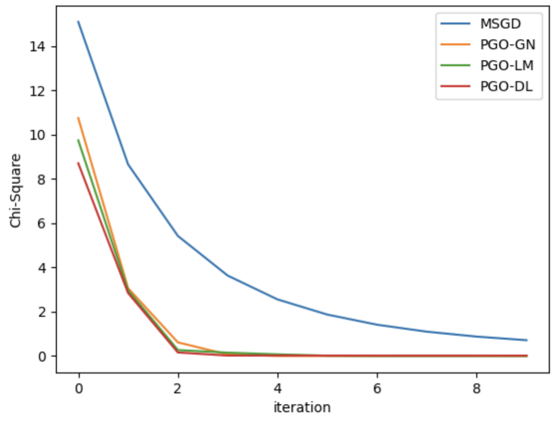

Fig. 12. The comparison of chi-square from MSGD (blue) and PGO with three different solvers: Gauss-Newton (orange), Levenberg Marquardt (green), and Powell's dog leg (green)

underlying reason is the fact that the MSGD method constructs the linear system by approximating the matrix M as mentioned in section III-A instead of direct construction. Furthermore, due to the innate convergence rate of gradient-based methods, the number of iterations to reach the same optimality gap is larger than the second-order methods that are used in PGO. However, the computational speed of gradient-based methods is significantly faster than the second-order methods. Fig. 11 and 12 display the loss from Eq. 2 and chi-square per each iteration based on MSGD and three different solvers behind PGO, which evaluate on the M3500 dataset. From both graphs, all solvers behind PGO can reach the local optimum within merely around 4 iterations, while MSGD requires more. This is because Newton-based methods can convert super linearly after a few iterations in the damped phase, while gradient-based methods slowly convert. In terms of the three solvers behind PGO, their convergence rate for this dataset is almost the same due to less effect of perturbation terms from the augmented matrix. For positive definite systems (H), Gauss-New is the best solver among them, because the other two are affected by the introduction of the augmented matrix, stucking with gradient steps for many iterations. However, for a positive semi definite case, Levenberg Marquardt and Powell's dog leg could perform better than Gauss-Newton due to the ability to handle the singularity of the system.

V. Conclusion

For our experiment, PGO is the better solver for SLAM backend, compared to MSGD in both efficiency and performance. Even though Gauss-Newton, Levenberg Marquardt, and Powell's dog leg as a solver for PGO can perform almost the same, we would like to choose Levenberg Marquardt with a small lambda for this task. This is because Gauss-Newton might not be robust for a positive semidefinite system, while solving h_{dl} in Powell's dogleg might be difficult for some cases, leading to no existence solution in high-dimensional. Hence, Levenberg Marquardt method could be one of the best choices for PGO, even the speed for a positive semidefinite matrix can be significantly slower than Gauss-Newton.

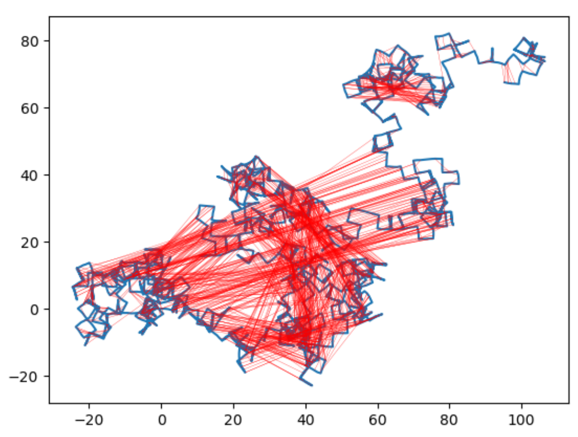

Fig. 13. M3500 dataset with 0.1 rad noise

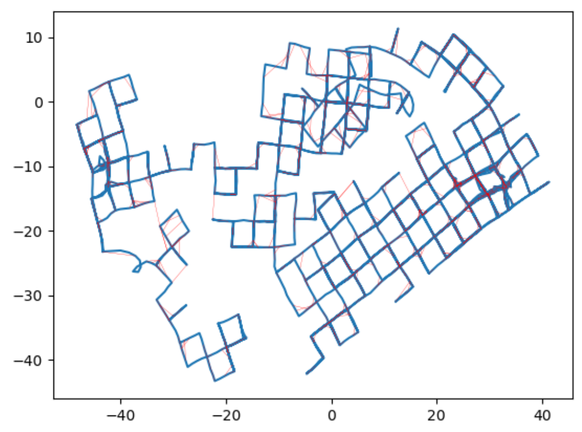

Fig. 14. The optimization result on M3500a based on PGO

VI. Future Work

In general, the optimization solver is designed to perfectly fit with datapoints, but not with noise added to the system. Fig. 13 shows the M3500 dataset with 0.1 radian gaussian noise, and the result of PGO after 10-iteration optimization is illustrated in Fig. 14. As expected, the result of PGO cannot produce the high accuracy due to overfitting with noise. One way to tackle with this is to integrate robust kernel with the loss function in order to reject outliers. Hence, the next step is to introduce robust estimation techniques such as huber loss [10], max-mixture [11], and graduated non convexity [12]. Besides that, we plan to implement heuristic PGO in order to perform real-time, while omitting some irrelevant nodes to reduce computational cost during the optimization process, before expanding it to distributed pose graph optimization [13].

References

[1] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[2] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[3] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 2262–2269.

[4] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[5] G. Strang, *Introduction to linear algebra*. SIAM, 2022.

[6] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[7] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*. Springer, 2006, pp. 105–116.

[8] Å. Björck, *Numerical methods for least squares problems*. SIAM, 1996.

[9] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, "A linear approximation for graph-based simultaneous localization and mapping," 2012.

[10] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4331–4339.

[11] E. Olson and P. Agarwal, "Inference on networks of mixtures for robust robot mapping," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 826–840, 2013.

[12] H. Yang, P. Antonante, V. Tzoumas, and L. Carlone, "Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1127–1134, 2020.

[13] C. Li, P. Yi, G. Guo, and Y. Hong, "Distributed pose-graph optimization with multi-level partitioning for collaborative slam," *arXiv preprint arXiv:2401.01657*, 2024.