ROB312 TP5 : Graph SLAM with ICP

Dajing GU

January 2020

1 Introduction

1.1 Course summary

In the last class, we have studied the **EKF SLAM**, one of the mapping methods with backtracking. However, the increased complexity of computation with the number of landmarks $O(N^2)$, the data association problems and the limitation of linearization have limited the performance of this algorithm in a dense and huge map.

Some other methods like **Fast SLAM** and **Graph SLAM** are then be created. Particle filtering are used in Fast SLAM by breaking down the problem into trajectory / landmarks. Graph SLAM is the main stream method of SLAM, which separates the problem of SLAM into two parts: frontend and backend. The frontend is responsible for the data association and graph creation, while the graph optimization is finished in the backend.

1.2 Introduction of TP

In this practical work, we will work on two SLAM approaches based on Iterated Closest Point (ICP): a simple incremental SLAM approach and a simplified Graph SLAM algorithm. The provided code makes it possible to read laser scan datasets associated with the odometry of the robot during the acquisition, to register the scans using ICP and to map the environment.

2 Incremental SLAM

In this section, we will have a look at the **Incremental SLAM**.

2.1 Question 1

2.1.1 Algorithm of `icpIncrementalSLAM.py`

The principle of the algorithm is to update the next scans in the *scanList* and to add the satisfactory present scan to the *map*:

- For every one of *step* scans (key frame scan) in the *scanList*, the closest scan in the *scanList* will be used as a reference to compute a new scan position. (The distance between two scans is a combination of an euclidean distance with rotation differences.)
- ICP will then be performed to the original scan with its reference.
- The *scanList* will be updated by correcting the following scans.
- The present scan after correction will be added to the *map* if it is far enough from all the scans already in the map (> `distThresholdAdd`).

2.1.2 Difference between `icpIncrementalSLAM.py` and `icpLocalization.py`

The difference between the two algorithm is located at the method of choosing the reference scan and at the update of the *map*.

- In `icpLocalization.py`, it is the scan after *step* steps (the next key frame) that is used as the reference scan, while in `icpIncrementalSLAM.py`, it's the nearest scan is used.
- There's no update of map and no threshold to realize the update of *map* in `icpLocalization.py` but in `icpIncrementalSLAM.py`, there's `distThresholdAdd` that is responsible for it.
- The present scan is never corrected in `icpLocalization.py`.

2.2 Question 2

In this question we analyze the consequence of a closed loop in the loop detection. closed loop detection, also known as loop detection, refers to the ability of a robot to recognize that it has reached a certain scene and make the map closed.

In the code, it is realized by the parameter of `disrThresholdAdd`. When the distance between the present scan and the present map is far enough, the present scan will be added to the map. This explains the meanning of 'incremental' in the name of this algorithm. The future scans of odometry pose will be corrected but the previous scans already added in the map remain unmodified.

When such a distance is smaller than the value of `disrThresholdAdd` (the closed loop), which means the present pose of the robot is very close to the pose already noted in the map, the map will not be updated but the future scans of odometry pose will still be corrected.

2.3 Question 3

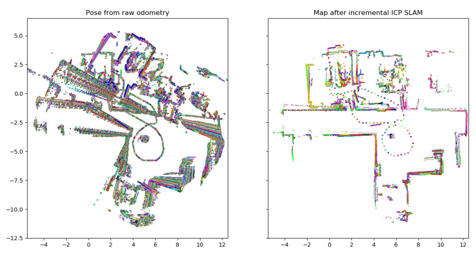
In this question, we analyze the role of the `step` and `distThresholdAdd`.

2.3.1 Role of the `step`

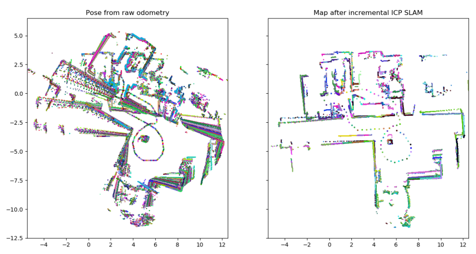
As have already been studied in TP1, the `step` refers to the key frame frequency. The value of `step` will influence the computation time and the quality of ICP.

The `distThresholdAdd` is fixed to **0.3** in this subsection. The results of different `step` are shown in figure 2.1. The sub-figure in the left are the direct superposition or stitching of pose from raw odometry, while the right sub-figure shows the map after incremenral ICP SLAM.

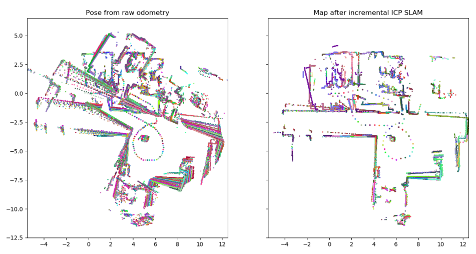
The change of the quality of localization is not proportional to the change of step. A too small step (e.g. step ≤ 2) causes too much time for computation, which brings us with a localization not so satisfactory and a long computation time. The small bias in each scan will also be accumulated to be a huge one, which greatly influence the quality of final localization. When there is a too big step (e.g. step ≥ 15), the quality of localization will neither be acceptable since there lacks enough superposition area between two scans so the ICP can not be well applied. A moderate value of step (e.g. step $= \mathbf{3}$) can be tried.



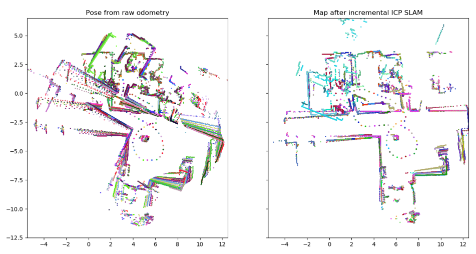
(a) Step = 1, Thres = 0.3



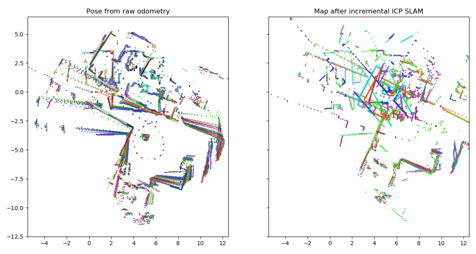
(b) Step = 3, Thres = 0.3



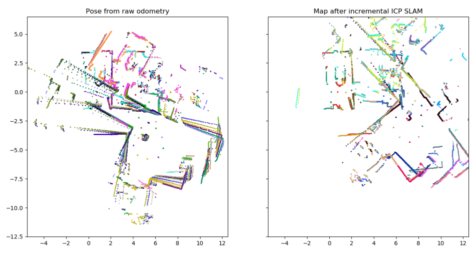
(c) Step = 5, Thres = 0.3



(d) Step = 10, Thres = 0.3



(e) Step = 15, Thres = 0.3

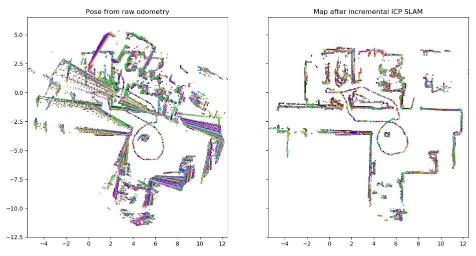


(f) Step = 25, Thres = 0.3

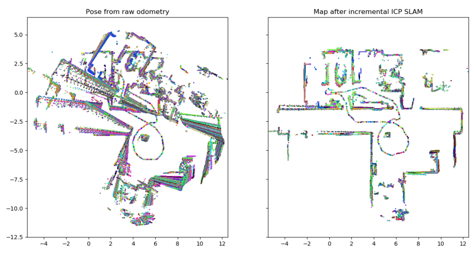
Figure 2.1: Figure of different steps

2.3.2 Role of the `distThresholdAdd`

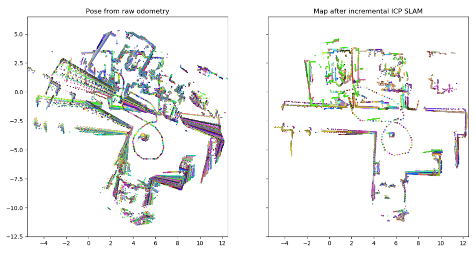
The `distThresholdAdd` is used to filter the poses used to be added to the map, those who are too close to the poses already exist in the map will not be added. In this subsection, the `step` is fixed to 3.



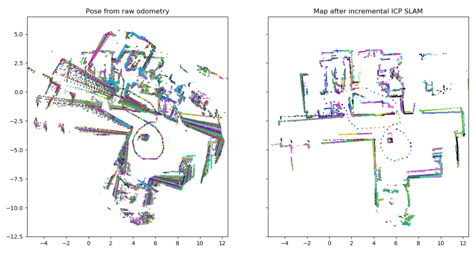
(a) Step = 3, Thres = 0.01



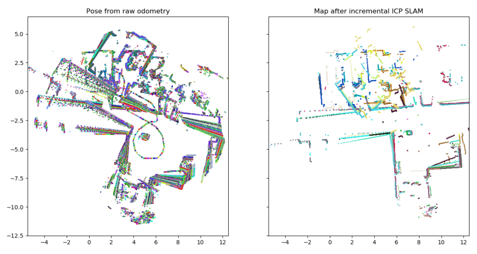
(b) Step = 3, Thres = 0.1



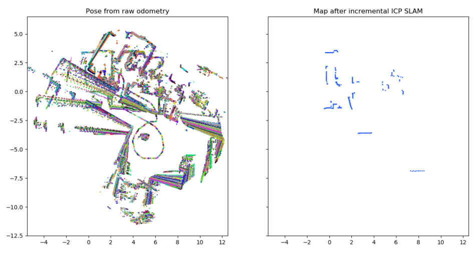
(c) Step = 3, Thres = 0.2



(d) Step = 3, Thres = 0.3



(e) Step = 3, Thres = 1



(f) Step = 3, Thres = 10

Figure 2.2: Figure of different thresholds

The results of different `distThresholdAdd` are shown in figure 2.2. The sub-figure in the left are the direct superposition or stitching of pose from raw odometry, while the right sub-figure shows the map after incremenral ICP SLAM.

It can be see from the figure 2.2 that the smaller the value of `distThresholdAdd`, the more possible it is to add the present scan to the map. So the difference between the map of time **t** and the map of time **t+1** is very small for a small threshold. An example of a very small `distThresholdAdd` = 0.01 is shown in the figure 2(a), too much information is upgraded in the map.

On the contrary, for a bigger value of `distThresholdAdd`, there is more difference between the map of time **t** and that of time **t+1**. Less information is upgraded in the map. An example of a too big `distThresholdAdd` = 10 is shown in the figure 2(f), where there is too little information to create a completed map.

2.3.3 Compromise of the parameters

An appropriate `step` = **3** which is not too big or too small and an appropriate `distThresholdAdd` = **0.2** which is relatively small are chosen as the best compromise of the parameters. the result of which is shown in the figure 2.3.

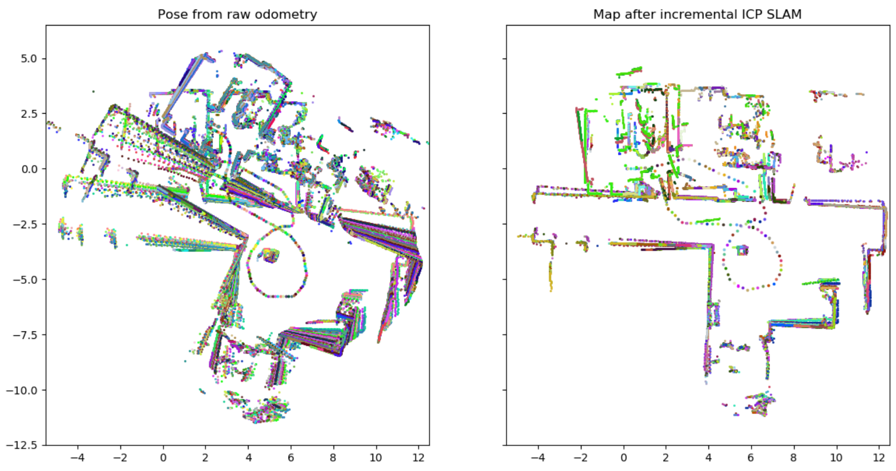


Figure 2.3: Best parameters (`distThresholdAdd` = 0.2 and `step` = 3)

3 Graph SLAM

In this section, we will study the **Graph SLAM**.

3.1 Question 4

3.1.1 Algorithm of `icpGraphSLAM.py`

The algorithm of the **Graph SLAM** is constituted by three main parts: addition of scans to the map, graph creation and graph optimization.

- **addition of scans to the map** : This part is similar to the principle explained in question 1. For every one of *step* scans (key frame scan) in the *scanList*, the closest scan in the *scanList* is used as a reference to compute a new scan position. The *scanList* will be updated by correcting the following scans. The original scan after correction will be added to the *map* if it is far enough from the present scan (> `distThresholdAdd`).
- **graph creation** : In every iteration, a list of scans which are close enough to the present scan will be obtained in the help of another parameter `distThresholdMatch`. For every scan in the list *closeScans*, the correspondent scan already added in the map will be used as the reference scan and ICP will then be performed to the chosen scan and the reference scan. A combination of the relative pose and the absolute pose will be used as the vertex to create a graph.

- **graph optimization** : The graph optimization is performed towards satisfying the edge constraints set by us, and finally an optimized map is obtained by computing new pose as mean of positions from neighbors.

3.1.2 Difference between `icpLocalization.py` and `icpGraphSLAM.py`

The difference between **Graph SLAM** is the update of the map:

- In `icpLocalization.py`, the map is updated by adding new scans to the map. The scans already added to the map will never be modified.
- However, in `icpGraphSLAM.py`, the map is updated not only by adding new scans but also by modifying the scans already added to the map (graph optimization). In the help of `disrThresholdMatch`, a list of scans close enough to the present scan will be obtained. The posotion of the scans in the map will be optimized by computing new pose as mean of positions from their neighbors.

3.2 Question 5

When a loop is closed, the robot recognizes that it has reached a scene already visited.

It is firstly realized by the parameter of `disrThresholdAdd`. When the distance between the present scan and the map is far enough, the present scan will be added to the map. The future scans of odometry pose will also be corrected but the previous scans already added in the map remain unmodified.

Then in the help of `disrThresholdMatch`, a list of scans which are close enough to the present scan will be obtained. A combination of the relative pose and the absolute pose between the present scan and the reference scan will be used as the vertex to create a graph. By satisfying the edge constraints, the map is optimized by computing new pose as mean of positions from neighbors. All the scans in the map can be corrected.

3.3 Question 6

In this question, we analyze the role of the `distThresholdAdd` and `distThresholdMatch`.

It should be noted that when `distThresholdMatch` < `distThresholdAdd`, the previous scans in the map can no longer be corrected because there are no enough scans in the *closeList*, the two algorithm become the same. So in this question, the value of `distThresholdMatch` should be set always bigger than that of `distThresholdAdd`.

3.3.1 Role of the `distThresholdAdd`

As is explained in question 3, the `distThresholdAdd` is used to filter the poses used to be added to the map, those who are too close to the poses already exist in the map will not be added. In this subsection, the `step` is fixed to 3 and `distThresholdMatch` is fixed to 0.8.

The smaller the value of `distThresholdAdd`, the smaller the difference between the map of time **t** and the map of time **t+1** and more information is upgraded in the map.

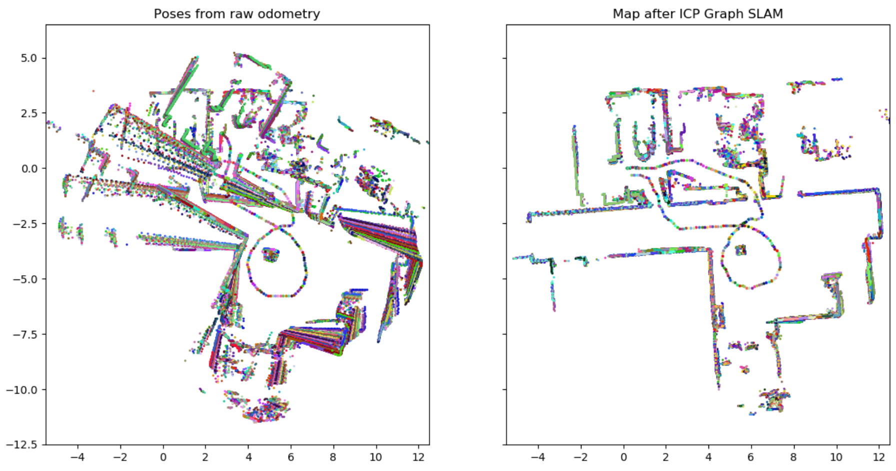


Figure 3.1: `distThresholdAdd` = 0.01 and `distThresholdMatch` = 0.8

An example of a too small `distThresholdAdd` = 1 is shown in the figure 3.1, it can be seen that there is too little information to create a completed map.

On the contrary, for a bigger value of `distThresholdAdd`, there is more difference between the map of time **t** and that of time **t+1** and less information is upgraded in the map. An example of a too big `distThresholdAdd` = 10 (`distThresholdMatch` < `distThresholdAdd`)is shown in the figure 3.2, it can be seen that there is too little information to create a completed map, just like the case of `icpIncrementalSLAM.py`.

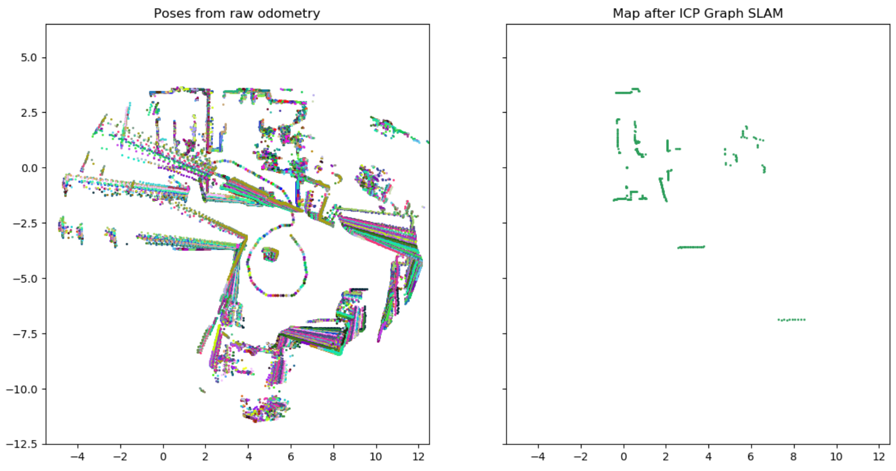


Figure 3.2: `distThresholdAdd` = 10 and `distThresholdMatch` = 0.8

3.3.2 Role of the `distThresholdMatch`

In this subsection, the `step` is fixed to 3 and `distThresholdAdd` is fixed to 0.3.

The `distThresholdAdd` is used to obtain a list of close scans which is used in the graph optimization. The bigger the `distThresholdAdd`, the more scans are included in the *closeList*.

If `distThresholdAdd` is too small, there will be few scans who are far enough from the map, whose result is presented in the figure 3.3. It has nearly the same performance as in the `icpIncrementalSLAM.py`.

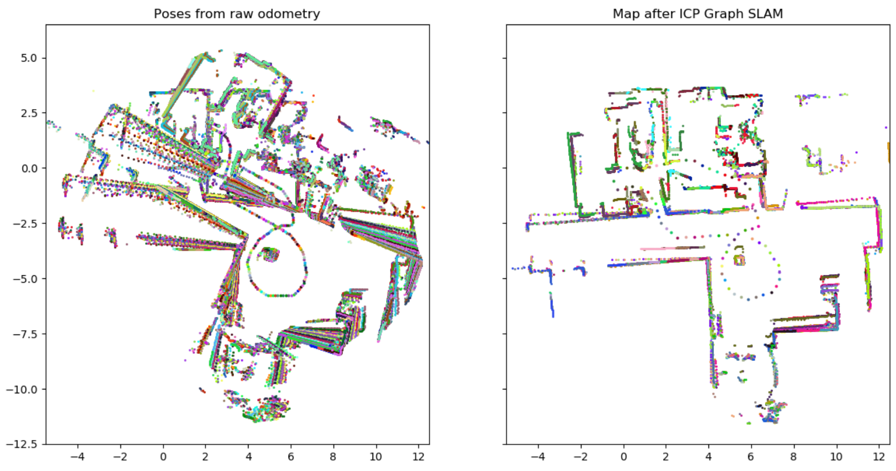


Figure 3.3: `distThresholdAdd` $= 0.3$ and `distThresholdMatch` $= 0.01$

If `distThresholdAdd` is bigger, more scans will be included in the list *closeList*. If it's too big, some scans not close enough will also be mis-included, which may influence the quality of graph optimization, as is presented in teh figure 3.4. What's more, the too big *closeList* causes also a too long computation time.

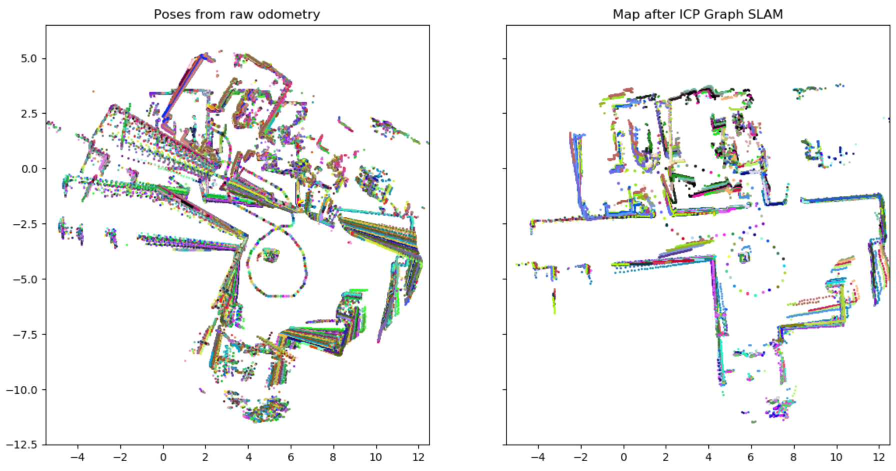


Figure 3.4: `distThresholdAdd` $= 0.3$ and `distThresholdMatch` $= 1.5$

3.3.3 Compromise of the parameters

`distThresholdAdd` = **0.25** and `distThresholdMatch` = **1** are chosen as the best compromise of the parameters. the result of which is shown in the figure 3.5. Compared with the result of the `icpIncrementalSLAM.py` presented in the figure 2.3, the map is better localized thanks to the graph optimization even though there still exists some uncertainty of localization in the map.

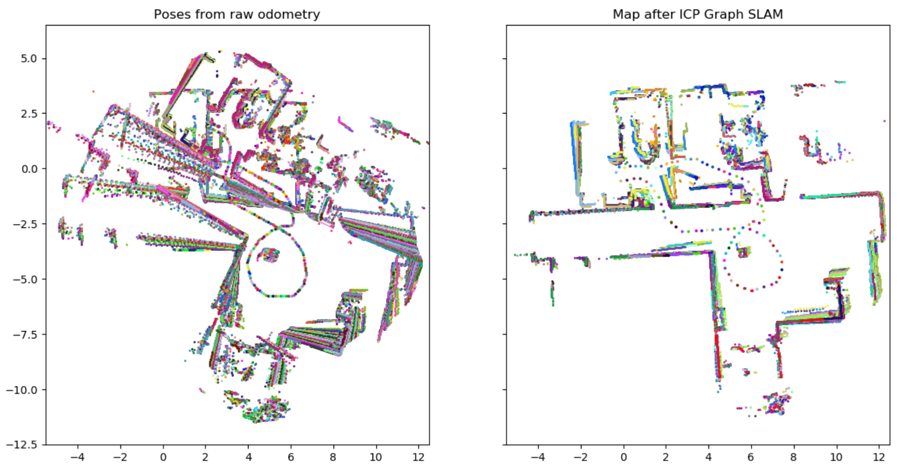
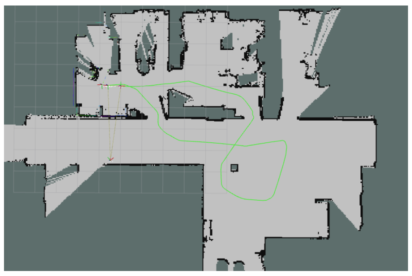


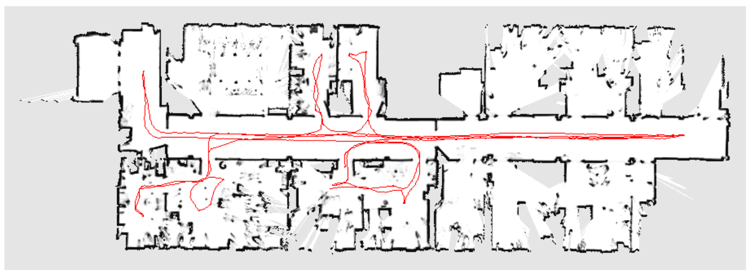
Figure 3.5: Best parameters (`distThresholdAdd` = 0.25 and `distThresholdMatch` = 1)

4 Influence of the environment

After the analysis of the two algorithms, we'll now have a look at the influence of the environment.



(a) Environment U2IS



(b) Environment FR079

4.1 Environment U2IS

In this environment, we set the parameters `minResolution = 0.05` and `best_matching = 0.85` in `icp.py`.

4.1.1 Incremental SLAM

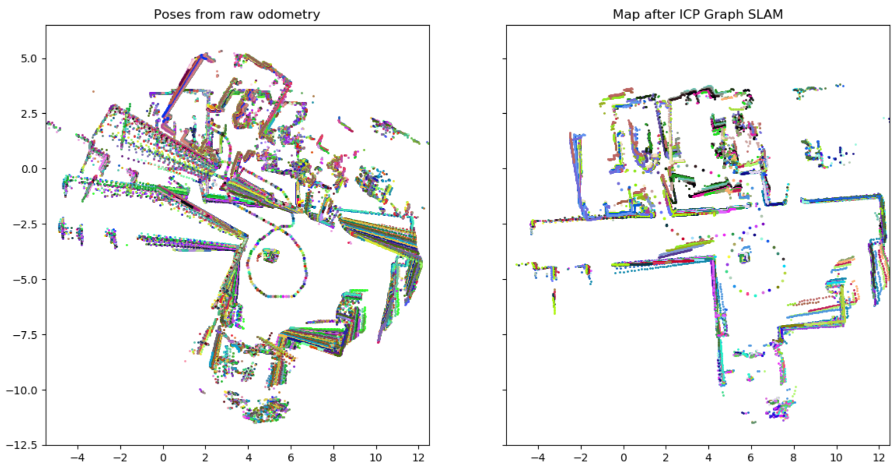


Figure 4.1: Result of Incremental SLAM with `distThresholdAdd` = 0.2 and `step` = 3

4.1.2 Graph SLAM

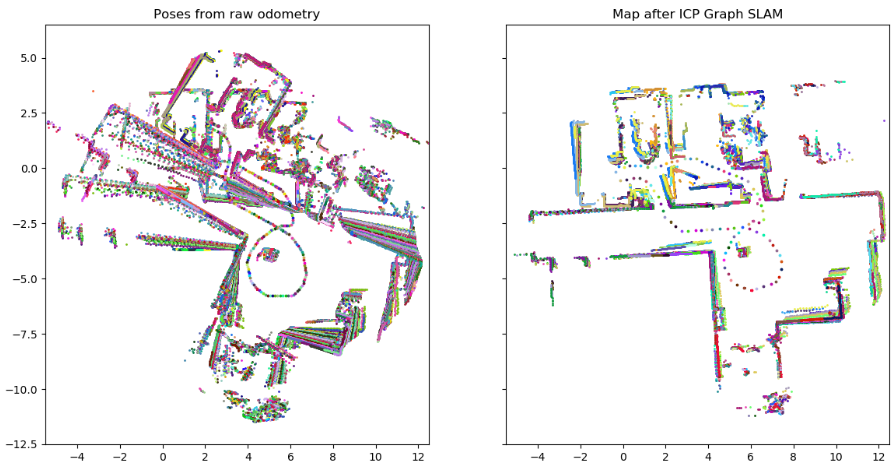


Figure 4.2: Result of Graph SLAM with `distThresholdAdd` = 0.25 and `distThresholdMatch` = 1

4.2 Environment FR079

In this environment, in order to have a better effect, we set the parameters `minResolution = 0.07` and `best_matching = 0.7` in `icp.py`. The relatively good parameters are found in the help of my classmate: Zheyi Shen.

4.2.1 Incremental SLAM

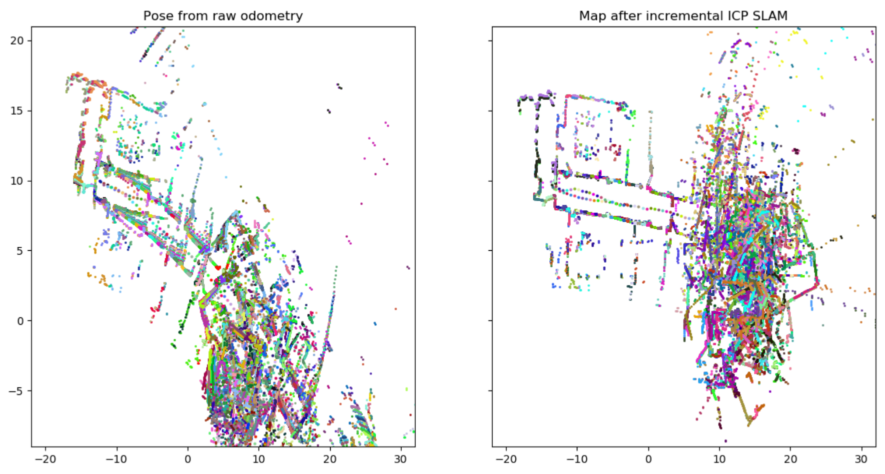


Figure 4.3: Result of Incremental SLAM with `distThresholdAdd` = 0.15 and `step` = 5

4.2.2 Graph SLAM

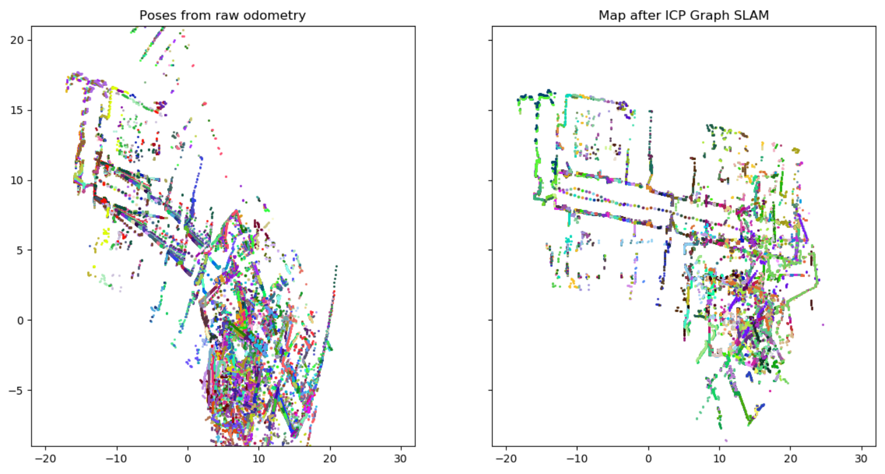


Figure 4.4: Result of Graph SLAM with `distThresholdAdd` = 0.15 and `distThresholdMatch` = 0.3

4.2.3 Conclusion

It can be seen from the figures above that no matter for the Incremental SLAM or Graph SLAM, the performance in the environment FR079 is always much worse than in the environment U2IS.

By comparing the two environments, we can find that the U2IS environment is much simpler than that of the FR079. There are fewer rooms in U2IS and it's much easier for the robot to find details in the map. However, there are more rooms which are more difficult to detect directly from the corridor so the robot has to enter into the room.

What's more, there is more superposition for the trajectory and more turns (change of direction) of the robot in FR079 than in U2IS, which also increases the difficulty.

In conclusion, it is the complexity of the environment and the nature of the trajectory that makes makes SLAM much more difficult to perform in the environment FR079.