Computer Vision 2 - Iterative Closest Point

Maurice Frank - 11650656 (UvA)
maurice.frank@posteo.de

David Biertimpel - 12324418 (UvA)
david.biertimpel@student.uva.nl

Leonardo Romor - 12261734 (UvA)
leonardo.romor@student.uva.nl

(All contributed equally)

April 24, 2019

Introduction

In this assignment we explore the Iterative Closest Point (IPC) algorithm by applying it on 3D point clouds showing a person rotating around its own axis. For this purpose, we first assess the IPC's performance in a quantitative manner by analyzing its accuracy, speed, stability and tolerance to noise using different sampling strategies. Afterward, we reconstruct the 3D model of the person by estimating the transformations between the successive frames and merging all point clouds into a single one. After testing the stability of the reconstruction, we discuss drawbacks of the current ICP implementation and finally suggest further improvements.

1 Iterative Closest Point

Interative Closest Point (IPC) algorithm tries to find an rigid transformation between to set of points that minimizes their Root Mean Square Error (RMSE). In each iteration we find the nearest point in the target point set for each point in the source set and then estimate the transformation using those matches. The source points are transformed with the new transformation bringing them closer to the target set. This procedure is repeated until the RMSE is sufficiently stable.

1.1 Implementation

We implement the ICP method using Python 3.7. We take multiple design decision:

Closest Point Finding the closest point in the target set for each point in the source set is not a trivial task. Naïvely searching the whole set is not practical (as in cannot provide how long it might take on our hardware). Instead we employ tree structures for this as first proposed by [5]. A further discussion of this is in Section 3.2.

RMSE We want to iterate the transformation until the RMSE of the matching points is converged. Instead of checking for equivalence of to the previous RMSE we introduce a tolerance. For more mis-aligned or noised reconstructions using equivalence did not halt for us. We set the tolerance to $10e-5$.

Z-Outliers We remove points from both source and target sets that have a depth value higher than 1. This restriction is specific to this data as it removes the scanned points from the background behind the desired person.

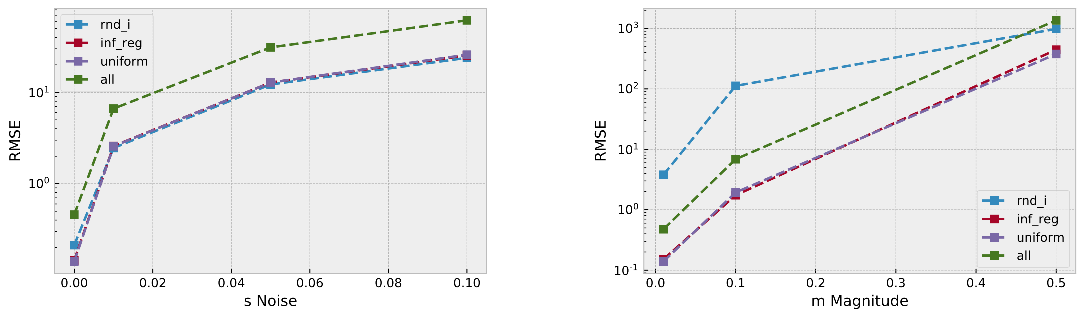

Figure 1: *Left* behaviour of minimum RMSE under different noise strengths s and *right* under different transformations with magnitude m. `rnd_i` is Random uniform and `inf_reg` is sampling from informative regions. Note that the most left values in the noise plot (no noise) are under unmanipulated conditions.

Minimum distance After finding the closest points for the source set we remove matches whose distance is bigger than 0.1. Thus we do not estimate the transformation on point matches that are too far apart. Again this set threshold is specific to this data.

We investigate different sampling techniques to sub-sample from the source point set:

All We do not sub-sample and match all source points against the target.
Uniform We sample uniformly from the source set with a fixed sample size.
Random Uniform In each iteration of ICP we uniformly sample a new set of source points. After estimating the transformation only those source points get transformed.
Informative Regions We try to locate informative regions in the source set to sample from. For this we use two features: normal sampling, hue histogram. In the normal sampling we compute the cosine similarity between each source normal and a fixed anchor normal and in the hue sampling we calculate the hue of each source points color. Next we build two histograms for those two distributions and uniformly sample from both. Given a sample size we sample $\frac{1}{2} \cdot \text{sample\_size}$ from the normal sampling and hue sampling, respectively. The intuition here is that we try to sample for regions with different face orientations and different color values.

1.1.1 Experiments

We investigate the performance of the different sampling techniques with respect to a) accuracy b) speed c) stability d) tolerance to noise.

To test for accuracy and speed we check for the minimum RMSE and the time taken. To test for **stability** we random sample rigid transformations and apply them to a given source set. The transformation is defined by the three angles $\phi, \psi, \theta \in [0, 2\pi \cdot m]$ and the translation $x, y, z \in [-m, m]$ where the magnitude $0 < m < 1$. The translation is relative to the size of the source set. Increasing m shows the stability of the ICP to initially more mis-aligned set pairs. To test for **noise** we add noise from $\mathcal{N}(0, s)$ to the source points. Again the variance of the noise is relative to the width and height of the source set. We test at different values of s.

In Figure 1 we plot the achieved RSME of the different sampling techniques under different noise and transformation strengths. Further in Figure 2 we plot the runtime in seconds under the same experiments. As expected all methods degrade in accuracy of the reconstruction under increasing noise or under worse initial conditions. *All* sampling is more susceptible to noise than the other techniques while under random rigid transformations also random uniform performs worse than our anticipated best method of the informative regions. Further we see that the informative regions and uniform

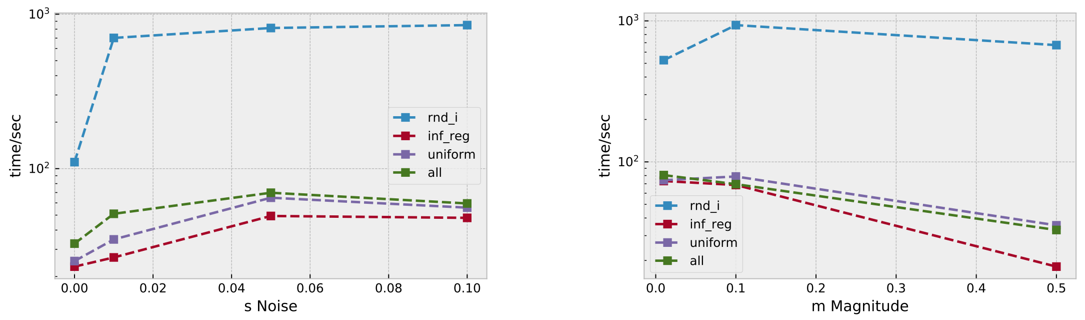

Figure 2: *Left* behaviour of runtime under different noise strengths s and *right* under different transformations with magnitude m. Again note that the most left values in the noise plot (no noise) are under unmanipulated conditions.

sampling achieve interchangeable results and that uniform random sampling is distinctly slower than all other methods.

Under these results we suspect our sampling from informative regions to actually sample similar to uniform sampling which thus would induce that the samples are not actual stemming from informative regions.

2 Merging Scenes

After quantitatively evaluating the ICP algorithm we now further assess its capabilities by reconstructing a whole 3D model of the person from the consecutive point clouds. This procedure can also be thought of as estimating the imaginary camera poses around the person. We will evaluate this part in a rather qualitative manner and give insights on which parts of the reconstruction succeed and where problems arise. After reconstructing the 3D model with all available point clouds, we further test the ICP's robustness by performing the reconstruction with every second, fourth and tenth frame. Finally, we investigate if successively merging the point clouds after each transformation is beneficial for the reconstruction process.

2.1

For the following experiments we use uniform sampling with a sampling size of 5000 points. We use uniform sampling as it is simple and without relying on additional algorithms or data a reasonable baseline. The sample size is set to 5000 because it is about 10% of the average point cloud size and thus will most likely cover a good portion of the points.

(a)

In the 3D reconstruction process, we project each point cloud in the space of the 99-th frame. For this purpose, we set in each iteration the i-th frame as our target and the $(i+1)$-th frame as our base point cloud (Note that we interpret the terms *base* and *target* such that the base point cloud adapts to match the target point cloud). As we iterate over the frames, we save each estimated rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$ as a rigid transformation. We do not update $\boldsymbol{R}$ and $\boldsymbol{t}$ right away since the multiplication of small decimals may result in numerical underflow. After estimating each transformation, we iterate a second time over all the frames and successively bring them in the space of the 99-th frame. To achieve this, we merge all frames on the way and apply the current transformation to the accumulated point cloud at each iteration. In this way, each i-th point cloud is $(99 - i)$ times transformed by the corresponding saved transformations.

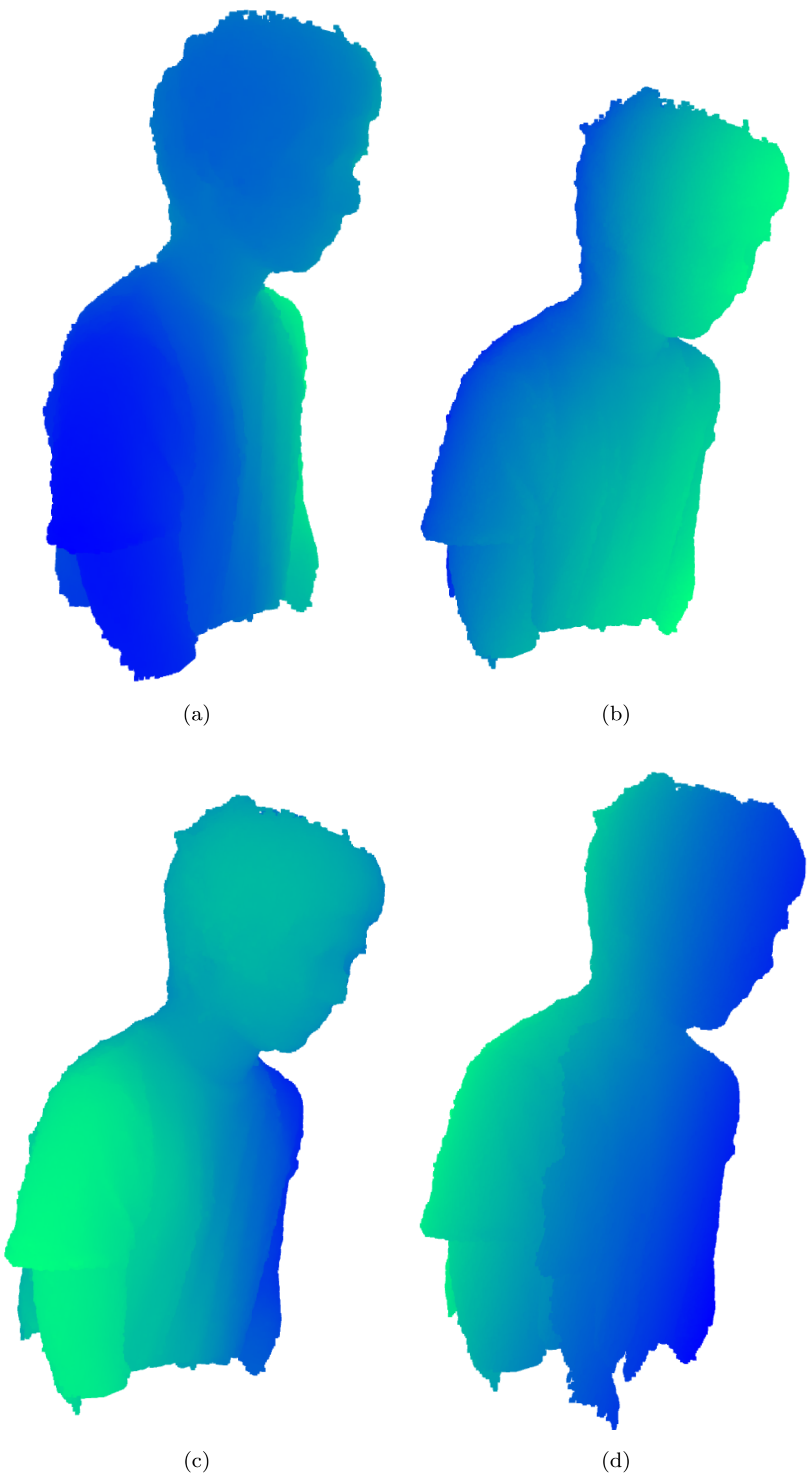

Figure 3: Results of the 3D reconstruction estimated over different iterations. (a) Shows the first 25 iterations, (b) Shows the reconstruction until iteration 50, (c) Shows the reconstruction until the 75 iteration, (d) Shows the total iterations.

In Figure 3 we show the merged point clouds in different stages of completeness including 25, 50, 75 and 100 frames. We immediately notice that the 3D reconstruction of the point clouds consisting of

25, 50 and 75 frames worked fairly well. While the 25 frames reconstruction is almost perfect, we observe minor noise and artifacts in the reconstructions composed of 50 and 75 frames. Only in the case of the reconstruction using all 100 points the noise increases notably and the model shows shifts around the shoulders and in the face (e.g. when looking at the nose).

This introduced noise can be explained by two major sources: Firstly, we know that ICP is likely to converge only to a local minimum, especially if the point clouds differ from each other. Although in our case the difference between the point clouds of the individual frames is rather small, it is likely that the estimated rigid transformation contains a small error that gradually adds up over the frames. Secondly, we find that in some cases the RMSE in the ICP algorithm does not decrease sufficiently and the ICP converges prematurely with a comparatively high error. The frames in which we observe such a behavior are simultaneously the frames in which we see the noise in the reconstruction to significantly increase. The resulting spikes in the error over the iterations are shown in figure 4. In the case where the reconstruction considers all the points (stride: 1) we do not observe any significant spikes. This is in line with our observation that the noise built up gradually and only became noticeably apparent over the last 25 frames.

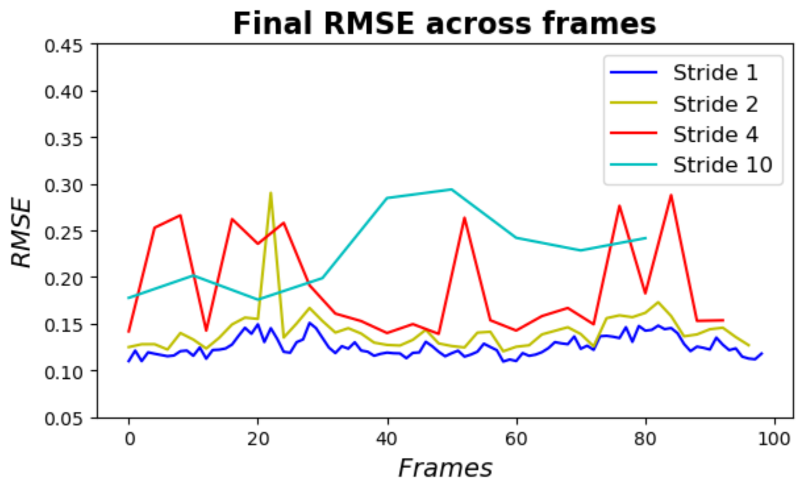

Figure 4: Depiction of the final Root Mean Squared Error, when calculating the transformation matrices with ICP, for each frame and stride.

(b)

After reconstructing the 3D model with every frame, we further evaluate the robustness of the ICP and perform the reconstruction by only using every second, fourth and tenth frame. For this purpose, we introduce a stride in our functions determining the gap between the consecutive frames. Besides this change, the principle of estimating the rigid transformations stays the same. In figure 5, we show the results achieved with both the first 25 and all the frames for each of the three stride types. Looking at the results, it is immediately apparent that the higher the stride, the more noisy the results get. Although the results obtained with a stride of 2 are still near being satisfactory, the results obtained by considering every 10-th frame show severe levels of noise. At a stride of 4, the result with the first 25 frames still looks acceptable as it holds only a few artifacts (see 5c). However, if we look at the total 100 frames, we can see that the error accelerates considerably (see 5d). In the case of a stride of 10, the result achieved with both 25 and 100 frames contain significant artifacts. These qualitative observations are also supported by the RMSE plotted across the different frames (see 4). The higher the stride, the more frequent are the cases in which the error does not converge sufficiently and gets stuck at an unusually high value. Where a stride of 2 produces only one such outlier, a stride of 4 already has 7 and a stride of 10 results in the error being constantly high. In conclusion, it can be said that the camera pose estimation changes to the effect that the results become worse in any case when a stride is introduced.

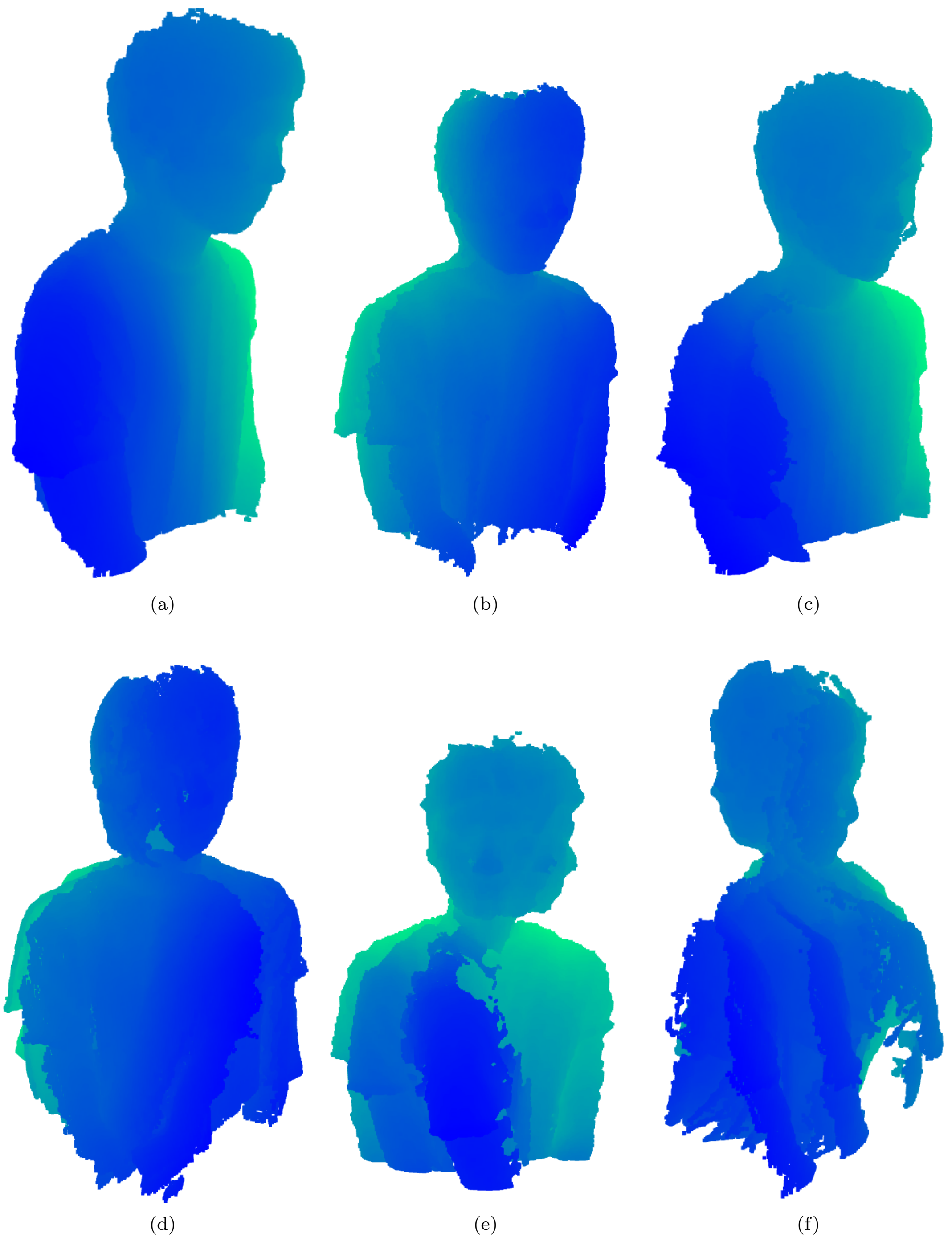

Figure 5: Results of the 3D reconstruction estimated over different iterations. (a) & (b) Show the first 25 iterations and total iterations with a stride of 2. (c) & (d) Show the same frames obtained with a stride of 4 and (e) & (f) show the same frames with a stride of 10. The reconstruction performance clearly decreases as the stride increases.

2.2

In the two previous subsections, we used only the point clouds of the i-th and $(i+1)$-th frames to estimate the respective rigid transformation between these two frames. This technique led to partially satisfactory results, but especially in the last frames, noise and artifacts were introduced into the reconstruction. In the following, we investigate whether it is beneficial for the reconstruction to merge the target frames while iterating over the frames to provide more points to the source point cloud.

The procedure of estimating the rigid transformation does not significantly change beside that after merging the already accumulated target point cloud with the transformed $(i+1)$-th frame, we need to map the entire accumulated target point cloud into the space of the next frame.

Since the constant merging of the target frames, the accumulated target will reach a size which makes building up the *PyNNDescen* or *KDTree* infeasible. Therefore, we follow the assumption that early added target points are not as important as newly added points. This causes us to thin out the older entries when the target cloud has reached a certain size. We do this by deleting every second point in the last 20% of the indexes, resulting in 10% of the target points being deleted. This makes the computation more feasible.

After applying this new procedure to our data, we end up with the results shown in figure 6 and 7. Although, the RMSE graph in figure 7 promises a successful convergence without any significant noise, we observe significant artifacts in our reconstructed 3D model (see 6).

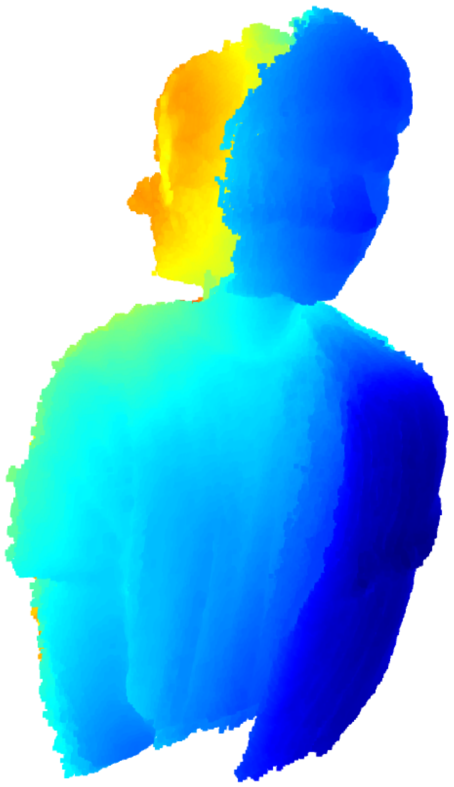

Figure 6: Illustration of the 3D reconstruction obtained by iteratively merging and estimating the camera poses using all consecutive frames.

From our own rationale the proposed procedure should likely have further improved the 3D reconstruction. Therefore, we assume an implementation error from our side.

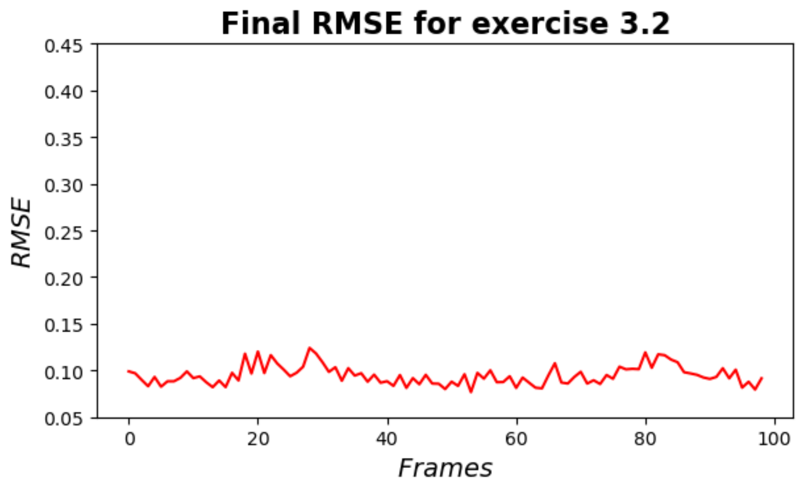

Figure 7: Illustration of the 3D reconstruction obtained by iteratively merging and estimating the camera poses using all consecutive frames.

3 Questions and Additional Improvements

3.1 Drawbacks of ICP

ICP has some glaring drawbacks. First the algorithm assumes that every point that might get sampled from the source set has a corresponding match in the target set. That might not be the case, actually it is almost always not the case. Next the method can only produce rigid transformations. Most pointset registration situations ask for more complicated transformation or at least spline-wise rigid transformations. As such already scaling the source set will degrade the performance of the method.

3.2 Tree-building methods

We investigate the speed and efficiency of different tree building methods for finding nearest neighbors. As using brute-force methods for finding nearest neighbors in, here typically, 50.000 3D-points is too slow to even prepare any results we leave it out. The classical improvement for finding nearest neighbors are tree structures, most notably KDTrees [5]. First we use a KDTree as implemented in the `scipy` library, based on [3]. Second we also try the NN-Descent method [2] for which the authors provide a python implementation [4]. Comparative results are listed in Table 1. The Python implementation of NN-Descent is 3.5x faster than the Python implementation of KDTree [1] but the Cython version [2] is by far the fastest. As we did not know about the cKDTree method until *very* recently, all result are run with PyNNDescent if not noted otherwise.

Method	Mean duration
KDTree	75.2 ± 10.80
cKDTree	4.56 ± 0.70
PyNNDescent	21.45 ± 2.27

Table 1: Comparison of the mean duration and std for the alignment of two frames for different tree construction methods. We picked 10 random frames and their subsequent frames, respectively. KDTree is pure `Python` implementation while the cKDTree is in `Cython`.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html`
[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html`

3.3 Improvements of ICP

From the our short experiments two improvements are imperative. First any use of a tree-based nearest neighbor search is preferable to the brute-force of vanilla ICP. While we resort to the two aforementioned methods more advanced stochastic methods are considerably faster. We refer to [1] for this. Second the sampling is important to be improved compared to vanilla ICP. We tried to infer informative regions by sampling from normal-angle and hue histograms which resulted in no improvement in performance. We propose to use the actual rgb color data to infer more concrete regions of interest. This could be done by finding the bijective transformation between the point coordinates of the source set and the image plane. With that one could detect regions of interest in the image (e.g. SIFT) and map those feature locations to points in the source point cloud.

Conclusion

In this report we showed our experiments with the iterative closest points method for point-set registration. In the first part of the exercise, we defined multiple ways of sub-sampling and analyzed their performance. While we showed the proneness of ICP to rapidly degrade in performance under minimal noise of stronger mis-alignment we could not make out a reliable improvement to the algorithm that would mitigate that. We did succeed though in considerably increasing the speed of the method showing that the method of closest-point retrieval is the elementary bottleneck for speed. Using the right sophisticated method here reliable increases speed. The experiments do show that the basic ICP can be improved but that does not change its inherent limitations that makes the algorithm rarely the right tool for the job. In the second part of the exercise, we successfully used the ICP algorithm to reconstruct the 3D model of the person in most situations. However, our methods showed minor to severe noise and artifacts especially in the last frames. In the exercise 2.2 we did not succeed in improving our previous implementation by iteratively merging the target frames.

References

[1] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *CoRR*, abs/1807.05614, 2018.

[2] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586. ACM.

[3] Songrit Maneewongvatana and David M. Mount. It's okay to be skinny, if your friends are fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, volume 2, pages 1–8.

[4] Leland McInnes. A Python nearest neighbor descent for approximate nearest neighbors: Lmcinnes/pynndescent.

[5] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. 13(2):119–152.