Computer Vision 2: Iterative Closest Point

Gabriele Bani
11640758
gabriele.bani@student.uva.nl

Andrii Skliar
11636785
andrii.skliar@student.uva.nl

Andreas Hadjipieris
11730064
andreas.hadjipieri@student.uva.nl

1 Introduction

In this report, we will explore the capabilities of the Iterative Closest Point (ICP) algorithm. The ICP algorithm will try to find the best transformation matrix of a point cloud to another point cloud. This procedure makes a final point cloud that contains all the point clouds merged together. A spectrum of versions of the algorithm were implemented ranging from the very basic brute-force method to more advanced methods like iterative sampling. In addition more intuitive approaches have been made so we can deviate from brute force as much as possible. Weighting of the points in respect to their contribution on the surface as well as clustering the points from informative regions, this algorithm has many possible extensions. We explore the capabilities of some of these extensions.

2 ICP

2.1 General ICP

In this section, we describe the ICP algorithm we implemented. In general, the ICP algorithm takes as input a source point cloud and a target point cloud, and outputs a rigid transformation ϕ (rotation + translation) which minimizes some matching error between the source and the target point cloouds. In our case, the error is the RMS error, which is calculated from all point of the source to their nearest neighbors in the targets. After every iteration, the source points get transformed to so that the RMS decreases, and after the algorithm has finished running, the outputs are both the set of transformed points and a transformation from the source coordinates to the new coordinates.

The algorithm, after a first initialization of the rotation matrix and translation vector, iteratively applies the following two steps until convergence (which is detected in our case, by a fixed maximum number of iterations or a change in the RMS smaller than a certain threshold). The first is finding the closest point between each point in the source from the target. The second is finding a rotation matrix and translation vector which minimize the distance between the points of the two sets. This is done in our case using SVD decomposition. By applying those two setups iteratively, the algorithm is guaranteed to converge monotonically to a local minima, since at every SVD the RMS diminishes, and at every iteration the points are transformed using the newly found transformation. At every iteration, the source points are transformed using the rotation and translation matrices found in the same iteration, and it is kept track of the total transformation needed to transform data in the original source space to the current transformed space. These total transformation are what will be used in later sections to transform the data from one camera coordinate system to another.

2.2 Fast closest point with KDTrees

For point matching, we rely on KDTrees, since they provide a very fast way to find the nearest neighbors between two sets of points, while also calculating their distances. In the basic version

Table 1: My caption

	Full Mean	Full std	Iterative Mean	Iterative std	Uniform Mean	Uniform std
Run 1	8.082	1.342	4.039	0.771	3.790	0.967
Run 2	8.180	1.467	3.974	0.700	5.340	0.967
Run 3	8.020	1.425	3.922	0.613	3.757	0.600

of the algorithm, we one tree, which stores data of the full target matrix, which is used to calculate errors. When sampling, we also use another tree, which indexes only the target points which have been sampled.

2.3 Rigid transformation

To calculate the rigid transformation, we rely on the description of (cite 1). We introduced some small modifications, however. The main change is that the calculation of the matrix R is not done by VWU^T, but instead we first compute $X \odot w$, where w is the diagonal of W (which is a diagonal matrix), so that we save one expensive matrix multiplication and reduce the effective complexity of the calculation or R. Notice also that in our implementation, the data matrix has every point as a row, so many operations are transposed respect to the original description.

2.4 Preprocessing

As a side note, we would like to mention some little preprocessing steps that we did before running the algorithm. Since many of the point cloud description files contained points from the background, we filtered the data such that those would be removed. In particular, we filtered every data point with $z > 1.7$. This resulted only in a few outliers overall. Then, we generated c̈leanf̈iles, which are the actual input to our files. Additionally, we make use of normals of points in an outlier detection strategy, so we also clean normal files correspondent to the point cloud files. The normal files are created from the data points using a matlab script.

2.5 Sampling

Trying to match all the points of the source and the target can clearly be a poor choice in a number of cases. First of all, the data might be too large and prevent the algorithm to run as fast as needed. The most simple form of sampling is uniform, which selects with uniform probability n data points at the beginning of the ICP call. Alternatively, in order to avoid extreme cases in which a particularly bad set of points is chosen at the beginning, we can sample again at each iteration of the ICP. In our code, we call this method ïterative ·̈ Notice that for simplicity, we decided to sample from either no or all the two point clouds.

2.6 Comparison of sampling methods

We compare now the three different sampling methods described quantitatively, in terms of error and time. We use a sampling size of 5000 datapoints. As error, we consider the RMS between every two consecutive frames, and plot it in ffig. 1. For these first experiments, we run aways 30 iterations of ICP per frame, because we want to compare whether the total error and reconstruction error change with different sampling methods. What we find is not what we expected: all the sampling methods produce almost the same errors through the process, and only differ in the amount of time needed to converge. Since the number of iterations was fixed, this means that the size of the data (which is larger for full experiment), plays an important role in the running time. The results displayed are averaged over three runs (with different seeds), and we can observe that for this data, the sampling method has indeed no big impact. In fig. 2 we can also see 2d projection of the obained 3D reconstruction, which looks satisfying but is not consistent on the head and he arms of the person.

Figure 1: The errors for Full, uniform and iterative sampling. The graph shows the errors for the transformation from one frame to the next

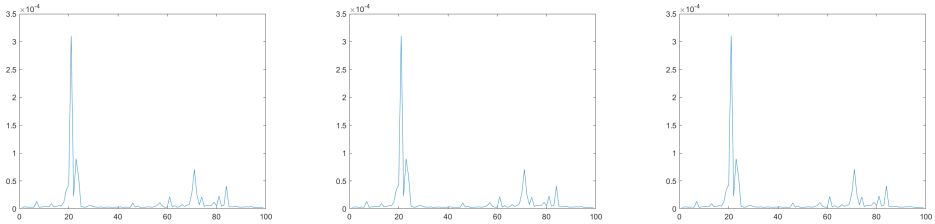
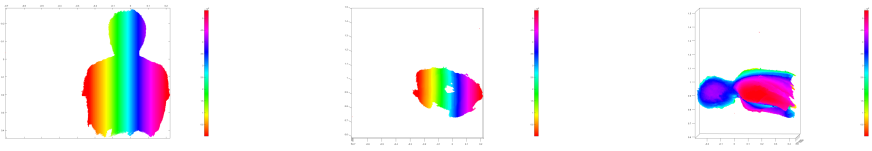


Figure 2: X and Y axis projection of the point cloud (left). X and Z axis projection of the point cloud (middle). Y and Z axis projection of the point cloud(right).



3 Merging Scences

Here, we would like to discuss the approach to merging frames in order to get final 3D-reconstructed figure. First, we would like to present the implementation and, afterwards, discuss the results of different approaches.

3.1 Implementation

We have utilized previously discussed ICP implementation to estimate camera poses between two frames and merge them afterwards. In our implementation we have two main approach to estimation of camera poses:

1. Consecutive - given step size (i.e. 1, 2, 4, 10), we find transformation between two consecutive frames with specific step (i.e. each 2nd frame). Afterwards, only frames, for which pose has been estimated will be merged.
2. Iterative - pose is being estimated between each two consecutive frames (i.e. f_1 and f_2). Afterwards, they are merged and pose between merged frame (f_{12}) and next frame (f_3) is calculated. This goes on until the final frame.

During pose estimation, we are building a data structure, which will be used for efficient merging. It looks as following for frames from 1 to 4: $[f_1, (f_2, t_{1\to2}), (f_3, t_{2\to3}), (f_4, t_{3\to4})]$. Here, f_i stand for a frame i and $t_{i\to i+1})$ stands for a transformation between frames i and $i+1$. **Note**, that depending on the chosen approach (either consecutive or iterative), we get different transformation matrices in each item of the list. Also, using such list allows us not to specify step size during the actual merging as we have all the data stored in that list. During merging, we use functional programming approach and apply reduce function to the described list. At each step, we apply transformation to the accumulator and concatenate next frame with the transformed accumulator. It can be visualized for first three frames as following:

- $Acc := f_1$
- $Acc := (t_{1\to2} @ Acc) + + f_2$, where @ stands for multiplication between homogeneous transformation matrix and source point cloud, $++$ stands for concatentation between two point clouds.

Figure 3: X and Y axis projection of the Threshold point cloud (left). X and Z axis projection of the Threshold point cloud (middle). Y and Z axis projection of the Threshold point cloud(right).

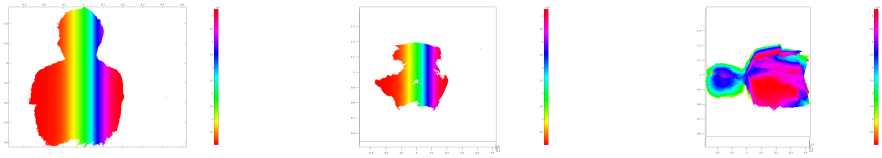
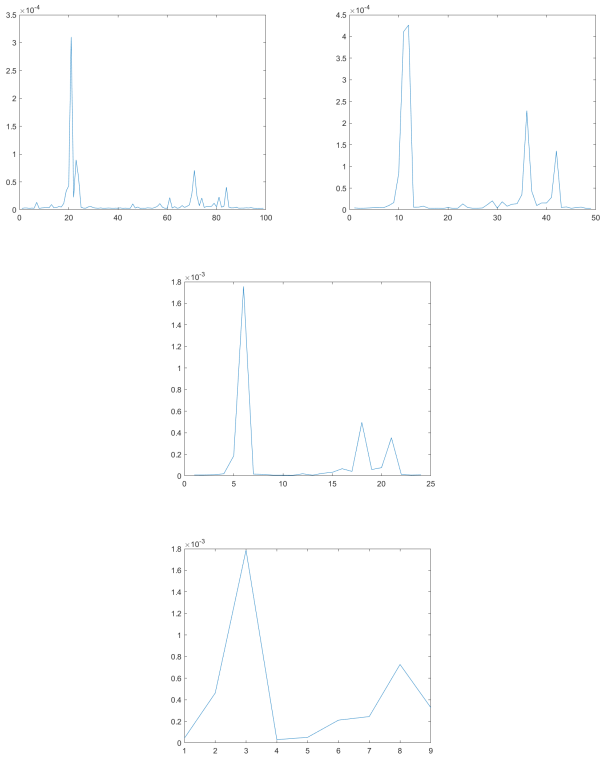


Figure 4: Top-Left: step 1 error, Top-Right: step 2 error, Bottom-Left: step 4 error, Bottom-Right: step 10 error,



- $Acc := (t_{2\to 3}@Acc) + +f_3$, where @ stands for multiplication between homogeneous transformation matrix and source point cloud, $++$ stands for concatentation between two point clouds.

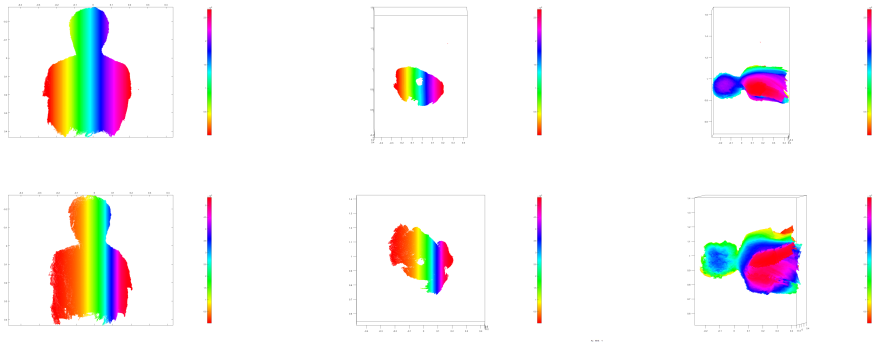
In the end of the reduce function, we will have all the point clouds transformed using corresponding transformation matrices and merged together.

3.2 Results and discussion

3.2.1 Consecutive approach

We tried this approach with various number of frames and have concluded, that the actual result highly depends both on the data and the actual step size between frames for which pose is being estimated. As in our case consecutive images are pretty close to each other and therefore correspond to the similar parts of the final object, we can see that though using step size of one (so estimating transformation for each pair) works best, running the same algorithm with step size of 2 still gives roughly similar, but worse, results fig. 5.

Figure 5: Left: step 2 projections, Right: step 10 projections



However, if we run it with step size of 10, results produced will be pretty bad as we are missing a lot of features of the actual object. Therefore, in the final 3D-reconstructed image, we can see that the data points in final point cloud are quite sparse and can't properly reconstruct the real object.

Note, though, that if the frames are even more similar (i.e. consecutive frames correspond to practically the same part of the object), better reconstruction is possible as the transformation matrix to be estimated can be much simpler than when two frames are far from each other in world coordinates. Also, in that case even higher sampling rate can give sufficient results. Also, possible improvement can be done to handle the error accumulation. As there might be some errors when estimating transformation matrix between each pair of consecutive frames, when erroneous transformations are applied to the same frame, final result might have very large error compared to the ground truth. Possible solution would be to apply less transformations to each frame as then error accumulated would be lower. We have utilized this idea by running scene merging between two halves of the frames. More specifically, instead of calculating $\{t_{1\to2}, t_{2\to3}, \dots t_{99\to100}\}$, we calculate $\{t_{1\to2}, t_{2\to3}, \dots t_{49\to50}\}$ and $\{t_{100\to99}, t_{99\to98}, \dots t_{51\to50}\}$ and merge the resulting point clouds to get final reconstructed object. However, for us this approach didn't show any visible differences.

3.2.2 Iterative approach

We have tried to run iterative approach to scene merging, however, we found it to be very inefficient and therefore decided not to compare it to the previously mentioned results. The reason for it was that as it has to find transformation between next frame and all the previous frames merged, ICP gets more and more computationally expensive with each frame. This happens because of the KD-Tree, that we use for finding correspondence between points in source and target frames. As the number of points that have to be compared increases, so does the complexity of the search.

Also, we expect the results to be much worse using that approach as we are trying to match a lot of points, that are contained in source point cloud, with much smaller (in terms of number of points) target point cloud, which would mean, that we can minimize the error by projecting only a subset of the source point cloud correctly. However, this might imply that the result will be much worse for all other datapoints, so, in the end, we can get very low error while still having transformation matrix incorrect. These assumptions are made under the consideration that the difference between two consecutive frames is small, so, if we try to transform merged point cloud, we might get points from frame, which is very far from the current frame (i.e. f_1 and f_{50},) transformed correctly, but this would result into very bad transformation in general as the error got accumulated over 50 frames.

4 Questions

1. What are the drawbacks of the ICP algorithm? Note, that we are discussing vanilla ICP here, which doesn't have any improvements and uses simple rigid motion computation. The main drawbacks of such ICP algorithm are following:

- Pure ICP is incapable of handling outliers. The algorithm relies in fact on the assumption that the points in the source and the target correspond, but this might not be the case. Very simply, one frame might contain points that even when transformed, do not correspond to points in the target because they are simply unique to one view. Also, the data might contain outliers due to the precision of sensors. Outliers typically have a higher distance to their nearest neighbor, since there should be no match for them. This fact can be harmful if outliers are not removed,, but it can also be used to detect the outliers themselves and remove them from the set of points actually used for calculating the transformation. Notice that outliers are not removed entirely from the data-set, because we still want to project them and see them in other coordinate spaces.
- Vanilla ICP is sensitive to initialization, and it suffers from the fact that it cannot escape local minima, which makes the initialization a crucial step.
- Rigid motion computation only uses rotation and translation and is therefore sensitive to scaling. If we have different scaling between two objects, that we are trying to match, vanilla ICP would not be able to match them correctly as it would have to re-scale objects to match them properly, which it can't. This results in having the assumption that the distance between the camera and the object we map is constant. A lot of work has been done to add the scaling factor in the ICP algorithm[5] [6] [4].
- Finally, it is very clear that the global error accumulated over many time-frames can lead to poor 3D reconstruction. When we run the algorithm with only a few frames, the results are typically the results are typically visually better than the one fully reconstructed. It may happen in fact, that if even one transformation from one frame to another fails to be learned correctly, all frames that would be projected in the target space using that transformation will be negatively affected by it.
  One simple possible solution would be, in our case, projecting all the images in the coordinate space of the 50th frame. However, this did not lead to visible improvements in our case.

2. How do you think the ICP algorithm can be improved, beside the techniques mentioned in linked papers, in terms of efficiency and accuracy?
   - **Outlier detection:** A simple way to detect outliers can be a fixed threshold on the distance between point matches. When this is higher than a certain threshold, then we do not use those points for calculating the rigid transformation. This may result in poor results since it is not clear how to choose an effective threshold without trial and error. Another simple alternative can be rejecting a certain fixed percentage of matches, the one with highest distances. In this case, we might either remove points which actually add a small distance to their neighbor, of even fail to remove outliers when they are present in more than the percentage specified. Finally, points can be rejected by exploiting information about the normals of the points, by fixing a threshold which represents the maximum dot product between the two points of a match.
   - **Weighting points** Another way to improve the efficiency of ICP is weighting point matches in a way such that points in more interesting regions are given more weight when iteratively finding the best transformation from one point cloud to another. One idea that we implemented is the one suggested in [7], where weights are defined as $w_i = 1 - d(i)/d_{max}$, where $d(i)$ is the distance of the i^{th} match, and d_{max} is the maximum distance value between the matches. Since the fraction assumes values from 0 to 1 (from closest to furthest), the whole weight will be zero for distances close to the maximum one, and closer to one for smaller ones. This is a desired effect: we are giving more importance to points close to each other, which is likely because they are actual matches.
     Another way we weight points is using normals. In this case, we just assign as weight of a datapoint the dot product between it an its nearest neighbor in the target set.
     With weighting, one can also obtain the same effect as not considering points at all for the calculation of the transformation. However, in practice typically weighting works well when there are lots of points, and the improvement in results seems to be dependent on the data.
   - Parallelizing the ICP algorithm . As described, parallelizing can be very helpful in terms of speed, but it can also help in terms of accuracy by running many times the algorithm with different initialization. This is to minimize the dependency on the

initialization, and on average this would lead to better transformations than just running the algorithm once.

- Adding the scaling factor in various ways [5] [6] [4]. These techniques range between the spectrum of updating a scaling factor at each iteration, or the iterative algorithm with the SVD algorithm and the properties of parabola incorporated to compute the translation, rotation and scale transformations at each iterative step.
- This paper[1] introduces a new parameter to ICP which describes how much deformation should the source have to reach the target. With the addition of the stiffness parameter, the convergence of the ICP can be achieved much faster and accurately by deforming the surface of the point clouds according to this parameter to get closet to the target iteratevely. This extension also works for a bigger spectrum of solutions since its not using only the rotation-translation transformation matrix.
- Sub-setting points in the point cloud [3]. It searches for the corresponding pairs on subsets of the entire data, which can provide structural information to benefit the registration. A simulated annealing combined with ICP scheme is used to improve the registration accuracy, at the same time, reduce the computation cost.
- One interesting idea that we have tried was to replace minimization of error of the ICP algorithm with a number of PID correction controllers. The intuition behind this idea was that we have a series of rotation and translation matrices consisting of 6 variables (which represent the 6 degrees of freedom given to the algorithm to move so it can reach its target), and which represent the translation and rotation in each of the 3 axis. We saw the task as a game that a player has a controller with 6 knobs and tries to figure out the best combination of these to "move" his surface around the 3d space. The PID loops, one for each degree of freedom would try to minimize its specific error, which for example the error for the PID responsible for the x-Translation would be the difference between the x-axis of the two point clouds. We have tried to choose meaningful errors for each degree of freedom to make the loops affect each others performance the less as possible. The drawbacks of this approach is that one has to manually tune the PIDs for the current 3D space. Fortunately this procedure needs to be done only ones if we know our environment correctly, however it can be very time-consuming since the best combination of 6 PID loops needs to be found. A lot of work has been done to automatically tune the the parameters of a PID loop [8] [2] but we could not report results since time was limited. Our implementation showed promising results with the source oscillating on the target with smaller and smaller steps. What made us choose this algorithm for this task is the fact that is supposed to be scalable. If the results on the 6 degrees of freedom were good enough, and we had enough time we would have added 3 more PID loops to model the scaling.

5 Additional improvements

5.1 Outlier removal: results

We now compare quantitatively and qualitatively the results obtained with different outlier removal methods. As baseline, we continue to use the standard ICP without sampling. To recap, we have experimented with three methods:

- Threshold: we fix a threshold of 0.002, and do not consider the points whose nearest neighbor in the target is further than the threshold
- Variance: similar to threshold, we do not consider points whose distance from the nearest neighbor is more than 2.5 times the standard deviation of the match distances

We can see that the error graph for threshold is slightly different from the graphs we obtained before. If we look at the qualitative result of the 3D reconstruction in fig. 3, we can see that indeed the result is different from the previous ones, and it is worse. This is most likely due to a poor choice of the threshold. The thresholding method is very harsh, and we can observe that the time needed to converge is actually much higher than with other methods. One possible interpretation of this could be that the threshold is so invasive that it deletes points which should have been matched, and thus interferes with the convergence. On the opposite, variance method shows promising results, at least in terms of speed. In terms of qualitative results, it is almost the same as the original method.

Table 2: Results for Outlier removal and weighting. We show mean and standard deviation over the 100 frames in one full ICP run (i.e, for Baseline, every transformation from one frame to another took 13.45 seconds on average)

	Baseline	Threshold (Outliers)	Variance (Outliers)	Weighting	Normals (weighting)
Mean (s)	13.45	18.85	10.11	8.13	8.02
Std (s)	4.28	12.01	3.85	2.19	2.37

5.2 Data point weighting: results

We report results on one method of data weighting, described in the previous section, which assigns weight to data points as $w_i = 1 - d(i)/d_{max}$. Also, we experiment on weighting using the normals, which means that we assign as weight of a point the dot product between the point and its nearest neighbor. With a better selection of points, we expect faster convergence. Indeed, as seen from table 2, the running time is lower than with other methods. We can conclude then that from our experiments, weighting methods allow faster convergence without harming the qualitative aspect of the solution found.

6 Conclusion

We have implemented the ICP algorithm and different improvements for it, and discussed its strong and weak points, along to other possible improvements. We have run different version of the algorithm, and discovered that in our case, factors such as sampling type were of little importance for the convergence of the algorithm. Although we found that the running time when sampling was lower, the number of iterations was fixed, which means that the decrease in running time is due mostly to the size of the data. Surprisingly, both quantitative measures such as the approximation errors and qualitative ones such as the shape of the reconstructed point cloud, showed almost no difference while changing sampling method. This might mean that even uniform sampling maintains most of the information about the data in our case.

About the data, one very remarkable result that emerges from the experiments is that there seem to be a pattern in the reconstruction errors of the frames. In all error graphs, we see a high peak around the 20th frame, possibly indicating that the 20th frame is very noisy or anyway different from it neighbors, which is very unusual. If we treat as a quantitative measure the sum of the reconstruction errors between adjacent frames, then this is dominated my the error around frame 20.

Finally, we have seen how improvements like weighting of data points can allow faster convergence, and that outlier rejection, if with properly tuned parameters, can also have a positive impact on the convergence of the algorithm. This time, the convergence condition was the tolerance between two consecutive RMS values. We have also experimented on whether skipping frames can have a positive impact, and discovered that in our case, it lead only to worse reconstructed models.

The biggest question mark left at the end of these experiments, is why all the non degenerate local minima found by the algorithm resulted in very similar 3D reconstruction. This reconstruction was never perfect, with holes on the head, and visible errors on arms area. For sure, the fact that we always initialized the rotation matrix and translation vector in the same way plays an important role, much more than the small optimization tricks that we implemented such as outlier rejection and weighting.

References

[1] Brian Amberg, Sami Romdhani, and Thomas Vetter. Optimal step nonrigid icp algorithms for surface registration. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[2] Karl Johan Åström and Tore Hägglund. *Automatic tuning of PID controllers*. Instrument Society of America (ISA), 1988.

[3] Junfen Chen and Bahari Belaton. An improved iterative closest point algorithm for rigid point registration. In *International Conference on Machine Learning and Cybernetics*, pages 255–263. Springer, 2014.

[4] Shaoyi Du, Nanning Zheng, Lei Xiong, Shihui Ying, and Jianru Xue. Scaling iterative closest point algorithm for registration of m–d point sets. *Journal of Visual Communication and Image Representation*, 21(5-6):442–452, 2010.

[5] Shaoyi Du, Nanning Zheng, Shihui Ying, Qubo You, and Yang Wu. An extension of the icp algorithm considering scale factor. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 5, pages V–193. IEEE, 2007.

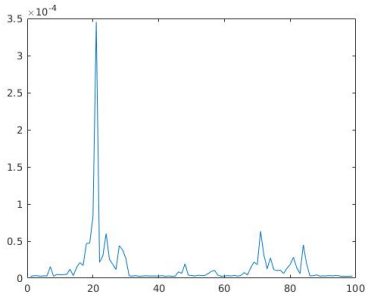
[6] SY Du, JH Zhu, NN Zheng, JZ Zhao, and C Li. Isotropic scaling iterative closest point algorithm for partial registration. *Electronics letters*, 47(14):799–800, 2011.

[7] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. 2001, 10 2001.

[8] Mang Zhuang and DP Atherton. Automatic tuning of optimum pid controllers. In *IEE Proceedings D (Control Theory and Applications)*, volume 140, pages 216–224. IET, 1993.

Figure 6: The errors for thresholda and variance outlier removal

(a) Threshold



(b) Variance