Assignment 1. Iterative Closest Point - ICP

Jeroen van Wely, Niek IJzerman & Jochem Soons
11289988, 11318740, 11327030
Computer Vision 2
MSc Artificial Intelligence
University of Amsterdam

April 19, 2021

1 Introduction

Within this report we will discuss the Iterative Closest Point (ICP) algorithm - an algorithm designed for geometric alignment of three-dimensional objects. We will start in chapter 2, where we will give a general introduction to the ICP algorithm and discuss the data on which we will deploy the ICP algorithm. Within this section we will also discuss and perform experiments for multiple methods in which efficiency and accuracy of the ICP algorithm could be improved. In chapter 3, we will use the ICP algorithm to merge different scenes (i.e. frames) and compare different merging approaches. In chapter 4, questions about characteristics of the ICP algorithm are answered. Chapter 5 will be dedicated to some personal suggestions in which the ICP algorithm could potentially be improved. Finally, we will discuss and conclude our research.

2 Iterative Closest Point - ICP

The ICP algorithm was first introduced by Besl and McKay [2]. The goal of ICP is to find a spatial transformation that minimizes the distance (e.g. Root Mean Squared Error) between point cloud A_1 and point cloud A_2. Mathematically this can be defined as

$$RMS\left(A_1, A_2, \psi\right) = \sqrt{\frac{1}{n} \sum_{a \in A_1} \|a - \psi(a)\|^2}, \tag{1}$$

$$\min_{\psi: A_1 \to A_2, t \in \Re^d} \sum_{a \in A_1} \|Ra + t - \psi(a)\|^2. \tag{2}$$

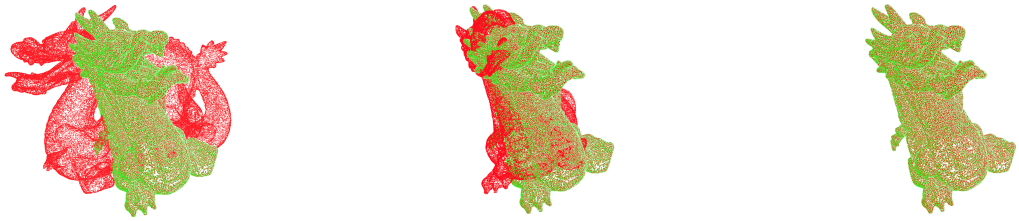
where R is the rotation matrix and t is the translation vector in d dimensions. ψ is used to create one-to-one matches between point clouds A_1 and A_2. To define the camera movement between point clouds A_1 and A_2, R and t that minimize equation 2 are used.

To be able to find R and t, we iteratively go trough the following steps:

1. Initialize $R = I$, $t = 0$.
2. Transform the base point cloud A_1 with R and t.
3. Find the closest points for each point in the base point set (A_1) from the target point set (A_2) using brute-force approach.

4. Refine R and t using Singular Value Decomposition.
5. Calculate RMS between transformed A_1 and A_2
6. Go to step 2 unless RMS is unchanged.

If the RMS converges and eventually does not change anymore, this indicates that we have found a local optimum in which the rotation matrix R and translation vector t are converged. As an example, in figure 1 different steps in the ICP merging process using for the dragon dataset are shown.



(a) before merge. **(b)** intermediate merge. **(c)** final merge.

Figure 1: Different states of ICP merge for base point cloud (red) and target point cloud (green) of dragon dataset.

2.1 Data

For our experiments we will utilize multiple point cloud data sources. First of all, we will use the source.mat (base point cloud) and target.mat (target point cloud) which were included in the original assignment. Secondly, we have downloaded the dragon1.xyz (base point cloud) and dragon2.xyz (target point cloud) from an external source. Both these cloud point pairs are single frames (i.e. one base cloud and one target cloud). We will refer to them as "MAT dataset" and "Dragon dataset", respectively.

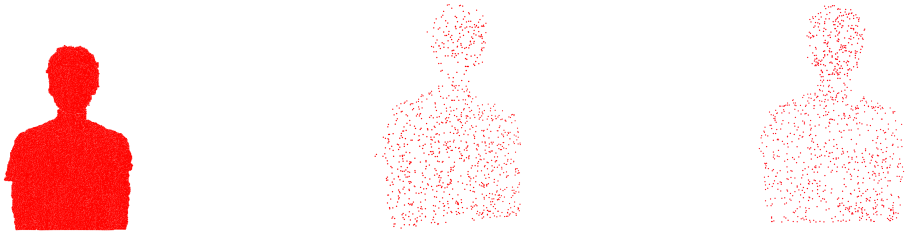
Furthermore, we will utilize 100 consecutive scenes included in the assignment, where each scene consists of a depth image, mask image, point cloud file, normals point cloud file, regular image .mat file of the scene. We will make use of the point cloud files in our experiments. Furthermore, as a pre-process step we have removed outliers (i.e. background cloud points) from each scene.

2.2 Point selection techniques

To be able to find rotation matrix R and translation vector t that minimize the RMS, we need to sample points from the base point cloud. There exist multiple ways in which we can select these points. Within this section we will experiment with four of these possible sample methods.

The first sample technique that will be addressed will use all all points within the source cloud to construct matrix R and vector t. Uniform sub-sampling (i.e. taking a subset of points from the base cloud) will be our second sampling method. Third, as an extension on the sub-sampling technique, we will sub-sample new points each iterations in which we refine R and t. We will refer to this sample method as 'random sub-sampling'. Fourth, we will sub-sample points one time at the beginning of the iteration process from informative regions. This will be done by choosing points such that the distribution of normals among selected points is as large as possible, as introduced by [1]. We will refer to this sampling method as 'normal-space sampling'. As a visualization of all sample methods, in figure 2 the complete sampled point cloud (i.e. all points), a sub-sampled/random sampled point cloud and a normal-space sampled point cloud.

We will deploy these techniques on the MAT and Dragon datasets. To be able to compare all methods, we will report on their accuracy, speed, stability and tolerance to noise. Results can be found in section 2.5.



(a) All points sampled. **(b)** uniform/random sub-sampling. **(c)** Normal-space sampling.

Figure 2: Different sample methods visualized.

2.3 Improve ICP using kd-tree

Regular ICP uses a brute-force approach to find the closest points in the transformed base point cloud and target cloud. As this approach is computationally quite expensive, kd-tree can be deployed. Kd-tree is an effective method to reduce computation by storing by storing our target cloud points in a tree data structure. Each node in the tree holds a point of k dimensions. As our points are tree-dimensional, each node also contains tree properties. Just like a binary search tree, we can place nodes (i.e. cloud points) by going down the tree. We go left in the tree when a specific dimension value of the node is smaller than the similar dimension value of the node we want to place and right when a specific dimension value is bigger than the similar dimension value of the node we want to place. Each layer we go deeper in the tree, we alternate the dimension we compare between the tree node and node to place. Eventually, we place the new node as a leaf somewhere in the tree.

Similarly as placing a node, we can look for a target cloud point which is the closest to a transformed node from the source point cloud. While brute-force matching requires us to check and compare all target cloud points to find the point closest to a transformed source point, kd-tree enables us to only check and compare a small sub-selection of all target cloud points to find the closest point. This is due to the fact that at each node we either visit the left leaf or right leaf. Due to this search technique, using kd-tree it is computationally less expensive to search for closest points.

To be able to compare regular ICP and ICP using kd-tree, in the next section (section 2.5) we will report their accuracy, speed, stability and tolerance to noise for the different point selection techniques introduced in section 2.2. Similarly to the brute-force ICP, we will deploy the ICP with kd-tree on the MAT and Dragon source-target pairs.

2.4 Further imrovements

Before we head over to our results, we will discuss some other tweaks to the ICP algorithm that are noteworthy to mention. First of all, in accordance with [3], we implemented two different techniques for weighing pairs that are found in the matching process within the ICP algorithm. When running our code, you can either 1) set a defined threshold that rejects all matched pairs for which the distance is greater than set threshold, and 2) we implemented a weighing of pairs based on their distance, formulated as:

$$Weight = 1 - \frac{Dist(p_1, p_2}{Dist_{max}} \tag{3}$$

2.5 Results

The averaged results over three separate runs for the Dragon and MAT source-target pairs, including standard deviation, are displayed table 1. Several things can be concluded from looking at these results. We will discuss them one by one.

Accuracy: First of all, we can see that the RMS error, which serves as a proxy of transformation accuracy, is very low for all techniques on both datasets, for both the traditional ICP algorithm as the improved ICP algorithm using kd-tree. This indicates that the ICP algorithm has no real difficulties in transforming the MAT and Dragon source frame to its respective target frame. In fact, for the Dragon point clouds, it can transform the source perfectly, achieving a RMS error of 0.0 for all techniques. For the MAT data, the error of all different sampling techniques is very similar (also between the ICP variants), so there are no significant differences in terms of accuracy to report here.

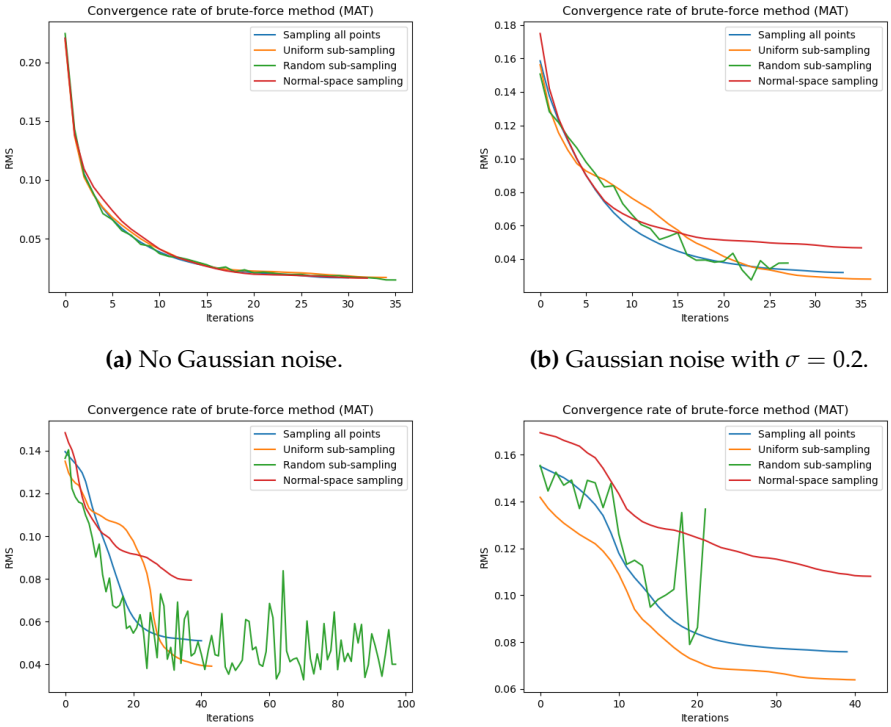
Speed: Secondly, we also see similar results for the number of iterations the ICP algorithm needs to converge, which is a characteristic for speed of convergence. Here, only for the random sub-sampling technique we see an increase in the number of iterations until convergence, which can be explained by oscillations in RMS score that arise from sampling different random points at each iteration, which prevents the difference of RMS scores to be lower than the set threshold of convergence. However, analyzing the speed of convergence becomes more interesting when looking at the total runtime of the ICP algorithm. Here we notice two things: 1) the runtime of the 'all point' sampling mechanism is significantly larger than of the sub-sampling methods, which is naturally unsurprising as the ICP algorithm has less datapoints to deal with, and 2) the speed, in terms of seconds, is much higher for the improved ICP algorithm that works with kd-trees. This is also not surprising, as we expected the ICP algorithm to decrease the computational complexity. So an important conclusion here is that number of required iterations does not tell the whole story here: looking at the required runtime gives much more insight when analyzing speed.

Stability: Thirdly, when analyzing stability, we see that all different variants of the ICP algorithm consistently succeed in successfully finds the transformation between the source and target frame, as the standard deviation between runs is quite low. Only for the random sub-sampling, we see that it can happen that the algorithm needs a lot of iterations to converge. This can again be explained by fluctuations in RMS scores that arise from the stochastic sampling process that occurs in every iteration, instead of just one time at the start as is the case with the uniform sampling approach.

Robustness to noise: Lastly, we can analyze the robustness to noise by looking at the figures 3 and 4. Here we added Gaussian distributed noisy points to our existing point clouds (both source and target) using different values of sigma. We added a relative 10% of noisy points to the original point clouds (which for the MAT data equals an extra 640 points). For both the regular brute-force and improved ICP algorithm we can notice a similar pattern for increasing noise in the data. For $\sigma = 0.2$, we see all sampling variants still performing relatively good, although the normal-space sampling seems to be less tolerant to the extra added noisy points. When σ increases to 0.5, we see the normal-space sampling performing significantly worse than the other sampling methods, while the random sampling method oscillates and does not reach stable convergence. Finally, when the noisy is increased to $\sigma = 1.0$, we see the same trend continuing: normal-space sampling and random sampling are outperformed by the sampling techniques using all points or using uniform sampling. This trend is not too surprising: it is understandable that inferred normals from point clouds become noisy themselves when a lot of noisy points are present, which hinders the technique to indicate regions of interest that can be used effectively for ICP, while random sub-sampling at each iteration becomes in essence very noisy when a lot of points are noisy. Overall, it is somewhat surprising that the uniform sampling strategy seems to perform best for different noisy values. This just shows that empirical evaluation of complex computer vision techniques can yield results that can be surprising.

	MAT			Dragon		
Technique	RMS	# Iterations	Time (s)	RMS	# Iterations	Time (s)
Brute-Force						
All points	0.0167 (0.0)	31.00 (0.0)	40.5 (1.11)	-*	-	-
Uniform sub-sampling	0.0165 (0.0004)	30.67 (3.21)	1.6 (0.20)	0.0 (0.0)	14.67 (0.58)	74.30 (1.91)
Random sub-sampling	0.0137 (0.0004)	66.67 (24.95)	3.46 (1.37)	0.0 (0.0)	15 (0.0)	77.82 (4.63)
Normal-space sampling	0.0153 (0.0015)	35.67 (1.15)	2.18 (0.14)	0.0 (0.0)	15 (0.0)	78.11 (4.01)
KD-Tree						
All points	0.0138 (0.0)	43.00 (0.0)	4.38 (0.65)	-	-	-
Uniform sub-sampling	0.0177 (0.0012)	37.00 (13.08)	0.04 (0.01)	0.0 (0.0)	15.67 (0.58)	0.22 (0.06)
Random sub-sampling	0.0156 (0.0030)	123.33 (116.42)	0.15 (0.13)	0.0 (0.0)	15.33 (0.58)	0.23 (0.04)
Normal-space sampling	0.0146 (0.0020)	37.33 (2.51)	2.21 (0.12)	0.0 (0.0)	14.67 (0.58)	1.81 (0.04)

Table 1: mean values of final RMS, # iters and time (s) for MAT and Dragon datasets taken over 3 runs including standard deviation between brackets. No additional noise added. *: due to limited computational capacity, these experiments were not carried out.



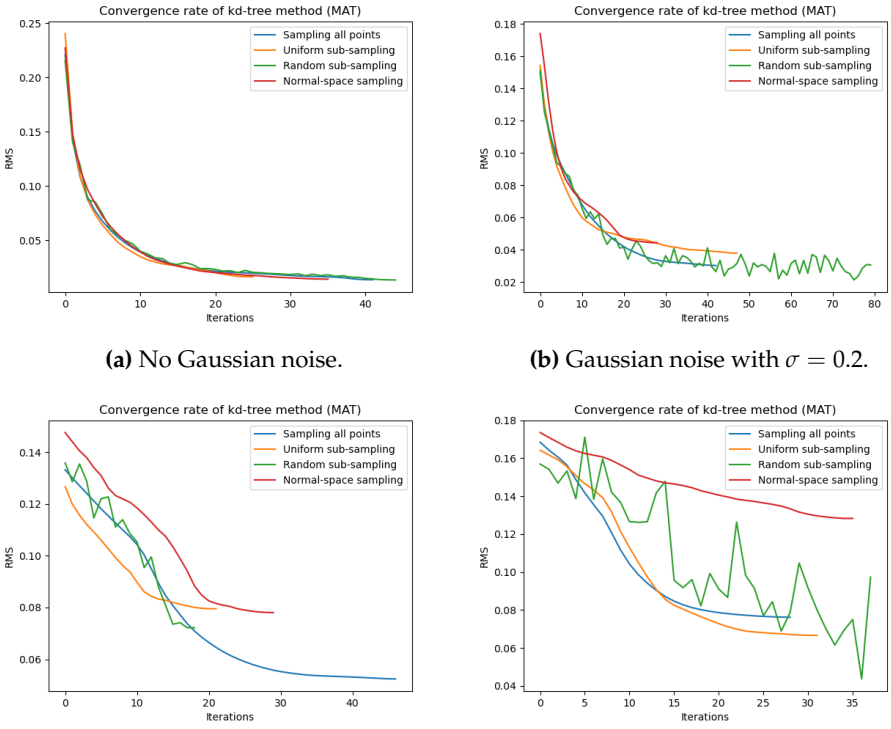
(a) No Gaussian noise.

(b) Gaussian noise with $\sigma = 0.2$.

(c) Gaussian noise with $\sigma = 0.5$.

(d) Gaussian noise with $\sigma = 1.0$.

Figure 3: Effects of random Gaussian noise to the RMS score convergence for different values of sigma, for the brute-force ICP algorithm.



(a) No Gaussian noise.

(b) Gaussian noise with $\sigma = 0.2$.

(c) Gaussian noise with $\sigma = 0.5$.

(d) Gaussian noise with $\sigma = 1.0$.

Figure 4: Effects of random Gaussian noise to the RMS score convergence for different values of sigma, for kd-tree ICP algorithm.

3 Merging Scenes

In the previous section (section 2) we discussed and used multiple ICP variants to merge single base-target cloud point scenes. In this section, we will utilize ICP to merge multiple consecutive scenes. We will merge datasets of frames of a man spinning on a chair. This set contains 100 frames, which together capsuled exactly one complete rotation. Thus combining each of these frames should give us a 3D representation of the man spinning on the chair. First, in section 3.1 we will merge frames by transforming and appending each new frame to the previous frame, which has then just been transformed to its processing frame. This will be done in two different ways: (1) by merging all consecutive frames and (2) by merging intermediate frames (i.e. every 2^{nd}, 4^{nd} and 10^{nd}). Second, in section 3.2 we will merge all consecutive frames as well. However, instead of merging to the start-frame, each new frame is merged with the entire mesh up to that point.

For the task of merging we tested which variant of ICP performed best. This is an ICP variant using spreadout sampling (this is explained in section 5); A sample size of 1000; convergence threshold of 1.5e-6; no distance threshold; no target sampling and using KD-Tree closest point search. One can this variant with corresponding settings have been used unless stated otherwise. The results when using normals-space sampling where comparable.

3.1

3.1.1 Merging each frame to previous frame

In this section, the frames are merged together by merging each new frame to the previous transformed frame. This is done by computing the rotation matrix and translation vector of the new frame with the original untransformed previous frame. We then compute the transformation on the current frame with this rotation matrix and translation vector. this is then transformed with the rotation matrix and translation vector used to transform the previous frame with the frame before it. We keep doing this until we have reached the start frame. Then we proceed to the next frame to be transformed.

In figure 5 the results of using this strategy our shows for merging all 100 frames. As one can see the results are as one would expect: A clear 3D image of a man.

(a) frontal view. **(b)** side view. **(c)** back view.

Figure 5: Merging of scenes with step size 1. For each merge the frontal, side and back view is displayed.

3.1.2 Adding step sizes

In figure 6 we present the results of the same merging strategy, however here we have not used all 100 frames. Some frames have been skipped by using a step size bigger than one. The first row corresponds with the merge with a step size of 2; the second row with a step size of 4 and the last row with a step size of 10. It is noticeable that as the step size increases the quality of the mesh decreases. This is due to the fact that the point clouds from frames further apart are less similar and thus they have fewer points overlapping with each other. The ICP algorithm wills still try to merge these points, however. This can results in some wrong merges as is clearly noticeable in the merge where a step size of 10 has been used.

(a) step size 2, frontal **(b)** step size 2, side **(c)** step size 2, back

(d) step size 4, frontal **(e)** step size 4, side **(f)** step size 4, back

(g) step size 10, frontal **(h)** step size 10, side **(i)** step size 10, back

Figure 6: Merging of scenes with different step sizes. For each merge the frontal, side and back view is displayed.

3.2

3.2.1 Merging each frame to current merge

In this section the 100 frames have been merged as well, however here another merging strategy has been used. Every frame (base) is transformed to merge up till that point (target). After it has been transformed it is concatenated to the merge (new target) and we move on to the next frame (new base). The problem with this strategy is that most ICP variants are not able to compute a merge within a regular time frame. This is due to the target size increasing as new frames are merged. The only variant of ICP that can be used is therefore one that uses sampling on the target point cloud. This variation of the ICP algorithm is what produced the merge presented in figure 7. Noticeable is how the quality of the merge is substantially less compared to the other merges discussed in the previous section. This is due to the sampling of the target cloud. Especially when merging later frames (frames 50 to 100) the set of samples obtained from the target cloud becomes less likely to contain points that are also present in the base point cloud because the target contains mainly frames that aren't similar

to the base frame at all.

(a) frontal view. **(b)** side view. **(c)** back view.

Figure 7: Merging each frame to current merge

Overall, it can be concluded that merging with the former strategy and a step size of one produces the best results. However, we can't truly compare the two merging strategies because we were unable to use the same ICP variant. An ICP variant with more efficient target cloud sampling would allow us to do so.

4 Questions

4.1 What are the drawbacks of the ICP algorithm?

First of all, as we have seen with matching cloud points using a brute-force approach, ICP can be computationally expensive. This is mostly the case for the brute-force matching approach. The kd-tree approach uses a more efficient method for point matching and reduced the computational costs significantly. Furthermore, the ICP algorithm converges to a local minimum. Although the global optimum could be reached, we are therefore not guaranteed to find the best solution. The fact that the ICP algorithm is susceptible to outliers and noise could be considered a drawback as well. This has also been demonstrated in section 2.5. Although we are able to apply outlier detection as a pre-processing step, the performance of the ICP algorithm might still be harmed due to noise and outliers.

4.2 How do you think the ICP algorithm can be improved, beside the techniques mentioned in [3], in terms of efficiency and accuracy?

A possible way the ICP algorithm could be improved is by changing the sampling method such that a sub-sample of points is taken for which the distance to each other is maximized. We will refer to this technique as 'spreadout sampling'. By applying this sample technique we ensure that the points are not all sub-sampled from the same region but are equally distributed in the point cloud, which might improve merging performance. Furthermore, this sample technique could be considered efficient in the sense that we only need to sample points once at the beginning of the iteration process. For reference, we have visualized this sample technique in figure 8. In the next section (section 5) we will also run experiment for this sample technique and report its performance.

5 Additional improvements

Besides the improvement we mentioned in section 2.4 (Further improvement, we also implemented our suggestion in section 4.2: applying spreadout sampling in an attempt to improve the results of

ICP. In spreadout sampling we sub-sample a pre-specified number of points of which the distance to each other is maximized (see figure 8 as example). In table 2 we have reported the averaged results over three separate runs for the MAT and Dragon source-target pairs, including standard deviation. For comparison, we have also reported the results of sampling techniques introduced in section 2.5. Results are only shown for the kd-tree matching approach as it outperforms the brute-force matching approach in terms of efficiency (i.e. speed).

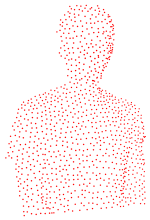


Figure 8: Spreadout sampling (1000 points).

	MAT			**Dragon**		
Technique	RMS	# Iterations	Time (s)	RMS	# Iterations	Time (s)
KD-Tree						
All points	0.0138 (0.0)	43.00 (0.0)	4.38 (0.65)	-	-	-
Uniform sub-sampling	0.0177 (0.0012)	37.00 (13.08)	0.04 (0.01)	0.0 (0.0)	15.67 (0.58)	0.22 (0.06)
Random sub-sampling	0.0156 (0.0030)	123.33 (116.42)	0.15 (0.13)	0.0 (0.0)	15.33 (0.58)	0.23 (0.04)
Normal-space sampling	0.0146 (0.0020)	37.33 (2.51)	2.21 (0.12)	0.0 (0.0)	14.67 (0.58)	1.81 (0.04)
Spreadout sampling	0.0156 (0.0023)	36.33 (6.02)	0.09 (0.01)	0.0 (0.0)	14.00 (0.0)	3.66 (0.13)

Table 2: mean values of final RMS, # iters and time (s) for MAT and Dragon datasets taken over 3 runs including standard deviation between brackets for original sampling methods and our proposed 'spreadout sampling' method. No additional noise added. *: due to limited computational capacity, these experiments were not carried out.

As visible in table 2, spreadout sampling neither performances better nor worse on the MAT dataset. The RMS, # Iterations and Time (s) all seems quite similar to the other sampling methods. On the Dragon dataset, both the RMS and # Iterations seem quite comparable to the other sample methods. Although margins are slim, the time the sampling method takes seems a bit higher compared to the other methods though.

Given these results, it is difficult to determine whether our proposed spreadout sampling method could be beneficial to improve the results of ICP. Additional experiments could be helpful to gain more insights into the quality of the spreadout sampling method. The fact that our reported results are in the same ballpark as other sampling methods could be considered promising though.

6 Discussion and Conclusion

Within this research we have experimented with different sampling methods and different point matching techniques that can be used in the ICP algorithm. First, it was noticeable that the brute-force matching approach for ICP produced produced similar results as the kd-tree matching approach in terms of the accuracy (i.e. RMS) and # of iterations before converge. The kd-tree approach, however, was significantly faster than the brute-force matching approach. This characteristic of the

kd-tree approach makes it very suitable for real-world applications in which speed is of importance. Furthermore, we have implemented two special point sampling methods: Normal-space sampling and spreadout sampling. Both sampling methods proved to be efficient and accurate, but did show significant improvements with respect to other sampling methods. We also experimented to what extend different sampling techniques were robust against noise. From our results, it was concluded that normal-space sampling can handle noise the worst. Uniform sampling was able to handle noise the best. This was quite surprising to us as we expected that uniform sampling would sample noise points quite often.

A Self Evaluation

All members contributed equally to the assignment. The most time-consuming process was programming the ICP algorithm and all its variants. To reduce the workload of each member of the group, we all contributed small parts of the final code. Furthermore, the analysis of our results writing of our report was done jointly by discussions over Zoom and in person.

References

[1] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 859–868, 2018.

[2] 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 586–606. International Society for Optics and Photonics, 1992.

[3] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.