Analysis for Graph-Based SLAM Algorithms under g^2o Framework

Tianxiang Lin, Jiayin Xia, Qishun Yu, Kerou Zhang and Ben Zhou

Carnegie Mellon University

May 2021

Contents

1 Introduction

Simultaneous Localization and Mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. The back-end of SLAM can be classified into two processing methods: filter methods (e.x. Kalman Filter), and nonlinear optimization (e.x. graph-based optimization). Although filter methods used to be popular among researchers in SLAM, state-of-the-art research topics in the SLAM field are mostly in graph-based optimization recently. This is because filter methods, such as Kalman Filter, are not suitable for large environments with a huge amount of data in visual SLAM. In visual SLAM, the number of camera poses and image feature points is huge, but the Jacobian in graph-based method is sparse, which means it can speed up. Compared to filter methods, the efficiency of nonlinear optimization is significantly higher in visual SLAM. That's why our project focus will be on graph-based optimization.

2 Graph Optimization

Graph-based optimization is a SLAM optimization problem represented by a graph that is composed of vertexes and edges. In the graph, the poses are represented by vertices and the observation equations are represented by edges. Therefore, the objective function can be written as

$$x^{\star} = \min_{x} F(x)$$

$$F(x) = \sum_{k=1}^{n} e_k(x_k, z_k)^T \Omega_k e_k(x_k, z_k)$$

where Ω is the inverse of the covariance matrix of poses, e is the error function representing difference between real and estimated measurement. By minimizing $F(x)$, we can find a set of poses x that best desribes measurements z.

2.1 Graph Optimization Frameworks

General Graph Optimization (g2o) is a framework used for optimizing nonlinear least-squares problems that can be represented as a graph. It has performance that is comparable to specialized algorithms while being able to accept general forms of nonlinear measurements[3]. Looking at the structure of g2o shown in Figure 1, it can be seen that the core of the framework is the SparseOptimizer, which is a HyperGraph. The SparseOptimizer has an OptimizationAlgorithm that uses either Gauss-Newton, Levenberg-Marquardt, or Powell's dogleg. The OptimizationAlgorithm includes a Solver, which calculates the Jacobian of sparse matrices and Hessian matrix, and a linear solver which solves $H\Delta x = -b$ using PCG, CSparse, or Choldmod [2]. g2o however is a non-robust optimizer. In the presence of outliers, the performance of g2o drastically decreases[5].

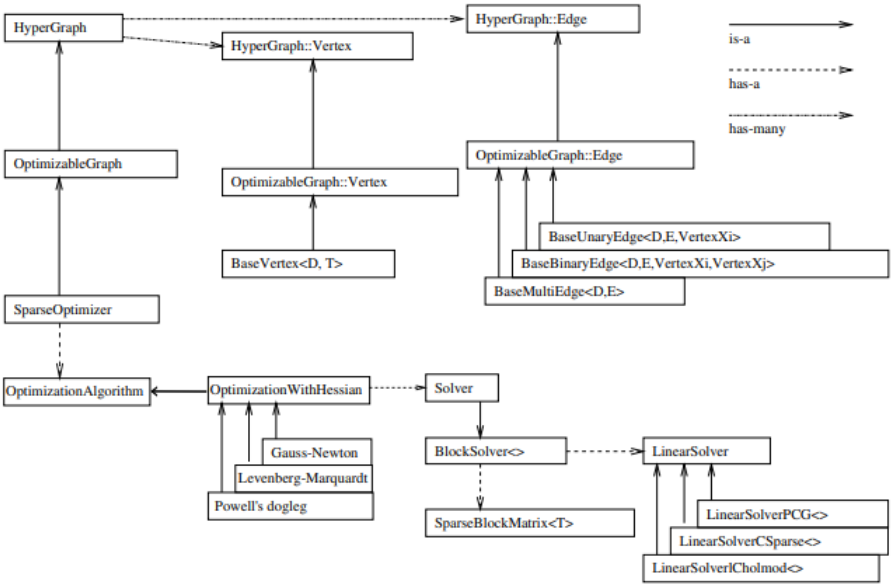


Figure 1: The Structure of g2o

2.2 Graph Optimization Algorithms

The G2O framework is formulated as a nonlinear least-squares optimization. Given the robot measurements, the optimizer is able to formulate the solution with the least total of the square of the residual error between the expected and actual states. In the project, we uses Gauss-Newton, Levenberg-Marquardt, and Powell's dogleg methods as our traditional optimizer. The naive implementation of standard algorithm are introduced below.

Gauss-Newton Given residual function of the states, Gauss-Newton method iteratively solve the optimization by minimizing the sum of squares. Starting with an initial estimate x^0, Gauss-Newton first linearize the nonlinear system with $h_i(x^t + \Delta) \approx h_i(x^t) + H_i\Delta_{gn}$ and assemble into $|| A\Delta_{gn} - b ||^2$. The Δ_{gn} can be further solved by the normal equations:

$$A^T A\Delta_{gn} = A^T b$$

and the resulting Δ_{gn} is used to update x^{t+1}. This process is repeated until convergence.

Levenberg-Marquardt Levenberg-Marquardt method associates Gauss-Newton method with gradient-descent method by introducing an additional parameter λ into the normal equation resulting in the equation below:

$$(A^T A + \lambda I)\Delta_{lm} = A^T b$$

The damping factor λ is used to adjust the reduction of the sum of residual error. When the damping factor λ is small, the equation behaves closer to the Gauss-Newton method, while high value λ makes the equation behaves like gradient-descent with small steps. The resulting Δ_{lm} is used to update x^{t+1}. This process is repeated until convergence.

Powell's dogleg Similar to the Levenberg-Marquardt method, Powell's dogleg method associates Gauss-Newton method with gradient-descent method. At each iteration, the method explicitly maintains a trust region T and calculate the Gauss-Newton update Δ_{gn} and steepest descent update Δ_{sd}. Then, the update of Δ_{dl} is selected based on these three parameters, as shown in the algorithm.

Algorithm 1: Powell's dogleg method

```
Result: Return Dogleg update Δ_dl
for each, i ∈ I do
    if Δ_gn < T then
        Δ_dl = Δ_gn
    else
        if Δ_sd > T then
            Δ_dl = Δ_sd (at the boundary T)
        else
            Δ_dl = Δ_k (intersection between Δ_sd and Δ_gn at the
             boundary T)
        end
    end
end
```

2.3 Robust Kernels

For the optimization problem of SLAM, as soon as the real world is involved, outliers and ambiguities may occur. Optimization is sensitive to outliers, if the data point is very far away from our model, the error gets squared and amplified quite substantially. In order to be more outlier robust, g2o framework is able to replace the error function by a more robust cost function like huber for example. In this section, three robust kernel functions are introduced and tested for robustness.

Huber In g2o, Huber cost function is implemented to deal with data outliers. The conditional expression is shown below, where e is the residual error. The cost function is quadratic when e is small but linear for large e. Compared to other, even more robust cost functions, the Huber kernel has advantage that it is still convex and thus does not introduce new local minima in the error

function [3].

$$L_\delta(e) = \begin{cases} e^2, & \text{if } \mid e \mid \leq b \\ 2b \mid e \mid -b^2, & \text{otherwise} \end{cases}$$

Geman-McClure Originally introduced as a nonlinear filter for film restoration in image processing [1], the Geman-McClure cost function also penalize more when the error term e is small, as shown in Figure 2 and equation:

$$L_\delta(e) = \frac{\frac{e^2}{2}}{(k + e^2)}$$

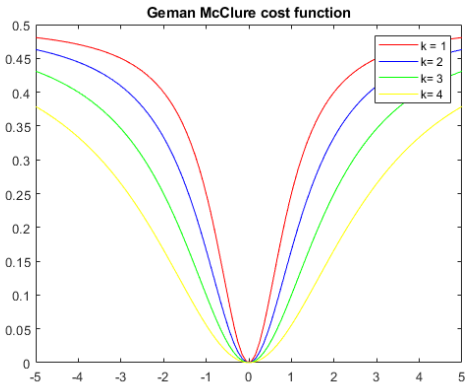


Figure 2: Geman McClure cost function

DCS The key idea of Dynamic Covariance Scaling (DCS) is the introduction of additional weights to constraint outliers. The method adds additional scaling factor s_{ij}^2 to the information matrix.

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{ij} \mathbf{e}_{ij}(X)^T \left(s_{ij}^2 \Omega_{ij}\right) \mathbf{e}_{ij}(X)$$

For every constraints, an individual scaling factor is added to the standard least squares equation to down rate the error if the current state is considered to be an outlier. The scaling factor is defined as:

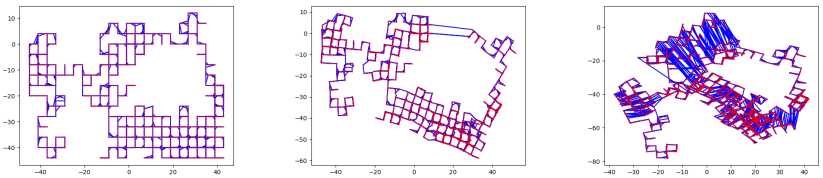
$$s_{ij} = \min\left(1, \frac{2\Phi}{\Phi + \chi_{ij}^2}\right)$$

Where Φ is the parameter that we can define to influence the effect of the down rate. Scaling factor is set to be 1 if the constraints is close to the mean estimate but it will weight down the constraints that are far away from the mean estimate.

3 Project Settings

3.1 Datasets

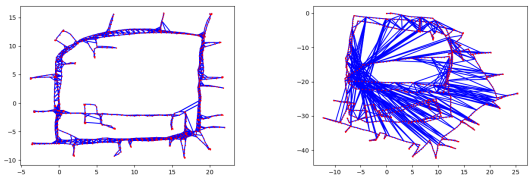
Manhattan 3500 Dataset Manhattan 3500 is a synthetic dataset by Olson et al.[4], which consists of 3500 poses and 5600 constraints. For initialization procedure, we use the dataset provided by Olsen[4] and g2o[3]. Meanwhile, we propose a method to add noise on ground truth to test optimization algorithms and robust kernels, which will be fully discussed in the section 3.2 and 3.2.



(a) Mahattan 3500 gt (b) Mahattan 3500 g2o (c) Mahattan 3500 Olson

Figure 3: Manhattan 3500 Dataset

Intel Dataset Intel Dataset, acquired at the Intel Research Lab in Seattle, is the dataset whose pose graph was obtained by processing the raw measurements from wheel odometry and laser range finder. For initialization procedure, we use the dataset provided by g2o[3]. We also add loop closure and Gaussian noises on initial poses on the ground truth to test the efficiency and accuracy of selected algorithms and robust kernels.



(a) Intel gt (b) Intel dataset with Noise

Figure 4: Intel Dataset

3.2 Noise on Initial Estimates

Two types of different errors are added to the datasets mentioned above. First method is adding noise on initial estimates, which means errors on the vertices in the g2o files. A Python script is written to process the datasets. We added three Gaussian Distribution noises to the x, y, and theta values correspondingly. The Gaussian Distributions are all with zero means, and the values of standard

deviation of x, y are two times larger than the standard deviation of theta. Multiple trials are implemented for standard deviation of x and y ranging from 0.5 to 2.5 on Intel Dataset. Further experiments are conducted on the Intel Dataset with poor initial estimates to evaluate the optimization performance with and without robust kernels.

3.3 Loop Closure Constraint Outliers

Another type of error added on datasets is adding outliers to loop closure constraints. A dataset generation Python script from Vertigo, an extension library of g2o[3] is used for this type of error data generation. Loop Closure Constraint Outliers are added additionally to the original edges of the g2o dataset files. Specifically for this project, we added local loop closure outliers. This is assumed to be more challenging than other types of outliers since the optimization algorithm need to distinguish the edge outliers which are not far off from the original edges. We used Manhattan 3500 ground truth dataset as the input file of the Python script and added different numbers of local loop closure outliers ranging from 1 to 10. Further experiments are conducted with different robust kernels to evaluate the optimization performance.

4 Experiments and Discussion

4.1 Optimization Algorithm Performance to Poor Initial Estimates

To test how different optimization algorithms perform when given a poor initial estimate, we used the Manhattan 3500 dataset with varying amounts of noise added to the ground truth values and optimized with the GN, LM, and dogleg methods. We then plotted the chi-squared value verses the number of iterations for each of these methods. The results can be seen in Figure 5. The first four graphs are when we added an increasing amount of noise to the ground true dataset to simulate poor initial estimates. The fifth graph is the g2o dataset, and the last graph is the Olsen dataset.

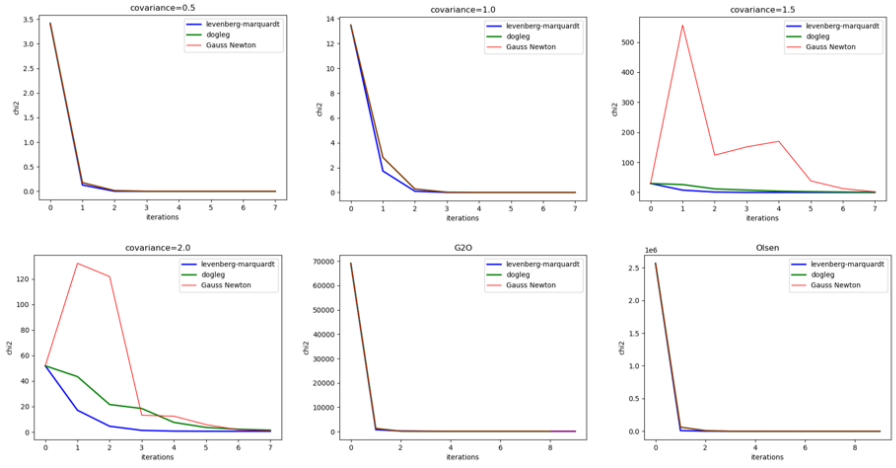


Figure 5: The Structure of g2o

From these graph, we can see that the GN method is not a stable method, as the chi-squared value can fluctuate a bit. However, similarly between all three methods, as the amound of noise added increases, it becomes impossible for these optimization algorithms to reduce the chi-squared value to a relatively small value.

4.2 Kernel Performance to Poor Initial Estimates

We then tested the affects of applying a robust kernel when doing optimization on poor initial estimates. Using the g2o intel dataset as the ground truth, we added an increasing amount of Gaussian noise to the grout truth to simulate bad initial estimates. We then optimized each dataset with the dogleg method both using and without using the Huber kernel. The results of these optimizations can be seen in Figure 6.

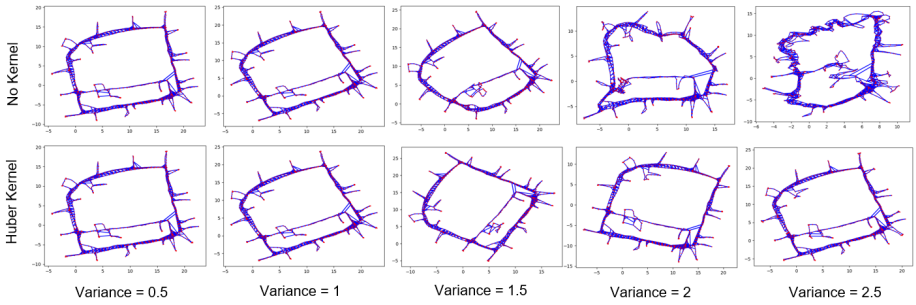


Figure 6: Optimization output with and without Huber kernel

The images from left to right show the output of the optimization when Gaussian noise with an increasing variance is added to the initial guess. The top row shows the output when no kernel is used, and the bottom row shows the output when the robust Huber kernel is used. Just from a visual comparison, it can be seen that the Huber kernel greatly increases the accuracy of the optimization, especially as the initial estimates get worse. This is also reflected in the chi-squared values of these outputs.

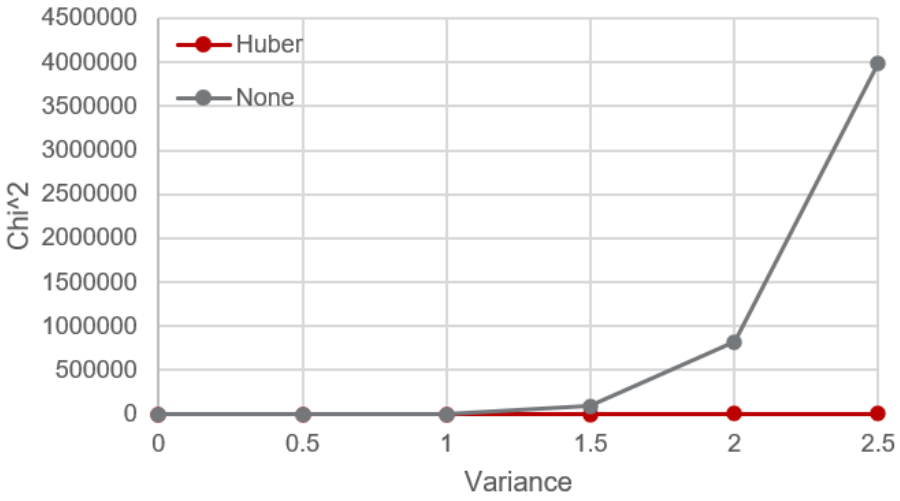
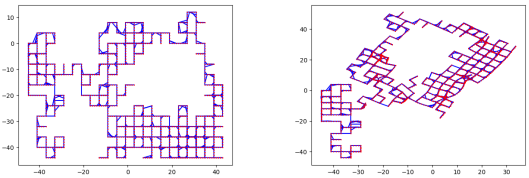


Figure 7: Chi-squared of optimization with and without Huber kernel

As can be seen from Figure 7, the chi-squared value of the optimization when not using any robust kernel increases drastically as the amount of noise increases. However, when applying the Huber kernel, the chi-squared values are barely increasing.

4.3 Kernel Performance to Outliers



(a) Ground Truth

(b) Optimized with 10 Incorrect Loops

Figure 8: Robust Kernel's Reaction on close loop constraint outliers

Finally, we tested the affects of applying a robust kernel when doing optimization on false positive loop closure constraints. We used Manhattan 3500 as our testing dataset for this experiment, added up to 10 incorrect loop closure, and optimized with Powell's dogleg algorithm.

According to our experiments, shown in Figure 8, the optimized results deteriorated with the increase of false positive loop constraints. Adding 10 incorrect loop constraints to the ground truth, the optimized result became unrecognizable compared to the ground truth, even though chi-square value converged.

5 Conclusion and Future Work

From the experiments we make in the previous sections, Gauss-Newton Law might converge fastest, but its convergence will show sawtooth-like behaviors. And Levenberg-Marquardt and Powell's dogleg Algorithm are more robust to outliers. Furthermore, robust kernels like Huber, Cauchy, Geman-McClure, and DCS are helpful against poor initialization. However, they are not efficient enough to loop closure outliers according to our experiments.

There are two interesting directions where our experiments and implementations can be extended. First of all, two datasets are not enough to make general conclusions, so we want to perform more experiments on different datasets. And we want to implement kernel robust against false positive loop closures.

References

[1] Stuart Geman, Donald E. McClure, and Donald Geman. A nonlinear filter for film restoration and other problems in image processing. *CVGIP: Graphical Models and Image Processing*, 54(4):281–289, 1992.

[2] Giorgio Grisetti, Rainer Kummerle, Hauke Strasdat, and Kurt Konolige grisetti. g2o: A general framework for (hyper) graph optimization. https://vincentqin.gitee.io/blogresource-4/slam/g2o-details.pdf. Accessed: 2021-3-15.

[3] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, Shanghai, China, May 2011.

[4] Edwin Olson, John Leonard, and Seth Teller. Fast iterative alignment of pose graphs with poor initial estimates. pages 2262 – 2269, 06 2006.

[5] Heng Yang, Pasquale Antonante, Vasileios Tzoumas, and Luca Carlone. Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection. *IEEE Robotics and Automation Letters*, 5(2):1127–1134, 2020.