Iterative Closest Point for 3D Model Construction from Depth Camera

Bram Kooiman 11415665
Zvonimir Dujmic 11029935

April 2018

1 Introduction

Iterative Closest Point (ICP) algorithm finds the transformation between two point clouds that minimizes the Root Mean Squared Error (RMS). We will use this algorithm to construct a 3D model of a person from a collection of photos taken with a depth-camera.

Per iteration, ICP determines what points in the target cloud lie closest to the points in the base cloud and finds the transformation that brings these points closer together with Singular Value Decomposition (SVD) of the covariance matrix. The iterative process continues untill RMS does not notably decrease further.

We evaluate a few optimisations of ICP and apply the most appropriate one for 3D model construction. We want to find the optimisation of ICP that makes it available for real-time use (minimum RMS for convergence time under .5 seconds) and an optimisation that balances performance and speed best.

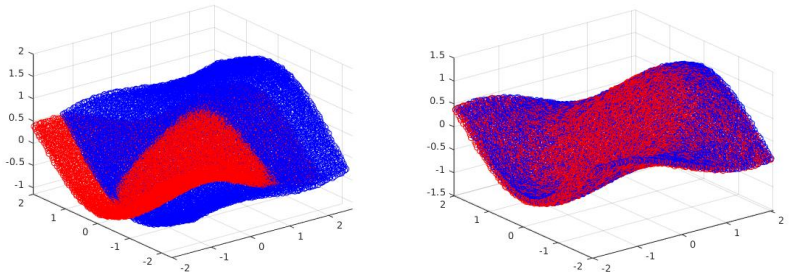

Figure 1: Alignment by ICP (unoptimised)

2 ICP optimisation

In direct ICP, all points in the base cloud are mapped to all points in the target cloud. Given that a cloud can contain over sixty thousand points, this may require some computational time. The process could be optimised in a number of ways. We discuss optimisation by randomly sampling a subset of points from the cloud, either once before all iterations (uniform sampling) or before each individual iteration (random sampling). We also discuss sampling from points in a way that makes sure we find a set of points that represents all surface orientations of the cloud (sampling by surface normals). In conclusion, we consider the following ICP algorithms:

- Unoptimised ICP
- Uniform sampling
- Random sampling
- Uniform sampling over surface normals
- Random sampling over surface normals

We discuss the performance of each of them based on time to convergence, RMS at convergence, stability (dependable results) and tolerance to noise.

2.1 Optimisions of ICP

- Uniform Sampling.
  The ICP algorithm is performed on a subset of the point cloud. Both the base cloud and the target cloud are downsampled. This reduces the required computing time for finding the closest point in the target cloud for each of the points in the base cloud. If the orginal cloud is dense enough, the surface will still be recognisable from the downsampled cloud.

  However, it can be expected that the final fit has a higher RMS, indicating a poor fit. It may even happen that the point clouds don't align properly for even a low RMS, because unlucky sampling may have selected a cloud that does not represent the original cloud well.

  This is a typical trade-off between performance and speed, controlled by the parameter p that specifies the relative size of the subsample to the original cloud.

  "Uniform Sampling" can be interpreted in two ways. It can either mean sample 'uniformly from the surface' - which means you take a point and skip a few according to the geometry of the surface in a regular fashion - or it can mean 'sample from a uniform distribution', which means that all points have the same probability to be drawn, but their geometric location is ignored. Our interpretation is the latter.

- Random Sampling.
  As opposed to uniform sampling, random sampling takes a subset of the original cloud for each iteration. This may help to mitigate the problem of an unlucky initial sampling, but it may also require more computational effort because the sampling step has to be performed more often.

- Uniform Sampling over Surface Normals.
  The idea of sampling over surface normals is based on the way that a aligning two 2D surfaces can be ambiguous. It is those regions where there is local texture and shape that can help to disambiguate. If we sample points directly we might accidentally filter out the informative points. An excellent example given in [1] considers a mostly 2D surface with grooves for which convergence would be cumbersome if those grooves are accidentally sampled out. For each point we determine the surface normal by approximating the surface with the four nearest neighbours.

  Surface normals are transformed to spherical coordinates and sampled in a 2D histogram based on their azimuth and elevation. The bins are sorted smallest to largest by the amount of surface normals that fall within them. Sampling is then performed over smaller bins first. The algorithm tries to sample equally many points from each bin. If a bin is too small to accommodate the need the algorithm adjusts itself to sample more, yet still equally, from the remaining bins. This makes sure the resulting subsample represents a distribution where all surface normal directions are present as equally as possible.

- Random Sampling over Surface Normals.
  Surface normals are sampled every iteration, instead of only once. This is done as efficiently as possible by first establishing a histogram sorted on the amount of normals that fall within a bin only once. Every iteration samples from this sorted set, but the transformation from cartesian to spherical coordinates, the discretisation and sorting need only be performed once.

2.2 Performance of Optimised ICP

Common practice is to measure speed of convergence by the amount of iterations required to converge, because it's a more stable measure than time. However, the amount of computational effort that an iteration requires varies widely over the algorithms considered here. Instead we choose to measure convergence by time after all. Time can be affected by many things apart from the algorithm (hardware, background processes, temperature, etc...). We make as fair a comparison as possible by performing all runs on the same pc and averaging the performance over 10 independent runs. Convergence is defined as a change in RMS smaller than 1% between two consecutive iterations.

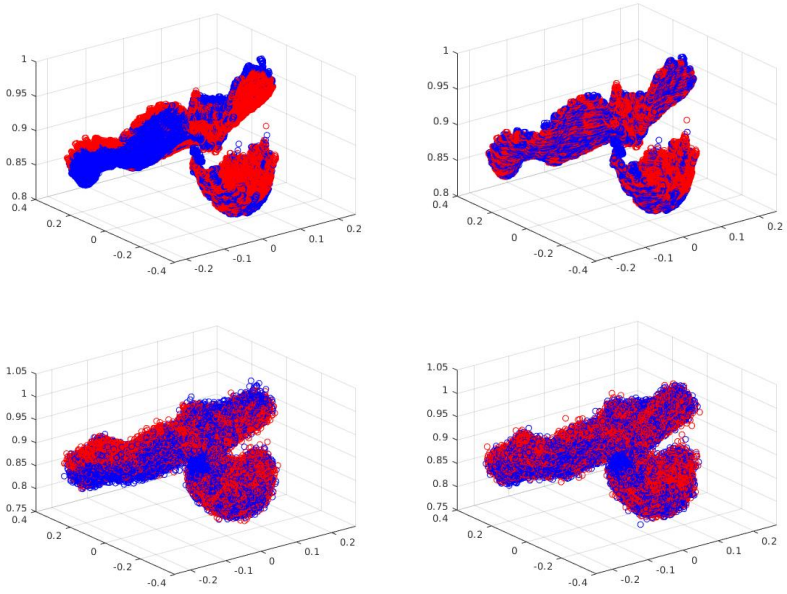

Figure 2: alignment (left to right) for a case free of additive noise (top) and with 3D gaussian noise with $\epsilon = .01$

In order to test whether the algorithm is tolerant to noise we add a 3D gaussian noise over the xyz coordinates of the points in both clouds (this in turn influences the surface normals, so they need to be recalculated). In figure 2 we can see cloud alingment for a (additive-)noise-free and noisy case.

Sampling by surface normals requires a set hyperparameter for the amount of bins per azimuth and elevation. Table 1 shows the experiments on which we based the decision to set this hyperparameter to 10. Stability, performance and tolerance to noise are comparable over all values of n_{bins}, though time to convergence seems to increase with it. For a small binsize we see that the time to convergence actually decreases for a noisy case. We pick a binsize for which the performance is dependable under noise, $n_{bins} = 10$.

ϵ	n_{bins}	it_{conv}	$time_{conv}$	RMS_{conv}(e-3)	$time_{std}$	RMS_{std}(e-5)
0	4	8.9	0.76	4.8	0.05	10.
.01	4	8.6	0.71	7.2	0.08	6.
0	**10**	**8.9**	**0.75**	**4.9**	**0.05**	**7.**
.01	**10**	**8.7**	**0.75**	**7.3**	**0.08**	**7.**
0	20	8.9	0.80	4.8	0.03	5.
.01	20	8.6	0.80	7.3	0.10	10.
0	40	8.6	0.96	4.9	0.05	4.
.01	40	8.4	1.03	7.3	0.09	9.

Table 2: Performance of ICP optimisations. The setting that we think is best for a real-time application and the settings we'll be using in the next sections are highlighted

ϵ	sampling	sample by	p	it_{conv}	$time_{conv}$	RMS_{conv}(e-3)	$time_{std}$	RMS_{std}(e-5)
0	unoptimised	default	1	15	121.	2.3	0	0
.01	unoptimised	default	1	17	135.	3.6	0.84	0
0	**uniform**	**default**	**0.2**	**13.4**	**3.1**	**3.3**	**0.11**	**5.e-05**
0	**uniform**	**normals**	**0.2**	**11.3**	**2.7**	**3.9**	**0.08**	**2.e-05**
0	uniform	default	0.1	11.3	0.89	4.1	0.08	11.
.01	uniform	default	0.1	8.4	0.66	7.2	0.07	8.
0	uniform	normals	0.1	8.9	0.72	4.8	0.04	6.
.01	uniform	normals	0.1	8.7	0.72	7.3	0.06	5.
0	**uniform**	**default**	**0.07**	**10.9**	**0.43**	**4.7**	**0.07**	**10.**
.01	**uniform**	**default**	**0.07**	**7.3**	**0.30**	**8.2**	**0.02**	**7.**
0	uniform	normals	0.07	8.1	0.34	5.4	0.02	7.
.01	uniform	normals	0.07	7.9	0.35	8.1	0.04	8.
0	uniform	default	0.04	8.1	0.11	6.0	0.01	17.
.01	uniform	default	0.04	6.4	0.09	9.6	0.01	11.
0	uniform	normals	0.04	7.9	0.13	6.6	0.01	11.
.01	uniform	normals	0.04	6.4	0.12	9.6	0.01	12.
0	random	default	0.1	56.8	4.5	4.1	2.8	6.
.01	random	default	0.1	20.8	1.7	7.3	1.1	6.
0	random	normals	0.1	30.9	2.5	4.9	1.8	3.
.01	random	normals	0.1	17.9	1.5	7.3	1.2	4.
0	random	default	0.07	70.9	2.9	4.6	4.6	8.
.01	random	default	0.07	13.2	0.5	8.1	0.25	6.
0	random	normals	0.07	36.4	1.6	5.5	2.2	6.
.01	random	normals	0.07	14.1	0.62	8.1	0.45	7.
0	random	default	0.04	38.6	0.64	5.9	0.60	11.
.01	random	default	0.04	19.1	0.30	9.7	0.17	8.
0	random	normals	0.04	28.5	0.50	6.6	0.44	8.
.01	random	normals	0.04	18.7	0.36	9.6	0.35	11.

Table 2 shows the performance, stability and tolerance to noise of optimisation methods for various settings. We see that low values of p converge faster, but to a worse alignment (higher RMS). All runs have a stable RMS_{conv} whose standard deviation is approximately one hundreth of the RMS. Time is stable for uniform sampling, but has a higher standard deviation for random sampling. Moreover, random sampling takes more time in general without an increase in performance. We think this is because on every iteration the sample is different, so it chases a different goal each step.
For a real-world application we'd want to minimise converged RMS in safely below .5 seconds. $p = .07$ seems to be the best choice. Sampling over surface normals does not appear to have a significant benefit in this test, so we pick sampling over points to be the best optimisation for a real-time application.

The task considered in the next section does not need to be performed in real-time and we can assume the point clouds to have a level of noise corresponding to $\epsilon = 0$. We pick the parameters that provide the best trade-off between speed and performance without the hard boundary of .5 seconds to convergence; uniform sampling with $p = .2$. We'll be experimenting with sampling over points as well as sampling over normals.

3 Merging Scenes

The previous section describes how to use the ICP algorithm to transform one point cloud as close to another point cloud. In order for this algorithm to be applied for the construction of a 3D object, it is necessary to connect multiple point clouds from different angles to a single point cloud that will faithfully represent objects from reality. For this purpose, we experiment with a few merging algorithms described in further texts.

The first way of merging scenes (according to assignment 3.1) is to take each two consecutive frames of given data and then find transformation matrix that maps the previous frame to the next. The transformation is then applied to the full 3D model, which consists of all previously processed frames. Its results are shown in figure 5. (figures are sorted from worst to best).

Another way of merging (according to assignment 3.2a) is to map the entire current datacloud to the next incoming frame by finding the transformation directly between them. In that case we sample points from the entire model processed so far, not just from the previous frame. In order to speed up the process, we take every other frame instead of every frame (1st, 3rd, 5th, etc...). Its results are shown in figure 3.

A good thing to check is if the merge will be better if we do use every frame (merging according to assignment 3.2b). Its results are shown in figure 4.

Originally, we would have expected the 3.1 method to be more unstable than

the 3.1 methods. If a single frame of the image is noisy or converges badly, the faulty transformation is applied to all frames. If this happens a few frames in a row the errors will accumulate without having a chance to cancel each other out. The intuition is then that if we sample from all previous frames (like in the 3.2 methods), a bit of noise will not be problematic, because the slow drift of the model will be countered by the weighting of the earlier frames. At least, so speaks the intuition. From the results we can see that the 3.2 models are a poorer alignment than the 3.1 model. Our sense is that for front-facing and rear-facing depth clouds the large flat surfaces on the back and chest want to align, whereas there would have been a more realistic model the shoulders aligned better. By using consecutive frames there is no 'chest' to inappropriately converge to when we are considering target clouds that map the back of the person.

Although table 2 shows worse performance scores for sampling by normals, the 3D model does appear to align better with this method. Its results are shown in figure 6.

For a more comfortable view of the pointclouds the reviewer is invited to run the appropriate section of 'main.m'. Constructing the 3D model can take a few minutes. The process can be sped up by setting the p-value in 'merge_scenes.m' to a lower value, but it is discouraged as the resulting model will be worsened by it.

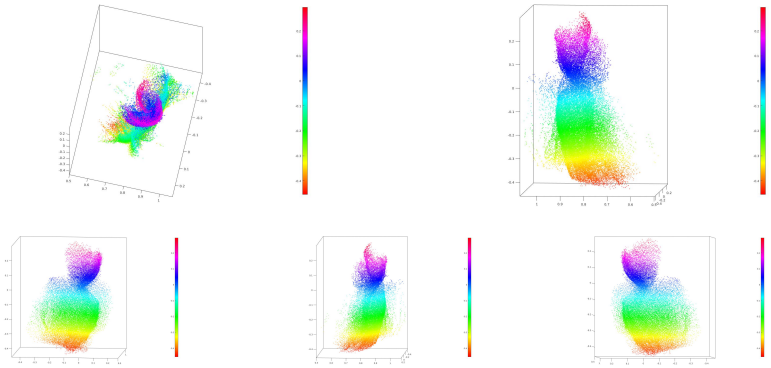

Figure 3: 3D model obtained by sampling points from all previously processed frames, using every other available frame (method according to 3.2a). The model shows 50000 points, approximately one fiftieth of the total points in the cloud. The top view shows that the back and chest sections want to align, at the expense of proper shoulder alignment. This bad alignment makes the side views hard to read.

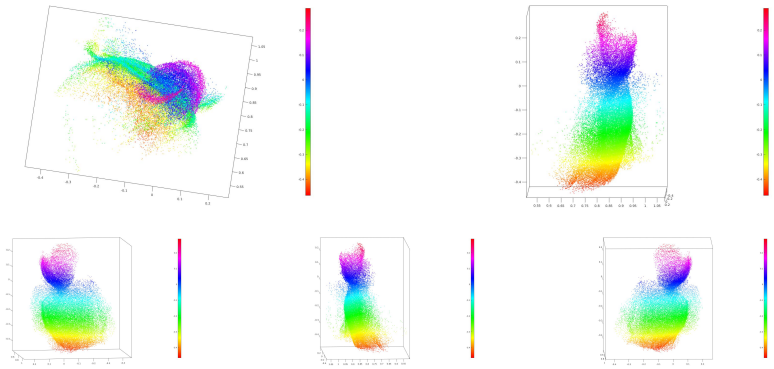

Figure 4: 3D model obtained by sampling points from all previously processed frames, using all available frames (method according to 3.2b). The model shows 50000 points, approximately one hundreth of the total points in the cloud. The model shows no improvement over the 3.2a method, and can even be argued to be worse

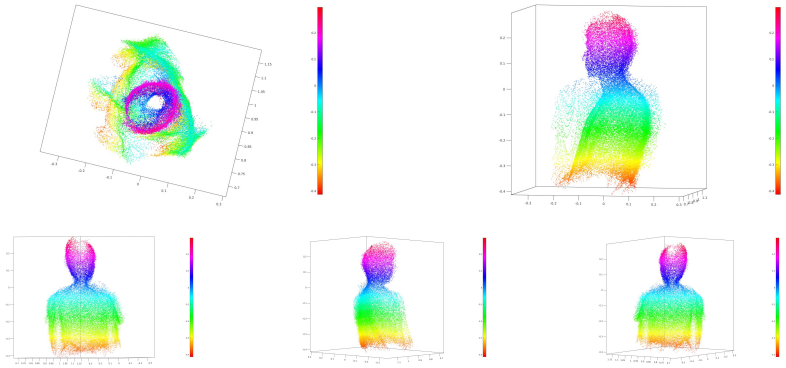

Figure 5: 3D model obtained by sampling points from consecutive frames, using all available frames (method according to 3.1). The model shows 50000 points, approximately one hundreth of the total points in the cloud. Though the side views seem fair, the top view shows that the model is not fully realistic.

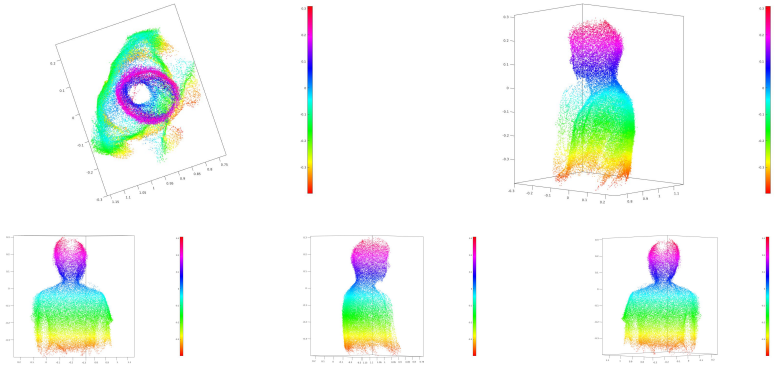

Figure 6: 3D model obtained by sampling points from consecutive frames by surface normal direction, using all available frames (experimental method). The model shows 50000 points, approximately one hundreth of the total points in the cloud. The top view shows that the model is not perfect, though it seems the best one yet, judging by the sharp modeling of the back surface.

The fact that we find that the best algorithm on this task is not the same as the best algorithm found in table 2 may be due to the fact that those performance scores were found on the transformation from the first to the second frame only. This transformation may not have been representative for the task at hand or for ICP in general.

4 Drawbacks

ICP only converges if the 'initial guess' is already close. Likewise, if ICP gets stuck in a local optimum it cannot infer that it is not the global optimum.

If you would try to align a small patch to a larger cloud it would certainly converge to an alignment where the smaller patch is in the center of the cloud, even though it might fit better on, say, the corner. The clouds to be compared have to be on the same scale.

A quite general drawback; there is a trade-off between speed and performance where we cannot have best of both worlds. If we'd want to have a real-time application, it will not perform as well as it could. If we want the best performance there is, we have to sit tight to wait for convergence.

The problem with the 3.1 method may yet be that errors are allowed to accumulate, which would not be the case for the 3.2 method. However, the 3.2 methods tend to wrongly align the back to the chest surface. It seems to be a choice between the lesser of two evils, though we'd really want to circumvent both.

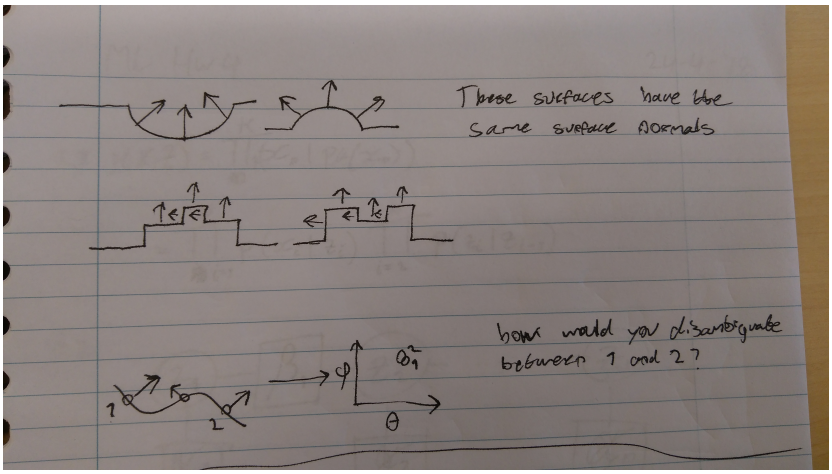

Figure 7: Skepticism on the rotation by normals approach

5 Ideas

As mentioned before, the words 'uniform sampling' can be interpreted in two ways. We interpreted it as 'sampling points from a uniform distribution', but it may be more beneficial to sample regularly according to geometry.

Like mentioned before, the 3.1 method may allow errors to accumulate, but the 3.2 methods wrongly align the back to the chest surface. It may be worth considering a batch method where points are sampled from the past few frames (say 4). This way, there is more data to prevent error to accumulate, but not so much that the chest will be present when the back is being modelled.

In an earlier section we described an approach that makes use of surface normals. An idea is to make use of surface normals in a different way in order to find the rotation matrix R. If we represent the surface normals as a point cloud we would get sort of an unequal distribution of points on a sphere. We could perform ICP on two point clouds in this space to find R. Translation could be found by evaluating the centroids of the point clouds. Some reservations to this approach may be had. Like the questions posed in Figure 7, some highly different surfaces have similar surface normals, and some points that are very far away in their geometry are close in normal-space (in spherical coordinates θ and ϕ). However, the intuition remains that the two spheres share densely or thinly populated areas on their surfaces that can be aligned. It may be worth an experiment to see whether it is a viable method.

Accuracy may be improved by de-noising the point clouds before ICP. De-noising can be done by interpolating the positions of the nearest neighbours and have the updated position lie somewhere between the original measurement and the position as voted by the neighbours. RMS would certainly decrease, because the surfaces considered are smoother. A visual inspection would have

to conclude whether the resulting 3D model is also better for it.

References

[1] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

6 Appendix: Self-evaluation

The intro to the 3rd chapter is written by Zvonimir, the Drawbacks and Ideas chapters are written in tandem and the rest is written by Bram. The idea of using normals as we implemented came from Bram, but the idea of how we could also use normals (described in the Ideas-section) was by Zvonimir. Preprocessing of a datacloud (removing background) was written by Zvonimir. The code that runs and tests ICP and its optimisations was written by Bram. Zvonimir tried his hand at constructing 3D models using the ICP algorithm, but was not able to finish. Bram wrote the code to merge scenes into a 3D model as well. The depracated code is included in the hand-in, but it is not required to review it.