Computer Vision
ICP, EGC, SFM

Georgios Methenitis, Marios Tzakris, Paris Mavromoustakos

University of Amsterdam

June 2014

***Abstract*- In this technical report we are presenting implementation details along with experimental results obtained for two separate classes of algorithms. We will discuss the Iterative Closest Point (ICP) algorithm and how variations in different parameters of the implementation affect its performance. In addition, this report addresses the Structure from Motion algorithm. We will show how 3D structure of a scene can be obtained from 2D images taken from multiple viewpoints.**

1 Introduction

Iterative Closest Point (ICP) as first described in [1] is an alignment algorithm used to iteratively estimate the closest match between points in two point clouds or surface meshes. Section 2 discusses the details of this procedure tested on a given dataset of point clouds representing a human, viewed from various camera positions. In addition, a variety of modifications of the algorithm are analyzed in terms of speed, accuracy, stability and tolerance to noise. Furthermore, section 3 and 4 describe the Structure from Motion algorithm (SFM) [2]. SFM is a process of estimating shape and motion parameters of a 3D scene from 2D images depicting the scene from multiple views. In particular Section 3 addresses the task of computing the intrinsic projective (epipolar) geometry between two views and obtaining the global camera poses by chaining the derived geometrical measurements pairwise, while in Section 4 the method of extracting 3D structure of the scene given the chaining results, is explained.

2 Iterative Closest Point

Iterative closest point algorithm (ICP), as the name suggests, is a method of registering surfaces and 3D point clouds. The main steps of this method are, first finding the closest points between a base surface and a target one, and secondly computing the rotation and translation of the target surface with respect to the base one. This process continues iteratively until a mean square distance metric does not change any more.

The rotation and translation of a three dimensional surface have six degrees of freedom, which

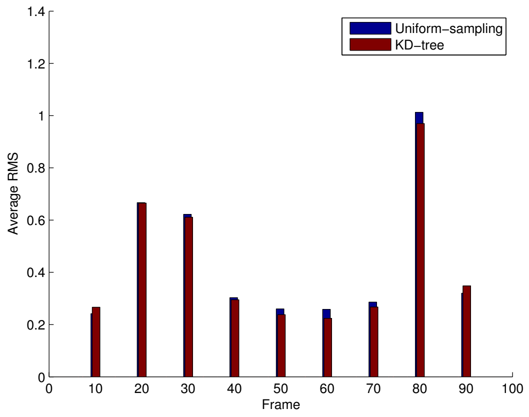

Figure 1: RMS error comparison between KD-tree implementation and the uniform sampling.

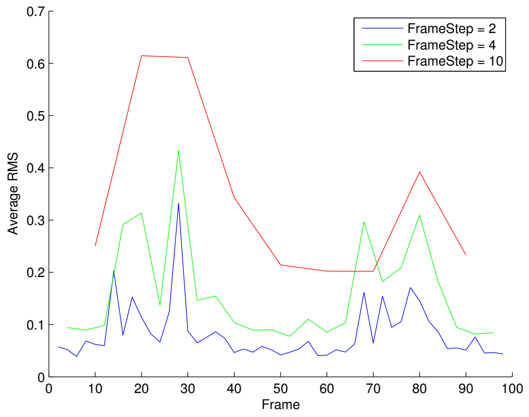

Figure 2: Average relative RMS error metric for merged kd-tree implementation for frame-step $1, 2, 4, 10$.

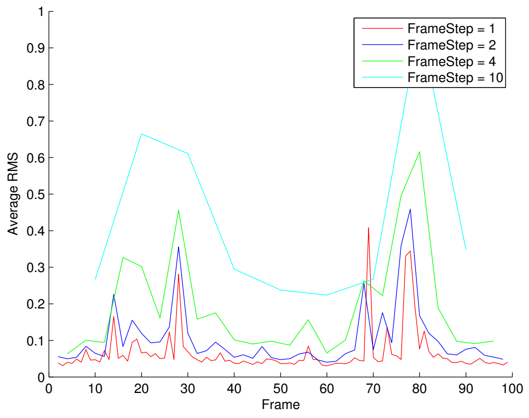

Figure 3: Average relative RMS error metric for kd-tree implementation without merging for frame-step $1, 2, 4, 10$.

the algorithm is guaranteed to converge to a local optimum every time. What is more, two major aspects regarding the performance as well as the convergence of the algorithm are the selection of the points from the base surface and the search method for the closest ones to the target surface. A good selection policy of the points can lead to an optimal registration of the surfaces and a very efficient performance computationally-wise.

A few variants have been applied on the ICP algorithm, in which performance improvements resulted from different techniques for, closest point selection, point rejection and error metric variations. For point selection instead of using all available points, a uniform sub-sampling or random sampling among them can lead to faster detection of the closest points. Also, selecting points from more informative regions of the surfaces can lead to better performance since more points are needed if the gradients of the surface are of higher value. As far as matching points are concerned, we can use a few variants apart from searching the whole space of the target surface. A projection of each point in the reference surface to the target one can give us an accurate estimation about where the closest point lies. Also color and angle measurements between surfaces can give us useful information about where to find these points.

Dataset

Our dataset consists of 100 3D point clouds that represent surfaces deriving from multiple views of a human body.

2.1 Implementation

In order to implement our algorithm we followed the work done in "Implementation of a 3D ICP-based Scan Matches". The first and most computationally expensive step of the algorithm is to find for each point of the target cloud the closest point there is in the base one. After doing this we end up having two sets of points with size equal to the target cloud's size. These point clouds need to be centered so we can eliminate any translation the target point cloud might have. Thus, we normalize the base and target cloud's closest point sets subtracting their centers from each of their points. Applying singular value decomposition to the two normalized clouds helps us estimate the rotation of the target cloud with respect to the base. This process continues re-computing the closest points of each cloud and re-transforming the target cloud until the error is minimized.

Given the rotation, we transform the center of the target cloud by multiplying the homogeneous coordinates of it with the estimated rotation matrix subtracting it from the base cloud's center point. The resulting 3D point will serve as the translation of the target cloud with respect to the base one. Applying the estimated translation and rotation transformation onto the target cloud we acquire the new target cloud from which we can now compute the average euclidean distance from the base cloud. This process runs iteratively until the average euclidean distance is minimized.

Another important aspect of the algorithm is the way the merging of the two point clouds is done. There are two methods applied, the first is to merge the two point clouds as one and consider it as the base cloud comparing it with the next frame, and the second one is to store the merged point cloud and proceed considering the target cloud of the first iteration as the base cloud of the second.

2.2 Results

We applied three different techniques for closest point selection including KD-tree, uniform sampling and brute force. Brute force searches all the points of the base for each of the target. KD-tree stores all elements of the base cloud in a KD-tree and then minimizes the complexity of finding the closest point of each target point. Uniform sampling searches the closest points of a constant number of target points using brute force again. A comparison in the error of uniform sampling and KD-tree implementation is presented in figure 1. In this figure we can see the average RMS error when we merge every tenth frame with the base, without merging the result in the base cloud each time.

To check also how the number of frames we skip influences the performance of the algorithm we also conducted an experiment using the KD-tree implementation for every $1, 2, 4, 10$ frames. As we expected the error appears to be larger as we skip more frames. This result is illustrated in figure 2 when we merge the target cloud with the base cloud in each frame. For the implementation without the merging the same results are presented in figure 3.

The resulted merged surfaces are depicted in

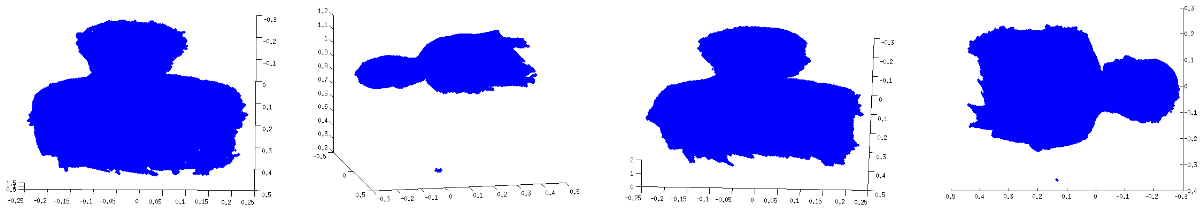

Figure 4: Visualization of the merged pointclouds using the ICP algorithm with different closest point selection techniques. From left to right: a) brute force, frame step 10, no merging b) Uniform sampling, frame step 4, with merging c) KD-tree, frame step 2, with merging d) KD-tree, frame step 1, without merging.

figure 4. There we can see the merged point clouds that represent the connected point clouds through the sequence of frames in the database. The results clearly show that the final point cloud is indeed the human that was scanned.

2.3 Discussing the ICP algorithm

We showed in the previous figures that merging the transformed point cloud with the base cloud every time clearly minimizes the RMS error. However, the results without merging the cloud are still adequate, given that the computational time does not explode as in the merged version of the algorithm. Additionally, the frame step also affects the performance of the algorithm in both cases, with merging and without. As we expected a minimum error is found when we apply the ICP algorithm in every frame.

Even though the ICP algorithm produces sufficient results for surface registration it still has major drawbacks when it comes to its computational cost which makes it really time demanding. The techniques presented earlier in this section improve its complexity without harming the performance.

It is noteworthy that using the kd-tree search significantly speeds up the icp algorithm. In particular, the brute-force implementation takes around 30 minutes to converge for each frame while the kd-tree amplified version requires around 2 minutes. What is more, skipping 1,3,5,10 frames also reduces the runtime of the algorithm analogously.

Another issue that arises when it comes to the performance of the algorithm is the fact that the surfaces to be merged need to be close to each other in terms of camera viewpoint, so the merging will converge to a quality solution.

Additional techniques with which ICP algorithm could be improved, are the following.

First, to add some constrains for the selected closest points we could say that the maximum distance of two selected points must not exceed the distance of the centers of masses of the two point clouds.

Another improvement could come from finding the two planes that are averaged through the point clouds. From these planes we could only sample points in some interval distances from each plane and find the closest in the same distances in the other point cloud. Through this technique we make sure that points are only taken from similar relative distances from the two planes, fact that can lead to a good selection of points.

3 Epipolar Geometry and Chaining

In this section we are going to describe the first part of the Structure from Motion algorithm, which is to exploit prior knowledge about the scene. This is done by finding corresponding image points in multiple views and compute their epipolar geometry which is expressed algebraically by the fundamental matrix which can be decomposed to obtain the required projection matrices.

3.1 Dataset

The data we were required to use consists of two individual datasets. The first one contains 49 frames, each one of them depicting a house from a different angle. The second contains 16 frames, each one of them depicting a teddy-bear from a different view. In each frame, there is a slight movement of the target object with respect to the previous frames.

3.2 Implementation

An important step of the algorithm is to identify local features in each image which are consistent in most of the frames. For this purpose we used the Scale-Invariant Feature Transform (SIFT) (vl_ sift function in vlFeat library) algorithm to detect points of interest in each frame. In order to restrict the search process of the SIFT detector an active contour based segmentation was used. By applying this technique we were able to segment the frames into background and foreground regions, allowing us to focus only on the points that exist on the object of interest as presented in figure 5.

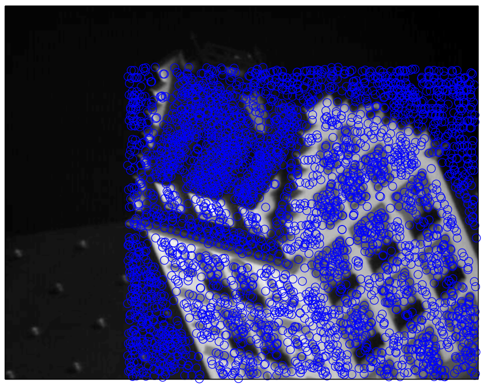

Figure 5: Feature points extracted by the SIFT detector found on the object of interest

By finding feature points in every frame we are able to identify matches in consecutive frames. This is done by using the built-in function vl_ ucbmatch of the vlFeat library of MATLAB which returns the desired corresponding feature points and rejects the ambiguous ones. The vl_ sift command without parameter tuning gives only a limited number of feature points as illustrated on the first image of figure 6. By setting a higher edge threshold which serves to filter out edge-like features we were able to obtain more key points. Another parameter we tuned was the number of levels per octave. By increasing this number in principle it returns more refined keypoints, but in practice the selection may be unstable due to noise. Feature matches between two consecutive frames of the House dataset with tuned parameters, are displayed on the second image of figure 6.

The next step is to apply centering to the points, namely subtract the centroid of the fea-

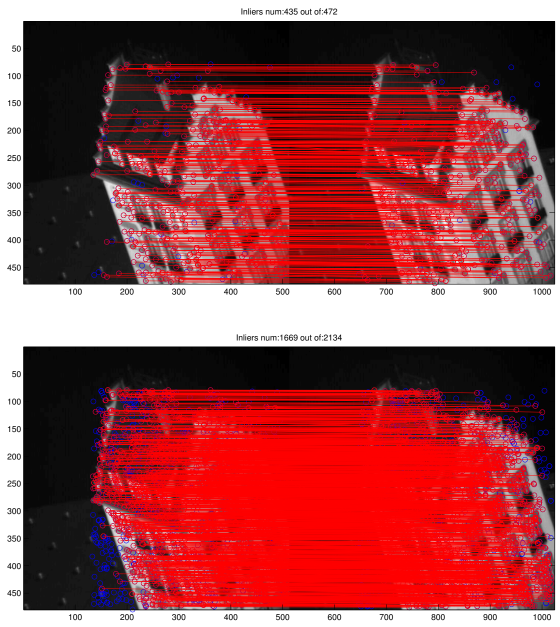

Figure 6: Above: Feature matches between two consecutive frames without tuned parameters, Below: Feature matches between two consecutive frames with parameter tuning: edgeThresh: 30, Levels: 30.

ture points (remove translation) according to :

$$\hat{x} = x_{ij} - \frac{1}{n}\sum_{k=1}^{n} x_{ik} \tag{1}$$

To estimate the fundamental matrix deriving from the corresponding image points extracted by the SIFT detector the normalized Random Sample Consensus (RANSAC) eight-point algorithm [3] was implemented. Application of this algorithm leads to a more stable result than the simple eight-point implementation. The steps of the algorithm are:

1 Pick 8 random matches from the list of the corresponding points.

2 Construct a matrix A from the matches and get the transformation matrix from the last column of V of the Singular Value Decomposition of A.

3 Count the number of inliers (points that agree with the transformation matrix) by computing the Sampson distance (if distance is smaller than a certain threshold then we consider the point as an inlier). By experimenting with different threshold values, we concluded to 0.5 being the most efficient.

4 Repeat steps 1 to 3 until the majority of the points are inliers.

3.3 Chaining

The last step is to construct the Point View Matrix which contains normalized point coordinates (inliers). An $m \times n$ matrix is initialized, where m is the number of frames and n is the total number of feature points. For every point (column) a 1 is added to the matrix's cell if this feature point is present in frame (row) n, else it remains 0.

3.4 Discussion

In fact, it is not always the case that many points will be present in all frames due to the fact that the frames may have considerably large difference in view among them. In figure 7 the feature point view matrices for the house and teddy-bear datasets are illustrated. We can see that the teddy-bear dataset produces a vastly more sparse point view matrix compared to the one the house dataset produces. In particular, no points are consistent through the whole teddy bear frame set. These point view matrices will be used in the 2nd part of the SFM implementation, in order to build a 3D structure of these two objects.

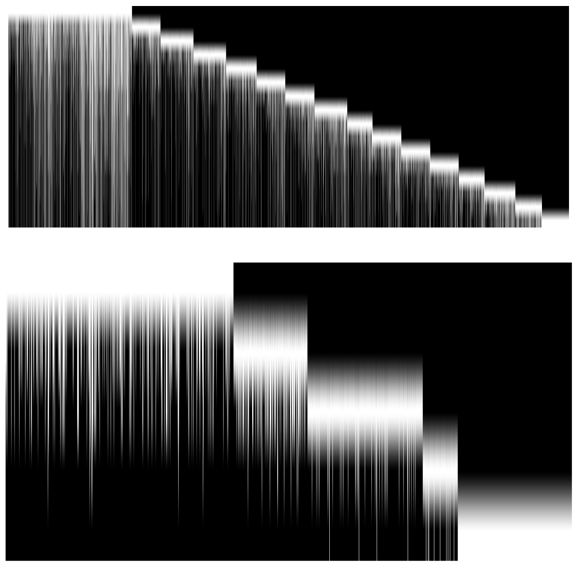

Figure 7: Point view matrix sequences, Horizontal axis denotes detected feature points, vertical axis are the frames. Lines starting from the top going to the bottom of the picture represent points detected in consecutive frames. From top to bottom: a) house dataset, b) teddybear dataset. Note that these Point View matrices result from 3-frame skipping (see 4.3)

4 Structure from Motion

The last part of the lab assignment discusses the final parts of the Structure from Motion (SFM) algorithm. In particular, using the output of the assignment's 2nd part, we derive the motion and shape matrix of a given set of points.

4.1 Implementation Details

As mentioned above, the data used by this part of the algorithm is an $m \times n$ matrix, where m is the total number of frames, and n is the total number of feature points extracted by the previous step of SFM. Given this "Point View" matrix, we gather feature points that persist through a sufficient number of frames, and construct a new $A : 2m \times n'$ matrix, where each row represents a 2D coordinate of each of n' feature points. Note that this matrix is dense enough to adequately describe 3D structure given 2D feature points.

After obtaining matrix A, 3D coordinates must be calculated for each feature point, with respect to every frame. In order to achieve that, we need to normalize the 2D coordinates in A. For each row of the A matrix, we calculate the centroid (the average coordinate over all feature points) which then we subtract from every point coordinate in the same row. By doing this, we relate every feature point to a 3D position in the resulting 3D structure.

Following normalization, SVD is applied to the (normalized) A matrix to obtain the motion & shape matrices: $D = U \times W \times V^T$

We need to reduce the rank of the resulting matrices to 3, in order to obtain 3D coordinates, so we use the 3 first columns of U, the 3 first rows of V^T and the top-left 3×3 matrix of W to obtain U_3, W_3, and V_3^T. Lastly, the matrix & shape functions are given by: $M = U_3 W_3^{\frac{1}{2}}$, $S = W_3^{\frac{1}{2}} V_3^T$

4.2 Dealing with affine ambiguity

The process described before, implies a decomposition that is subject to affine transformation[4]. Taking that into account, we further factorize matrices $M \rightarrow \hat{M}$ & $S \rightarrow \hat{S}$

according to Morita and Kanade (1997)[4]. In more detail, we calculate matrix A^{-1} and A where A represents an affine transform that transforms $\hat{M}$ into M and A^{-1} transforms $\hat{S}$ into S. Applying these matrix manipulations erases the affine ambiguity, under the assumption that image axes must be perpendicular[4].

Looking at figure pair 11 - 12 we can see how the lines of the structure are more realistic after the affine ambiguity elimination.

4.3 Frame sampling

We have observed that many feature points are detected at non-informative positions, implying a noise factor in our SFM implementation. In order to reduce the noise, while also increasing the speed of the algorithm, we sampled though the frame set, by skipping 1, 2, 3, 4, 5 or 8 frames. In the results section we can see how this parameter changes the 3D structure, and conclusions on it are derived in the discussion section.

4.4 Results

Below, we present the results obtained by the SFM implementation.

4.5 Discussion

This last step of the SFM algorithm gave us the opportunity to observe the results in practice. Given a set of frames, we used SFM to construct a 3D model of the object presented.

However, while experimenting with two different sets of frames (House & Teddybear), we discovered that the teddybear frameset is not capable of producing an efficient 3D model, while the house frameset does. An explanation to why this happens lies not in the actual implementation of the algorithm, but the dataset itself. It

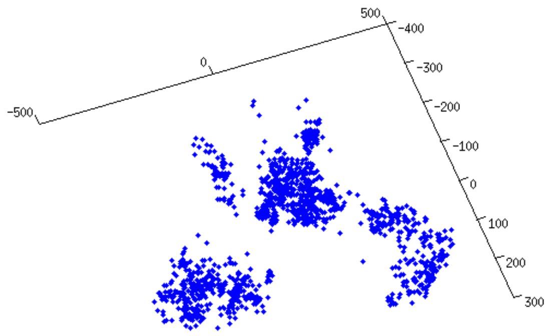

Figure 8: 3D structure of the teddybear frameset, sampling all frames, with affine ambiguity correction.

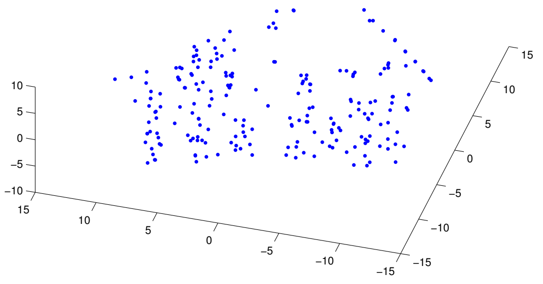

Figure 9: 3D structure of the house frameset, sampling all frames, without affine ambiguity correction.

is a fact that the teddybear frameset contains a 360° y-axis rotation of the object, resulting in significantly less feature points being persistent through a large number of frames. On the other

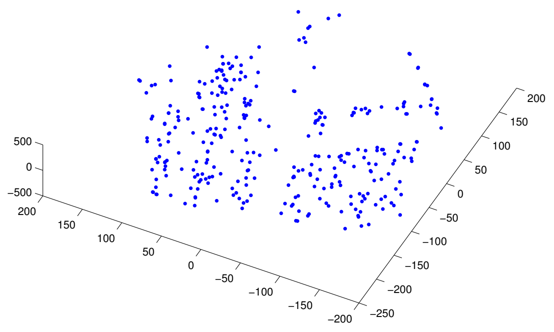

Figure 10: 3D structure of the house frameset, sampling every 2 frames, with affine ambiguity correction.

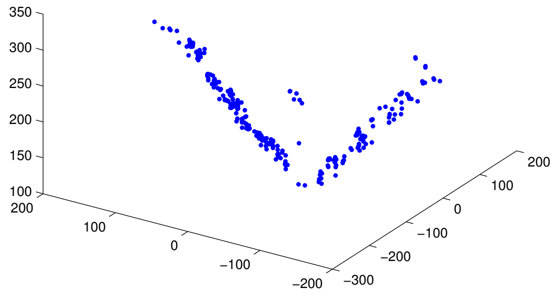

Figure 12: 3D structure of the house frameset, sampling every 3 frames, with affine ambiguity correction.

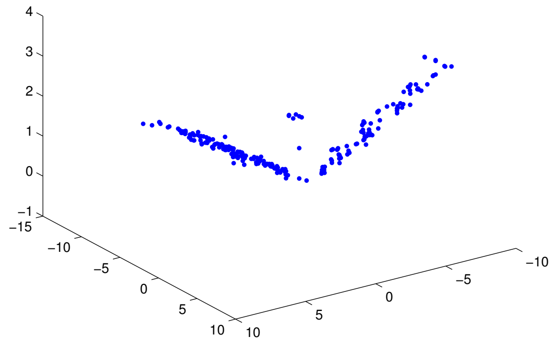

Figure 11: 3D structure of the house frameset, sampling every 3 frames, without affine ambiguity correction.

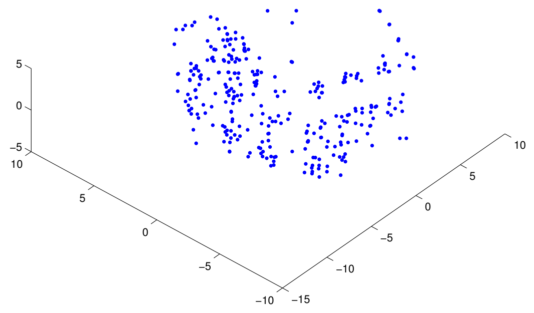

Figure 13: 3D structure of the house frameset, sampling every 4 frames, without affine ambiguity correction.

hand, the house frameset does not contain such steep change in point of view, but only a slight z-axis rotation. As a result, the number of feature points detected in all frames is higher, thus, the 3D model reconstructed by SFM looks more realistic. One can detect an unrealistic reconstruction for the teddybear dataset in figure 8.

Furthermore, regarding the frame sampling

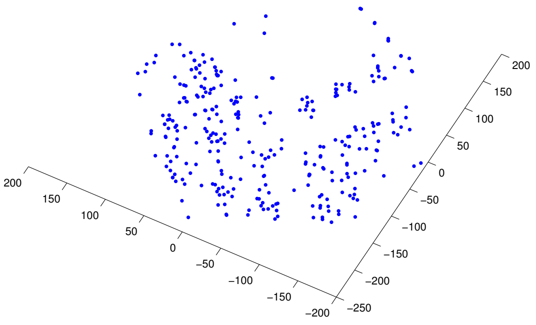

Figure 14: 3D structure of the house frameset, sampling every 4 frames, with affine ambiguity correction.

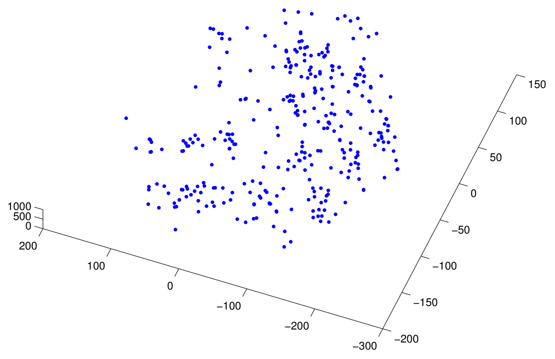

Figure 15: 3D structure of the house frameset, sampling every 5 frames, with affine ambiguity correction.

feature introduced, we can observe an improvement between figure 9 and figure 13, in terms of how realistic the structure looks. We can see that many new feature points are introduced,

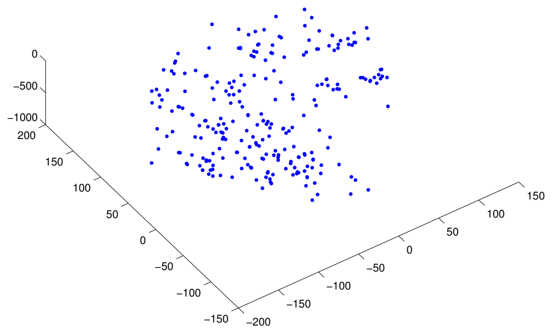

Figure 16: 3D structure of the house frameset, sampling every 8 frames, with affine ambiguity correction.

while in figure 14, almost no noisy points are present.

Despite the improvement achieved, we can observe in figures 15 and 16 that sparse sampling can lead to undesired results. The structure of the house is inconsistent, especially in figure 16 where feature points are consistent only for a small part of the house structure.

5 Printable 3D Model

We were assigned with the task of structuring a triangular watertight mesh, given a 3D point cloud representing a human. Figure 17 represents a visualization of this point cloud.

5.1 PCL (C++)

In order to build a printable 3D model given a point cloud, we used the Point Cloud Library (PCL)[5]. PCL has built-in functions in order to

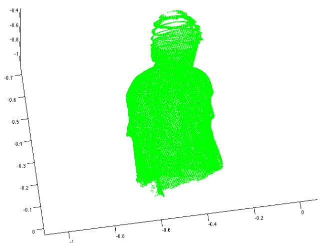
Figure 17: Visualization of a 3D point cloud of a human

read and visualize a 3D point cloud using triangulation. It can also provide with functions to make the 3D structure watertight. We visualized the output using ParaView, see figures 18, 20 and 19.

5.2 MyCrust (MATLAB)

In order to visualize a 3D point cloud in MATLAB, we used the "myCrust.m" toolbox (`http://www.codeforge.com/read/218867/MyCrust.m__html`). The functions it supplies construct a triangular mesh given a point cloud as input. In order to retrieve the data, we used the given readPcd() function, while we also used the MATLAB built-in function imfill() to create a watertight model. The latter function will "fill the holes" of every plane in the whole dataset.

Snapshots of the visualization are presented in figures 23, 22 and 21.

6 Conclusion

This lab assignment has been an educational and challenging task. We were able to observe how algorithm implementations work in practice by

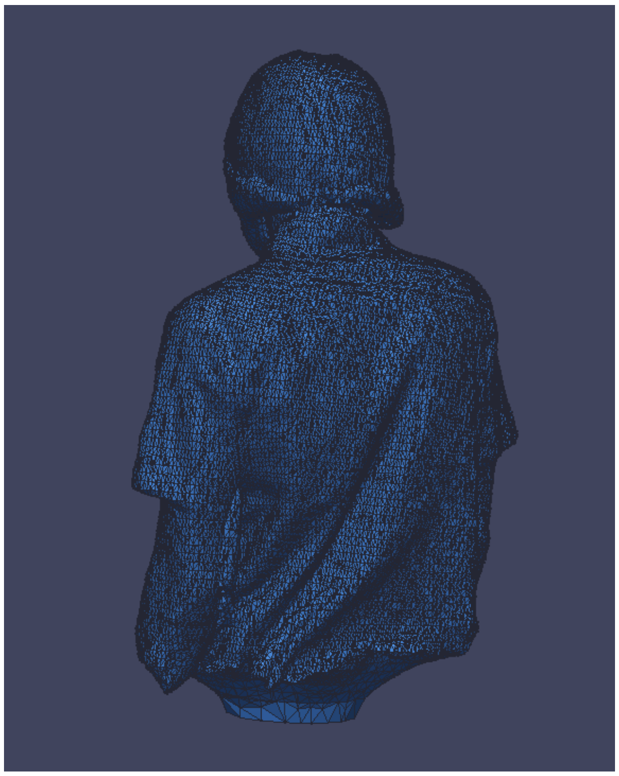
Figure 18: Printable triangular mesh, back view

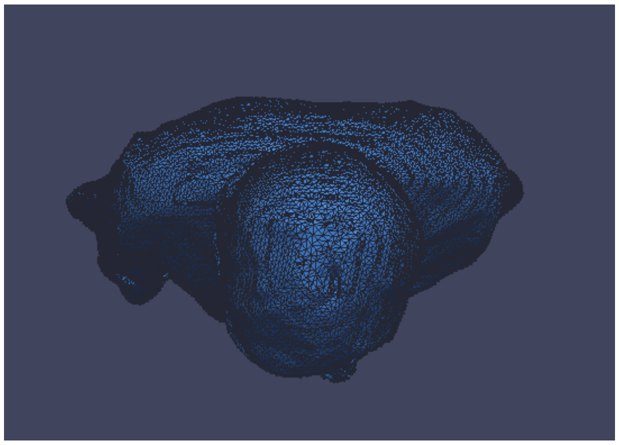
Figure 19: Printable triangular mesh, top view

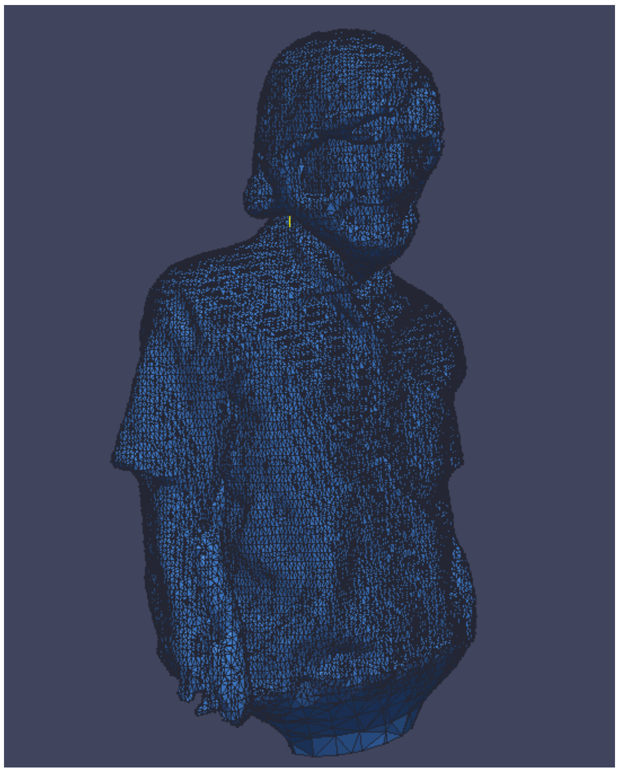

Figure 20: Printable triangular mesh, front view

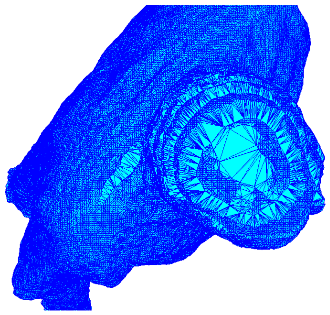

Figure 21: Printable triangular mesh, top view

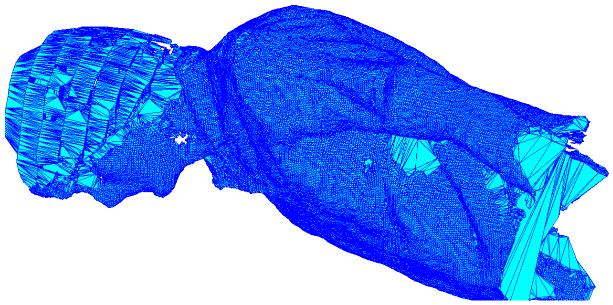

Figure 22: Printable triangular mesh, side view

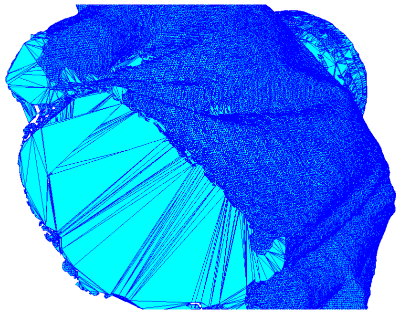

Figure 23: Printable triangular mesh, bot view

visualizing the results, and see how variations of these methods affect the constructed models.

We have realised that 3D model reconstruction is a computationally expensive and complex procedure, however, choice of the correct features, descriptors and point sampling methods can result to an efficient algorithm.

References

[1] Besl, Paul J., and Neil D. McKay. "Method for registration of 3-D shapes." In Robotics-DL tentative, pp. 586-606. International Society for Optics and Photonics, 1992.

[2] Rothganger, Fred, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. "3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints." International

Journal of Computer Vision 66, no. 3 (2006): 231-259.

[3] Richard I. Hartley. “In Defence of the 8-point Algorithm.” (1997)

[4] Toshihiko Morita and Takeo Kanade, “A Sequential Factorization Method for Recovering Shape and Motion From Image Streams”, Fellow, IEEE, 1997.

[5] Radu Bogdan Rusu and Steve Cousins, “3D is here: Point Cloud Library (PCL)”, IEEE International Conference on Robotics and Automation (ICRA), May 9-13 2011.