Laser scan matching through ICP Correction

David FILLIAT – ENSTA Paris

September 9, 2024

1 Question 1

Question 1: Implement points filtering (removing points too close to each other in the 'scan_points' array) and show the consequences (on error, variance, computation time, ...) as a function of the parameter values.

The following code removes the points in the scan that are too close from the previously accepted one, then add the first valid point to the point list and use it as the new reference. Note that it assumes there is not too much noise as this might lead to leave points that are too close to each other if a point between them in the scan is farther than the threshold. This is however quite rare in practice and this approximation performs well.

```
    minResolution = 0.05
    filtered_scan_points = [scan_points[:, 0]]
    for pt in scan_points.T[1:]:
        if np.linalg.norm(filtered_scan_points[-1] - pt) > minResolution:
            filtered_scan_points.append(pt)
    filtered_scan_points = np.stack(filtered_scan_points, axis=1)
```

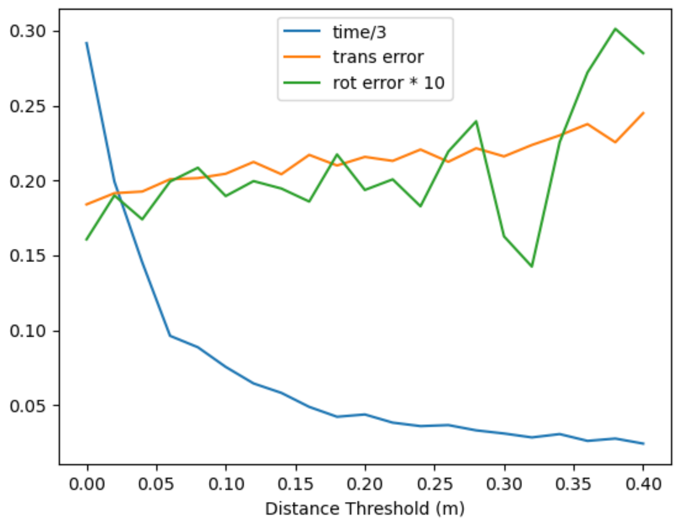

Figure 1: Effect of distance threshold on point filtering.

Testing this code with a parameter between 0 and 0.4 lead to the results reported in Figure 1. We can see that computation time steadily decrease, which is simply linked to the fact that there are less points to process afterwards. The performance almost steadily decrease, as less points are present and the discretization linked to the minimum spacing brings more and mode noise. Note also that the erratic behavior of the rotation error above 0.25 is linked to a higher variance in the results, which is apparent in the mean because we always use the same scans here for which some specific threshold are more adapted than other. Overall, we are then facing a quite common computation/precision tradeoff. A threshold around

0.05 seems to be a good compromise, which can be increased or decreased depending on the application requirements.

2 Question 2

Question 2: Implement match filtering by keeping a ratio $r < 1$ of the best matchings in the 'matched_scan_points' array. Try different values for r and show the consequences (error, variance, computation time, . . .).

The following code sorts the distance vector and takes the distance threshold corresponding to the ratio `r` of best values. Then points with distances below this value are selected.

```
    r=0.8
    sorted_dist = np.sort(distance)
    threshold = sorted_dist[int(r*(len(sorted_dist)-1))] # distance giving r\% match
    valid = distance <= threshold
    matched_scan_points = filtered_scan_points[:, valid]
    index = index[valid]
```

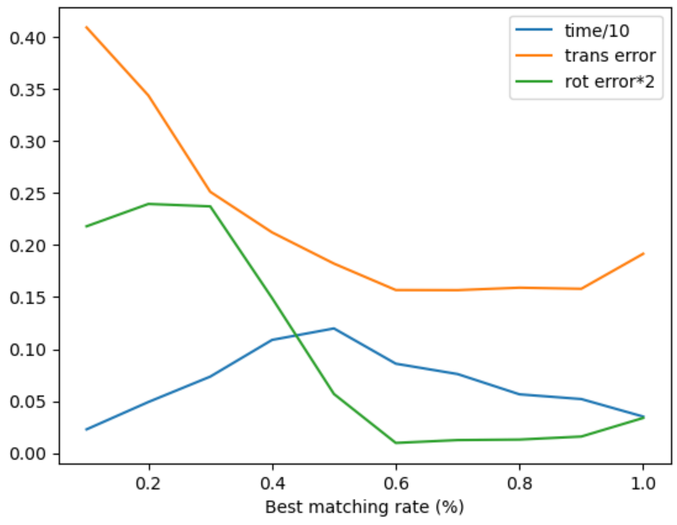

Figure 2: Effect of threshold on matching filtering.

Testing this code with a parameter between 10% and 100% lead to the results reported in Figure 2. We can observe that the computation time first grow (below 50%) and then decreases again. The initial growth is linked to the fact that more and more matching are used. The decrease after 50% is linked to the fact that with more matching, the updates are better and the algorithm converges with less iterations. The performance is quite bad with a low number of points and steadily increases up to 90%, but increases again at 100%. This is linked to the fact that there are almost always a few bad matching that are removed when the 10% of worst matching are discarded. A reasonable compromise is therefore to keep between 80% and 90% of the matchings.

3 Question 3

Question 3: Use the best setting you found with the `icpLocalization` function. Analyse visually the quality of the result as a function of the `step` parameter and of the parameters of the previous filtering. Propose a reasonable compromise between the quality of the localization and the computation time.

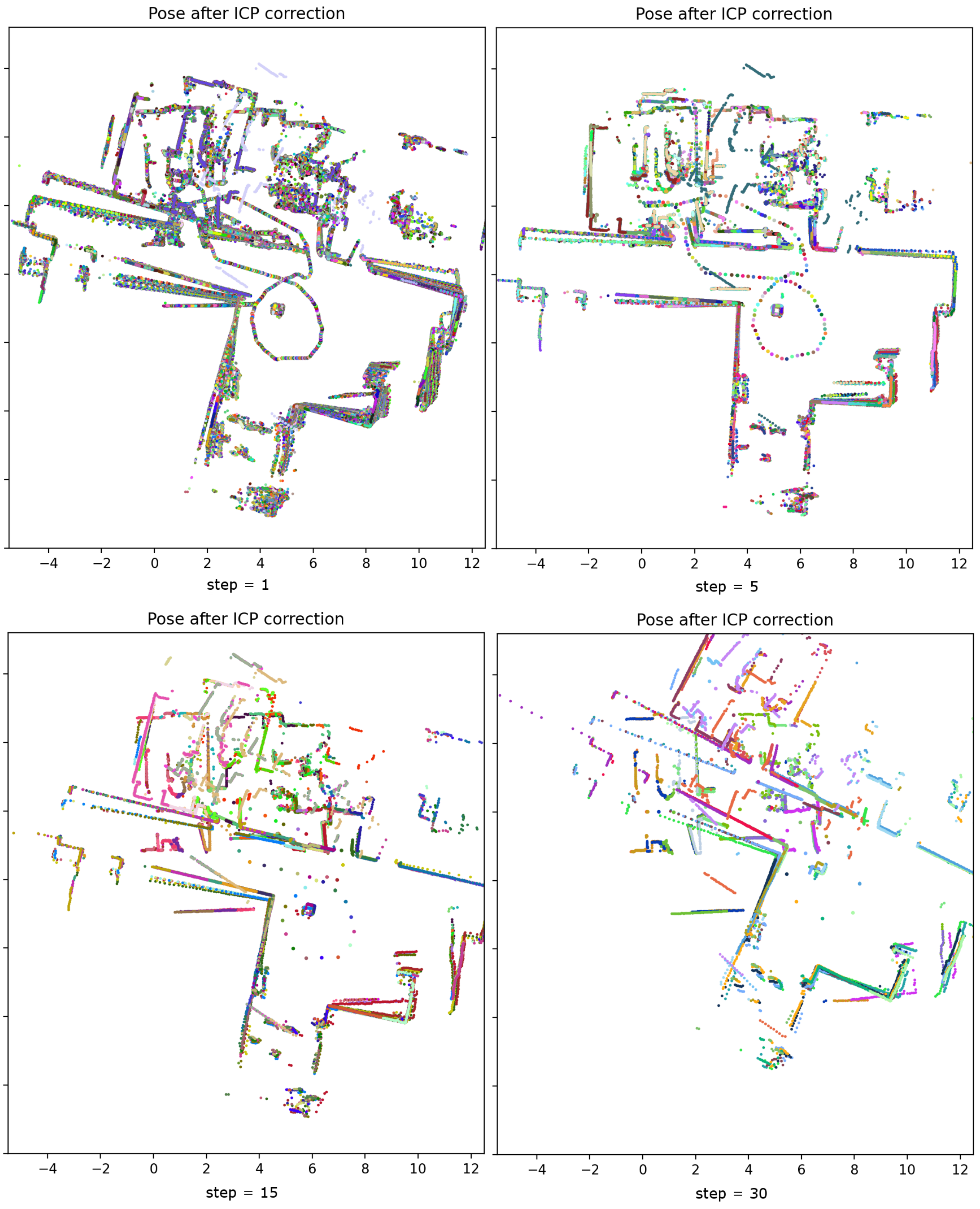

Figure 3: Effect of `step` on map quality.

Figure 3 shows the effect of the `step` variable on the map quality and Figure 4 shows the corresponding computation time. We can see that the best map is obtained for `step=5`, with a reasonable computation time and a good coverage of the environment features.

For `step=1`, the map quality is quite low because, while this configuration with very close scans is more favorable for ICP, applying ICP for each scan leads to accumulation of small errors at each iteration that lead to a worse map than for `step=5`. The computation time is also very high, because ICP is used more often, and the map contains much more scan which also add computation time for finding and matching to the nearest neighbor. We can also see that there are much more scans than needed to represent correctly the obstacles.

For `step=15` and `step=30`, the computation time continue to decrease, but the map quality degrades because ICP is performed on scans that are far from each other and therefore do not overlap correctly in some situations. It is for example the case when living the two rooms at the top and entering the corridor: matching scan taken inside and outside the room leads to a poor overlap and bad position estimate.

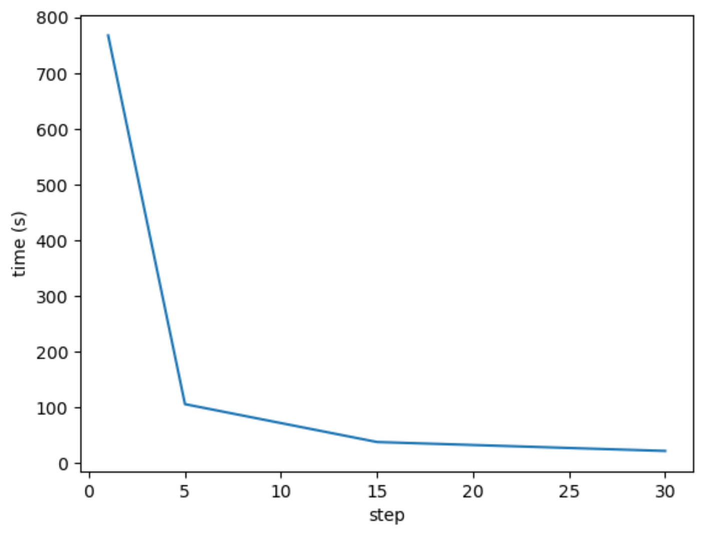

Figure 4: Effect of `step` on the computation time.

4 Question 4

Question 4: Implement filtering of the matchings by keeping only pairs of neighbor matching whose end-points are approximately at the same distance. Try different threshold values and show the consequences (error, variance, computation time, . . .).

The following code computes the distance between consecutive points of the scan and the distance between their correponding points in the reference. It then compares the two distances and keep only the matchings if the reference points are closer than the scan points distance plus a `threshold`. Note that filtering matchings where reference points are closer does not work well ass it prevents matching two points to the same reference which happens quite often.

```python
valid = [True]
for i in range(len(index)-1):
    dscan = np.linalg.norm(filtered_scan_points[:,i]-filtered_scan_points[:,i+1])
    dref = np.linalg.norm(ref_points[:,index[i]]-ref_points[:,index[i+1]])
    if dref > (dscan + threshold):
        valid.append(False)
    else:
        valid.append(True)
```

```

matched_scan_points = filtered_scan_points[:,valid]
index = index[valid]
```

Figure 5 shows the effect of the `threshold` parameter on the performance of the ICP algorithm. Note that the number of matching filtered decreases as the threshold increases. A value of 0.5 correspond to almost no filtering. We can see that the translation error is quite stable when the threshold is > 0 and not really improved by the filtering. The rotation error however is much smaller with a small threshold, i.e., a stronger filtering. The computation time is quite stable, with a bit longer time for small threshold (there are more matching kept), and large threshold (less matching, but slower convergence lead to more ICP iterations). A threshold around 0.05 or 0.1 seems to be a good compromise between performance and computation time.

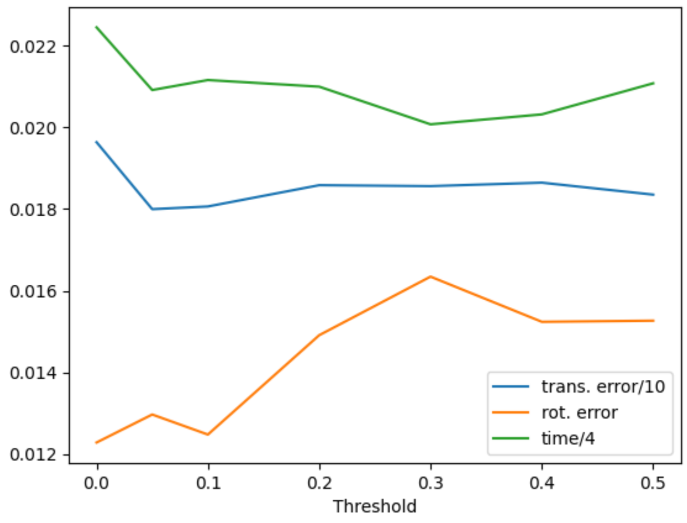

Figure 5: Effect of `threshold` on ICP performance.