ROB312 TP1 : Laser scan matching through ICP

Dajing GU

December 2020

1 Introduction

In this practical work, the laser scan correlation method named Iterated Closest Point (ICP) and some of its variants are implemented and tested. The basic code has been given, which is able to read laser scan datasets associated with the odometry of the robot during the acquisition and to implement the ICP method.

Two dataset are used in this TP, U2IS and FR079 and all the results on them are analysed in this report. However, dataset FR079 may be not suitable since the ICP test results on it are alsways very bad.

2 Principle of ICP

The Iterative Closest Point (ICP) is an algorithm employed to match two surface representations, such as points clouds or polygon meshes. The steps are shown in the part followed:

- Initialize points p_i from the target point cloud P / `dat`
- Filter points
- Repeat until convergence
  - ⋆ Find the closest points q_i from source point cloud Q / `ref`
  - ⋆ Filter correspondences (p_i, q_i)
  - ⋆ Calculate the rotation matrix R and the translation matrix t to minimize the error function: $d = \frac{1}{n}\sum_{i=1}^{n} \|p'_i - q_i\|^2$, with $p'_i = R \cdot p_i + t$
  - ⋆ Calculate the mean distance d_{means} between p'_i and q_i. Stop iteration if d_{means} small enough.

3 Questions

3.1 Q1 : Implement points filtering

In this question, the points too close to each other in the 'data' scan are removed by calculating the euclidean distance between a point and its neighbour (the point adjacent), which helps to reduce the computation and to improve the accuracy. The initial total number of data is 461, the results of points filtering with different thresholds are shown in table 3.1.

Threshold	Number of points	Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
0	461	1.84×10^{-1}	1.97×10^{-4}	1.72×10^{-2}	3.04×10^{-5}	1.27
0.02	264	1.93×10^{-1}	2.90×10^{-4}	1.60×10^{-2}	4.51×10^{-5}	0.86
0.05	91	1.90×10^{-1}	1.02×10^{-4}	1.32×10^{-2}	1.00×10^{-5}	0.35
0.2	31	2.25×10^{-1}	9.40×10^{-4}	1.14×10^{-2}	5.23×10^{-5}	0.26
10	1	4.28×10^{-1}	5.74×10^{-2}	1.38×10^{-1}	6.21×10^{-3}	0.17

Table 3.1: Results of points filtering with different thresholds using dataset U2IS

The data of the table 3.1 show that, the bigger the threshold of the distance is, the fewer points will be retained and thus less t_{mean} is needed. As for the errors and variances of the transformation matrix, the minimum will be obtained when the number of points retained are relevantly small but still enough for finish the computation. When there are too few points, the errors and variances will grow again. Finally, a threshold of **0.02** is chosen.

3.2 Q2 : Implement match filtering

In this question, the match filtering is implemented by keeping the `XX`% best matching. The computation can be reduced and the accuracy can be improved. The results of match filtering with different values of `XX` are shown in table 3.2.

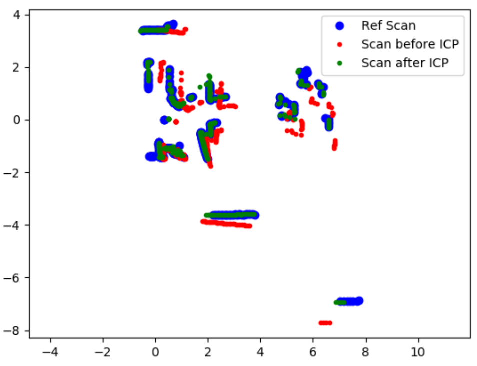
In order to have a better idea of the result of the match filtering, the Threshold is fixed to **0.02** in this question.

`XX`(%)	Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
10	4.06×10^{-1}	2.83×10^{-2}	1.37×10^{-1}	6.28×10^{-3}	0.6
30	3.77×10^{-1}	4.55×10^{-2}	8.54×10^{-2}	3.22×10^{-3}	1.27
50	1.71×10^{-1}	1.85×10^{-3}	1.69×10^{-2}	4.63×10^{-4}	2.19
60	1.88×10^{-1}	4.27×10^{-3}	6.20×10^{-2}	7.53×10^{-3}	1.75
70	1.58×10^{-1}	4.60×10^{-6}	5.1×10^{-3}	1.15×10^{-5}	1.43
80	1.59×10^{-1}	9.28×10^{-6}	5.31×10^{-3}	5.51×10^{-6}	0.99
90	1.62×10^{-1}	5.62×10^{-5}	7.76×10^{-3}	1.80×10^{-5}	1.28

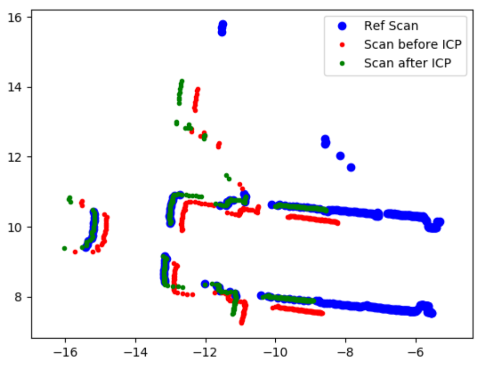
Table 3.2: Results of points filtering with different thresholds using dataset U2IS

It can be seen from the table 3.2 that once the matching filter is implemented, a bigger t_{mean} is needed to finish the iteration. A small value of `XX` will make fewer matching pairs retained, which explains the short t_{mean} and the relevant high value of errors and variances. On the contrary, a big value of `XX` will leave some wrong matching pairs retained, which causes a longer t_{mean} and higher value of errors and variances. A compromise between the t_{mean} and the accuracy should be made to achieve results. Finally, an `XX` of **70%** is chosen.

The illustration of result of ICP is represented in figure 3.2. In each sub-figure, there are 3 kinds of points: red, green, and blue. The blues points are the source points cloud as `ref`, the red points are the target points cloud which we want to align. And the green points are the points cloud after ICP. It can be seen that green points are closer to blue points than the red ones, which proves the effect of ICP.



(a) U2Is dataset



(b) FR079 dataset

Figure 3.1: Result of closest-point match in ICP test

3.3 Q3 : ICP Localization

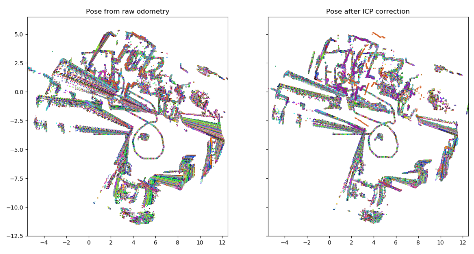
In this question, the `icplocalization` function is used to watch the effect of the ICP in localization. The results of different `step` are shown in figure 3.2. The sub-figure in the left are the direct superposition or stitching of the images scanned by the camera. The sub-figure shows the result of image stitching after an operation of icp every `step` times. The choice of step will influence the computation time and the quality of stitching.

As have been analysed in Q1, a big threshold of distance will remove more points, which brings us with a small t_{mean}. However, when too many points are removed, there may not be enough points for computation, which may cause big values of errors and variances of transformation matrix. In Q2, we've found that a small value of `XX` will make fewer matching pairs retained, which brings a smaller t_{mean}, but the error may be bigger for lack of enough correct matching pairs. So a reasonable compromise should be made between the accuracy and the t_{mean}.

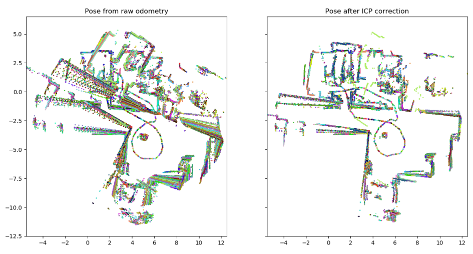
In Q3, threshold is fixed to **0.02** and `XX` is fixed to **70%**. Similar to the conclusions drawn in Q1 and Q2m Figure 3.2 the time spent in localization decreases with the increase of the value of step. The bigger the step is, the faster the localization.

The change of the quality of localization is not proportional to the change of step. A too small step (e.g. step ≤ 3) causes too much computation, which brings us with a localization not so satisfactory. The small bias in each scan will 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 a real-time correction. So a moderate value of step (e.g. step = **7**) can be tried.

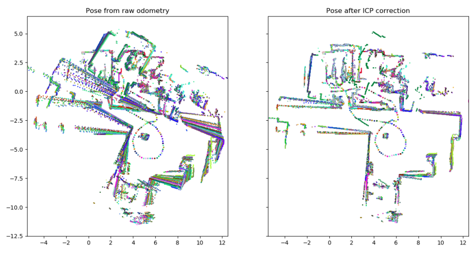
In conclusion, threshold = **0.02**, `XX` = **70%**, step = **7** is chosen.



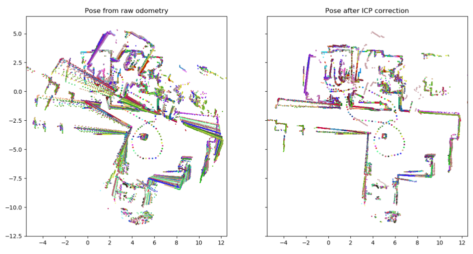
(a) Step = 1



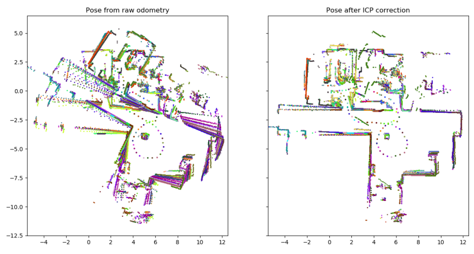
(b) Step = 3



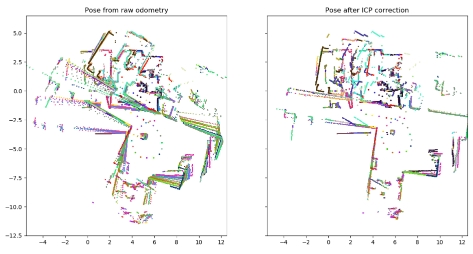
(c) Step = 5



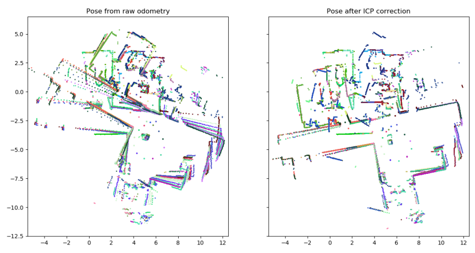
(d) Step = 7



(e) Step = 10



(f) Step = 15



(g) Step = 20

Figure 3.2: Figure of different steps

3.4 Q4 Implement matching using 'normal shooting'

3.4.1 Normal-shooting

In this question, instead of the nearest neighbor matching, the "normal shooting" matching is used. This time the closest points q_i from the source point cloud is no longer filtered by the euclidean distance, but by the normal vector N and the tangent vector T.

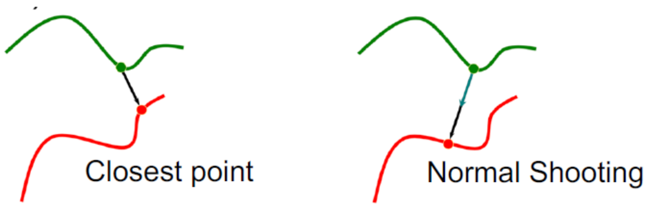


Figure 3.3: Difference between closest point and normal shooting

The tangent vector T is defined by two adjacent points in target point cloud `dat`. For each point A in target point cloud `dat`, we calculate the vector N of this point to all the points in source point cloud `ref`.

According to the formula: $cos(\theta) = \frac{T \cdot N}{\|T\| \cdot \|N\|}$, and thus by choosing the point in ref whose $cos(\theta)$ is the smallest will be chosen as the matching point of B. (A, B) will be a matching pair obtained by the method 'normal shooting'. It should be noted that *normal shooting* method is mainly used in the case of smooth curve, which means the normal vector of each point in target point cloud `dat` can be really found in the source point cloud `ref`.

Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
4.44×10^{-1}	1.62×10^{-3}	6.37×10^{-2}	2.22×10^{-4}	20.62

Table 3.3: Results of normal-shooting match for U2IS dataset

Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
12.31	2.76	2.30	8.18×10^{-1}	15.30

Table 3.4: Results of normal-shooting match for FR079 dataset

The data set we used in this practical work is discrete, which can not ensure the availability of the normal vector for every point. What's more, different from the code for Q1 and Q2 where the fast **KD tree** is used, the cycles used in this question cost much more time, so a bigger threshold has to be used to retain fewer points for computation. A tutorial of MIT [1] has used **BSP tree** to accelerate, but there's no precise explanation or implementation steps.

With **0.5** as thread for distance, the results of normal-shooting match are shown in figure 3.4.

3.4.2 Nearest-normal-shooting

Considering the huge error of the method of normal-shooting match, a combination of the closest-point matching method and the normal-shooting method is implemented. Which means that a filter is used after the normal-shooting match to eliminate the wrong normal vector calculated.

[1] http://groups.csail.mit.edu/graphics/classes/6.838/F01/lectures/IterativeAlgs/ICP/normal.html

Thanks to the elimination of wrong normal vectors, the nearest-normal-shooting method has a better performance than normal-shooting method for the dataset of U2IS, with less computation time and higher accuracy. However, for the dataset of FR079, it doesn't perform better.

Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
1.85×10^{-1}	1.55×10^{-2}	2.19×10^{-2}	3.35×10^{-4}	26.68

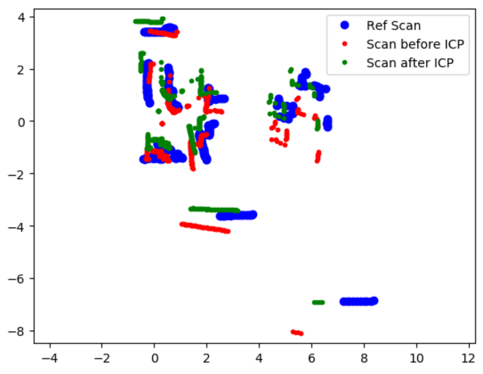
Table 3.5: Results of nearest-normal-shooting match for U2IS dataset

Err_T	Var_T	Err_R	Var_R	t_{mean} (s)
12.85	5.22	2.16	6.78×10^{-1}	19.64

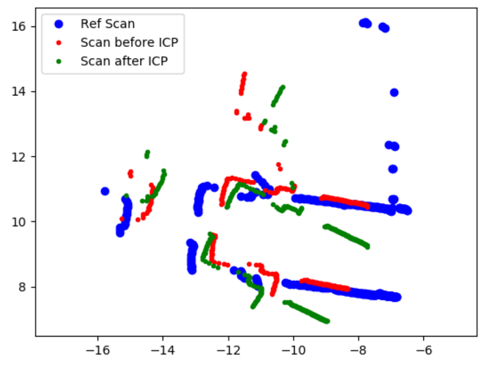
Table 3.6: Results of normal-shooting match for FR079 dataset

3.5 Results of two normal-shooting methods

In this section, the figures of two normal-shooting matching methods are represented.



(a) U2IS dataset



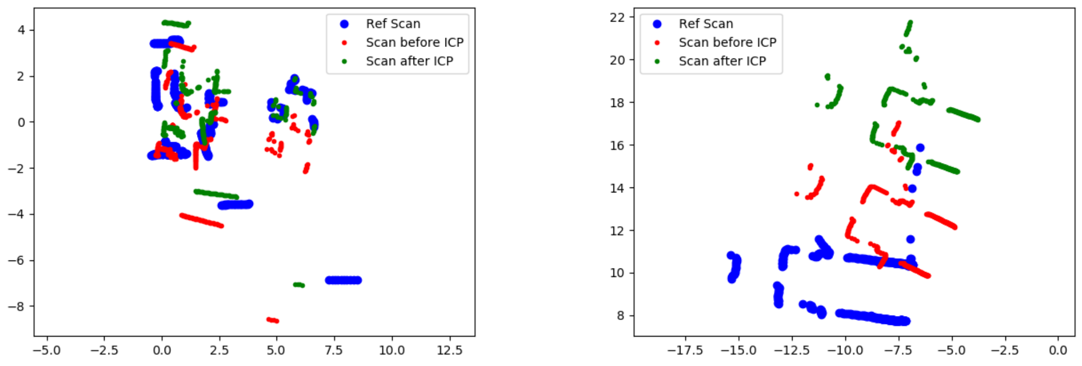
(b) FR079 dataset

Figure 3.4: Result of nomral-shooting match in ICP test

A Codes

A.1 Code for too-close points filtering

```
for i in range(dat.shape[1]-1):
    if np.linalg.norm((dat.T)[i+1] - (dat.T)[i]) < 1:
        delete_list.append(i)
dat_filt = np.delete(dat, delete_list, 1)
```



(a) U2IS dataset (b) FR079 dataset

Figure 3.5: Result of nearest-nomral-shooting match in ICP test

A.2 Code for match filtering

```
XX = 0.7
data_index = np.argsort(distance)[:int(len(distance) * XX)]
index = index[data_index]
dat_matched = dat_filt.copy()[:, data_index]
```

A.3 Code for normal-shooting match filtering

```
## closest-normal-shooting matching
norm_idx = np.zeros(dat_filt.shape[1], dtype=int)
cos_list = []
for i in range(dat_filt.shape[1] - 1):
    vect_tan = dat_filt[:, i + 1] - dat_filt[:, i]
    cos = np.ones(ref.shape[1])
    for j in  range(max(0,index[i]-10), min(index[i]+10, ...
                                  ref.shape[1])):
        vect_norm = dat_filt[:, i] - ref[:, j]
        cos[j] = abs(np.dot(vect_tan, vect_norm) / ...
                                     (np.linalg.norm(vect_tan) ...
                                     * ...
                                     np.linalg.norm(vect_norm)))
norm_idx[i] = np.argmin(cos)
cos_list.append(cos.min())
dat_matched = dat_filt
datat_index = norm_idx

XX = 0.5
data_index = np.sort(np.argsort(cos_list)[:int(len(cos_list) * XX)])
index = index[data_index]
dat_matched = dat_filt.copy()[:, data_index]
```