Computer Vision 2: Assignment 1

Angela van Sprang, Oleg Litvinov, Emily Mes

April 22, 2022

1 Introduction

This brief research report describes our findings for some experiments using the ICP algorithm. We discuss the algorithm itself and analyse some improvements to improve the speed and quality. For the matching step, we have implemented the brute force approach, the kdtree approach and the z-buffer approach. For the sampling step, we have implemented uniform sampling, random sampling, multi-resolution sampling and gradient sampling. Furthermore, we have applied our implementation to a real data set of a student to reconstruct a 3D model by global registration.

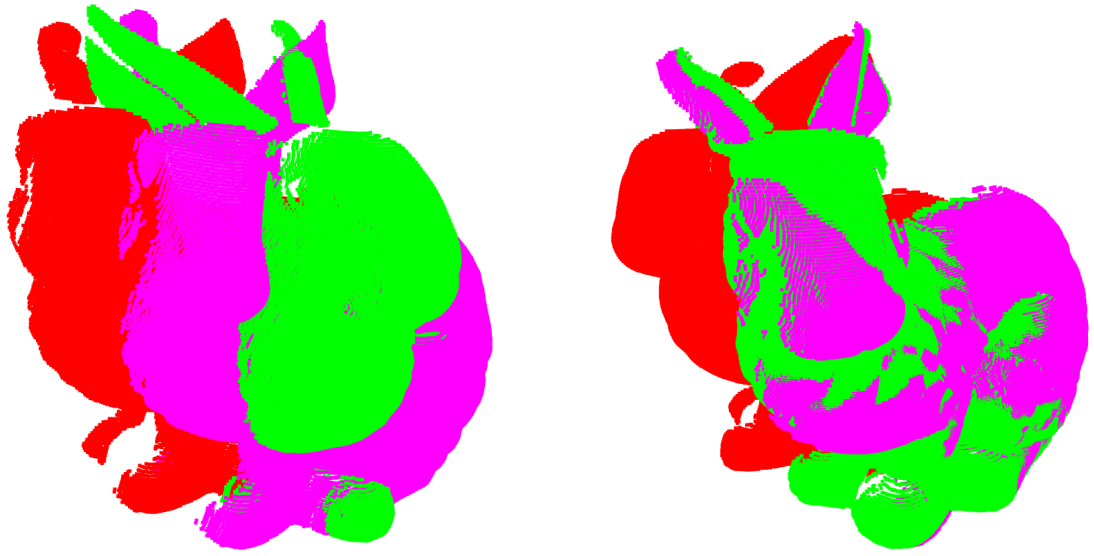
2 Basic workings of ICP

Iterative Closest Point (ICP) is a method for finding a spatial transformation between 2 given point clouds, which are A_1, A_2, the source and target respectively. This spatial transformation (rotation matrix R and translation vector t) is found by minimizing the distance between the point clouds by Root Mean Square Error (RMSE). The algorithm of ICP works as follows.

1. R and t are initialized to be the identity matrix and zero translation vector.
2. A_1 point cloud is transformed using the current values of R and t.
3. the points from A_1 are matched to the closest point in A_2. This matching step could be done on a brute force way, or be optimized making use of K-D Tree or Z-buffer, which are discussed later.
4. the new R and t are found using SVD between the matched points.
5. Go to step 2. This loop will continue until RMSE is converged within a certain threshold or for a pre-defined number of iterations.

3 ICP Implementation

After implementing the above mentioned steps of the ICP algorithm, we suspected that we have not implemented the algorithm to work as well as it should. We suspect there has been a bug somewhere that sometimes gives us unexpected results. This would sometimes result in no convergence or convergence at an unexpected place. Unfortunately, we were not able to solve this problem for all cases. While our results showcase some basic working of the ICP algorithm, we expected the algorithm to sometimes work better and therefore align better with the target data set than it did. As can be seen in Figure 1, we can definitely see that it is going in the right direction, since the constructed point cloud (purple) is between the original starting point (source, red) and the desired end point (target, green). However, from what we have read in the literature about ICP, we would have expected the constructed point cloud (purple) to exactly align with the target point cloud (green). Unfortunately, we were not able to reproduce that behaviour and fix the potential bug in our ICP implementation. On the other hand, we saw the ICP perform the expected and desired behaviour. We will come back to this matter in the discussion. Therefore we were able to improve our initial ICP algorithm by the following improvements, and make interesting comparisons with the different improvements.



(a) Incorrect (b) Correct

Figure 1: Result of initial ICP implementation on the bunny data set. Red is the source point cloud, green is the target point cloud and purple is the constructed point cloud. So the closer the purple cloud is to the green cloud, the better.

4 Improvements

4.1 Sampling

We have tried different sampling methods with the goal of saving time and memory each iteration by using a lower number of points, while keeping quality as high as possible. Sampling also helps to ensure both cloud points have the same number of points. The different methods are described below:

1. **All points.** When we aim to process all the points, we check if the clouds have a different number of points. If so, we randomly select points of the bigger cloud to make it the same size as the smaller cloud. Moreover, if a brute-force cloud matching approach is combined with all points, we select 1 point per 100 from both clouds to make the computation possible.

2. **Uniform sampling.** We sample uniformly from both the source and target cloud before iteration. We sample the same number of points from the target and the source. This number is calculated as the minimum number of points between two clouds divided by 50.

3. **Random sampling.** We sample both clouds uniformly in each iteration. For each iteration we transform the original source cloud with the current (intermediate) values of R and t. This way, the algorithm can make use of more points in the source and target than just one set of samples, while still use a lower number of points each iteration. This number of points is calculated in the same way as in the uniform sampling.

4. **Multi-resolution sampling.** We implemented this method as described in [2]. We divide the number of points in each cloud by a factor $N = 4$ in each iteration, but keep the number of points of the reduced data sets above the threshold of 50 points. The intuition behind this method is to create coarse to fine matchings over the number of iterations to obtain a faster convergence. In general, a lower resolution matching implies more important transformations and therefore a faster convergence.

5. **Gradient sampling**. We take the points in the source and target clouds that have the highest 2nd order norm of the gradient (before iteration). This way, we take the points into account that are in the most informative regions of the clouds.

4.2 kd tree

An optimization of the brute force matching step is making use of a K-D tree. A K-D tree is a binary search tree where space partitioning is used to organize the points in K-D. Note that in this case we represent points in 3D, so K=3. Our implementation makes use of the KDTree function of the *sklearn* package.

4.3 z-buffer

Another optimization of the matching step we implemented was making use of a z-buffer [1]. This technique is often used in computer graphics to account for occlusion of the 3D objects on the 2D screen. In this case, we initialize for both point clouds a buffer. The size of both buffers is equal and is dependent on the minimum bounding box of the union of the 2 point clouds. Then every 3D point of the source and target cloud are orthogonally projected onto their corresponding buffers. When multiple points are projected on the same cell, the point that is closer to the buffer is chosen (to account for occlusion). In practice, this means that we keep the point with the smallest z-value. The points are stored in the buffer by keeping track of their indices in the original clouds. Then we can match every point in the source buffer by going over the target buffer in a m x m window. We chose m=3, because it was a good trade-off between efficiency and performance. When multiple points are matched, we choose the closest one by means of Euclidean distance. This results that we are able to get a matching from the points in the source cloud to the points in the target cloud, whereafter we are able to continue the ICP steps mentioned above. While [1] proposes a more elaborate multi-z-buffer method, we only make use of the mono-z-buffer. The authors mention that this approach works well especially when the overlap between the two surfaces is sufficiently planar. However, a downside of this simplified approach is that when we have strongly curved overlap, the matching algorithm does not converge properly. We can also observe this difference with our two synthetic data sets, as can be seen in Figure 2. Since we want the constructed point cloud (purple) to be as close as possible to the target point cloud (green), we see that both data sets do have some possible improvements to be made. But the most interesting part is the noticeable difference in performance between these 2 data sets. We can see that for the wave data set, the z-buffer matching could be better, but performs fine. Especially because we see that the constructed cloud in between the target and source clouds, meaning that it has moved towards the correct direction. On the other hand, we see that for the bunny data, the z-buffer matching technique was not successful, because we can see that the constructed (purple) cloud moved further away from the source (red) and target (green) clouds. This is in line with what was mentioned before; a drawback of the mono-z-buffer is that it has problems with strongly curved overlap, which is the case with the bunny data set. To improve this, we could use the multi-z-buffer approach (as proposed by [1]), which produces a more homogeneous resampling and is therefore more robust to strongly curved overlap.

In our implementation, we placed the z-buffer referential in the origin of the target point cloud. To align both point clouds to the z-buffer referential, we transformed the source point cloud with the more recently found R en t. So we transformed the source point cloud to the target point cloud. If we would want to change the z-buffer referential, we should align the coordinate system of both point clouds to the z-buffer referential, which requires the transformation from both point clouds to the arbitrarily placed z-buffer referential, which can not be trivially found. Therefore, in practice, the z-buffer referential is most easily placed at the origin of one of the point clouds. However, the choice of the z-buffer referential is not important to the quality of the registration. It would only complicate finding the transformations.

4.4 Analysis

We tried to align each model to the other. All the models start from 1000 points. Only the combination of 'all' and 'brute-force' start with 1500 dots to show how long it is.

Having the vanilla ICP implemented as well as sampling methods, k-d tree, and z-buffer, we evaluated:

1. speed
2. accuracy
3. stability

(a) Wave data (b) Bunny data

Figure 2: Z-buffer matching on the wave and bunny data sets. Red is the source point cloud, green is the target point cloud and purple is the constructed point cloud. So the closer the purple cloud is to the green cloud, the better.

4. tolerance

Each combination of the sampling and matching methods was launched for 5 times.

4.4.1 Speed

Time was evaluated as a sum of initialization (uniform and gradient samplings) and the ICP running times before convergence with the epsilon value of 0.0001. From the plot 3 we can easily see that brute-force cloud matching takes always more time than k-d tree. Apart from that, multi-resolution sub-sampling outperforms all the other methods because of the fast point number decrease.

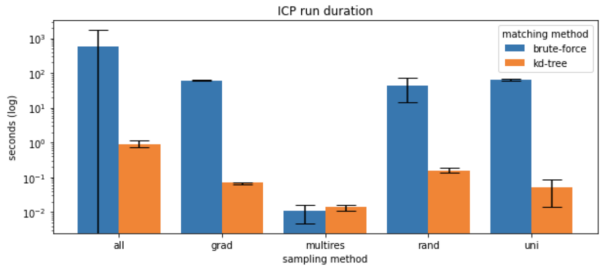


Figure 3: ICP running time

4.4.2 Accuracy

Accuracy was evaluated as the last obtained RMSE value during of ICP with the epsilon value of 0.0001. From the plot 4 we can see that matching method has almost no impact on the accuracy. What is not surprising, when we consider all the points available, final RMSE score is better (lower) in comparison to the other models. Gradient-based sampling also has quite small RMSE values because it captures the corner points which are important to match.

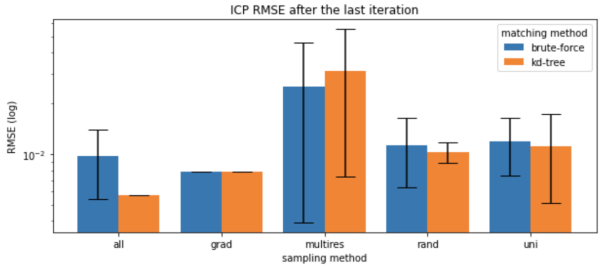


Figure 4: ICP RMSE

4.4.3 Stability

To evaluate stability, we calculated how often the algorithm converged before a hard stop on the 15th iteration with the epsilon value of 0.0001. For 0.001 all the combinations of hyperparameters lead to convergence. But in this case, for example, grad methods don't converge at all while the other combinations do not show any reasonable pattern.

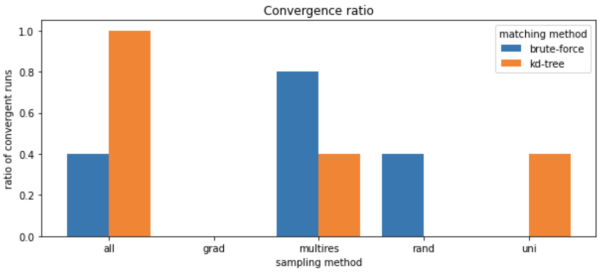


Figure 5: ICP stability

4.4.4 Tolerance

To evaluate tolerance, we launched each setup with different decreasing values of epsilon for 5 times. We assume that the lowest value with which algorithm converged before the hard stop on the 15th iteration is the optimum "resolution". For each hyper-parameter combination we calculated the number of convergence cases like 6.

				cases of this values
sampling	method	epsilon	converged	
all	bf	1.000000e-07	False	5
		1.000000e-06	False	5
		1.000000e-05	False	5
		1.000000e-04	False	3
			True	2
		1.000000e-03	True	5

Figure 6: ICP tolerance

For brute-force matching method we have the following values:

- all points sampling: 1e-3
- gradient-based sampling: 1e-3
- multi-resolution sampling: all the resolutions sometimes converge

- random sampling: 1e-3
- uniform sampling: 1e-3

For K-D tree matching:

- all points sampling: 1e-4
- gradient-based sampling: 1e-3
- multi-resolution sampling: all the resolutions sometimes converge
- random sampling: 1e-3
- uniform sampling: 1e-3

5 Global Registration

We have been given 100 overlapping point clouds made by a Kinect sensor that has rotated around a student. We have tried to stitch together a full 3d model by finding all transformations between the frames and stitch them together. We did this following two different approaches described below.

Note that there were noise and artifacts present in the data. By inspection, we found that these could be removed by setting a threshold of 2 in the z-dimension as the points of interest lied within this range.

5.1 Estimate pose every N frames and merge at the end

In the first approach we estimate the camera poses (i.e. R, t) every N frames of given data using our ICP algorithm implemented earlier, where $N \in \{1, 2, 4, 10\}$. So, for every N^{th} frame we can transform the cloud into the coordinate system of the $(N+1)^{th}$ frame. By merging the clouds (i.e. transforming the clouds into the same coordinate system), we should obtain a full 3d model.

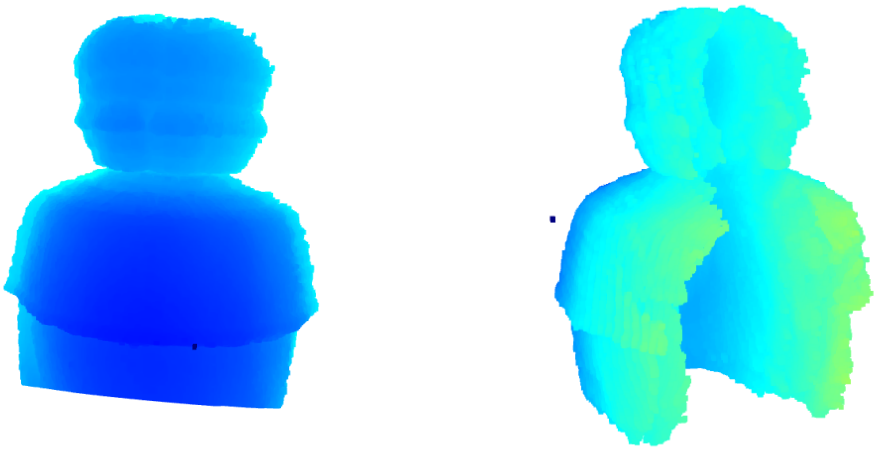
We used multi-resolution sampling and kd-tree matching to obtain our results. From our previous work, we experienced good and fast results in this setting.

Unfortunately, the merging does not produce sufficiently accurate results. This can be seen in Figures 7, 8, 9 and 10, which show visualizations of the merged point clouds for the different values of N. Multiple remarks can be made about these results:

- The different clouds are successfully put in the same coordinate system, because the different clouds are localized at the same point and form one new cloud;
- The different clouds are not well transformed to each other, since it seems from the front of each merged cloud that the person has multiple noses facing different sides. This should not have occurred;
- The back of the person is not well recovered, which can be seen from the side pictures of the results.

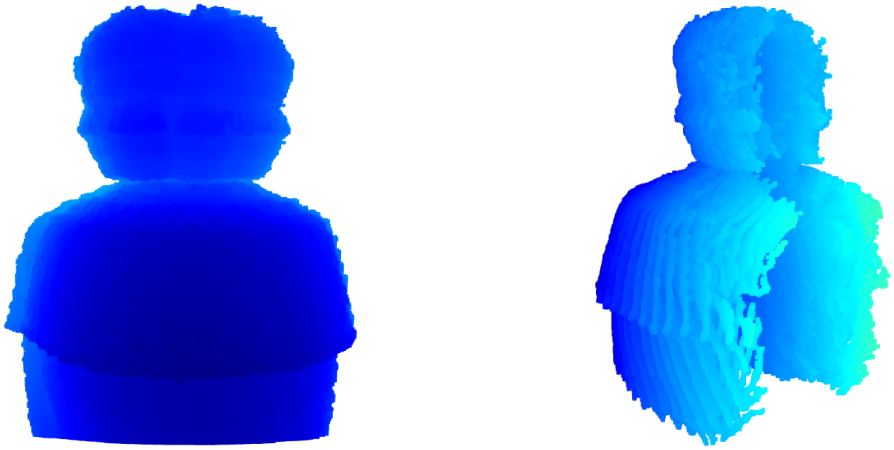
With regards to the different values of N, we can see that the merged clouds are smoother for lower values of N. This is as expected, since more clouds are included in the ICP and transformed for lower values of N. This is especially well visible when looking at the sides of the cloud points.

Sampling and kd-tree matching improves the speed significantly. Without a sampling method and with brute-force matching, it would take the algorithm extremely long for any value of N, so this is no feasible option. We have not performed an exhaustive grid search over all methods and parameters (as this is explicitly not asked for by the assignment), but we have witnessed that multi-resolution sampling improves the speed more than the other sampling methods (as implemented previously). This is probably due to the fact that most iterations are done near the chosen threshold of 50 points, while the other sampling models sample more points. We have also witnessed that the quality is not impacted a lot by choosing another sampling method.



(a) Front (b) Side

Figure 7: Merging at end for $N = 1$



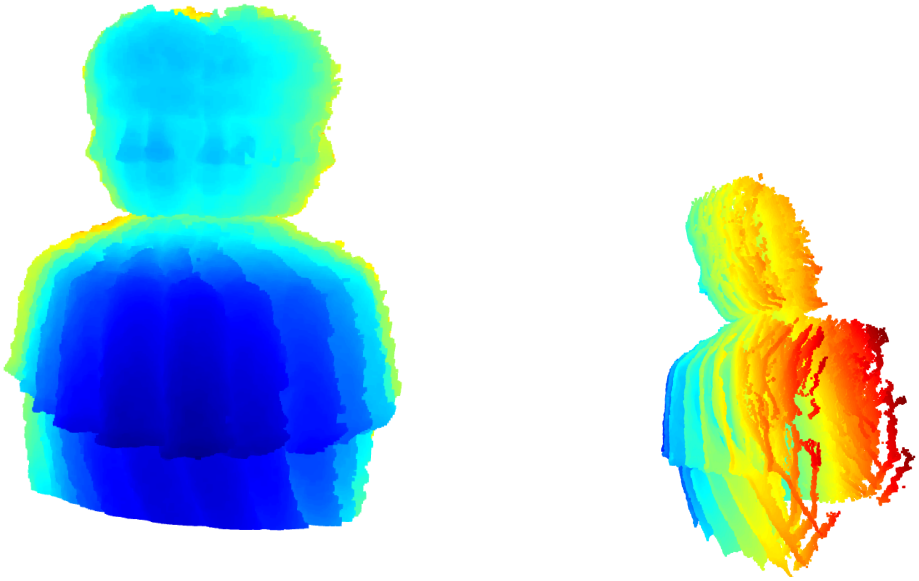
(a) Front (b) Side

Figure 8: Merging at end for $N = 2$

5.2 Estimate pose and merge every N frames

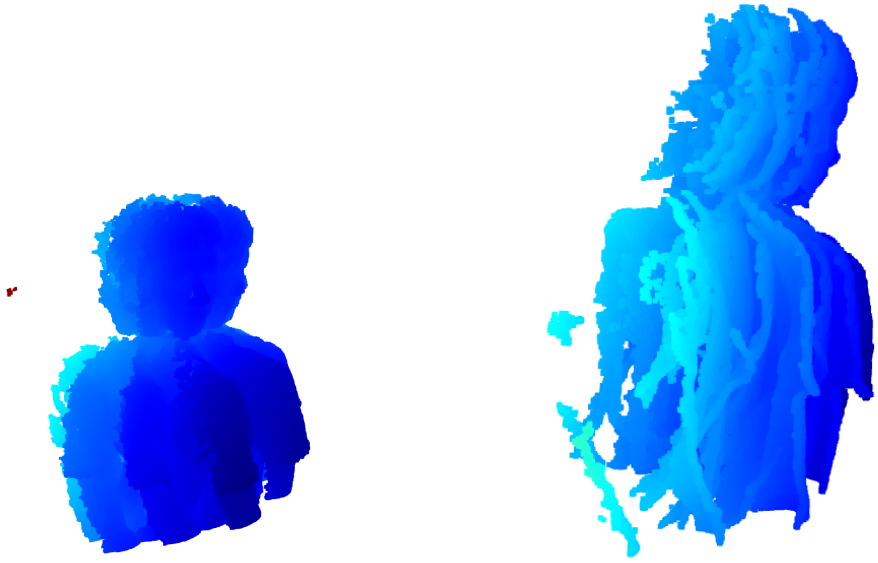
In the second approach we merge the point clouds every N frames instead of at the end. So, we still estimate the camera poses (i.e. R, t) every N frames using our ICP algorithm as implemented earlier. However, we already merge the frames for which we have obtained the camera poses and determine the remaining camera poses with the merged frames. In this approach, we only consider $N \in \{4, 10\}$. At the end of the estimation of poses, we already have the merged final frame.

We used uniform sampling (before iteration) and kd-tree matching to obtain our results. Sampling is important for this approach, since it becomes unfeasible to take the entire merged source as source cloud after a few iterations. This is why we uniformly sample the same number of points that are in the target cloud from the merged cloud. We sampled less points for higher values of N to maintain a feasible run-time.



(a) Front (b) Side

Figure 9: Merging at end for $N = 4$



(a) Front (b) Side

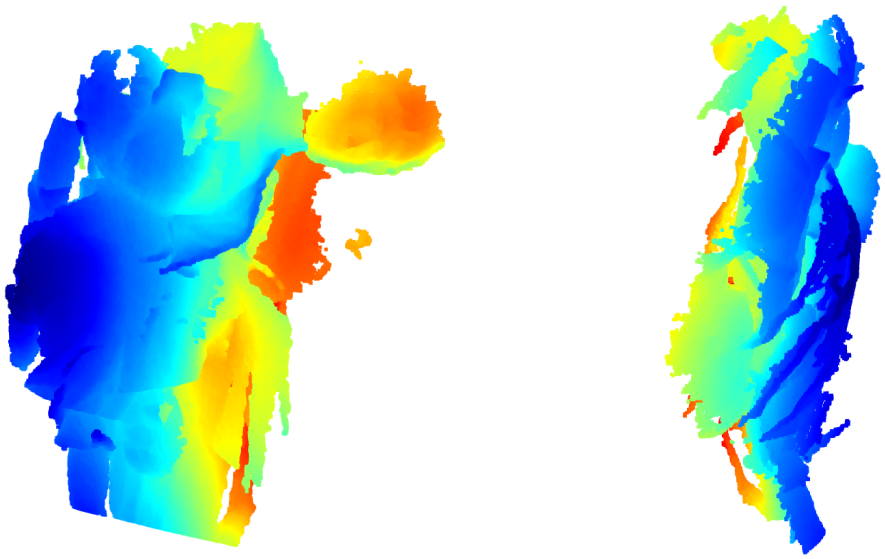
Figure 10: Merging at end for $N = 10$

The estimated camera poses do change in comparison with the previous estimates from the first approach. In the current approach, we are able to match points from all previous clouds to the target. For example, we could try to match a point in the person's back to his front, which will result in a different transformation than if we only took points from his front.

This estimation does not provide better results. In fact, this estimation performs worse than the previous approach. The results of this approach for the different values of N are given in Figures 12 and 11. The following remarks can be made regarding these Figures:

- The clouds are put in the same coordinate system (up to a certain degree), since they are all overlapping;
- The clouds are not well transformed to each other, since they are not well aligned. This should not have occurred;
- It seems that this approach performs worse for lower values of N, which is probably because the more clouds are incorporated into merging, the more distorted the matchings in the ICP become.

We would also like to mention as a result that this approach of merging the cloud points at every step did sometimes not converge (or at least unfeasibly slow). We think this is due to unfortunately sampling difficult points from previous clouds as described previously.

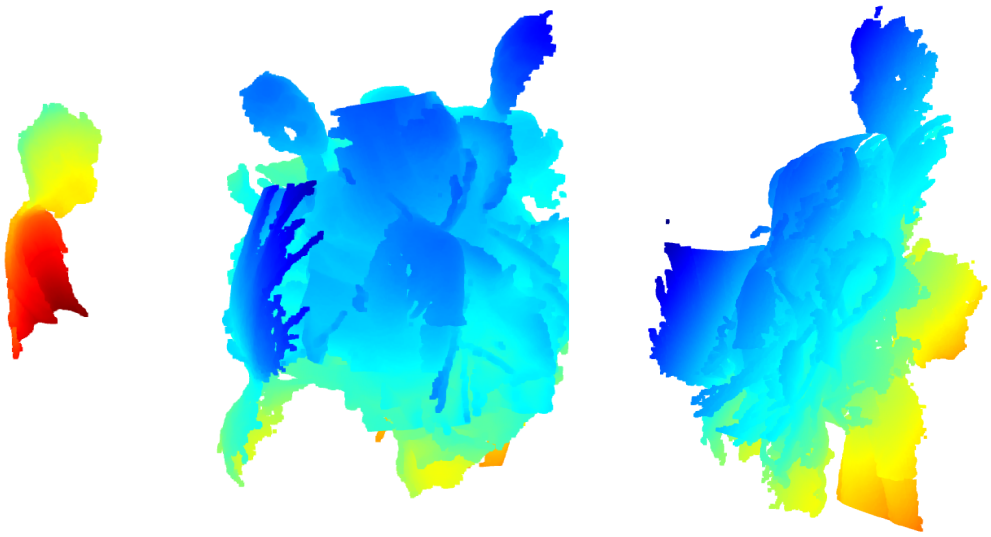


(a) Front (b) Side

Figure 11: Merging every $N = 10$ frames

6 Discussion

As can be seen in the qualitative results above, we were not able to always reproduce the workings of the ICP algorithm as well as it should. We observed cases where the ICP algorithm was not able to converge at all, or converge not close to the target point cloud. Other times, the ICP algorithm would perform as perfectly as expected. While this may be partially due to the drawbacks of ICP, it could also be caused by a bug in the ICP algorithm, which we were not able to solve in the span of this assignment. One of the drawbacks of the ICP algorithm is that some random initialization is involved, meaning that the ICP algorithm is not deterministic. It could converge in a local optimum, while we would want it to always converge in a global optimum. In [3], the ICP algorithm is evaluated extensively, in which the ultimate conclusion is that its convergence in a global optimum can not be guaranteed, and therefore possible local optimum convergence is inevitable. Other drawback of the ICP algorithm are its possibly slow convergence and the way it deals with outliers. For example, if we would have 2 almost identical point cloud, but if one of the point clouds has an extreme outlier, the point clouds would not be matched neatly, because that one point cloud would also be taken into account when calculating the error function. Therefore, it could be beneficial to apply some outlier detection and removal algorithm to both point clouds before applying the ICP algorithm. As for the other 2 possible drawbacks (possibly converging in a local optimum and its slow convergence), a possible improvement could be to combine Stochastic Gradient Descent with the ICP algorithm to make it possible to escape out of a local optimum, as is done by [4]. From the results, the authors were able to conclude that,



(a) Front (b) Side

Figure 12: Merging every $N = 4$ frames

if local optima were present, their stochastic variant of the ICP algorithm was more robust and more precise that the standard variant, which would most definitely result in better accuracy. Furthermore, as stochastic gradient ascent is known to converge faster than standard gradient descent, we expect the stochastic variant of ICP to also converge faster, and therefore be more efficient.

7 Conclusion

As has been discussed in this research paper, the ICP algorithm should be a robust and stable algorithm to obtain the spatial transformation between 2 given point clouds. We have seen multiple sampling and matching approaches, of which resulted in quite some difference with regards to speed and quality. While we were not able to succeed in the most stable ICP implementation, we still see the possibilities of using ICP to reconstruct a 3D model. In future work, we hope to optimize our ICP algorithm by solving the (most likely) bug and overcome the biggest drawback of the ICP algorithm by trying out a Stochastic variant that is less likely to get stuck in a local optimum.

A Self-Evaluation

All team members contributed equally to the assignment. Everyone's contribution are listed below.

1. Emily made the first version of the ICP algorithm and she also implemented the kd-tree and z-buffer.
2. Oleg made the second version of the ICP algorithm and implemented gradient sampling. He also did the quantitative analysis of the results.
3. Angela worked with Oleg on finding the bug in the ICP and implemented the different sampling methods (except gradient sampling). She also implemented global registration.

References

[1] Raouf Benjemaa and Francis Schmitt. "Fast global registration of 3D sampled surfaces using a multi-z-buffer technique". In: *Image and Vision Computing* 17.2 (1999), pp. 113–123.

[2] Timothee Jost and Heinz Hügli. "A multi-resolution scheme ICP algorithm for fast shape registration". In: Feb. 2002, pp. 540–543. ISBN: 0-7695-1521-4. DOI: 10.1109/TDPVT.2002.1024114.

[3] Peng Li et al. "Evaluation of the ICP algorithm in 3D point cloud registration". In: *IEEE Access* 8 (2020), pp. 68030–68048.

[4] Graeme P Penney et al. "A stochastic iterative closest point algorithm (stochastICP)". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2001, pp. 762–769.