Improving SLAM Initialization with Modifications of MASAT Algorithm

EECE5550 Mobile Robotics Final Project Report

Hongzhen Xu
Khoury College of Computer Sciences
Northeastern University, MA

Shuyue Gao
College of Engineering
Northeastern University, MA

Ananya Tadigadapa
College of Engineering
Northeastern University, MA

***Abstract*—This project aims to test and improve the Multi-Ancestor Spatial Approximation Tree (MASAT) algorithm developed by the Károly Harsányi team for SLAM initialization. We proposed, implemented and experimented two modified versions of MASAT algorithm.**

***Index Terms*—SLAM (Simultaneous Localization and Mapping) Initialization, Pose-Graph Optimization, MASAT Algorithm, Breadth-First Search (BFS), Simple Average (MASAT-sa), Rolling Breadth-First Search (MASAT-rbfs), Non-Linear Factor Graph Optimization**

I. Introduction and Motivation

SLAM (Simultaneous Localization and Mapping) initialization refers to the process of setting the initial state of the SLAM system, which includes defining the starting position and orientation of the robot or sensor, as well as the initial map representation. Proper initialization is critical because it provides the foundation upon which subsequent localization and mapping steps are built. The initialization is important because starting with a good initial state can lead to convergence in shorter processing and avoid local minima or failure to find optimization.

In the paper "MASAT: A fast and robust algorithm for pose-graph initialization" [1], the authors present a novel algorithm for computing the initial structure of pose-graph-based Simultaneous Localization and Mapping (SLAM) systems. Their method utilizes a Breadth-First Search (BFS) on the graph to gather multiple estimates of a robot's position based on its previously processed neighbors ("fixed" neighbors). The initial location of each pose is then determined as the average of these estimates. By employing this initialization strategy, they significantly reduce the number of iterations required for optimization while maintaining low computational complexity.

For pseudocode of this algorithm and our explanation to make it easy for the readers' understanding, please see **Appendix A** and **Appendix B** attached at the end of this report. For implementation of this algorithm by these authors, please see references for the link to their github repository [2]. The implementation takes a ".g2o" file as input and generates a new ".g2o" file as output. The ".g2o" file contains nodes (vertices) and edges that define the graph, where:

- Nodes: Represent variables to be optimized (e.g., robot poses, landmarks).
- Edges: Represent constraints and measurements between these variables.

The **motivation** of our project comes from the interest in SLAM initialization problems. We have observed limitation of this algorithm MASAT: only "fixed" neighbors contribute to the initial estimate of current vertex. In early period of BFS search, there are not many fixed neighbors and thus the initialization is not accurate enough.

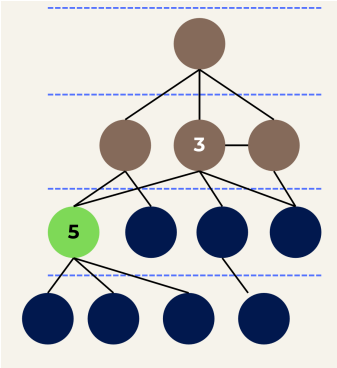

Fig. 1. Example of Limitation

II. Problem Statement

Key Observation from MASAT:

when at the i_{th} level, all previous level nodes are "fixed" and some of the current level nodes are "fixed". Only "fixed" nodes contribute to neighbor nodes' calculation.

This is the limitation and a headache of MASAT algorithm.

For example, in **Figure 1**, Node 5 uses node 3 for calculation, but node 3 has a sibling, many children and grandchildren that are not accessed yet. Then node 5 will be inaccurate in its initial estimate. The accumulated inaccuracy gets worse when BFS-based MASAT algorithm deepens to levels far away from 0_{th} level.

The **objective** of our project is to modify the MASAT algorithm to enhance the accuracy of the SLAM initialization, so that the initial estimate error before optimization is lower and so that the optimization runtime is lower.

III. Proposed Solutions

There are two proposed and implemented modifications of the MASAT algorithm, both with their benefits and drawbacks.

The implementation details and source code are available at https://github.com/hx394/SLAM_initialization [3]

A. PROPOSED SOLUTION 1: Simple Average

We call this approach "simple average" version of MASAT, or "MASAT-sa".

Two-Step Approach(A,B are containers such as arrays):

- Step 1: Run the MASAT algorithm to generate rough initial estimates assigned to A. Every node is "fixed" now.
- Step 2: Recalculate each node's average estimate and assign to B using all neighbors' static contributions from container A. Return B as the initialization result.

Algorithm 1 MASAT: Simple Average version

MASAT-sa (V, E)
Input: Raw measurements, vertices $v_i \in V$, edges $e_{i,j} \in E$
Output: An initial guess for the real pose-graph

$A = MASAT(V, E)$
for each vertex v_i **do**
  count=0
  for each edge $e_{i,j}$ **do**
    calculate estimate est_{ij} from each neighbor v_j in A using $e_{i,j}$
    count=count+1
  end for
  B[i] =$\sum_j est_{ij}$ $\div$ count
end for
return B

Please notice that after A is obtained in step 1, the calculation of each vertex(node) in B is using static values of all neighbors stored in A.

Time and Space Complexity Analysis

The time complexity and space complexity of BFS in MASAT are both O(V+E).

We just double the BFS in simple average version. So the time complexity and space complexity are 2O(V+E), which is also O(V+E).

In terms of time complexity, 2O(V+E)=O(V+E), because Big O notation disregards constant factors when describing the growth rate of an algorithm. For very large graphs, the factor of 2 becomes negligible compared to the growth of V+E, so both MASAT and MASAT-sa are O(V+E) in their time complexity and space complexity.

In Big O notation, multiplying the complexity by a constant factor does not change the asymptotic growth. The focus is on how the algorithm scales with the size of the input (e.g. V for vertices and E for edges), and constants like 2 are omitted.

In computer science, it means upper bound on the running time or the space required by the algorithm are the same between MASAT and its simple average version.

Key observations:

- Each node is only determined by neighbors(nodes).
- BFS creates levels.
- Neighbors come from current level and 2 adjacent levels. (3 levels: current level, parents level, children level)
- In MASAT algorithm, only some of current level neighbors and all upper level neighbors contribute to any node in current level.
- In MASAT simple average version algorithm, all nodes are "fixed" and static after MASAT procedure and stored in container A, so all neighbors from 3 levels contribute to simple average procedure.

Do we have a way to ensure the accumulated error during BFS deepening under control?

Yes! We repeat BFS with iterative deepening in next subsection!

B. PROPOSED SOLUTION 2: Rolling BFS

We call this approach "MASAT: Rolling BFS", or "MASAT-rbfs".

Algorithm 2 MASAT: Rolling BFS version

MASAT-rbfs (V, E)
Input: Raw measurements, vertices $v_i \in V$, edges $e_{i,j} \in E$
Output: An initial guess for the real pose-graph

$\ell_i = \{v \in V | v$ is on the i$^{\text{th}}$ BFS level $\}$
$a_i = \ell_0 \cup \ell_1 \cup ... \cup \ell_i$
for i from 0 to BFS max depth **do**
  $A = MASAT(a_i, E)$
  $a_{i+1} = A$
end for
for each vertex v_i **do**
  count=0
  for each edge $e_{i,j}$ **do**
    calculate estimate est_{ij} from each neighbor v_j in A using $e_{i,j}$
    count=count+1
  end for
  B[i] =$\sum_j est_{ij}$ $\div$ count
end for
return B

Iterative deepening Approach (A, B are containers. "i" is iterative index from 0 to max BFS level.):

- For i from 0 to max BFS level:
  Iterates through BFS levels from 0_{th} level to i_{th} level.
- Nodes from level 0,1,..., i-2 have all neighbor nodes as "fixed" nodes and "fixed" neighbors contribute to calculation of estimates.
- Besides, all nodes get an update from neighbor nodes' value updates in each iteration. Assign the initial estimates to container A.
- Then, recalculate each node's average estimate using all neighbors' static estimates from container A just the

same way as step 2 in simple average version algorithm, and assign to B. Return B as the initialized estimates waiting to be optimized.

Time and Space Complexity Analysis

This algorithm calibrates the accuracy of initialization gradually.

It retains the same space complexity as MASAT algorithm, but time complexity becomes O(V^2).

Consider extreme edge cases:

- Each level has only one node,each node has at most two neighbors(parent and child):
  $(1 + 2 + ... + V) * 2 = (1 + V) * V = O(V^2)$
- Fully connected graph: level 0 has 1 node, level 1 has V-1 nodes, each node has V-1 neighbors, algorithm terminates at level 1:
  $1 + (1 + V - 1)(V - 1) = O(V^2)$

Claim: if we have more than one node at any level, the iteration number shrinks by the size of the level minus 1.

EXPLANATION: Why we need container B?

- The MASAT-rbfs algorithm forms estimates in the container A. But the choice of starting node highly affects the heir nodes. The structure of shallow levels in their estimates is much more rigid than deeper levels, because shallow levels are visited many more times compared to deeper levels.
- What is more, suppose i_{th} level nodes depend on $(i+1)_{th}$ level neighbors and get updated; after that, $(i+1)_{th}$ level depends on other neighbors and gets updated as well. Then the i_{th} level nodes become inaccurate because of the changes in $(i+1)_{th}$ level estimates.
- By using each node's neighbors' static values in A and saving the average estimate in B, we ensure that in the last round, we smooth the static estimates in A and the initialization is more accurate. That's why we need container B.

C. Extra Initialization Method Implemented

We also take the challenge of dealing with misleading or incorrect data causing invalid constraint errors such as infinity by pruning the factor graph.

And, we also take the challenge of dealing with data insufficiency leading to disconnected graphs. This usually happens after pruning the factor graph.

Our **procedures** are:

- pruning the data by factor error using a determined threshold,
- finding connected components of the graph, and choosing the largest connected component,
- cleaning previous initial estimates.

In our extra experiment, we implemented this method for optimization on **Dataset 12**, which is a dataset full of incorrect values. The result is successful.

This is the SLAM initialization for **Dataset 12**, where the original data input causes runtime error during reading input but our initialization works successfully. The MASAT algorithm cannot correctly initialize this dataset as well.

For definition of the factor error, please take a look at **Appendix G**.

However, after this initialization, we did not find a way to export Nonlinear Factor Graph and estimate values of gtsam library into ".g2o" format file. If we could, then this part of SLAM initialization would be compatible with current MASAT implementation. This could be a potential extension of this project.

The block diagram of the proposed system, created by the team, with indication of which teammates worked on which components, is in **Appendix C**.

IV. Experiments and Results

A. Data Generation and Acquisition

We used two main sources of datasets in this experiment:

- The original dataset from the MASAT paper git repository: the main purpose is to validate the performance of the original algorithm and ensure a fair comparison of the results.
- Several specially designed simulation dataset: covering the following scenarios with the aim of testing the robustness of MASAT and the modified algorithm in different environments.
  - "**Dataset 12** Data Containing Several Incorrect Data" which is generated for challenging the MASAT algorithm;
  - "**Dataset 14** Noisy Closure Circle 2D" which is generated as a 2D loop closure;
  - "**Dataset 13** Sparse 2D Circle" which is generated as sparse 2D dataset;
  - "**Dataset 11** Sphere 2500" which is found on the internet as a minimum spanning tree graph.

With these datasets, we are able to comprehensively evaluate the performance of the algorithms in a variety of sparse graph and high noise environments.

B. Experiment Method

1) Compare Groups: We have four compare groups.

- original data;
- MASAT initialized data;
- MASAT-sa initialized data;
- MASAT-rbfs initialized data.

2) Datasets and Sources: please see **Appendix D**.

3) Evaluation Metrics:

- Use gtsam library to read the input, optimize and record several metrics;
- Compare error differences before and after optimization, convergence iterations, initialization runtime and optimization runtime;
- Compare total runtime as the sum of initialization runtime and optimization runtime;
- Count the best performance for each compare group for each metric;
- Generate plots of 2D and 3D graphs before and after optimization.

This is the way we quantify the performance.

C. Summary of Results

We attempted to run the experiments on multiple computers and the results were different. This may be due to parallel computing of multiple CPU's. For the purpose of rigorous scientific approach, the results were collected from the same computer.

The **running environment** is:

- Operating System: Oracle Linux(64 bit) ubuntu-20.04.6 by using Oracle VM VirtualBox
- CPU count:2
- 13th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz
- Memory:4096MB

Here is the brief summary of results in **Figure 2**.

Operating System:	**Oracle Linux(64 bit)ubuntu-20.04.6**		**Oracle VM VirtualBox**	
CPU:2	**13th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz**			**Memory:4096MB**
Summary best counts on all datasets	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	**14**	0	0	0
initial error before optimization	1	1	**7**	**7**
error after optimization	6	11	**12**	**12**
iterations of optimization	4	11	**13**	12
runtime of optimization	2	2	**5**	4
total runtime	3	**7**	3	0

Fig. 2. brief summary of results

For visualizing plots of these four compare groups before and after the optimization of all 14 datasets, please take a look at **Appendix E**.

Detailed numerical results for each dataset in a table: Please see **Appendix F**.

EXPLANATION: Unit of Error

The error value is a scalar that has no unit, which we have confirmed on the internet.

- In this experiment, the error is a very important evaluation metric to measure the accuracy of the algorithm in generating the initial estimates and the effectiveness of the subsequent optimization. When optimizing using the gtsam library, the optimization error is a scalar expressed by the negative log probability. Specifically, that is, it is equal to the sum of the residual squared Mahalanobis distances configured by the current factor graph. Therefore, this error is unitless, as it reflects only the value of the optimization objective in a statistical sense, and not a specific physical quantity.

- The optimization error can be used not only to quantitatively compare the performance of different algorithms on the same dataset, but also to analyze the trend of the changing error, helping us understand the speed of convergence of the optimization process of the algorithms as well as the effectiveness of the modified algorithms.

V. Conclusion

From our experiments on various datasets, we can see a range of results for the improvements on the MASAT algorithm. Overall, MASAT and our two proposed implementations have better results than original data in optimization. This aligns with the result in the original MASAT paper.

There are 14 datasets in total. All of them can be initialized with MASAT, MASAT-sa, MASAT-rbfs correctly except "**Dataset 12** Data Containing Several Incorrect Data". **Dataset 12** also causes runtime error when reading the original data as input. The only successful initialization and optimization with **Dataset 12** is by using our extra initialization method discussed in previous section. The compatibility between this method and MASAT variation algorithms' implementation is a potential future extension of this project.

The **TABLE I** shows the percentage change of metrics compared to MASAT.

	MASAT-sa	MASAT-rbfs
Initialization runtime	107.836	2958.046
Initial error	-35.094	-47.864
Post optimization error	0	0
Iterations of optimization	-2.272	-2.5
Optimization runtime	-3.627059061	-3.026571737
Total runtime	6.6135	1244.1741

TABLE I
PERCENTAGE CHANGE OF METRICS COMPARED TO MASAT

A. Total Runtime Analysis

Please take a look at the "**Figure 2** brief summary of results".

- Out of the 13 datasets except **Dataset 12**, MASAT algorithm has the best total runtime on 7 of them, which is the highest count among four compare groups.
- But The original data has the best total runtime on 3 of them.
- Our implemented MASAT simple average version has the best total runtime on 3 of them, as well.

This performance benchmarking with the original paper proves that in certain scenarios, our proposed algorithm

MASAT-sa can beat the former MASAT algorithm in total runtime.

The datasets that our MASAT-sa performs the best on total runtime are "**Dataset 2** CITY10K", "**Dataset 6** INPUT M3500 G2O" and "**Dataset 7** INTEL P G2O".

Despite of the inspiring outcome, we have no clue what is the causal similarity among the patterns of these 3 datasets. This is the realm of data mining, which could be a potential future extension of this project.

The mathematical reason that our algorithm MASAT simple average version beats former MASAT under some circumstances is that, both algorithms have the same time complexity of initialization runtime.

The total runtime is determined by the sum of initialization (data preprocessing) runtime and optimization runtime. The initialization runtime is in the magnitude of mili seconds, while the optimization runtime is in the magnitude of seconds.

When the extra initialization runtime of MASAT simple average version is less than the reduction of optimization runtime compared to former MASAT algorithm, our algorithm MASAT-sa is better than any other approach among 4 compare groups.

B. Statistical Analysis of MASAT-sa

Our of the 13 datasets except **Dataset 12**, MASAT-sa algorithm also has the best performance metrics "initial error before optimization", "error after optimization", "iterations of optimization", and "runtime of optimization" with the highest counts of datasets among 4 compare groups.

Best performed datasets / total datasets of MASAT-sa

- "initial error before optimization": 7/13 datasets;
- "error after optimization": 12/13 datasets;
- "iterations of optimization": 13/13 datasets;
- "runtime of optimization": 5/13 datasets.

This outcome illustrates the rationality of using our MASAT simple average version.

C. Analysis of MASAT-rbfs

As we expected, MASAT-rbfs has the best performance metrics "initial error before optimization" and "error after optimization" with the highest counts of datasets among 4 compare groups. It is 7 out of 13 on "initial error before optimization", 12 out of 13 on "error after optimization".

Our assumption was that smaller initial estimate error infers shorter runtime of optimization. This assumption is generally true, but "**Dataset 7** INTEL P G2O" shows a contrary result. The original data has smallest initial error before optimization without any form of initialization.

The original MASAT significantly worsens the initial estimate error before optimization of **Dataset 7** with its initialization. So do MASAT variations simple average and rolling BFS. However, the original data cannot even converge during the upper limit of 100 iterations of optimization. Initialization is necessary for optimization in this dataset.

Further more, in "**Dataset 11** SPHERE 2500", MASAT-rbfs has the worst initial error before optimization among 3 initialization approaches in this dataset, but it has the shortest optimization runtime. This is anti-logic to our assumption and objective.

The reason of unstable performance of MASAT and its variations will be discussed later in this conclusion section.

In reality, MASAT-rbfs time complexity is much higher than the former MASAT algorithm. When the initialization runtime costs too much, it does not worth applying this algorithm to get the lowest initial estimate error before optimization.

What is more, compared to MASAT and MASAT-sa, MASAT-rbfs gets the worst initial error before optimization in "**Dataset 5** Grid 3D", "**Dataset 9** SPHERE 3D", "**Dataset 10** TORUS 3D".

In "**Dataset 2** City 10k", MASAT-rbfs has only 45933 as initial error before optimization; MASAT-sa has 209396 as initial error before optimization. In contrast to our assumption and objective, MASAT-sa has better optimization runtime than MASAT-rbfs on this dataset.

However, the rolling BFS idea has the potential to be applied to any BFS-based algorithm in order to keep the whole structure of a graph constrained strictly and explore the graph gradually with intentions such as early stopping or extreme precision. It is also possible that iterative deepening BFS algorithm works better with some algorithms other than MASAT. This is also a potential future extension of this project.

D. Analysis of Optimizer

Typically, the reduction in post optimization error due to these modifications is small due to the already-effective nature of the strong optimizer we use in our experiments: Levenberg Marquardt Optimizer. In many of the experiments, such as the one on "**Dataset 7** INTEL_P_G2O", the error after optimization and the number of iterations were the same between MASAT and the two improvement algorithms. This occurs due to the strong optimizer - the gtsam library Levenberg Marquardt Optimizer.

The mechanism of this optimizer combines the stability of gradient descent with the fast convergence of Gauss-Newton. It also adaptively balances exploration (large λ) and refinement (small λ) where λ is the damping parameter. This optimizer is also known for its limitation of being sensitive to the initial guess.

In most of our experiments, the optimizer completes the optimization in few iterations. The count of convergence iterations of every dataset after any form of initialization is below 5, except "**Dataset 14** Noisy 2D Circle" and the special case "**Dataset 12** Data Containing Several Incorrect Data". Maybe we should try some weaker optimizers to see the gap in iterations among different initialization algorithms as potential future extension of this project, because optimization runtime highly depends on number of iterations.

There was only one dataset "**Dataset 10** TAURUS 3D" in which the MASAT-rbfs initialization causes iterations of optimization to actually increase compared to MASAT. Obviously, the error before optimization is tripled when we use MASAT-rbfs compared to MASAT in this dataset. The MASAT-rbfs initialization algorithm was not supposed to generate worse initial error before optimization.

E. Discussion: Why using initialization data as input can be worse than using original data as input in very few cases?

The reason behind is as follow:

- MASAT and its variations of initialization methods do not consider the weight(certainty) of each edge, which is given by the constraint of the noise models represented as information matrix in input data(.g2o format).
- This deficiency may cause the initialization to be inaccurate and worse than original data, even though we use approaches that exploit all neighbors of every node.
- Sometimes the weights of edges are extremely unbalanced, and a large number of such edges link to a large number of neighbors. This will cause our strategy to fail, because we average all neighbors' contributions. The original MASAT algorithm itself averages "fixed" neighbors' contributions, which will also fail in metrics compared to using original data as input without initialization for optimization.

But on the other hand, as the preprocessing of data, initialization should not fully optimize the graph.

F. Summary of this Project

This project successfully explored the MASAT algorithm, an efficient method using BFS to generate an accurate initial guess for a SLAM problem. We also tested 2 proposed modifications to the algorithm, using "simple average" and "rolling breadth-first-search" methods. Our experiments used existing and generated datasets which map to real-life scenarios for applications of MASAT.

Our experiments address the limitations of the MASAT algorithm in various scenarios by evaluating among compare groups. The experimental results show that these two modified versions of the MASAT algorithm have their advantages and disadvantages.

While MASAT-sa maintains the strength in terms of computational efficiency, it can also make full use of the voting contributions of all the surrounding neighbours, which can improve the overall global accuracy and total runtime in some datasets. On the other hand, MASAT-rbfs progressively refines the estimation through a BFS version of iterative deepening approach. (By the way, the original iterative deepening is not using BFS, but Depth First Search.) The purpose of proposing MASAT-rbfs was to achieve a more accurate initialization, but it becomes too cumbersome in terms of computation.

Both methods, although different, effectively reduce the initial estimate error before optimization and shorten optimization runtime. MASAT simple average version even beats the original MASAT in measurement of total runtime in some datasets in our experiments. This project can rigorously show the important role of initialization in SLAM optimization problems.

APPENDIX A
MASAT ALGORITHM PSEUDOCODE

This is the pseudocode of former MASAT algorithm from the paper of their authors.

Algorithm 1: MASAT algorithm.

MASAT (V, E, Z);
Input : Raw measurements, vertices $v_i \in V$, edges $e_{ij} \in E$ and measurements z_{ij} for each edge e_{ij}
Output: An initial guess for the real pose-graph
Initially every vertex is labeled "not fix"
set vertex v_0 as the origo and its label is "fix"
Choose all the neighbors of vertex v_0 into the set Q
while $Q \neq \emptyset$ **do**
  w = the index of a vertex from the set Q.
  for *edges* e_{wi} **do**
    if *vertex* v_i *is "not fix"* **then**
      put vertex v_i into the set Q
    else if v_i *is "fix"* **then**
      add measurement z_{iw} to our existing estimation
      Modify our existing estimation for the position of vertex v_w with our new estimation
  end
  set the position of vertex v_w
  vertex v_w is "fix"
  take vertex v_w out from the set Q
end

Fig. 3. MASAT algorithm pseudocode

APPENDIX B
OUR EXPLANATION OF MASAT ALGORITHM

This is the explanation from us in the purpose of easy understanding the MASAT algorithm.

The MASAT algorithm approximates the exact position of a node by using votes from its previously processed neighbors. The algorithm is as follows:

Input: Raw measurements, vertices $v_i \in V$, edges $e_{ij} \in E$, and measurements z_{ij} for each edge e_{ij}.
Output:An initial guess for the pose-graph.

1) Initialize every vertex as "not fixed".
2) Set vertex v_0 as the origin and label it as "fixed".
3) Add all neighbors of vertex v_0 to the set Q.
4) While $Q \neq \emptyset$:
   a) Let w be the index of a vertex from Q.
   b) For each edge e_{wi}:
      i) If vertex v_i is "not fixed", add v_i to Q.
      ii) If v_i is "fixed", use measurement z_{iw} to update the estimation of vertex v_w.
      More specifically, calculate the average of predicted poses of current vertex from the relations with all fixed neighbors.
   c) Set the position of vertex v_w and label it as "fixed".
   d) Remove v_w from Q.

The algorithm assigns each node's position based on an average of the positional votes from its neighbors. The calculation for each neighbor $x_j \in N_i$ is given by:

$$p_{x_j}(x_i) = (p_1, \ldots, p_d)$$

where d represents the dimensions.

APPENDIX C
BLOCK DIAGRAM

Block Diagram of the Proposed System	Tasks Completed and Components Worked on
Hongzhen Xu Who implemented (Graph2.cpp) as the simple average version of MASAT, and (Graph3.cpp) as the rolling BFS version of MASAT. Who implemented codes for running experiments on datasets: (experiment.py), and (ExperimentWithPrunedGraph.py).	Brainstorm and implement two modified algorithms based on the paper; Write codes for running experiments; Write codes for running experiments with strategy of prune and selection; Run experiments and collect results; Analyze results.
Shuyue Gao Who provided 7 datasets for our experiments and arranged overwhelming amount of experiment results into excel charts (comparison.xlsx).	Data collection and production; Experimental data sorting and result analysis; Run the experiment.
Ananya Tadigadapa Who wrote (generate-sparse-2D-circle.py) and (generate-noisy-circle-2D.py) to generate datasets.	Generation of 2 datasets to match MASAT application scenarios; Running of experiments; Analysis of experiment results.
Team work	Presentation, Slides and Report

Fig. 4. block diagram of the proposed system

APPENDIX D
DATASETS AND SOURCES

This is the table showing the datasets we use in our experiments.

Dataset	Source	Purpose
"10kHog-man"	MASAT repo	Validate and Compare
"FR079_P_g2o"	MASAT repo	Validate and Compare
"grid3D"	MASAT repo	Validate and Compare
"INTEL_P_g2o"	MASAT repo	Validate and Compare
"sphere3D"	MASAT repo	Validate and Compare
"sphere2500"	Internet	Minimum spanning tree graph
"sparse2Dcircle"	Generated for project	Sparse 2D dataset

Fig. 5. datasets part 1

Dataset	Source	Purpose
“city10k”	MASAT repo	Validate and Compare
“FRH_P_g2o”	MASAT repo	Validate and Compare
“input_M3500_g2o”	MASAT repo	Validate and Compare
“parking-garage”	MASAT repo	Validate and Compare
“torus3D”	MASAT repo	Validate and Compare
“DataContainingSeverallncorrectData”	Generated for project	Factor graph pruning
“noisyClosureCircle2D”	Generated for project	2D loop closure

Fig. 6. datasets part 2

APPENDIX E
PLOTS OF THE EXPERIMENTS

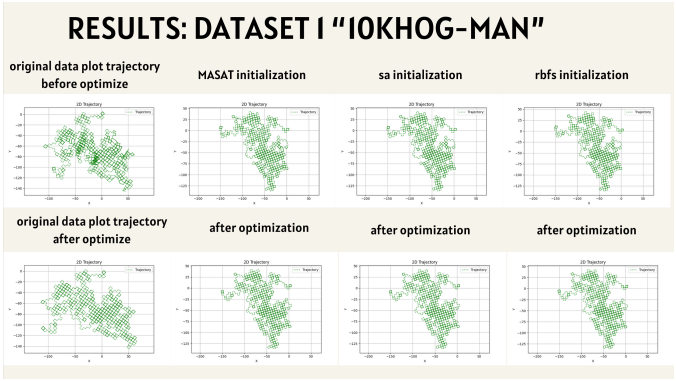

Fig. 7. dataset 1

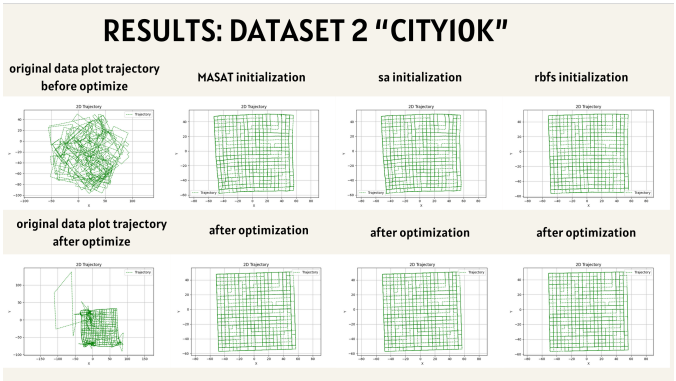

Fig. 8. dataset 2

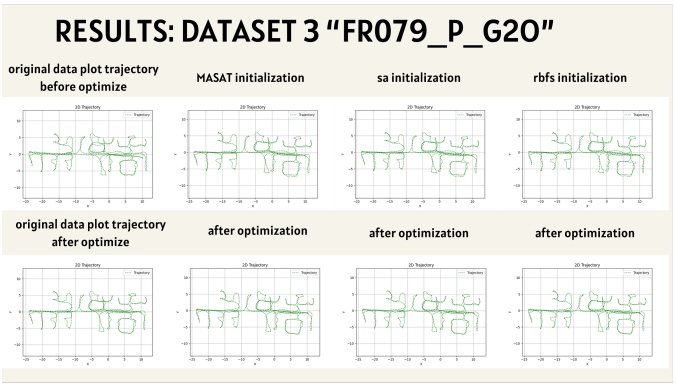

Fig. 9. dataset 3

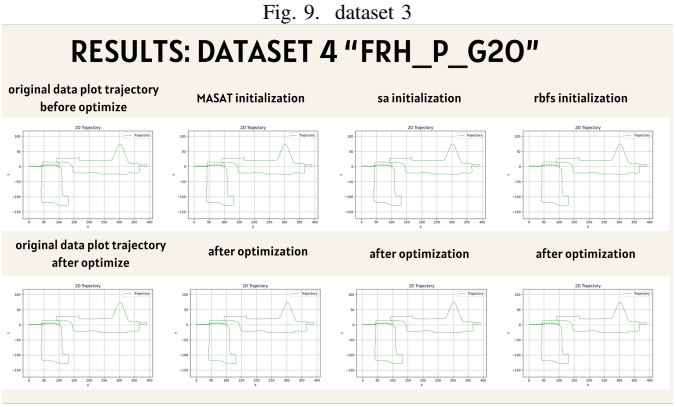

Fig. 10. dataset 4

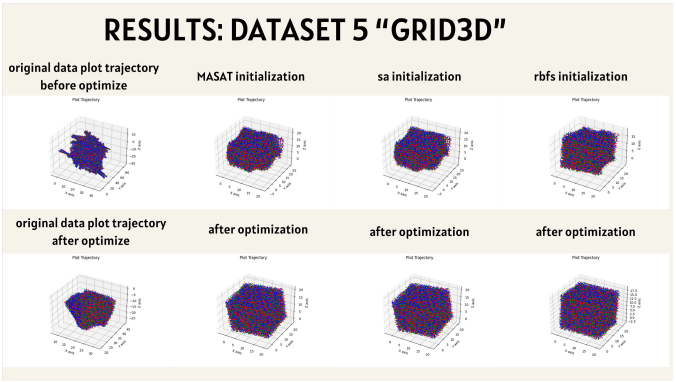

Fig. 11. dataset 5

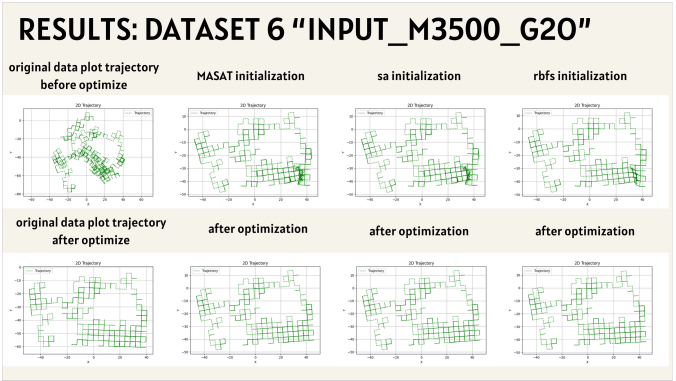

Fig. 12. dataset 6

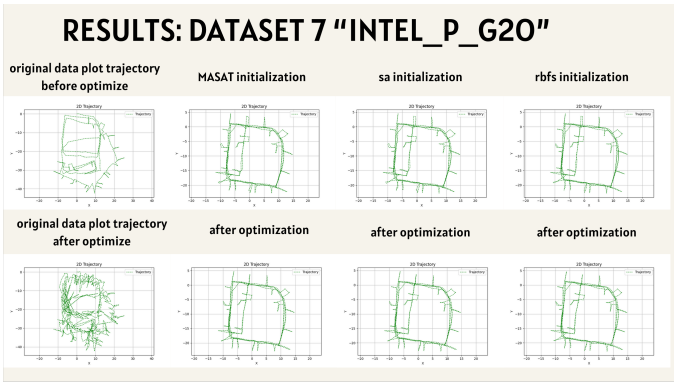

Fig. 13. dataset 7

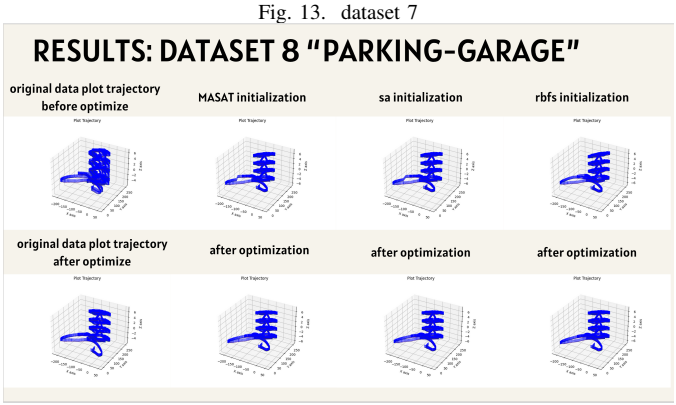

Fig. 14. dataset 8

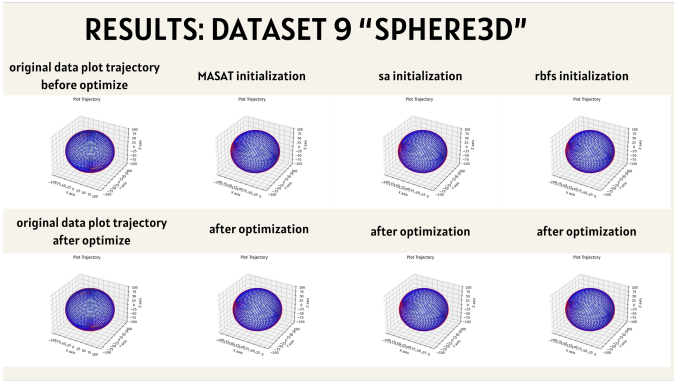

Fig. 15. dataset 9

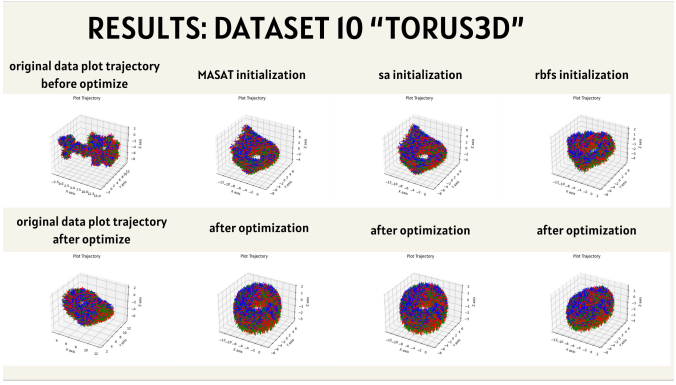

Fig. 16. dataset 10

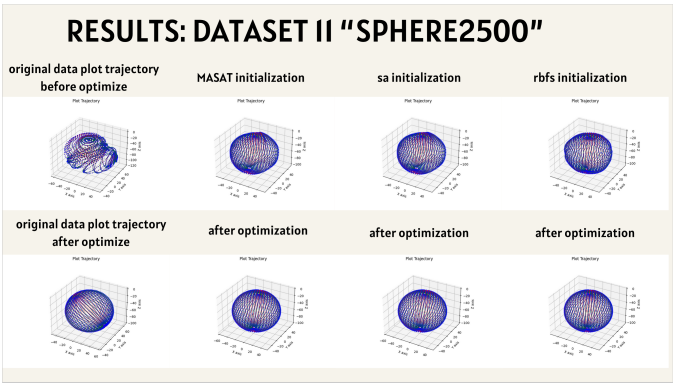

Fig. 17. dataset 11

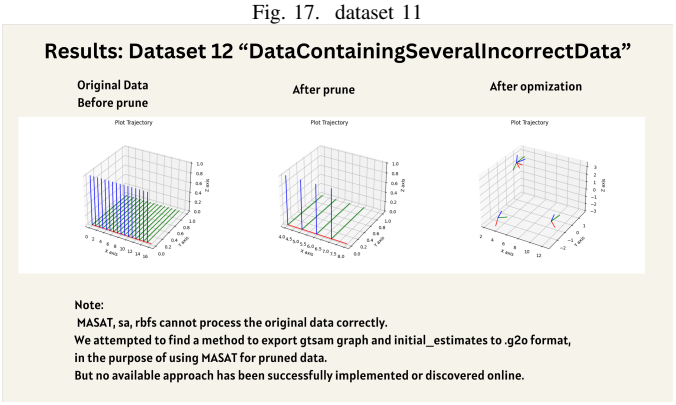

Fig. 18. dataset 12

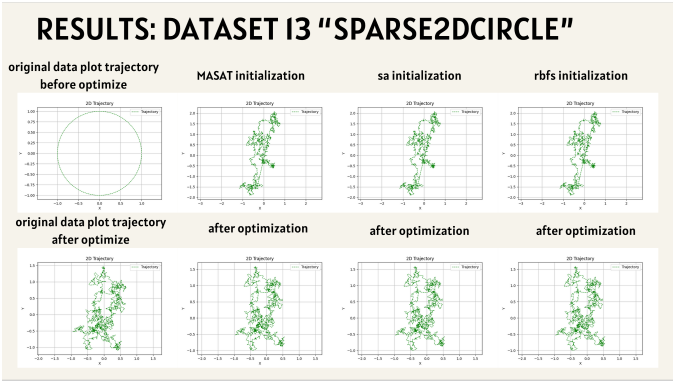

Fig. 19. dataset 13

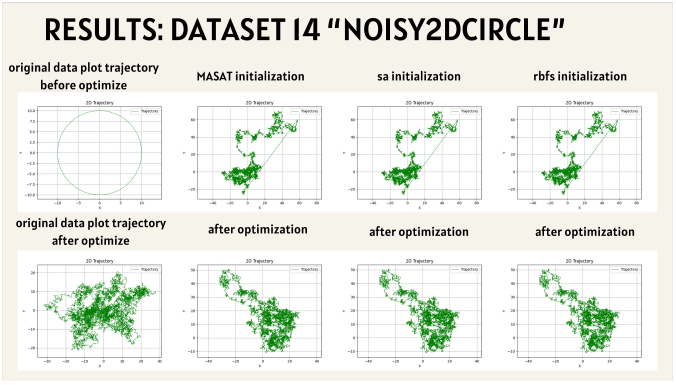

Fig. 20. dataset 14

APPENDIX F
TABLE OF ALL RESULTS

Operating System:	**Oracle Linux(64 bit)ubuntu-20.04.6**		**Oracle VM VirtualBox**	
CPU:2	**13th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz**			**Memory:4096MB**

Summary best counts on all datasets	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	**14**	0	0	0
initial error before optimization	1	1	**7**	**7**
error after optimization	6	11	**12**	**12**
iterations of optimization	4	11	**13**	12
runtime of optimization	2	2	**5**	4
total runtime	3	**7**	3	0

Dataset #1 10KHOG-MAN	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	99.67 ms	198.468 ms	14226 ms
initial error before optimization	3440446.7798	218974.1983	126043.5711	**92097.8377**
error after optimization	85797.3858	85797.3898	85797.3783	**85797.3761**
iterations of optimization	4	4	**3**	**3**
runtime of optimization	0.9834 seconds	0.8182 seconds	0.8061 seconds	**0.7787 seconds**
total runtime	0.9834 seconds	**0.9179 seconds**	1.0046 seconds	15.0047 seconds

Dataset #2 CITY10K	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	44.553ms	86.65ms	1642.11ms
initial error before optimization	16367858805.4112	418427.3508	209396.3935	**45933.2958**
error after optimization	299472801.0635	**15966.0018**	**15966.0018**	**15966.0018**
iterations of optimization	4	**3**	**3**	**3**
runtime of optimization	2.0051s	0.4408s	**0.3801s**	0.4061s
total runtime	2.0051 seconds	0.4854 seconds	**0.4668 seconds**	2.0482 seconds

Dataset #3 FR079_P_G2O	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0ms	3.341ms	5.023ms	194.029ms
initial error before optimization	4251.3868	2068.0006	756.7308	**205.3964**
error after optimization	18.8009	**18.8005**	**18.8005**	**18.8005**
iterations of optimization	4	**2**	**2**	**2**
runtime of optimization	0.0412s	0.0238s	0.0235s	**0.0200s**
total runtime	0.0412 seconds	**0.02714 seconds**	0.02852 seconds	0.2140 seconds

Dataset #4 FRH_P_G2O	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	4.552 ms	8.848 ms	581.042 ms
initial error before optimization	0.3410	0.02829	0.02484	**0.02280**
error after optimization	**9.6568e-05**	**9.6568e-05**	**9.6568e-05**	**9.6568e-05**
iterations of optimization	**2**	**2**	**2**	**2**
runtime of optimization	0.0370 s	0.0297s	0.0334 s	**0.0284 s**
total runtime	0.0370 seconds	**0.03425 seconds**	0.04225 seconds	0.6094 seconds

Fig. 21. table part 1

Dataset #5 GRID3D	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	134.98 ms	327.437 ms	4894.37 ms
initial error before optimization	95280764.1845	774453.1235	**589231.5635**	1031901.0716
error after optimization	357569.4874	42186.2507	**42186.2504**	42186.2505
iterations of optimization	100	**4**	**4**	**4**
runtime of optimization	1123.5207 s	**21.2651 s**	21.2897 s	21.6239 s
total runtime	1123.5207 seconds	**21.4001 seconds**	21.6171 seconds	26.5183 seconds

Dataset #6 INPUT_M3500_G2O	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	11.327 ms	21.894 ms	491.26 ms
initial error before optimization	1283333.8296	10588.2795	5731.2532	**1445.7827**
error after optimization	**68.9637**	**68.9637**	**68.9637**	**68.9637**
iterations of optimization	6	**4**	**4**	**4**
runtime of optimization	0.2227 s	0.1536 s	**0.1379 s**	0.1567 s
total runtime	0.2227 seconds	0.1649 seoncs	**0.1598 seconds**	0.6480 seconds

Dataset #7 INTEL_P_G2O	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	3.482 ms	6.318 ms	76.519 ms
initial error before optimization	**2471694.3100**	15164621055549.8	7148243919428.75	1822623475720.8276
error after optimization	217050.8442	**110.8397**	**110.8397**	**110.8397**
iterations of optimization	100	**5**	**5**	**5**
runtime of optimization	1.6421 s	0.1023 s	**0.0781 s**	0.1012 s
total runtime	1.6421 seconds	0.1058 seconds	**0.08442 seconds**	0.1777 seconds

Dataset #8 PARKING-GARAGE	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	34.881ms	80.168 ms	2459.04 ms
initial error before optimization	8363.6019	6.7071	**3.4876**	3.9071
error after optimization	**0.6342**	**0.6342**	**0.6342**	**0.6342**
iterations of optimization	**4**	**4**	**4**	**4**
runtime of optimization	**0.1381 s**	0.1394 s	0.1505 s	0.1445 s
total runtime	**0.1381 seconds**	0.1743 seconds	0.2307 seconds	2.6035 seconds

Dataset #9 SPHERE3D	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	119.214 ms	284.363 ms	3246.06 ms
initial error before optimization	67379969138.7516	88148429.2759	**68577508.3650**	95632255.0507
error after optimization	**126647.9106**	**126647.9106**	**126647.9106**	**126647.9106**
iterations of optimization	**3**	**3**	**3**	**3**
runtime of optimization	0.9438 s	**0.9138 s**	0.9779 s	0.9243 s
total runtime	**0.9438 seconds**	1.0330 seconds	1.2623 seconds	4.1704 seconds

Fig. 22. table part 2

Dataset #10 TORUS3D	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	57.399 ms	132.155 ms	3258.44 ms
initial error before optimization	2400615.0207	103393.6546	**79697.6812**	307110.7415
error after optimization	29981.4858	**12119.4771**	**12119.4771**	**12119.4771**
iterations of optimization	52	**4**	**4**	5
runtime of optimization	13.7226 s	0.7475 s	**0.7402 s**	0.9119 s
total runtime	13.7226 seconds	**0.8049 seconds**	0.8724 seconds	4.1703 seconds

Dataset #11 SPHERE2500	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	29.616 ms	68.692 ms	1673.8 ms
initial error before optimization	1305657.7118	**12807.1845**	**12807.1845**	35659.3783
error after optimization	**675.7425**	**675.7425**	**675.7425**	**675.7425**
iterations of optimization	5	**4**	**4**	**4**
runtime of optimization	0.4728 s	0.3047 s	0.3047 s	**0.2905 s**
total runtime	0.4728 seconds	**0.3343 seconds**	0.3734 seconds	1.9643 seconds

Dataset #12 DataContainingSeveralIncorrectData	Original Data	Pruned Data
running time of initialization	0 ms	0.0167 seconds
initial error before optimization	nan	**9400**
error after optimization	Runtime Error	**3.7072e-05**
iterations of optimization	Runtime Error	**49**
runtime of optimization	Runtime Error	**0.0111 s**
total runtime	Runtime Error	0.0278 seconds

Dataset #13 sparse2Dcircle	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	2.496 ms	4.217 ms	571.95
initial error before optimization	1429.2916	843.0872	**421.3642**	**421.3642**
error after optimization	**1.7184**	**1.7184**	**1.7184**	**1.7184**
iterations of optimization	**4**	**4**	**4**	**4**
runtime of optimization	**0.0298 seconds**	0.0359 seconds	0.0359 seconds	0.0299 seconds
total runtime	**0.0298 seconds**	0.0384 seconds	0.0401 seconds	0.6019 seconds

Dataset #14 noisy2Dcircle	Original Data	MASAT	MASAT_sa	MASAT_rbfs
running time of initialization	0 ms	26.434 ms	51.575 ms	68500.5 ms
initial error before optimization	14930.3579	9648.7064	**4808.3661**	**4808.3661**
error after optimization	4.2796	**0.04375**	**0.04375**	**0.04375**
iterations of optimization	100	**11**	**11**	**11**
runtime of optimization	11.8062 seconds	0.7577 seconds	**0.7462 seconds**	0.8031 seconds
total runtime	11.8062 seconds	**0.7841 seconds**	0.7978 seconds	69.3036 seconds

Fig. 23. table part 3

APPENDIX G
FACTOR ERROR EXPLANATION

In factor graph optimization, the factor error is typically defined as the residual error between the measurement and the estimated value. This error measures how much each factor deviates from the overall constraints in the graph. The general formula for calculating factor error is as follows:

1. Factor Error Formula

Given a factor f_i, its error can be represented as:

$$\text{error}(f_i) = \|h_i(X) - z_i\|^2_{\Sigma_i}$$

where:

- $h_i(X)$ is the predicted measurement derived from the state variables X via the observation model function h_i.
- z_i is the actual measurement from sensors or observations.
- Σ_i is the covariance matrix of the noise model, representing measurement uncertainty.
- $\|\cdot\|^2_{\Sigma_i}$ denotes the weighted squared norm, calculated as:

$$\|h_i(X) - z_i\|^2_{\Sigma_i} = (h_i(X) - z_i)^T \Sigma_i^{-1} (h_i(X) - z_i)$$

Without a noise model or with a unit covariance, the error simplifies to the squared Euclidean distance.

2. Interpretation of the Error

High Error Values: A high factor error indicates a large deviation between the measurement z_i and the estimate $h_i(X)$, possibly suggesting an inaccurate initial estimate or an outlier measurement.

Weighting and Covariance Matrix: The covariance matrix Σ_i reduces the impact of high-uncertainty measurements on the overall error, making the optimization more robust.

REFERENCES

[1] K. Harsányi, A. Kiss, T. Szirányi, and A. Majdik, "Masat: A fast and robust algorithm for pose-graph initialization," *Pattern Recognition Letters*, vol. 129, pp. 131–136, 2020.

[2] K. Harsányi, "MASAT_IG_for_SLAM." https://github.com/karoly-hars/MASAT_IG_for_SLAM/tree/master, 2019.

[3] H. Xu, A. Tadigadapa, and S. Gao, "Masat_modifications_and_experiments." https://github.com/hx394/SLAM_initialization, 2024.