Computer Vision 2 - Assignment 2
Structure from Motion

Yke Rusticus
yke.rusticus@student.uva.nl
11306386

Ewoud Vermeij
ewoud.vermeij@student.uva.nl
11348860

July 7, 2020

1 Introduction

In this work we aim to recover three-dimensional structures from a set of 2D images. In the analysis of scenes taken with uncalibrated cameras, the fundamental matrix can be used as basic tool (Hartley (1997)). Multiple methods exist for computing the fundamental matrix. However, the Eight Point Algorithm is known for its simplicity and is therefore used in this work. We explore multiple implementations of the Eight Point Algorithm: a ″basic″ implementation without any changes; one where coordinates are normalized in the process, which was shown to be effective by Hartley (1997); and lastly, with RANSAC. Subsequently, we perform ″chaining″ on our set of images, to build a representation of point correspondences for the different camera views (see Rothganger, Lazebnik, Schmid, and Ponce (2006)). We will then use this representation, the point-view matrix, to create a 3D reconstruction of our scene. The following sections describe the methods and implementation, the experiments and finally the conclusion.

2 Methods & Implementation

The first subsection shortly describes the data used. The following subsections describe the details of the Eight Point Algorithm, chaining, and lastly we describe how we applied structure from motion.

2.1 Data

We used 49 frames of a house taken from multiple viewpoints, shown in Figure 1.

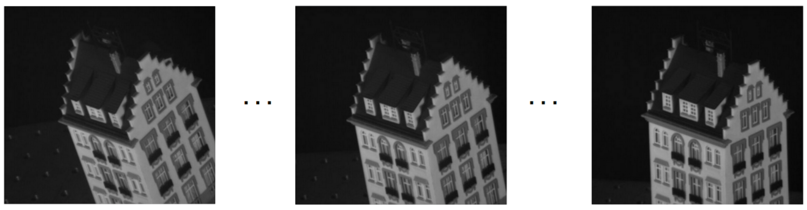


Figure 1: Three samples from the data used in this work. The viewpoint changes slightly from frame to frame.

2.2 Fundamental Matrix

In this section, we describe different approaches to approximating the fundamental matrix F, such that

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \underbrace{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}}_{F} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0, \tag{1}$$

for any i'th (corresponding) points (x_i', y_i') and (x_i, y_i) in two different views. In our implementation, the function `get_F` returns a the fundamental matrix, given two images I_a and I_b, and given the desired approach corresponding to the subsections below.

2.2.1 Eight-Point Algorithm

In our "standard" implementation of the Eight Point Algorithm, we first find matching points between I_a and I_b, using VLFeat SIFT descriptors and matching[1]. Then, using the resulting corresponding points p_a and p_b we call `eight_point_alg` that performs the main part of the algorithm. First, matrix $A \in \mathbb{R}^{n \times 9}$ is formed using p_a and p_b, as described in the assignment. Subsequently, we obtain the SVD from A, $A = UDV^\top$. The components of F are then given by the column of V corresponding to the smallest singular value. Then the rank of F is forced to zero by setting its smallest singular value to zero after performing SVD. F is then recomputed using the corrected singular value matrix and returned by the algorithm.

2.2.2 Normalized Eight-Point Algorithm

In the normalized version of the Eight Point Algorithm, the main part of the algorithm is identical as described before. In our implementation however, we normalize the points in p_a and p_b before we call `eight_point_alg`. This is effectively done by multiplying the points with normalization matrices T_a and T_b respectively. The fundamental matrix $\hat{F}$ that is returned by `eight_point_alg` is then denormalized as follows:

$$F = T_b^\top \hat{F} T_a. \tag{2}$$

The resulting fundamental matrix F is returned by `get_F`.

2.2.3 Normalized Eight-point Algorithm with RANSAC

Instead of using all (corresponding) points for the Eight Points Algorithm, we can iteratively find the best set of points to solve the problem. In this approach we sample 8 corresponding normalized points every iteration, for a total of N iterations. These points are fed to `eight_point_alg` similarly as described before, and the resulting fundamental matrix is denormalized to F. Using the point correspondances and F, we then calculate

[1] `https://www.vlfeat.org/overview/sift.html`

the number of inliers using Sampson distance and a given threshold. We keep track of the set of inliers per iteration. After iterating, the largest set of (normalized) inliers is used to calculate F again, which is then denormalized.

2.3 Chaining

To construct the point-view matrix (PVM) for consecutive views in the data set, we take the following steps in the function `chaining`. For every pair of consecutive frames I_i and I_j we first obtain the normalized matching points $\hat{p}_i$ and $\hat{p}_j$. If this is the first pair of frames we simply add all points to the PVM and move on to the next pair. For each new pair of frames we first add another (empty) row to our PVM. Then, with a Euclidean distance threshold we check for each point in $\hat{p}_i$ if it is already present in the previous row of the PVM. If for a given point $\hat{p}_i$ the smallest distance to a point in the previous row is smaller than the threshold, we add point $\hat{p}_j$ to the column corresponding to that closest point in the row we just added. If the smallest distance between $\hat{p}_i$ and the points in the previous row is larger than the threshold, we add a new column to the PVM and add $\hat{p}_j$ to the final row in that column. In that last case we have added a "new" point to the PVM. For visualization purposes, we have added the option to filter out all points that are absent in more than a given number of `frames_absent` frames.

2.4 Structure from Motion

With structure from motion, we will use the point view matrix (PVM) to construct a 3 dimensional point cloud of the object represented in the views. If we had all points in all views we could simply factorize the PVM matrix directly to recover the 3D point cloud. Unfortunately, we do not have all points in all views and will therefor construct dense blocks. Our function `get_dense_blocks` takes care of this. This function retrieves a set of dense blocks where each block contains a set of views. For each view we have a the set of points appearing in all views of that block. Our function has the possibility to set a range of minimum and maximum number of views per block and a minimum for the required number of points per dense block. We set range of views per block from 3 to 4 and the minimum number of points per block at 10.
Next, we will iterate over each block and factorize the points. After factorization, we retrieve 2 matrices: one for motion M, and one for structure S. We will use matrix S to create our 3D point cloud. Initially we save our point cloud S to the variable PC. For all further points clouds S, retrieved from the dense block we will first retrieve the common point between the so far constructed point cloud PC and the to be added point cloud S. With the intersected points, we use the Procrustus analysis to retrieve the transformation Z. Next, we specify two methods to add the new points from the transformed point cloud Z to PC. The first method takes all points from Z and overrides the same points in PC with the new points from Z. The second method checks for each point in Z if it already exists in PC. If not, the point is added to PC.

At last, our final construction regularly seemed to have some outliers. We decided to remove those from our final 3D point cloud, since some point clouds appeared to be slightly flat. The results of our experiments will be discussed in the next section.

3 Experiments

3.1 Epipolar lines

Figure 2 shows the epipolar lines drawn based on the estimated fundamental matrix obtained using ”standard” Eight-Point Algorithm, normalized, and the normalized algorithm with RANSAC. First of all, the orientation of the lines across the different methods is not consistent. In the first case in particular, the corresponding lines in the images themselves do not seem to make much sense. As we can see from the sampled epilines on the right of the figure, the epipolar constraint is not met in the first case, since the lines do not intersect with the points in any case. This result in line with the known instability of the standard algorithm. However, if we look at the normalized and the normalized with RANSAC approaches, we see much better results. In both cases, the epipolar constraint seems to be satisfied. Between the normalized and the RANSAC approach we see another difference (appart from the orientation of the lines). Namely, the images on the left for ”Normalized” show more points in the background compared to the images for ”RANSAC”. This is due to the fact that with the RANSAC approach, we only take points into account that agree with the fundamental matrix. This potentially removes unwanted points from the process, and is therefore expected to produce better results than the approach without RANSAC. Witch of the line orientations would be the most plausible between the last two approaches, is however not clear.

3.2 Chaining visualized

Figure 3 shows our chaining results and Figure 4 the visualized matrix from `PointViewMatrix.txt`. The given matrix was very clean, with no background points and long chains of points over all of the frames. We found it difficult to reproduce this result. If we filter only on the points that were present in all frames, we are left with only a handful of points. Moving on to filter the points that were present in over 40 frames, 30 frames, etc., we found that our image starts to resemble the given data more and more, until the matrix becomes too sparse and more background points become present. Our point-view matrix’ density is therefore dependent on how we filter it. In order to obtain more consistent point chains, a possible improvement might be to take into account point descriptors, to be able to really confirm whether a point corresponds to a point in a different view.

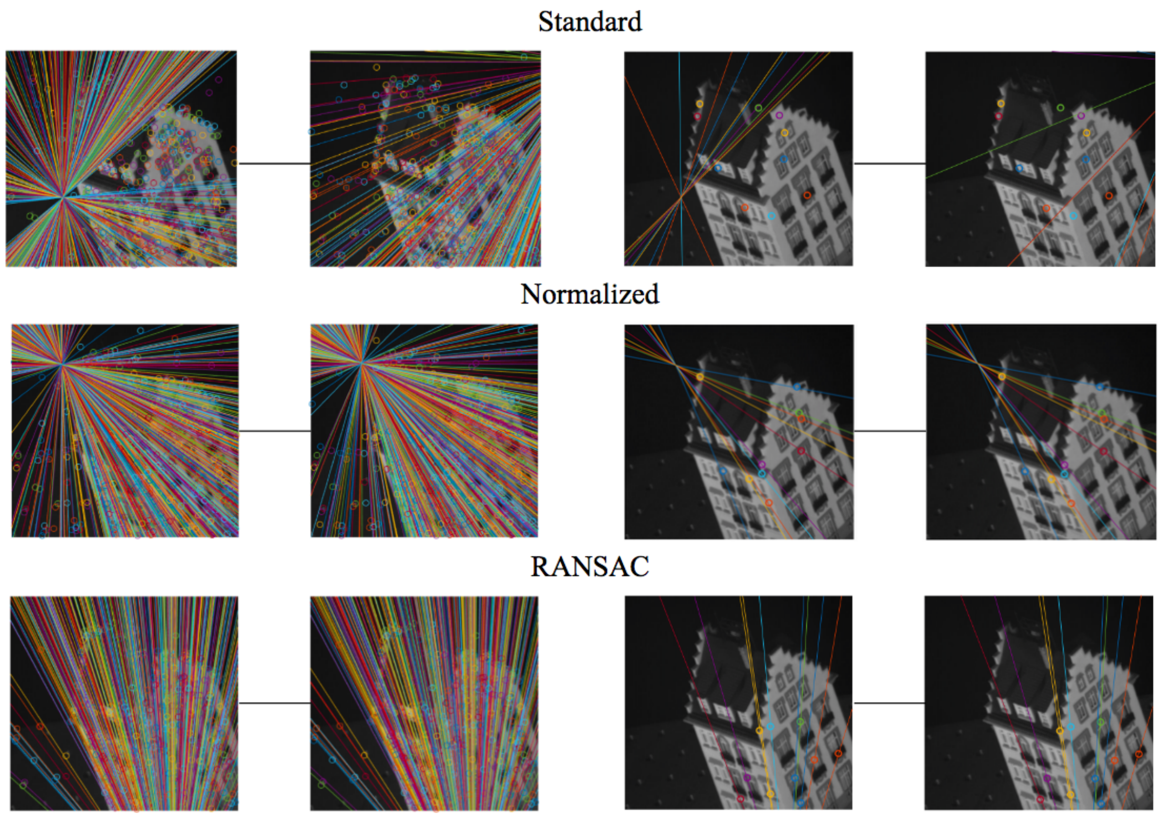


Figure 2: Epilines produced for image pair 1-2 using different Eight-Point Algorithm approaches. For each row, the two images on the left show lines for all corresponding points, and the images on the right show lines for a sample of 10 corresponding points.

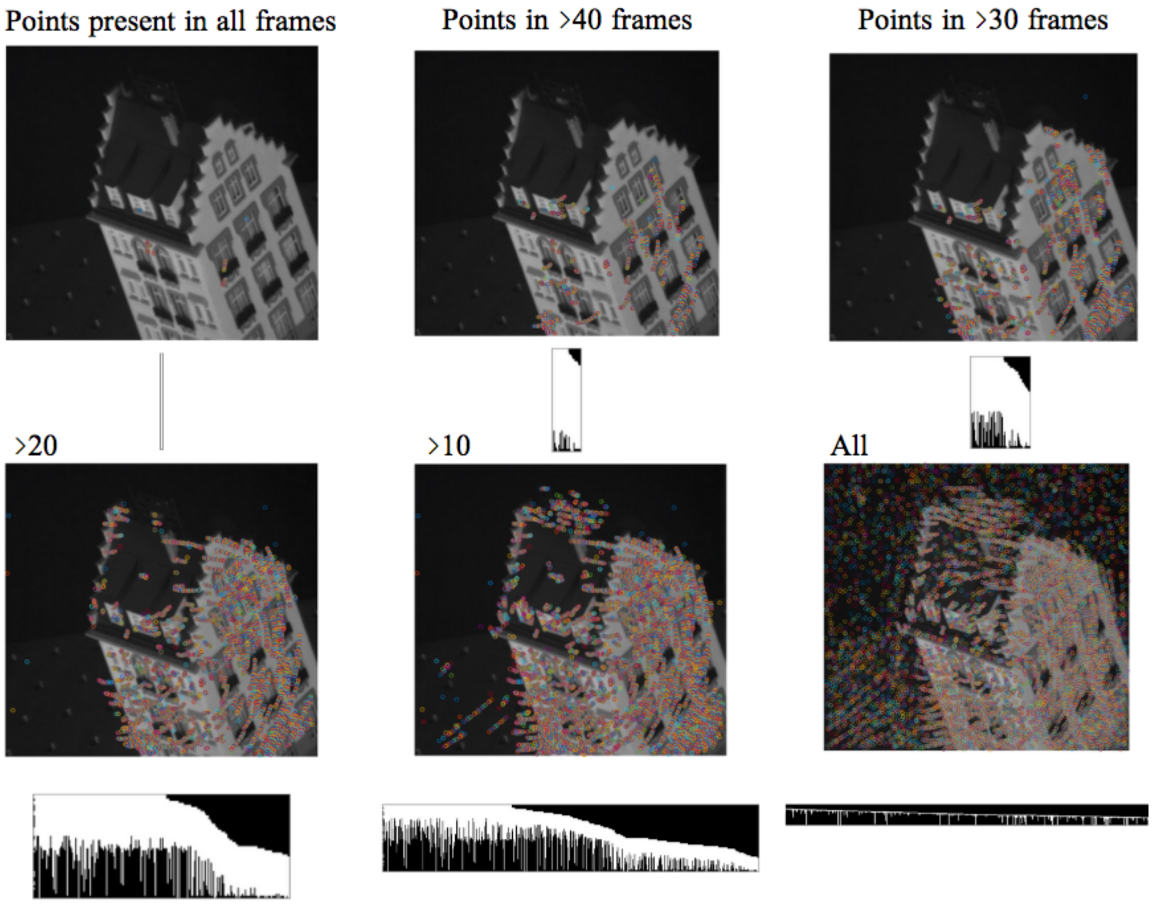


Figure 3: Each image shows scattered points from the PVM over the first image in the sequence. The points in each image are the points that are present in at least the specified number of frames in the sequence. Below each image, the PVM itself is visualized with non-empty values as white, and empty values as black. The bottom right image shows only part of its matrix, as it was highly sparse and therefore too large to visualize.

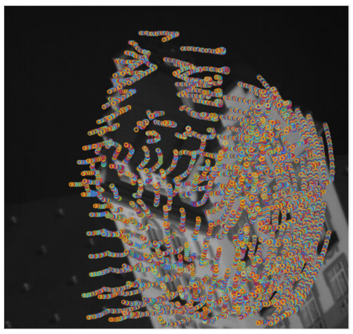
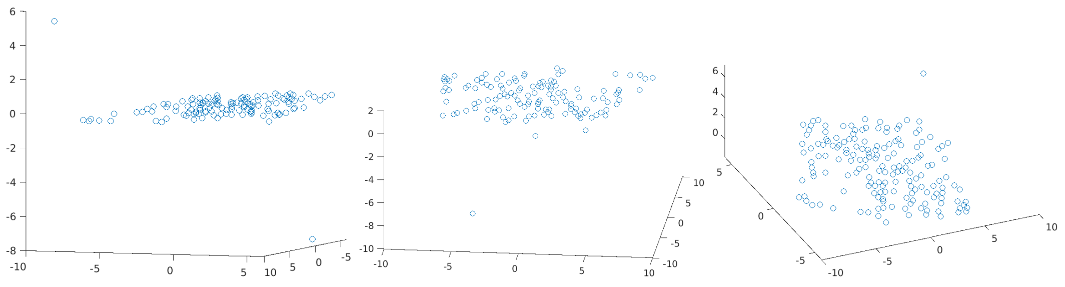


Figure 4: Chained points as given in the assignment as `PointViewMatrix.txt` visualized over the first image in the sequence. Each point in this matrix was present in each frame, so the matrix would look like the first example in Figure 3, but in this case with more points, so more columns.

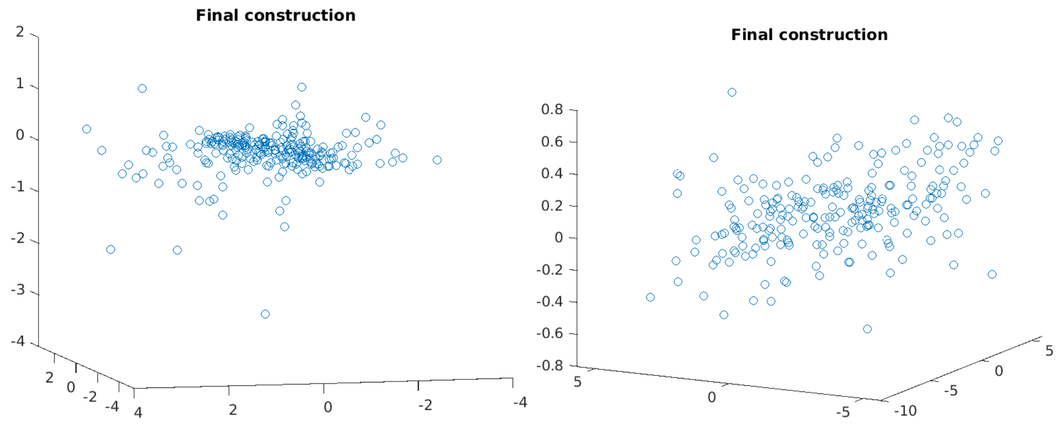
3.3 From 2D to 3D - SFM

In this section we will look at the results of the constructed 3D point clouds from Tomasi and Kanade (1992). We will first look at the point clouds generated. Figure 5 shows three point clouds generated from individual dense blocks. We can immediately see that all samples follow the same structure: A relatively flat point cloud with a few outliers. This is likely to be caused by the fact that we did not retrieve the optimal points initially, leaving us with point clouds not indicating anywhere near something that could look like a house. In figure 6 the two point clouds created with the override and empty method are shown. Neither representing a structure that could look like a house. This is not surprising, since the structures from the dense blocks are also not representing anything similar to a house. The override method does seem to gather all points in a more central position, where the empty method almost seem to have created a random point cloud. The override method updates all points on every dense block iteration. It is therefore possible that the last (few) dense block was relatively centered and added to the final point cloud. The empty method updates only points which have not been seen yet. Given that the dense block clouds do not seem to represent clear structures, merging the clouds with Procrustus does not return the wanted effect. Therefor, the cloud seems almost random.



(a) Dense block 1 (b) Dense block 2 (c) Dense block 3

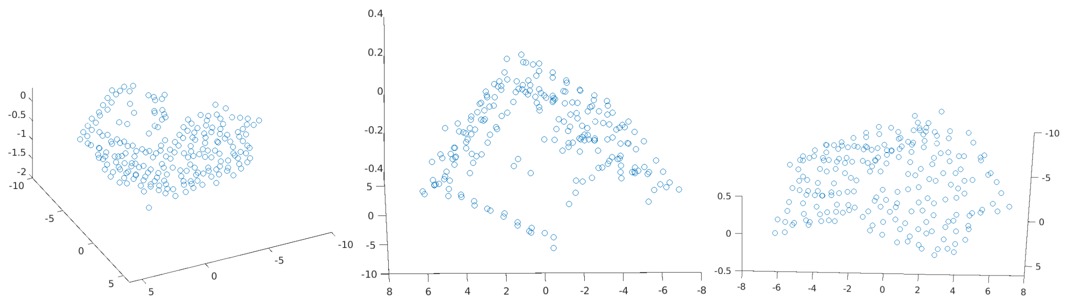
Figure 5: Different point clouds from dense blocks from the chained point view matrix.



(a) Override (b) Empty

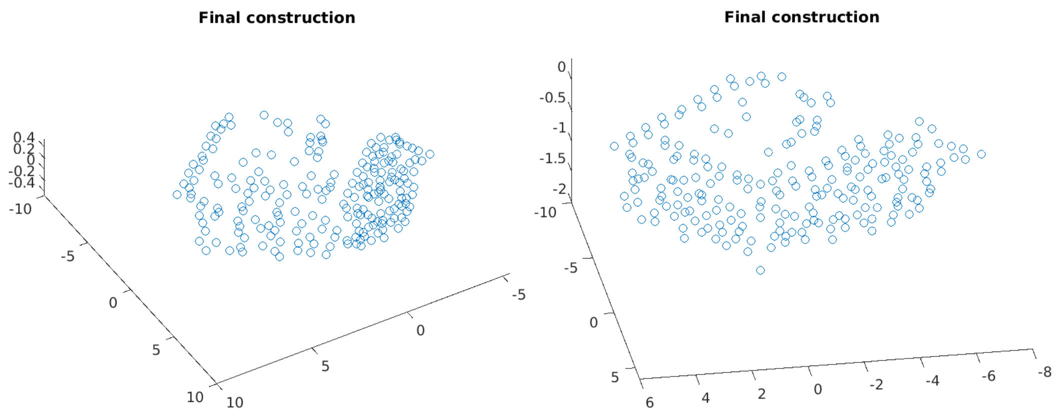
Figure 6: 3D point clouds using the point view matrix from chaining

We now shift our focus to figure 7. Here we use the PVM provided in the `PointViewMatrix.txt` file, which has ideal point information for us. Looking at the dense block samples, we can see clear structures that could represent a house or a roof of a house. Given these ideal point clouds from the dense blocks, it is obvious that we can construct very clear, 3 dimensional point clouds representing the house it should represent. We can see some differences between the override method and the empty method. The points from the override method seem to be more centered than the empty method, similar to the point clouds created with the chaining method. However, these differences are minor.



(a) Dense block 1 (b) Dense block 2 (c) Dense block 3

Figure 7: Different point clouds from dense blocks from the file provided point view matrix.



(a) Override (b) Empty

Figure 8: 3D point clouds using the point view matrix from the provided file.

4 Conclusion

We have shown that, given the epipolar lines, the normalized and RANSAC normalized eight-point algorithm show better results in terms of epipolar constraints. We expect RANSAC to provide the best results, due to the fact it only takes points that agree with the fundamental matrix.

We had some difficulties reproducing the ideal point view matrix given to us. Using only points existing in all views only gave us a handful of viewpoints. It was therefor a trade-off between using all points and using points existing in all views. We found that setting the `frames_absent` parameter between 20 and 30 gave the best point view matrix.

To construct a qualitative 3D point cloud representation given a set of points of interest is very dependent on the quality of the point view matrix itself. We have shown that given the algorithm of Tomasi and Kanade (1992) and an ideal point view matrix, it is possible to construct a clear 3D representation of the object shown from different view points.

References

Hartley, R. I. (1997). In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, *19*(6), 580–593.

Rothganger, F., Lazebnik, S., Schmid, C., & Ponce, J. (2006). 3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International journal of computer vision*, *66*(3), 231–259.

Tomasi, C., & Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *International journal of computer vision*, *9*(2), 137-154.

A Team member contributions

Ewoud Vermeij:

- Implementation of basis for fundamental matrix - Eight-point algorithm.
- Implementation of Structure from Motion.
- Implementation of Structure from Motion demo and visualizations.
- 50% of report.

Yke Rusticus:

- Extension eight-point algorithm to normalized and normalized + RANSAC.
- Chaining.
- Epiline and chaining visualizations.
- 50% of report.