University of Amsterdam

Assignment 1

Iterative Closest Point (ICP)

April 22, 2022

Students:
Gijs van Meer
12152706

Irene Papadopoulou
13869337

Marius Strampel
12066664

Lecturer:
prof. dr. T. Gevers

Course:
Computer Vision 2

Course code:
52042COV6Y

1 Introduction

This report shows our implementation of the Iterative Closest Point (ICP) algorithm. Several improvements to ICP are covered, along with a quantification of their impact on both ICP's speed, as well as performance. The model is used both on point cloud matching tasks, using wave and bunny clouds (see figure 1), as well as global registration by way of pose estimating using Kinect sensor frames. Lastly, we provide a brief overview of the biggest shortcomings of ICP and what related work has been done to mitigate those shortcomings.

1.1 ICP Implementation

We first implement the basic ICP algorithm, by following the steps described in Besl and McKay (1992). To test if our algorithm's returns a correct estimated transformation, we experiment with the provided dummy data; "wave" and "bunny". Each one consists of two arrays, one containing all the source (A_1) and the other one containing all the target (A_2) 3D points. The wave data has 6400 points for both source and target, while the bunny data has 40256 and 35336 source and target points respectively.

First, for every source point a we find it's match with the closest target point (in terms of euclidean distance). For example, two source points could match to the same target point and that is why it is not important for our source and target to have the same number of points. Then, using Singular Value Decomposition (SVD) we compute the optimal rotation matrix R and translation vector t that minimize

$$\sum_{a \in A_1} \|Ra + t - \psi(a)\|^2,$$

where $\psi(a)$ is a function that returns point a's closest match from the target points A_2. To do this, we first center both source points and their matches around the origin (to obtain centered matrices X and Y) and compute the covariance matrix S by multiplying the two ($S = XY^T$). We then compute the singular value decomposition $S = U\Sigma V^T$ and obtain the optimal R and t, by

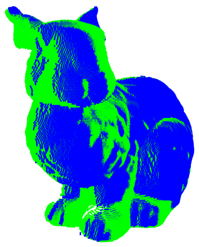
$$R = V \begin{bmatrix} 1 & & \\ & \ddots & \\ & & \det(VU^T) \end{bmatrix} U^T$$

and

$$t = \overline{y} - R\overline{x},$$

where $\overline{x}$ and $\overline{y}$ are the centroids of the source and matched target point clouds. Finally, we transform our original source points (A_1) with $A_1 = RA_1 + t$ and calculate the new closest matches (between the new source A_1 and original A_2 target points). In every iteration, we calculate the Root Mean Square (RMS) error between the new source points A_1 and their matches and we stop if the difference from the previous RMS is less than a certain threshold, or if we have reached the maximum number of iterations.

Figure 1 shows our implementation's results with the mentioned data. Noticeably, the algorithm runs considerably faster when using the wave data, since it needs to find fewer one-to-one point correspondences. In addition, we run the algorithm with down-sampled data, since for the bunny data the memory and computational time needs are too high. In Table 1 we compare our algorithm's speed with different data. As expected, the more points we have, the longer the algorithm needs to run. Additionally, when using the down-sampled bunny data the difference in RMS is minor compared to the difference in speed. Finally, in our experiments, we conclude that that the bunny data need a smaller threshold and more iterations to converge.


(a)

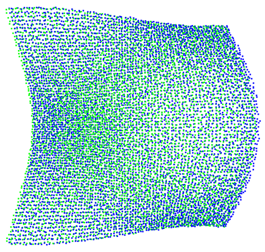

(b)

Figure 1: ICP results on the bunny (a) and wave (b). The target cloud point is in green and the transformed source after the ICP convergence is in blue.

	time (sec)	**threshold**	**RMS**	**convergence (iter)**
Wave (all points)	**8.84**	$1e-4$	**0.0167**	28
Bunny (all points)	5730.33	$1e-8$	**0.00164**	50
Bunny down-sampled (85%)	857.68	$1e-8$	0.00168	50
Bunny down-sampled (70%)	**358.38**	$1e-8$	0.00168	50

Table 1: **ICP speed results.** Comparison of speed between different data. When down-sampling, we down-sample from both source and target points. We run the algorithm with 50 as the number of maximum iterations. As expected, with more points, the algorithm needs significantly more time to converge.

2 Improving speed & quality

2.1 Subsampling

Several subsampling methods are compared in Rusinkiewicz and Levoy, 2001. This report will test three of these methods, as well as implementing its own version of a fourth method. These methods will be compared based on speed, accuracy, stability and the effect of noise.

2.1.1 Uniform sampling

The first subsampling method is the uniform subsampling. This method takes a randomly chosen sample from both the source and the target image, and then uses these samples to calculate the rototranslations for the entire source image.
An issue is that this method relies on obtaining a good subsample of points the first time the points are chosen, for which there exists a non-zero chance this does not happen. Another problem is noise, as the more the noise increases, the higher the chance that we select a large batch of inaccurate points.
The advantage is that due to having to choose points only once and doing this in a very simple way computation-wise we can achieve high speeds for approximating the correct rototranslation. This means that the uniform subsampling will be useful when we need fast approximations and don't care too much about exact alignment.

2.1.2 Random sampling

Random subsampling works in much the same way as the uniform subsampling, with one change. Instead of only creating the subset of images once before we iterate over the ICP algorithm, we now resample those points every iteration. This method reduces the chance of miscalculating based on having a bad batch of points, but increases computational cost.
For both random and uniform we sampled 1% of points by default.

2.1.3 Multi-resolution sampling

Multi-resolution subsampling works by starting at a low resolution of the image, meaning we sample only a small amount of points, and then increasing the amount of points we sample from as our adjustments to the target get smaller. Since the necessary rototranslations needed are not very precise at the start fewer points are needed. As the changes get smaller we are making more precise adjustments, which would require more points to make an accurate rototranslation. The steps are determined by a variable N. We start by some value larger than 100 as our total points sampled and multiply this by some value N until we get the full resolution of the image. Each time a threshold of RMS difference is crossed we increase the resolution of the image until we are working with the full resolution.
In the multi-resolution sampling paper (Jost & Hügli, 2002), the authors mention that using multi-resolution with kd-tree ICP (see section 2.2) can results in a 10x speedup in comparison to just kd-tree ICP. This is, however, a comparison done with an image with a resolution of 24 000 points, whereas our bunny data has 40 256 points. The paper is from 2002, meaning the kd-tree should have become more optimised in the years since. The authors also did not mention how they set their thresholds, complicating the comparison.
By default we ran Multi-resolution sampling using an N of 4, meaning we increase our image resolution by multiplying by 4 for each threshold reached.

2.1.4 Convex hull sampling

The final method is based on the informative region of an image. For this we use convex hull sampling. Normally methods would use either corner detection or edge detection to find the outline of the image. We instead decided to use a convex hull, which is the smallest set of points

that contain all other points within their borders. This way we extract the most important points within the edges of a point cloud.
First we create a triangle mesh from the point cloud and compute its vertex normals for the convex hull. We then extract the individual points of this hull. Finally, we use these as the source for the ICP. For these functionalities we use Open3D.
The issue with this method is that we might generate points that are not particularly close to the original mesh, resulting in some inaccuracies. This method should be a very stable method, as we only have to choose our points once and then continue with this set until ICP converges.

(a) (b)

Figure 2: Convex hull sampling on the bunny (a) and wave (b). The sampled points are in blue.

The mentioned points far away from the 3D point cloud can clearly be seen in figure 2a, we also see a lack of point distribution in the more simplistic wave model in figure 2b. These points will most likely lead to inaccuracies in the convergence step.
Other methods considered for this section were based on corner and edge detection itself (Ahmed et al., 2018) or a more complex version of the method implemented in this paper as seen in Bazazian et al. (2015).

2.2 Kd-Tree

Another method introduced to speed up the point matching required for ICP is called kd-tree (Bentley, 1975). The kd-tree algorithm builds a binary tree, where each non-leaf node partitions the cloud into two parts using hyperplanes aligned to some axis (x or y for 2D points). Points can then be queried on their closest points or closest approximate points based on where in the tree they end up. The average complexity search in kd-tree is $O(\log n)$, which is an enormous improvement on the $O(n^2)$ of ICP's standard exhaustive search. For this paper we used Scipy's implementation of kd-tree[1].

2.3 Z-buffer

The z-buffer is an improvement inspired by the computer graphics technique of the same name. It uses the z-value, the depth value in 3D space, of points to correctly identify which objects are occluded behind others when projected onto a 2D plane. This paper attempts to implement the mono-z buffer (hereafter referred to as z-buffer) as described in Benjemaa and Schmitt (1999). Matches are found by projecting points from their respective cloud onto some plane. The z-plane with the origin of the target cloud as a referential is used in this paper. This z-buffer referential used is of much importance, as this changes defines the 'camera angle', which obviously has a large effect on whether objects occlude each other. As such, the projection plane does not necessarily have to be the z-plane. Any arbitrary 2D plane could be used by selecting a direction of projection and directions orthogonal to it, which together form the z-buffer referential. We can then define

[1]https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html, accessed 22/04/2022

a plane somewhere along the direction of projection and simply project every point onto it using simple linear algebra.
We did not manage to implement a fully working version of the z-buffer improvement. It correctly shifts the source cloud towards the target cloud for a few iterations, before shifting back and forth, never quite converging. We do sometimes get good results for the wave data depending on the hyperparameters of z-buffer. Figure 16a in the Appendix shows such an example on wave, while figure 16b shows a typical result our z-buffer implementation on the bunny clouds.

2.4 Comparing the methods

We compare the methods on a few metrics. First of all we check how accurate the methods are by checking the value of the RMS obtained at each iteration, while also plotting the average number of iterations each method takes to run. Another test would be to see how stable the methods are, if they are stable we should see a smooth line and the method should be able to consistently finish within the chosen threshold of 100 iterations at most. The results are displayed in figures 3 through 7. All methods also apply the kd-tree method to their own subsampling method. We tested the methods on two point clouds of the Stanford Bunny. Each method was tested 25 times, after which we took the averages of these results.

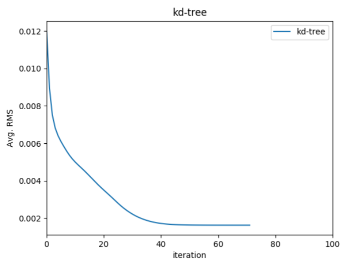


Figure 3: Using the kd-tree algorithm

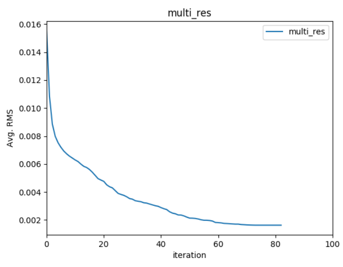


Figure 4: Using multi-resolution subsampling.

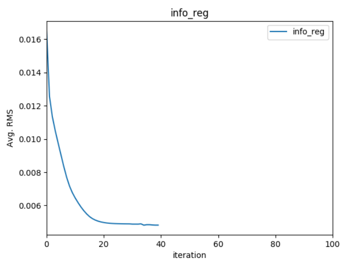


Figure 5: Using convex hull subsampling.

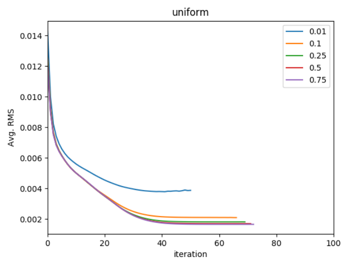


Figure 6: Using uniform subsampling.

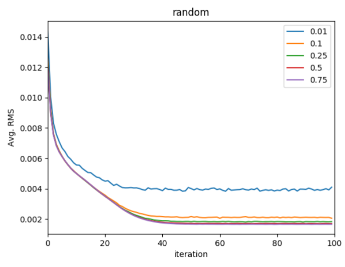


Figure 7: Using random subsampling.

By observing these results there are a couple of things we can notice. For these comparisons we will use figure 3 as the baseline to compare to.

In order to obtain a correct rototranslation the uniform and random subsampling (figures 6 and 7) should be run with at least 10% of points, but after this the differences become very minimal. The differences between the ratios for these two methods can be found in the appendix in tables 4 through 8. The random subsampling is clearly unstable, we see from the graph that the variance constantly shifts, since we are taking a new subsample of points each iteration.
It is clear that the convex-hull (figure 5) sampling method is still underdeveloped at this point, and would need further optimisation in order to achieve the desired accuracy. These optimisations could include removing the points too far away from the original point cloud, and adding more

points to achieve a more accurate outline of the point clouds. The method also takes too long for the results it achieves, being the second slowest with the worst results.
With multi-resolution (figure 4) we are not able to obtain the results as shown in Jost and Hügli, 2002. This is however due to the fact that our method seems to be faster in general despite having a larger point cloud (40 256 instead of 24 000 as per the original paper). We see in table 2 that the normal kd-tree takes us only 27 seconds to run, whereas it took Jost and Hügli around 504 seconds. Most likely the kd-tree used in this report is more optimised, since it also takes less time despite taking more iterations to converge, due to the increased point cloud complexity. Currently our method is about 3.46 times as fast as just using kd-tree. Not close to the 10 times reported by Jost and Hügli, but still an effective improvement. There is a slight instability when switching resolutions, but the method remains stable.

Method	**Avg time (sec)**	**threshold**	**Avg convergence (iter)**	**RMS**
Kd-tree	27.127	$1e-8$	72.0	0.00129
Multi-resolution	7.848	$1e-8$	82.88	0.00163
Convex hull	17.628	$1e-8$	40.4	0.00482
Uniform	0.153	$1e-8$	51.48	0.00387
Random	3.030	$1e-8$	100	0.00410

Table 2: **subsampling results.** Comparison of results on baselines mentioned before.

We also tested these methods on a variety of noise levels using a normal distribution, with μ equal to 0.05 times the point cloud's mean, and σ equal to 0.05 times the cloud's standard deviation. The results can be seen in figures 8 through 12. As the base line we took a noise rate of 0.1. The effect of the noise can be found in the Appendix (figure 17).

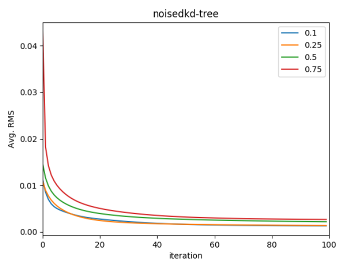


Figure 8: Kd-tree with added noise

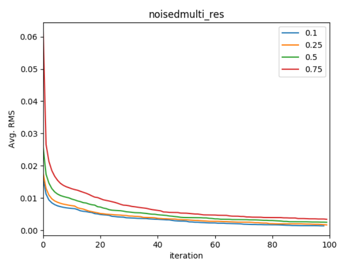


Figure 9: Multi-res subsampling with added noise

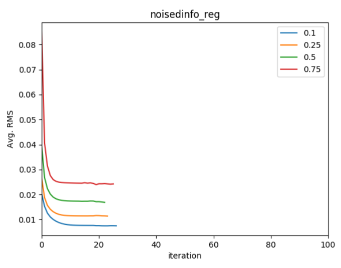


Figure 10: Convex-hull subsampling with added noise

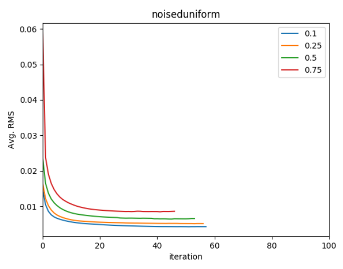


Figure 11: Uniform subsampling with added noise

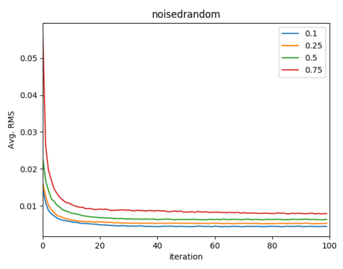


Figure 12: Random subsampling with added noise

We observe that two methods perform slightly worse, the uniform and random sampling (as seen in figures 11 and 12, with more detailed results in table 3). They achieve slightly worse RMS values which get worse as the noise level increases. These, however, were run under a subsampling ratio of 0.01, resulting in high chances of selecting inaccurate representations. Convex hull sampling performs the worst, as the shape of the image is changed resulting in the hull itself changing and no longer representing the point cloud outline. The other methods don't seem to be impacted much, both the multi-resolution (figure 9) and normal kd-tree (figure 8) perform consistently at low levels of noise, increasing slightly as the noise increases but never exceeding the random, uniform and convex hull methods.

Method	**Avg. time (sec)**	**threshold**	**Avg. convergence (iter)**	**RMS**
Kd-tree	25.141	$1e-8$	100.0	0.00129
Multi-resolution	7.009	$1e-8$	99.28	0.00136
convex hull	16.494	$1e-8$	26.72	0.00746
Uniform	0.230	$1e-8$	57.96	0.00428
Random	3.222	$1e-8$	100.0	0.00438

Table 3: **Noised subsampling results.** Comparison of results on baselines mentioned before.

3 Global Registration

In this section we explore global registration in which we will use ICP to stitch together a full 3D model of a student, from 100 overlapping point clouds. The point clouds where captured by a Kinect sensor rotated around the student. Our goal is to use ICP and find all rotations R_n and translations t_n between the frames, transform and merge all cloud points together to obtain the final 3D model. We do this with two different methods described in the following sections.

For both methods the provided point clouds are from real-world data and had to be cleaned. We remove background points, by removing the points that have a distance greater than 2 meters away from the camera.

3.1 Estimate pose every N frames and merge at the end

For this method, we first need to calculate all rotations R_n and translations t_n for every frame. We start by calling our ICP function with frame$_1$ as source and frame$_2$ as target to calculate R_1, t_1; the estimated rotation and translation which transforms frame$_1$ to frame$_2$. For the first iteration, our merged point cloud is simply the rototranslated source using R_1, t_1 (merged_pcd = R_1 frame$_1$ + t_1).

We continue with frame$_2$ as source and frame$_3$ as target to calculate R_2, t_2. Now, we transform the new source to the same angle as the new target and we get a new point cloud, called "source_transformed" for simplicity. In order to merge this new point cloud to our merged point cloud, we first transform the merged point cloud with R$_2$, t$_2$ and then we stack the transformed merged point cloud and "source_transformed" into one. We continue this process for all 100 frames provided and in each iteration we transform and merge our point clouds, to slowly build the 3D model of the student.

Using our insights from Section 2 since we now care for high accuracy instead of speed, we choose to not use any sampling methods. We simply run our ICP function with kd-tree, a threshold of 1e-10 and 200 as the number of maximum iterations.

In figure 13 we display our results when using all 100 frames. As can been seen, the 3D model of the student is accurate enough but not perfect. In particular in the side view ,13b, we notice that the arm is placed in more than one location. Also in the front view, 13a, we observe that left arm is slightly more to the front. The reason behind these mistakes could lie in some slightly incorrect estimations of R and t between frames that might have not converged or still include some background noise. Since this transformation is being applied to the merged point cloud in every iteration, this error can accumulate. For example, a lightly wrong estimation can lead to the merged point cloud being in a slightly different angle than the next frame used to estimate the new R, t.

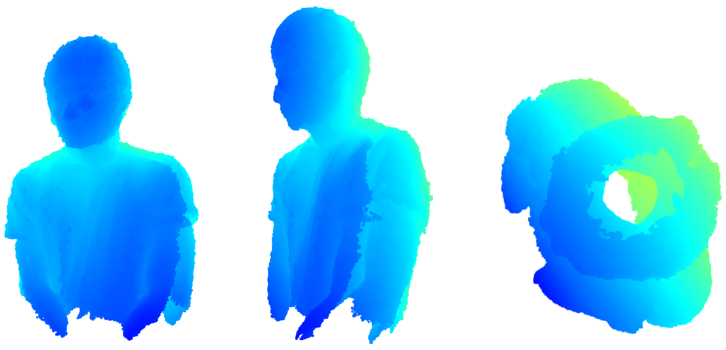

(a) Front View (b) Side View (c) Top View

Figure 13: 3D model visualization using all frames (N=1)

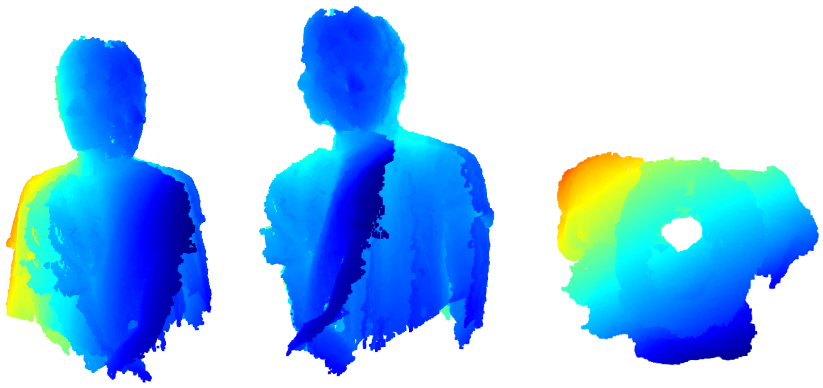

(a) Front View (b) Side View (c) Top View

Figure 14: 3D model visualization with N=4

We also repeat this process for different number of frames. In particular, we experiment with $N = 2, 4, 10$, which means that we skip and we calculate the R_n, t_n every 2, 4 or 10 frames. Results for the different N can be seen in figures 18, 14 and 19. It is clear that as N increases, the reconstruction gives worse results. That is as expected, since by skipping frames, there is a larger gap between the source and target points, making it harder for the ICP algorithm to accurately estimate the optimal rotation and translation.

3.2 Estimate pose and merge every N frames

This method follows the same logic as the previous method, except now we do not use the next frame as source in our ICP function, we instead use the merged point cloud we have so far. For example, in the previous method for N=4, we would use frame$_5$ as source and frame$_9$ as target to estimate R_2, t_2. Now, we use the merged point cloud (frame$_{15}$) as source and we keep the target the same (frame$_9$). We do the same for all frames. Since we are now using the merged point cloud as the source at each iteration and in each iteration this point cloud gets bigger, the computational time of ICP increases significantly. To alleviate this problem, we choose to use multi-resolution sampling, since from Section 2, we find that is the method with the best accuracy-speed trade off.

Figures 20, 15 show our results when using $N = 4$ and $N = 10$ respectively. From the figures, it is clear that the ICP algorithm fails to estimate the correct transformations with this method. In particular, the body of the student is not built at all from the back and there are many points placed incorrectly throughout the visualization. This is something we expect, since from the previous section we conclude that gap of 4 or even 10 frames will not lead to a good reconstruction.

With this method, we also await that the results will be even worse than Section 3.1, since now we do not use the real frame point clouds as source, so a wrong estimation of the camera pose by the ICP algorithm will not be able to recover in the next frames. It is also harder for the ICP to find a correct estimation, since it needs to match more and more points in every frame. When running the algorithm, we notice that the RMS values we get do not decrease below a certain value and might even get bigger between the different ICP runs, as the merged point cloud gets bigger.

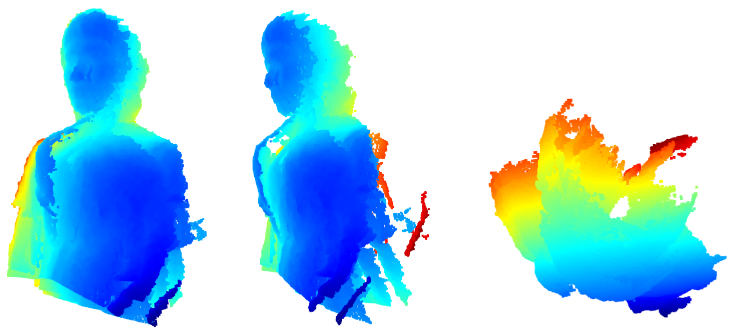

(a) Front View (b) Side View (c) Top View

Figure 15: 3D model visualization with N=10, merging every 10 frames and using merged frame as source in ICP

4 Discussion

Although simple to implement, the ICP algorithm does have several drawbacks. This section will provide a brief overview of the major drawbacks to ICP and showcase several different ways the literature has gone about solving them.

4.1 Speed

One of the biggest criticisms of the ICP algorithm is its slowness. This is largely due to two factors. Firstly, ICP's linear convergence (Pottmann et al., 2006), which can take a long time for complex problems. This can be improved upon in several ways. Pottmann et al. (2006) analyse and improve upon the SDM (Pottmann et al., 2004) model. SDM, and its variants, use quadratic approximations of the standard squared distance function quadratic function used in ICP. As such, SDM can obtain an estimate for the next position of each point by computing the second order Taylor approximant and minimizing the quadratic function. This is shown to allow for quadratic convergence, as opposed to ICP's linear convergence. Another method of improving ICP's linear convergence is to make use of Anderson acceleration as first described by Anderson (1965). This has been shown to significantly improve convergence speed, although this method offers no guarantees of global, nor even local convergence (Walker & Ni, 2011).
Secondly, ICP requires the distance to be computed between each source point and each target

point to determine the closest target point. This can be helped by using an efficient searching algorithm, such as kd-tree (Bentley, 1975). Several other techniques are outlined in (Rusinkiewicz & Levoy, 2001).

4.2 Sensitivity to data abnormalities

The standard ICP algorithm is notoriously susceptible to abnormalities in data, namely outliers, missing data, and partial overlap between the source and target cloud. Several heuristics have been proposed to prune or provide weights to source-target pairs to reduce the impact of outliers and missing data (Rusinkiewicz & Levoy, 2001). Such heuristics are often difficult to implement and hard to tune for optimal performance. To counteract this, Bouaziz et al. (2013) propose a new technique that makes use of sparsity to enforce robustness to outliers and missing data. To improve accuracy on partially overlapping data, Trimmed ICP has been developed (Chetverikov et al., 2002). Trimmed ICP has shown improved results over ICP for overlaps up to 50% by making frequent use of least trimmed squares in all stages of the algorithm.

4.3 Local minima

Another problem with ICP is that it does not ensure global minima, only local minima (Pottmann et al., 2006). One approach to solve this has been to find ϵ-suboptimal solutions: solutions that are at most some value ϵ away from the global optimum (Olsson et al., 2008). Yang et al. (2013) developed the first ICP variant guaranteed to converge to a global optimum.

4.4 Rigidity

The ICP algorithm only works on rigid registration or matching problems. For a full discussion on rigid, non-rigid and affine registration see Tam et al. (2012). Briefly, a registration problem is rigid if the source and target only differ by a rigid transformation, namely some combination of translation, rotation and reflection. A non-rigid registration problem, then, is one where source and target differ by a non-rigid transformation, which may include affine transformations, morphing, shearing, etc. Advances have been made to expand ICP's capability of solving non-rigid problems (Amberg et al., 2007; Du et al., 2010).

5 Conclusion

In this report we highlighted the ICP algorithm and some of subsampling methods. We found Muti-resolution to be the most time effective of these methods. Next, we were unable to implement the z-buffer so were not able to compare this method to kd-tree. Thirdly, we observed the capabilities and limitations of the ICP method in stitching multiple point clouds together. Finally we highlighted some key issues remaining in the ICP algorithm and the solutions implemented in literature to reduce these.

References

Ahmed, S. M., Tan, Y. Z., Chew, C., Mamun, A. A., & Wong, F. S. (2018). Edge and corner detection for unorganized 3d point clouds with application to robotic welding. *CoRR, abs/1809.10468*. http://arxiv.org/abs/1809.10468

Amberg, B., Romdhani, S., & Vetter, T. (2007). Optimal step nonrigid icp algorithms for surface registration. *2007 IEEE conference on computer vision and pattern recognition*, 1–8.

Anderson, D. G. (1965). Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, *12*(4), 547–560.

Bazazian, D., Casas, J. R., & Ruiz-Hidalgo, J. (2015). Fast and robust edge extraction in unorganized point clouds. *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 1–8. https://doi.org/10.1109/DICTA.2015.7371262

Benjemaa, R., & Schmitt, F. (1999). Fast global registration of 3d sampled surfaces using a multi-z-buffer technique. *Image and Vision Computing*, *17*(2), 113–123.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, *18*(9), 509–517. https://doi.org/10.1145/361002.361007

Besl, P., & McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *14*(2), 239–256. https://doi.org/10.1109/34.121791

Bouaziz, S., Tagliasacchi, A., & Pauly, M. (2013). Sparse iterative closest point. *Computer graphics forum*, *32*(5), 113–123.

Chetverikov, D., Svirko, D., Stepanov, D., & Krsek, P. (2002). The trimmed iterative closest point algorithm. *Object recognition supported by user interaction for service robots*, *3*, 545–548.

Du, S., Zheng, N., Ying, S., & Liu, J. (2010). Affine iterative closest point algorithm for point set registration. *Pattern Recognition Letters*, *31*(9), 791–799.

Jost, T., & Hügli, H. (2002). A multi-resolution scheme icp algorithm for fast shape registration, 540–543. https://doi.org/10.1109/TDPVT.2002.1024114

Olsson, C., Kahl, F., & Oskarsson, M. (2008). Branch-and-bound methods for euclidean registration problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(5), 783–794.

Pottmann, H., Huang, Q.-X., Yang, Y.-L., & Hu, S.-M. (2006). Geometry and convergence analysis of algorithms for registration of 3d shapes. *International Journal of Computer Vision*, *67*(3), 277–296.

Pottmann, H., Leopoldseder, S., & Hofer, M. (2004). Registration without icp. *Computer Vision and Image Understanding*, *95*(1), 54–71.

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the icp algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 145–152. https://doi.org/10.1109/IM.2001.924423

Tam, G. K., Cheng, Z.-Q., Lai, Y.-K., Langbein, F. C., Liu, Y., Marshall, D., Martin, R. R., Sun, X.-F., & Rosin, P. L. (2012). Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE transactions on visualization and computer graphics*, *19*(7), 1199–1217.

Walker, H. F., & Ni, P. (2011). Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, *49*(4), 1715–1735.

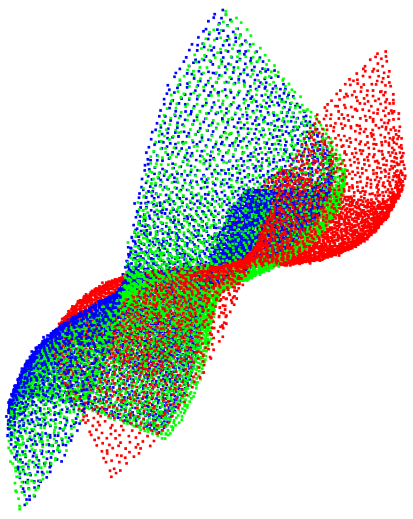
Yang, J., Li, H., & Jia, Y. (2013). Go-icp: Solving 3d registration efficiently and globally optimally. *Proceedings of the IEEE International Conference on Computer Vision*, 1457–1464.

Appendix

A Self-Evaluation

In this assignment Gijs worked on sampling (Section 2.1) and on running the results with the different sampling methods and noise, making the final plots. Irene worked on the implementation of the ICP algorithm (Section 1.1), as well as the merging of the cloud points (Section 3) with the help of Gijs for 3.2. Finally Marius worked on the implementation of Kd-Tree (Section 2.2), Z-buffer (Section 2.3) and reading literature for the Discussion.

B z-buffer results


(a) Wave

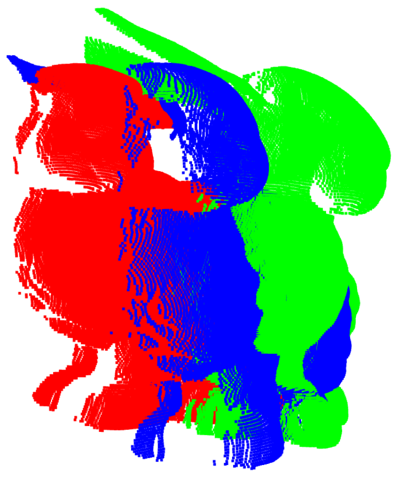

(b) Bunny

Figure 16: (a): z-buffer ICP on wave with $H, W = 15$ and $m = 3$, (b): z-buffer ICP on wave with $H, W = 100$ and $m = 5$.

C Sub sampling results

Ratio	Noise	Avg time (sec)	threshold	Avg convergence (iter)	RMS
0.01	0	0.153	$1e-8$	51.48	0.00387
0.1	0	2.229	$1e-8$	67.04	0.00209
0.25	0	6.880	$1e-8$	70.12	0.00181
0.5	0	12.990	$1e-8$	71.92	0.00169
0.75	0	33.071	$1e-8$	72.76	0.00165
0.01	0.1	0.230	$1e-8$	57.96	0.00428
0.01	0.25	0.452	$1e-8$	56.64	0.00515
0.01	0.5	0.598	$1e-8$	54.04	0.00657
0.01	0.75	0.720	$1e-8$	46.56	0.00862

Table 4: **Sub sampling results of uniform sub sampling.** Comparison of uniform sub sampling using different sub sampling ratios. Also with potential noise ratios

Ratio	Noise	Avg time (sec)	threshold	Avg convergence (iter)	RMS
0.01	0	3.030	$1e-8$	100	0.00410
0.1	0	22.712	$1e-8$	100	0.00205
0.25	0	49.175	$1e-8$	100	0.00183
0.5	0	113.276	$1e-8$	100	0.00169
0.75	0	214.237	$1e-8$	100	0.00789
0.01	0.1	3.222	$1e-8$	100	0.00438
0.01	0.25	3.433	$1e-8$	100	0.00438
0.01	0.5	3.420	$1e-8$	100	0.00633
0.01	0.75	3.647	$1e-8$	100	0.00789

Table 5: **Sub sampling results of random sub sampling.** Comparison of random sub sampling using different sub sampling ratios. Also with potential noise ratios

Noise	Avg time (sec)	threshold	Avg convergence (iter)	RMS
0	7.848	$1e-8$	82.88	0.00162
0.1	7.009	$1e-8$	99.28	0.00136
0.25	4.160	$1e-8$	100	0.00166
0.5	3.546	$1e-8$	100	0.00247
0.75	4.160	$1e-8$	100	0.00339

Table 6: **Sub sampling results of multi-resolution sub sampling.** Comparison of multi-resolution sub sampling with different noise ratios

Noise	Avg time (sec)	threshold	Avg convergence (iter)	RMS
0	17.628	$1e-8$	40.4	0.00482
0.1	16.494	$1e-8$	26.72	0.00746
0.25	17.916	$1e-8$	24.28	0.0113
0.5	18.715	$1e-8$	23.28	0.0169
0.75	16.337	$1e-8$	25.92	0.0242

Table 7: **Sub sampling results of convex hull sub sampling.** Comparison of convex hull sub sampling with different noise ratios

Noise	Avg time (sec)	threshold	Avg convergence (iter)	RMS
0	27.127	$1e-8$	72.0	0.00163
0.1	25.141	$1e-8$	100	0.00130
0.25	20.132	$1e-8$	100	0.00136
0.5	19.768	$1e-8$	100	0.00218
0.75	19.710	$1e-8$	100	0.00265

Table 8: **Kd-tree results.** Comparison of kd-tree without sub sampling with different noise ratios

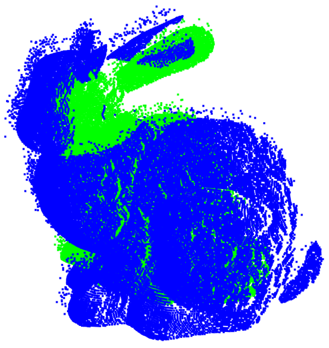


Figure 17: Bunnies with noise with a rate of 0.1 applied to it.

D Global Registration Results

D.1 Estimate pose every N frames and merge at the end

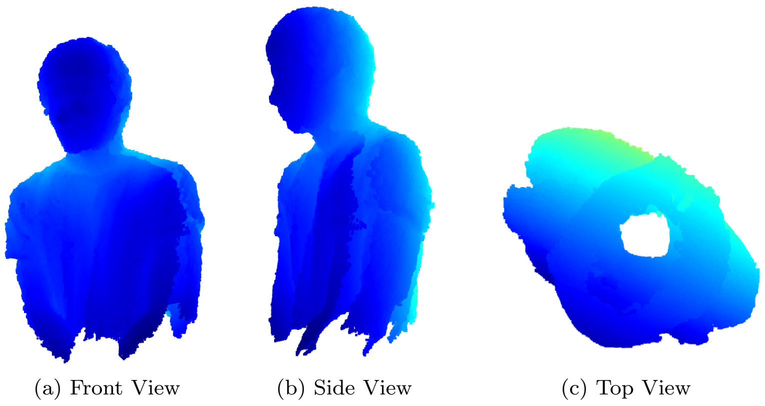


Figure 18: 3D model visualization with N=2

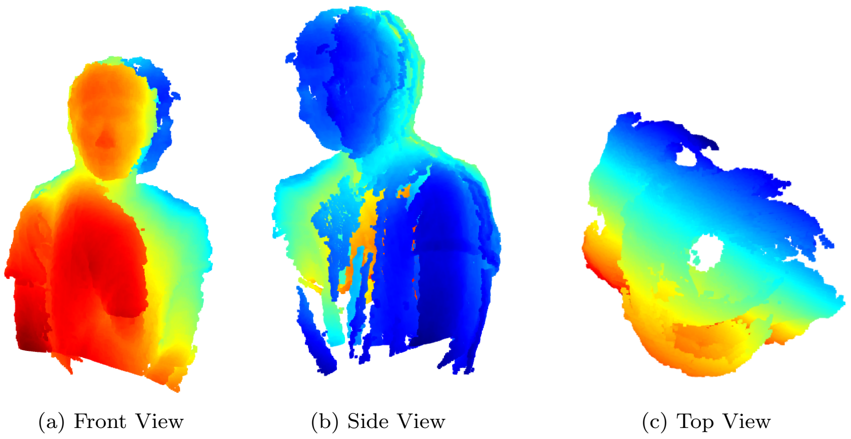
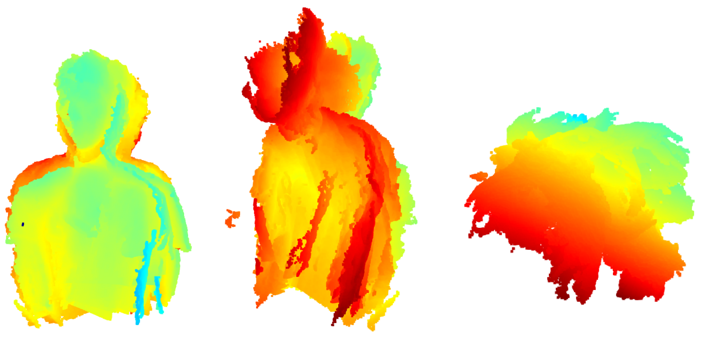


Figure 19: 3D model visualization with N=10

D.2 Estimate pose and merge every N frames



(a) Front View (b) Side View (c) Top View

Figure 20: 3D model visualization with N=4, merging every 4 frames and using merged frame as source in ICP