CV2 Assignment 1

Alexandra Lindt (12230642), Nikita Bortych (12342076)
Michiel Bonnee (11883936)

April 2019

1 Introduction

In this report we explore the recreation of a 3D figure based on consecutively taken 2D pictures. For this purpose, we make use of the Iterative Closest Point (ICP) algorithm for estimating an affine transformation between two three-dimensional point clouds. We modify the vanilla ICP algorithm using three different sampling methods and compare them based on the resulting accuracy, speed, stability and tolerance to noise. Further, we use the modified ICP algorithm to merge point clouds for images of a person taken from different angles into a three-dimensional reconstruction of this person. In the process, we experiment with different joining methods and also analyze them. In addition, we describe strengths of the ICP algorithms and discuss possible improvements.

2 Iterative Closest Point

Iterative closest point (ICP) [1] is an algorithm for finding the optimal spatial transformation between a two point clouds. The specific version of the ICP algorithm used in this assignment is also referred to as *point-to-point* ICP [3].

2.1 Implementation

The ICP algorithm takes two point clouds A_1 and A_2 of same dimensionality d as input. Point cloud A_1 is treated as source and point cloud A_2 as target, meaning that the algorithm finds a target transformation from the source point cloud to the target point cloud while the target point cloud remains fixed. The target transformation is an affine transformation and consists of a rotation represented by the rotation matrix $R \in R^{d \times d}$ and a translation represented by the translation vector t $\in R^d$. We operate on three-dimensional data, so $d = 3$. As given in the task description, we use the RMS as error function in our implementation. Before feeding the data to the ICP algorithm, the background points are discarded from the data by imposing a threshold of 1 meter in the z axis. Also it was noticed that some of the normals values are "nans". It might imply that that points are corrupted in the case of the point cloud as well. Hence all the

"nans" normals and corresponding points in the point clouds are filtered out. The core algorithm consists of two steps that are repeated until convergence (i.e. until the RSM between A_1 and A_2 is lower than a threshold):

1. We find for each point $p_1 \in A_1$ the point $p_2 \in A_2$ with the lowest distance to p_1. The result of this search can be expressed as a function $\psi(p_1) \ \ \forall p_1 \in A_1$. Since the point clouds we gonna use in this assignment contain several thousand points, we had to find an efficient way to compare all points of two point clouds with each other. For this purpose, we make use of KDTrees [4]. KDTree stands for k-dimensional tree, which is a data structure used for organizing coordinates. A KDTree is a binary tree where the leafs are k-dimensional data points, in the case of the implemented ICP algorithm $k = 3$. We use this KDTree to find the closest point pairs.

2. We find R and t such that the RMS error between $R \cdot A_1 + t$ and $\psi(p_1)$ is minimized. This is achieved by using singular value decomposition (exact procedure is decribed in [5]). [TODO: Maybe more here ? if time left]

After each repetition of these two steps, A_1 is updated with $A_1 = R \cdot A_1 + t$. This means that in the following repetition, A_1 will be closer to A_2 and the error between A_1 and A_2 monotonically increases. This way, the ICP algorithm converges towards the closest local minima, which it also is guaranteed to do [1]. Due to this property, the ICP algorithm works well for estimating the difference between point clouds that are not too different from each other.

2.1.1 Sampling methods

To reduce the ICP's computation time, we test a version in which we do not consider all points of a point cloud, but instead sample a sub-point cloud from the original one. The sampling is done in three different ways that are compared to figure out which one works best for the task at hand. The three sampling techniques considered are random sampling, uniform sampling, and informative region sampling. To provide a proper comparison to the full point cloud ICP, we further run the algorithm using all data points.

The **random sampling** technique simply picks points in the two point clouds at random to use in the algorithm. Every iteration another set of random points is picked from both the source as the target point cloud. These sampled points are then used to minimize the difference between both point clouds.

With **uniform sampling** the data is sampled following a normal distribution over the data. This means that the probability of a point getting picked depends on its distance to the mean. In our case this means that points towards the middle of the data array have a greater probability to get picked.

Informative region sampling is our implementation of a normal space sampling algorithm. The sampling is done by selecting points in such a way that normals are most uniformly picked as possible. We use the provided normal file of each image, and convert these to an angular space. Based on the position in

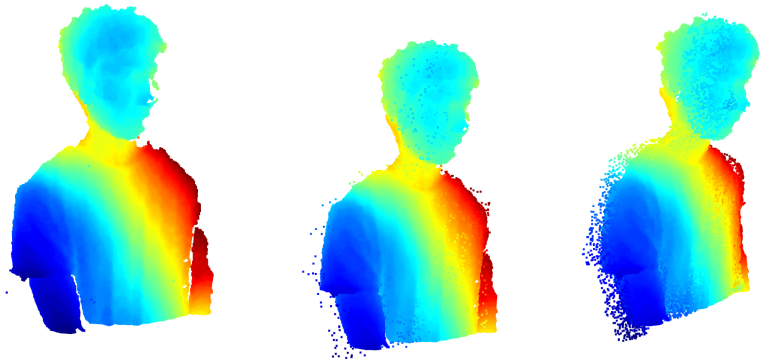

Figure 1: Introduction of visual noise with increase of the sample size. Left: sample of 10 pixels. Middle: sample of 1000 pixels. Right: sample of 10000 pixels.

this space the normals are binned, and the samples are selected uniformly from these bins.

2.2 Comparing sampling methods

We are comparing the sampling methods based on (a) accuracy, (b) speed, (c) stability and (d) tolerance to noise.
Unfortunately, we cannot compare all of them directly. The reason for this is that informative sampling needs to have normal point in order to perform. However, the "source.mat" and "target.mat" does not have the normals included and hence the dataset that has the human in it has to be used. On the other hand, the RAM of the computer running the experiments does not allow computing ICP on all of the points on the "human dataset". In order to solve the problem, 2 set experiments will be made: without the informative sampling on the "source.mat" and "target.mat" and without taking all the points on the first and second frame of the "human dataset". Convergence threshold used for the experiments is 0.0005 and we sample 1000 points.

(a) We define **accuracy** as the RMS between the source point cloud and the target point cloud. The rationale behind that is that the closer the transformed point cloud is to the target point cloud, the more accurate the algorithm is.

(b) We measure **speed** of the algorithm as the CPU time. Obviously, this varies from computer to computer, but if it is run on a single machine, then the performance becomes comparative. Also, the lower the CPU time, the higher the speed.

(c) **Stability** is defined as convergence of the algorithm dependent on initial

condition. To do measure that we change the initialization of the R matrix and t vector to be sampled from unit Gaussian distribution. Then we multiply both tensors by 10 to see how the change in magnitude of initial conditions will influence the convergence. We check the convergence by measuring the final accuracy, as well as the distance between the Affine transformation matrix and vector with and without the random initilisation with magnitude increase. The distances between the tensors will show us how sensitive the tensors are to the initial condition and will provide some insight into the ambiguity problem.

(d) **Tolerance to noise** is about convergence of the algorithm with input data with noise. In order to evaluate how robust the algorithm to noise in the input data, we augment both source and target point clouds with unit Gaussian matrix with the same dimensions as the data. This represents natural noise in, for instance, sensors that capture the points. We check the robustness to noise by measuring the accuracy with and without noise, as well as the distance between the Affine transformation matrix and vector with and without the noise. The distances between the tensors will show us how sensitive the tensors are to noise in the input data.

The performance of all these experiments can be seen in tables 1 and 2.

As we can see in table 1, for the case of the "human" dataset, all three sampling methods perform very comparatively with respect to accuracy and speed. Changes in the initial conditions seem not to influence the accuracy, hence we can say that all of the methods are stable. The affine transformation matrices, however, change a lot, depending on the change of the initial conditions. These big changes in the matrices with no changes in the accuracy reveal the ambiguity problem : the same affine transformation can be achieved by many different matrices. As for the influence of the noise, we can see that the accuracy degrades by 15 times . The size of the affine tensors also increases. However, accuracy degrates by the same factor across all of the sampling methods. This might indicate that all of the sampling methods are dealing in noise in similar way and the noise just made it harder for the algorithm to access the information contained in the data. The qualitative differences in the performance are illustrated in figure 2.

As we can see in table 2, for the case of the "wave" dataset, both sampling methods perform very comparatively with respect to accuracy and speed, however the "sampling all points" method has 3 times smaller RMSE at the cost of being 6 times as long. Changes in the initial conditions again seem not to influence the accuracy, hence we can say that all of the methods are stable. The affine transformation matrices, however, are again change a lot, depending on the change of the initial conditions. As for the influence of the noise, we can see that the accuracy degrades by a factor of 10 in case of sampling all the points and by a factor of 4 in case of the sampling methods. The size of the affine tensors also increases. Interestingly, accuracy degrates much more in the case of all points. This can be explained by the fact that more the all sampling looks at all of the noised points and is much more confused by it. In the case of other

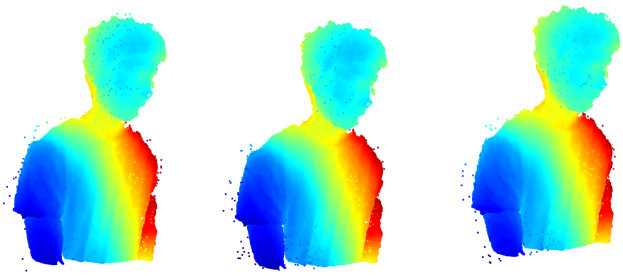

Figure 2: Different sampling methods on the "human" dataset. Left : informative sampling. Middle : random sampling. Right: uniform sampling.

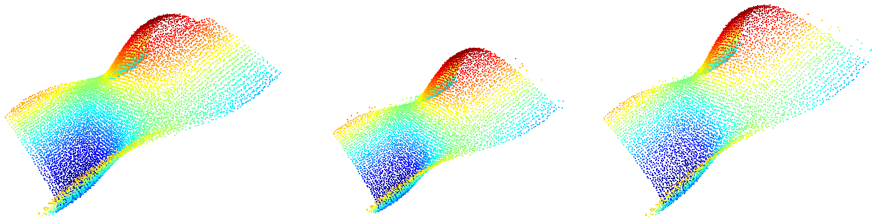

Figure 3: Different sampling methods on the "wave" dataset. Top-left : taking all points. Top-right : random sampling. Bottom: uniform sampling.

sampling methods, they degrade in performance by the same factor. This might indicate that both of the sampling methods are dealing in noise in similar way. The qualitative differences in the performance are illustrated in figure 3.

3 Merging

3.1

(a) In this section we use our implementation of the ICP algorithm to merge multiple point clouds into one overall point cloud. For this purpose, we are estimating the camera poses associated with each point cloud or frame in the input and then use them to transform the point clouds into the space of the 1st frame. With all point clouds in the same space, we can then merge the point clouds together and hopefully obtain a point cloud that represents the person depicted in the input images in 3D.
Within this procedure, we make use of our implementation of the ICP algorithm from the previous section. After receiving n single point clouds $[pc_1, \ldots, pc_n]$ as input, we concretely proceed as follows:

Test/Method:	Informative	Uniform	Random
Accuracy, RMSE	0.016	0.016	0.015
Speed, s	2.42	2.15	2.02
Stability, RMSE	0.016	0.016	0.016
Stability, $\|R_{normal} - Rrandom\|_2$	22.96	18.9	21.54
Stability, $\|t_{normal} - trandom\|_2$	8.4	9	7.5
Noise, RMSE	0.24	0.23	0.23
Noise, $\|R_{normal} - Rnoisy\|_2$	0.72	0.70	0.65
Noise, $\|t_{normal} - tnoisy\|_2$	0.45	0.45	0.44

Table 1: Evaluation of different sampling methods on the "human" dataset

Test/Method:	All	Uniform	Random
Accuracy, RMSE	0.028	0.087	0.082
Speed, s	51.3	8.06	8.03
Stability, RMSE	0.028	0.081	0.081
Stability, $\|R_{normal} - Rrandom\|_2$	22.1	32.82	32.82
Stability, $\|t_{normal} - trandom\|_2$	11.28	20.72	23.8
Noise, RMSE	0.20	0.35	0.32
Noise, $\|R_{normal} - Rnoisy\|_2$	0.49	0.43	0.53
Noise, $\|t_{normal} - tnoisy\|_2$	0.55	0.34	0.37

Table 2: Evaluation of different sampling methods on the "wave" dataset

1. First, we consider the last point cloud pair pc_n and pc_{n-1}. Using ICP, we obtain the transfomation $T_{n\to n-1}$. We transform pc_n into the space of pc_{n-1} using $T_{n\to n-1}$ and then append a certain percetance of the resulting point cloud (uniformly sampled) to the list variable `transformed_pcs`.

2. We continue with the next point cloud pair pc_{n-1} and pc_{n-2}. Like before, we obtain $T_{n-1\to n-2}$ with ICP andthen transform pc_{n-2} into the space of pc_{n-1}. Now, we also have to consider that the point-clouds in `transformed_pcs` (= at this timestep exactly one item) are still in the pc_{n-1} space. To also transform them into the current space, they all have to be transformed with $T_{n-1\to n-2}$ as well.

3. We go on as decribed in the previous two steps for the all point cloud pairs pc_i and pc_{i-1} until we reach the 1st point cloud. By this time, `transformed_pcs` should contain $n-1$ point clouds, all of them over the iterations transformed into the space of pc_1. We can then just merge all of these point clouds in one.

The results for this method are displayed in figure 4, which shows the outcome for different amounts of considered input frames. We can clearly see that the results gets more noisy the more input frames are considered. We assume that this happens because the small errors of each translation between two frames

add up in the long run and start to seriously lower the quality of this method's outcome from about 25 input frames upwards.

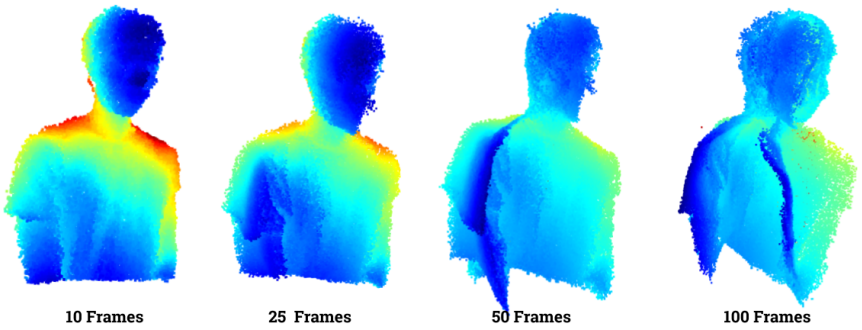

Figure 4: Outcome of the method described in 3.1(a) for different amounts of input frames considered.

(b) In order to create the overall point cloud using only every 2nd,4th or 10th frame of the n input frames, we proceed exactly as we did in (a). The only difference is that before we start, we reduce the input to only the features that we want to consider, i.e. we go from $[pc_1, \ldots, pc_n]$ to $[pc_1, pc_{1+i}, \ldots]$ when we want to consider each ith feature. The results for this method are displayed in figure 5. It is apparent that we can achieve a quite similar result if we only consider every $2nd$ frame. This comes as not a big surprise, as the input pictures are taken from quite close angles. It even looks like a little of the noise from the original result is removed. However, when only using every 4th frame, the result is much sparse than the original. Finally using every 10th layer did not give a too good output. It seems like here the angle between the single input images was to big to apply this method.

3.2

In this section we use our implementation of the ICP algorithm for the same task as in the previous section. However, unlike in the last section, we will be transforming the commutative point cloud that we accumulate along the way. We plan to achieve this by starting with the first frame and then estimating the affine transformation tensors by using the second frame as the target. Both frames are then merged by after the first frame is transformed. The outcome frame is assigned the variable `total`. Following, we use `total` to estimate the new affine matrix with the next frame as the target. We transform `total` with the new frame as target. This procedure is repeated until the end of the dataset is reached.

Within this procedure, we make use of our implementation of the ICP algorithm from the previous section. After receiving n single point clouds

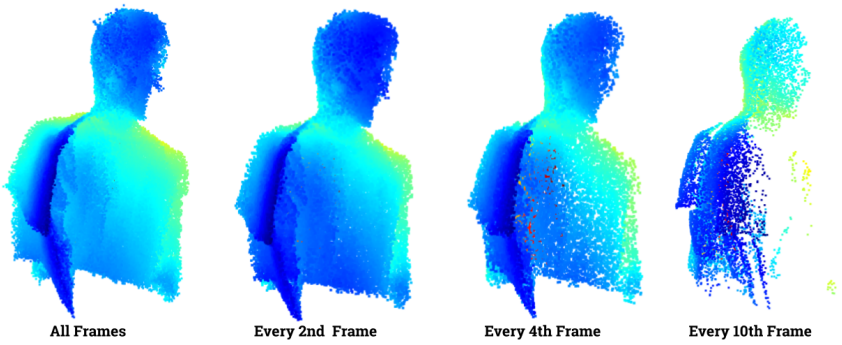

Figure 5: Outcome of the method described in 3.1(b) for different subsets of the input images.

$[pc_1, \ldots, pc_n]$ as input, we concretely proceed as follows:

1. First, we consider the first point cloud pair pc_1 and pc_2. Using ICP, we obtain the transformation $T_{1\to 2}$. We transform pc_1 into the space of pc_2 using $T_{1\to 2}$ and then assign a certain percentage of the resulting point cloud (uniformly sampled) as the point cloud variable `total`.

2. We continue with the next point cloud pc_3. However, now, we obtain $T_{total\to 3}$ with ICP. We follow by transforming total into the space of pc_3. We then merge total and the pc_3. and assign this as the new value for total.

3. We go on as described in the previous two steps for the all point clouds until pc_n.

The results of this procedure can be seen on figure 6. It is far from optimal and has a lot of noise and artifacts like double faces. Unfortunately, we were not able to reach a better result with this method. If we compare it to the results in 3.1, the result is different and comparatevely better.

4 Questions

The ICP algorithm works very well in minimizing distances between two point clouds. But, as with all algorithms, there are some drawbacks that might be improved.

4.1 ICP drawbacks

Since ICP is an iterative algorithm that identifies the closest points in each iteration, it can be computationally very heavy for big datasets[2]. We also

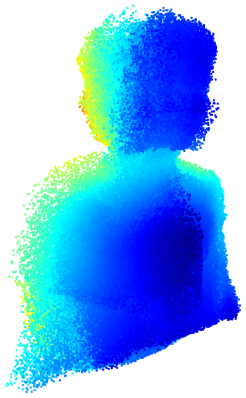

Figure 6: Result obtained with the method described in 3.2 .

experienced this problem to some extend, since it takes a while to run the ICP on all our 100 input images. As we have seen, this drawback can be encountered with a good sampling technique, but this does mean data is lost and the overall accuracy may suffer.

Another drawback of the ICP algorithm that we also already mentioned earlier in this report is that ICP always converges to the closest local minima. With some luck this local minima is also the global minimum, but obviously this is not always the case. ICP's performance therefore heavily relies on the initial guess.

ICP cannot handle outliers in the two input point clouds very well. Regardless of how off a point in a point cloud is from the other points, the ICP algorithm still tries to match it to a point in the output region and may therefore sacrifice the good fit of other points in order to minimize the overall error.

Another issue that we encountered in this assignment with the algorithm is that the error in the single translations estimated with the ICP adds up for cumulated translations. This resulted in quite poor results for the 3D construction method in section 3.1, approximately from 25 accumulated translations upwards.

4.2 Improvements on ICP

As discussed above, one of the drawbacks of the ICP algorithm is the convergence to local minima. One way to counter this is by running the algorithm multiple times and choose the result with the lowest RMS. This does make the algorithm even slower to converge. A better way would be to use additional information to make the initial guess more educated, this might push the algorithm more towards the global minimum.[2]

Another improvement on the ICP algorithm can be implemented with preprocessing the data. ICP is very sensitive to outliers which tells us that pre-

processing data is a very important step to gain better accuracy and speed. To improve on the preprocessing a possibility is to already do some mapping of points or areas that look-a-like. This will push the algorithm towards a more informed initial guess as well. Another option could be to set certain tresholds for point clouds of a specific dataset and then only consider points within them.

5 Conclusion

In this paper we implemented an ICP algorithm which is used in merging frames to create a 3D model of an object. The ICP algorithm works by calculating the optimal transformation between two point clouds A_1 and A_2. In order to gain insight into some design choices, multiple sampling methods are used, these include random sampling, uniform sampling, and informative region sampling. We found that informative region sampling works best but not by a lot. Another interesting observation we found is that using less sample data seemed to produce better results. We then used this ICP algorithm and an estimation of the different camera poses of the different frames to merge together these frames in order to construct a 3D model of the object in the images. This is done by transforming every frame into the space of the 0th frame, and then using the ICP algorithm append every frame to its previous frame. This creates a 3D model of the image.

6 Self-Evaluation

Everybody contributed equally to the report.

References

[1] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[2] Yaroslav O Halchenko. Iterative closest point algorithm. l1 error approach via linear programming.

[3] K-L Low and Anselmo Lastra. Reliable and rapidly-converging icp algorithm using multiresolution smoothing. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 171–178. IEEE, 2003.

[4] V Ramasubramanian and Kuldip K Paliwal. Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Transactions on Signal Processing*, 40(3):518–531, 1992.

[5] Olga Sorkine. Least-squares rigid motion using svd. *Technical notes*, 120(3):52, 2009.