EECS 568 Mobile Robotics HW4
Graph SLAM

PoKang Chen

Introduction and Reading File

The Graph Slam is the sate-of-the-art and a prevalent method dealing with the SLAM problem. The advantage of the Graph Slam is to simply construct the graph with factors and poses and utilizing optimizer to solve for better trajectories. This work focus on the backend optimization for that we read 2D and 3D g2o file of vertices and edges then optimized by Gauss Newton Optimizer(Batch Method) and iSAM(Incremental Method).

The structure of the g2o files are as following:

1. 2D:

Vertex: id_1, x, y, θ

```
VERTEX_SE2 0 0.000000 0.000000 0.000000
VERTEX_SE2 1 0.000000 0.000000 -0.000642
VERTEX_SE2 2 0.000000 0.000000 -0.001180
VERTEX_SE2 3 0.011002 -0.000975 -0.003562
VERTEX_SE2 4 0.641008 -0.011200 -0.007444
VERTEX_SE2 5 0.696016 -0.011716 -0.098726
```

Edges: $id_1, id_2, x, y, \theta, q_{11}, q_{12}, q_{13}, q_{22}, q_{23}, q_{33}$

```
EDGE_SE2 0 1 0.000000 0.000000 -0.000642 11.111271 -0.249667 0.000000 399.999840 0.000000 2496.793089
EDGE_SE2 1 2 0.000000 0.000000 -0.000538 11.111224 -0.209222 0.000000 399.999887 0.000000 2497.312169
EDGE_SE2 2 3 0.011003 -0.000962 -0.002382 5898.288051 69216.359995 0.000000 813797.504789 0.000000 2488.132420
EDGE_SE2 3 4 0.630039 -0.007981 -0.003882 11.129692 2.115033 0.000000 251.862638 0.000000 2480.702442
EDGE_SE2 4 5 0.055010 -0.000106 -0.091282 274.195567 -2936.281177 0.000000 32782.895793 0.000000 2099.259006
EDGE_SE2 5 6 0.004599 -0.003282 -0.797272 97512.798743 -544001.571431 0.000000 3035217.135802 0.000000 773.949086
```

2. 3D:

Vertex: id_1, (x, y, θ) , (qw, qx, qy, qz)

```
VERTEX_SE3:QUAT 0 0 0 0 0 0 0 1
VERTEX_SE3:QUAT 1 4.15448 -0.0665288 0.000389663 -0.0107791 0.00867285 -0.00190021 0.999902
VERTEX_SE3:QUAT 2 8.31419 -0.173106 -0.0129024 -0.00802114 0.00792915 0.00172397 0.999935
VERTEX_SE3:QUAT 3 12.6035 -0.1837 -0.0848858 -0.0103926 -0.00786875 0.0167457 0.999775
VERTEX_SE3:QUAT 4 16.8016 -0.108137 -0.0601645 -0.00253999 0.00220233 0.0263555 0.999647
VERTEX_SE3:QUAT 5 20.9607 0.0310604 -0.085476 -0.00864918 0.00189323 0.0207459 0.999746
```

Edges: id_1, id_2, (x, y, θ) , (qw, qx, qy, qz) + (21 digits of upper triangle of 6x6 matrix)

```
EDGE_SE3:QUAT 0 1 4.15448 -0.0665288 0.000389663 -0.0107791 0.00867285 -0.00190021 0.999902 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.00073 -0.000375887 0.0691425 3
EDGE_SE3:QUAT 1 2 4.15971 -0.0912353 0.0567356 0.00272799 -0.000777724 0.00363979 0.999989 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 3.99998 -8.64848e-06 -0.00634076
EDGE_SE3:QUAT 2 3 4.28985 -0.0247738 -0.00423717 -0.00251893 -0.015912 0.0148755 0.99976 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.004 0.000250029 -0.126914 3.999 -
EDGE_SE3:QUAT 3 4 4.19815 -0.0648981 -0.0412507 0.00809376 0.00983641 0.0096526 0.999872 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.00125 0.000335718 0.0777606 3.999
EDGE_SE3:QUAT 4 5 4.16073 -0.0800714 -0.00683181 -0.00610256 -0.000133946 -0.00562439 0.999966 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 3.99985 4.94074e-06 -0.001483
EDGE_SE3:QUAT 5 6 4.53017 -0.123911 -0.0250554 0.00437116 0.00742819 -0.00548444 0.999948 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.00082 0.000123487 0.0597205 3.99
```

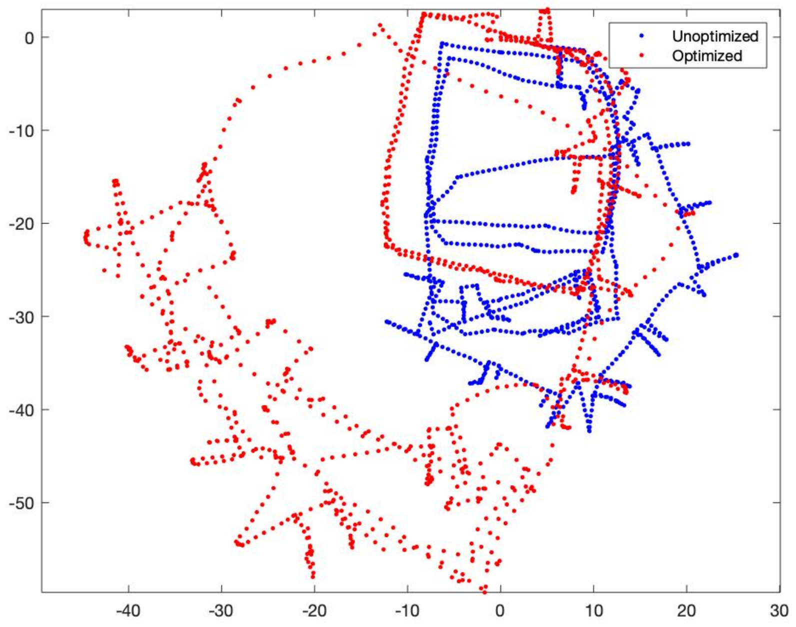
For 2D file, since I would need the access to each factor and pose, I further insert them into a vector as the return value from *readVertex()* and *readEdge()*.
The way I iterate though the **vertices** is as following.
$Key : pair\_vertex.first[i]->first$
$Pose : pair\_vertex.first[i]->second$
As to iterate through the **edges**, it should be as following:
$Key1 : pair\_edge.first[j]->first.first$
$Key2 : pair\_edge.first[j]->first.second$
$Factors : pair\_edge.second[j]$

For 3D file, I use basically the same code from dataset.cpp for *parse3DFactors()* and *parse3DEdges()*. From that, the poses and factors are obtained. The vertices are represented by (x,y,z) and (qw,qx,qy,qz) as the quaternion coordinate. The edges are represented by quaternion and the 6x6 information matrix.
The way I iterate though the 3D **vertices** is as following.
$Key : poses.find(i)->first$
$Pose : poses.find(i)->second$
As to iterate through the **edges**, it should be as following:
$Key1 : factors[i]->key1()$
$Key2 : factors[i]->key2()$

Task 1

A. Batch solution for 2D

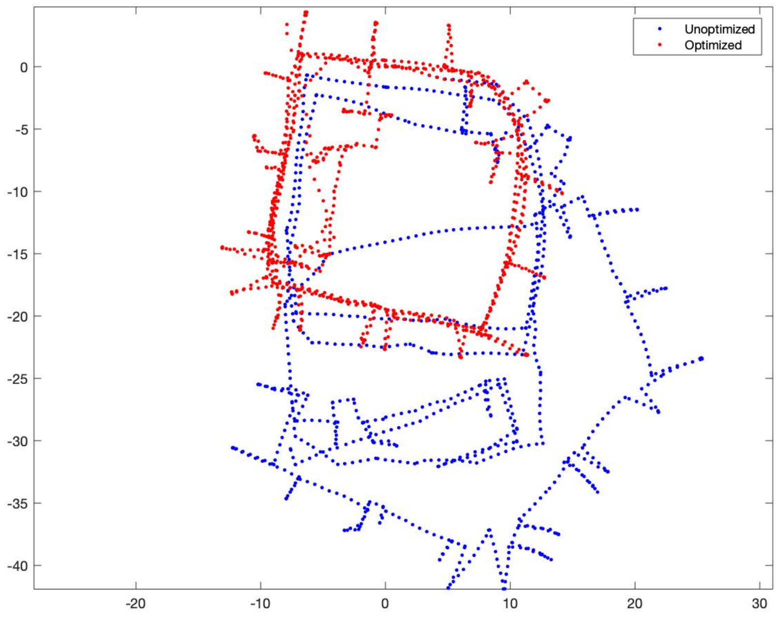
For Batch Method, the inputs of the Gauss Newton Optimizer is the graph and the initial vertices which read from the g2o file. Each of the edges is the constraint, or say the factor of the graph. The Batch Method feed the whole vertices(poses) and edges(constraints) together as a batch to the optimizer. The usual batch method would specify a size of the batch as one of the parameters, whereas in our case, we seen the whole dataset from the g2o file as a batch. So the batch size is same as the amount of data in the g2o file.



As in the plot of the unoptimized v.s. optimized trajectory of Intel g2o, it is obvious that it isn't a great solution actually. Although it seems on the track at first, to some certain extend, the optimization went off. The reason is that the initial value when solving the graph isn't optimal. Also the batch per step is way bigger than what optimizer expected. It is a more off-line optimized method.

B. Incremental Solution for 2D:

The concept of Incremental Solution is to basically incrementally optimize for each time step. After creating the graph and the initialEstimate(Values), a prior of the pose is added to the graph. Also the first pose read from the g2o file will also be added to the initialEstimate. We should first update the graph and the initialEstimate and optimize through function *calculateEstimate()*. Iterate through the vertex file of g2o and insert the

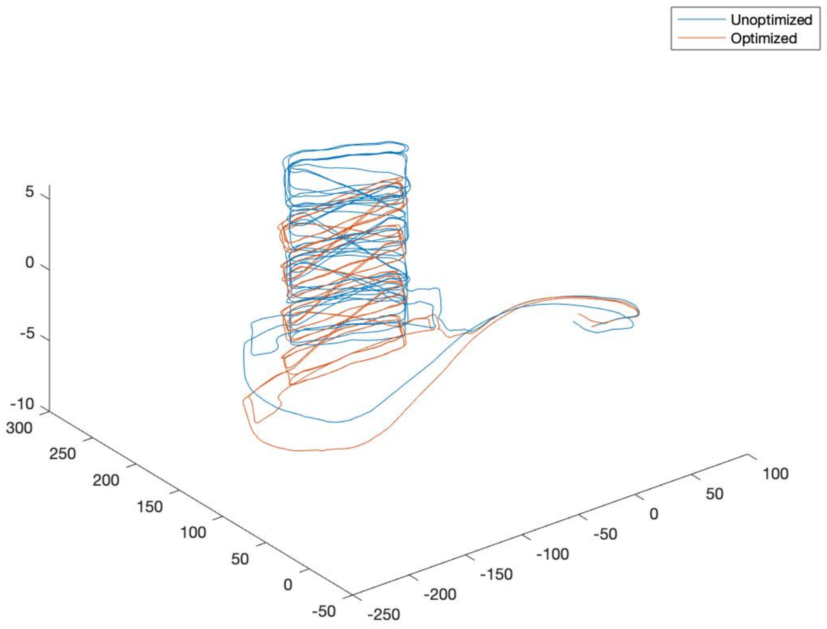


pose to the initialEstimate object. We then iterate through the edge file of g2o to find when the id_2 of the specific data equals the vertex number, it'll then add the edge to the graph. Finally the we update the value with isam function with the input being: **graph** and **initialEstimate**. For every iteration through the poses, the graph and the initialEstimate should be cleared and deleted to receive the new data from it then started the new loop.

Task 2

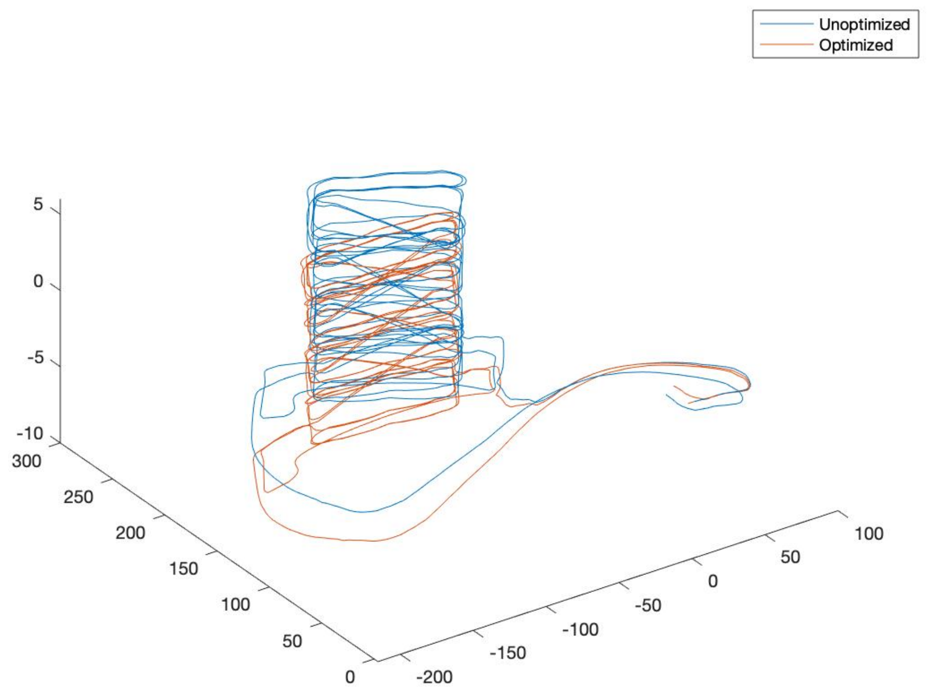
B. 3D Batch Method:

As for the 3D Batch Method, the situation is just the same as the 2D ones. What differs from 2D case is that the values of the input g2o is different from the 2D batch method. The dimension of the information matrix is 6x6 and the upper triangle value for the matrix is provided in the edge dataset. Other than that, basically it is just inserting the whole **factors** into the graph and set the **initial** as the whole poses read from g2o file. They are then set as the input of the Gauss Newton Optimizer. The following is the plot of the optimized and unoptimized dataset.



C. 3D Incremental Method:

For the 3D Incremental Method, the plot looks just as the batch method. The main reason may be that the initial value of this plot is better than the 2D dataset. The procedure of 3D Incremental Method is just like 2D Incremental Method. The inputs of isam update are as well **graph** and **initialEstimate.** We then utilize *calculateEstimate()*.


Unoptimized
Optimized