Computer Vision 2 - Assignment 2

Emma Hokken - 10572090
emmahokken@gmail.com
University of Amsterdam

Tim van Loenhout - 10741577
timvanloenhout@gmail.com
University of Amsterdam

1 INTRODUCTION

In this report, we note our findings for Assignment 2 of the course Computer Vision 2. First, we estimate a fundamental matrix using three different approaches. Second, we visualize a point-view matrix which displays the number of features present in each image. Lastly, we use this point-view matrix to reconstruct a 3D structure from 2D images.

2 DATA

The data consists of 49 images of a single house, where each image has a slight change in camera position.

3 FUNDAMENTAL MATRIX

Here, we constructed a fundamental matrix, using three different approaches; using Simple Eight-Point Algorithm, using Normalized Eight-Point algorithm and Normalized Eight-point Algorithm with RANSAC. Such a fundamental matrix describes the correlation between corresponding points in two images. These corresponding points are found by first finding the SIFT key point descriptors and then matches are constructed from the corresponding keypoints. Then noisy (background) matches are filtered out and the remaining points are stored as $(2xn)$ arrays. Finally, these arrays are converted to a homogeneous coordinate system, with $w = 1$, resulting in two point arrays of size $(3xn)$.

Using a fundamental matrix, the epipolar lines are obtained by taking the dot product between the fundamental matrix and the points.

For each approach, the results are shown for two pairs of images; one subsequent pair (image 1 and 2) and one pair with a larger viewpoint discrepancy (image 7 and 15).

In order to obtain a benchmark against which to compare our results, we also plotted the epipole lines obtained by using the OpenCV fundamental matrix (ref 1).

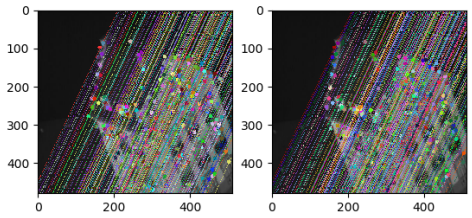
3.1 Eight-point Algorithm

In order to construct the fundamental matrix, we fist store the matches in a $(nx9)$ matrix A, such that $A \cdot fundamentalmatrix = 0$ (figure 2).

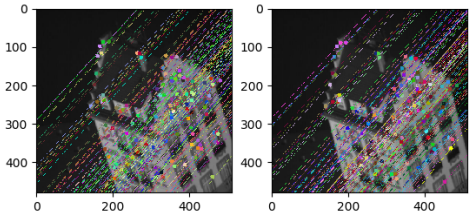
This matrix is then decomposed into UDV^T, where the column of V corresponding the largest singular value makes up the entries of the fundamental matrix F, which is then reshaped to size $(3x3)$. Finally, in order to make F rank 2, it is decomposed and recomputed after setting the smallest singular value in the diagonal matrix to 0.

3.2 Normalized Eight-point Algorithm

Here we first normalized the points before computing F. By normalizing, the points become centered around 0, with the standard deviation to the origin going from being 126.934 for image 2 and 126.774 for image 3 to both 1.414, which equals $sqrt(2)$.



(a) Image 2 and 3



(b) Image 7 and 15

Figure 1: Results OpenCV baseline fundamental matrix. Left: first image with epipolar lines corresponding to points on second image. Right: second image with epipolar lines corresponding to points on first image.

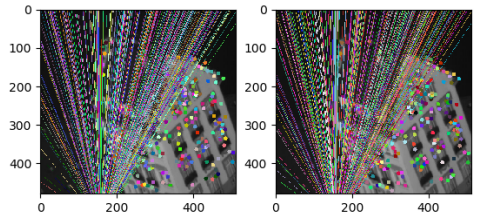
$$\underbrace{\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_nx_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix}}_{A} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

Figure 2: (nx9) matrix A.

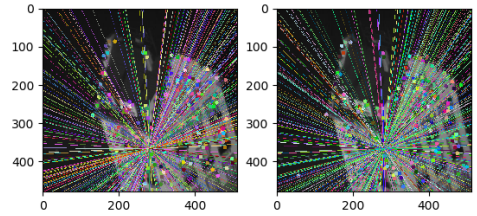
Using the normalized matches, F is computed, which is then applied on the original points to obtain the epipolar lines in figure 4.

3.3 Normalized Eight-point Algorithm with RANSAC

In order to improve the algorithm, we used RANSAC to filter out outliers. For this approach we used multiple subsets of 8 normalized matches to obtain a fundamental matrices, which are then used to compute the Sampson distance for each of the matches in the entire set. When this Sampson distance is below a certain threshold, a pair is considered an inlier. Finally, the largest set of inliers is used to compute the final fundamental matrix, which is applied on the points to obtain the results in figure 5. The Sampson distance threshold and the number of sampling iterations are tuned such

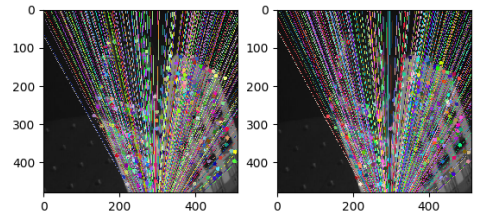


(a) Image 2 and 3

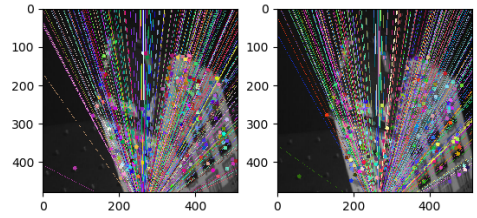


(b) Image 7 and 15

Figure 3: Results Normal Eight-Point Algorithm. Left: first image with epipolar lines corresponding to points on second image. Right: second image with epipolar lines corresponding to points on first image.



(a) Image 2 and 3



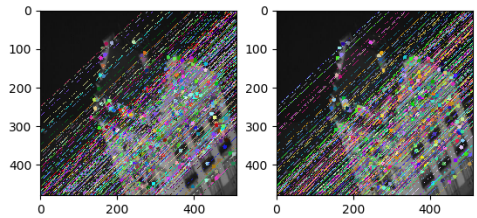
(b) Image 7 and 15

Figure 4: Results Normalized Eight-Point Algorithm. Left: first image with epipolar lines corresponding to points on second image. Right: second image with epipolar lines corresponding to points on first image.

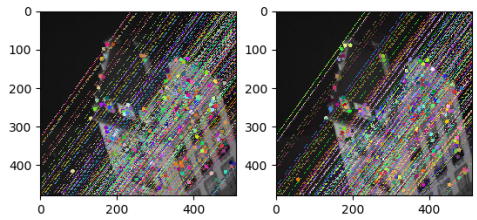
that the remaining number of inliers (figure 1) results in the most optimal epipolar lines.

	Original	After RANSAC
Image 2 and 3	357	159
Image 7 and 15	270	244

Table 1: Number of inliers after applying RANSAC.



(a) Image 2 and 3



(b) Image 7 and 15

Figure 5: Results Normalized Eight-Point Algorithm with RANSAC. Left: first image with epipolar lines corresponding to points on second image. Right: second image with epipolar lines corresponding to points on first image.

3.4 Analysis

When we compare the Normal and Normalized Eight-Point Algorithm results (figure 3 and 4), we see that using Normalized points improves the results. For instance, without normalizing the points, the epipolar lines do not intersect the points in image 2 and 3, however they do when using a fundamental matrix based on normalized points. For image 7 and 15, the improvement is even more significant, which might be caused by the bigger discrepancy between the two image viewpoints. The obtained benefit from using normalized points is also observed when looking at the epipolar constraints. As can be seen in table 2, in both cases, the average of AF over all matches does not equal 0, nevertheless the Normalized Eight-Point algorithm approaches this constraint more than the Normal Eight-Point algorithm.

The results get even better when filtering out the outliers. When looking at the results from the Normalized Eight-Point Algorithm with RANSAC (figure 5) we can see that the epipolar lines are very similar to the ones from the benchmark fundamental matrix from OpenCV (figure 1). As for the epipolar constraints, the average AF approaches 0 a bit more than when not using RANSAC, however this difference is relatively small and might not be significant.

	Normal	Normalized	Normalized with RANSAC
Image 2 and 3	0.04780	0.00073	0.00046
Image 7 and 15	1.22053	0.00816	0.00050

Table 2: Eight-Point Algorithm constraint results. Each value being equal to the dot product between A and the fundamental matrix (AF), averaged over all matches

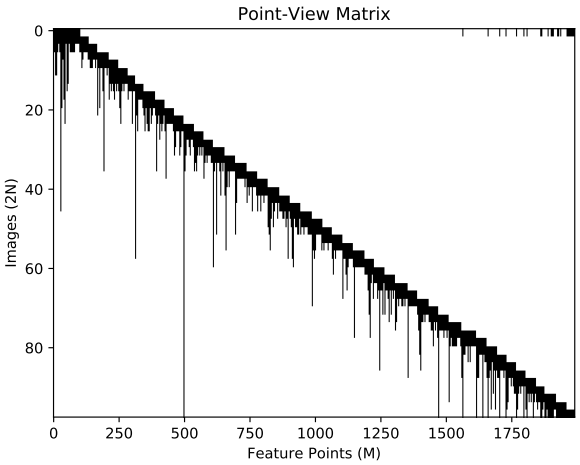


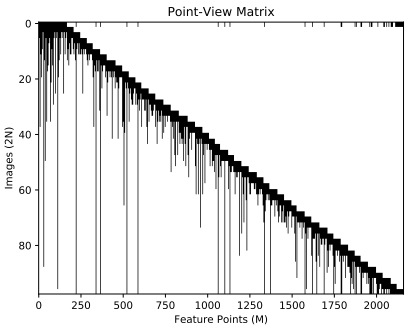
Figure 6: Representation of the generated point-view matrix

4 CHAINING

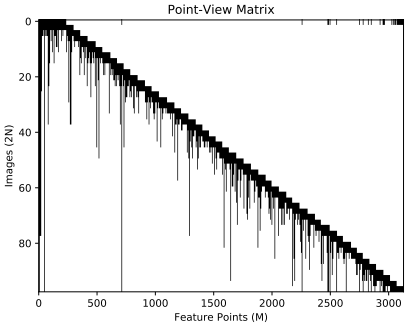
The generating of matches as described in the previous section can be done over multiple pairs of images, after which a so-called point-view matrix can be generated. In this matrix, rows represent the images and columns represent surface points (here in the form of (x,y) coordinates of each point). This creates a (sparse) matrix where one can find where each point occurs in which image.

For the construction of this matrix, a Pandas dataframe is used. The keypoints and their corresponding descriptors for two consecutive images are found using SIFT and matches between descriptors are found using OpenCV's Brute Force Matcher with an L1 norm. The found matches are sorted and only the best 25% are kept. We then iterate over all the resulting matches to determine their position in the point-view matrix. We find their correct position by checking whether there are already any points in the matrix that lie within a certain margin (in our case 20) of this new point. When this is the case, the new point is added to the column of the point that is close to it. When this is a completely new feature point, a new column is added to the matrix. The x and y coordinates are each added to their own row, resulting in a $2MxN$ matrix. A visualisation of the generated point-view matrix can be found in Figure 6.

This figure shows a relatively sparse matrix, but it still follows the structure one would expect. Because of the changing camera view, it is reasonable that we do not see every point in all images. However, we think that this matrix could be more dense. Hence, we attempted to play around with the distance threshold and the criterion for what is considered a "good" match, of which the results can be seen in Figure 7. When allowing for more points to be considered a "good" match, we noticed that the matrices became somewhat more dense. However, one should keep in mind that using all points is not necessarily a good thing. That is, when using all matches, matches that are present in the background of the image are also considered. Consequently, when using this point-view matrix for generating structure from motion, this can result in the reconstruction of points that are irrelevant for the structure. In the end, we think that using 50% of the points is a good compromise. This percentage results in a relatively dense matrix, while still filtering out irrelevant points.



(a) Point-view matrix generated with brute forced L1 matching using 50% of matches



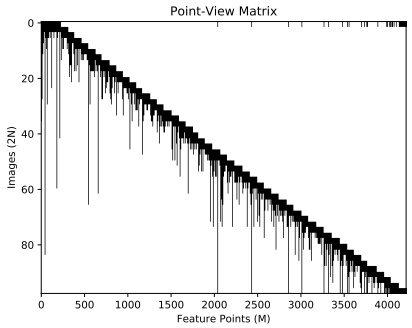
(b) Point-view matrix generated with brute forced L1 matching using 100% of matches

Figure 7: Point-view matrices generated with different percentage of matches.

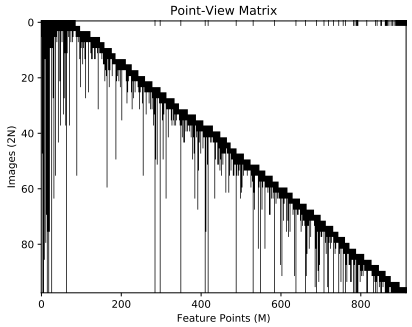
Another metric that could lead to a more dense matrix is the threshold that is used to determine if two points are the same (or sufficiently similar). Our default threshold was 10, but in Figure 8 we also tested a threshold of respectively 1 and 20. From these results we can see that the increasing the threshold makes the matrix more dense, but also decreases the number of points (as is expected). In order to obtain a balance in this trade-off between density and number of points, we stuck to our default of 10.

Furthermore, there are more methods for increasing the density of this point-view matrix. For instance, one could consider using a different method for finding feature points, or using a different method of finding matches. This will be more closely investigated in Section 6.

We were given a file (PointViewMatrix.txt) to check the correctness of our own matrix. A representation of the matrix from this benchmark matrix can be found in Figure 9. As can be seen, this matrix is much more dense. When using this benchmark matrix for affine structure from motion, no sampling is necessary and decomposition is directly possible. Our matrix on the other hand, is much more sparse. Some points are visible in almost all images (camera views), but most are only visible in a few. This means that



(a) Point-view matrix generated with brute forced L1 matching using 50% of matches, viewing points as similar when 1 away



(b) Point-view matrix generated with brute forced L1 matching using 50% of matches, viewing points as similar when 20 away

Figure 8: Point-view matrices generated with different definitions of similar.

if we want to derive structure from motion, we first need to sample dense blocks, as we will elaborate on in the next section.

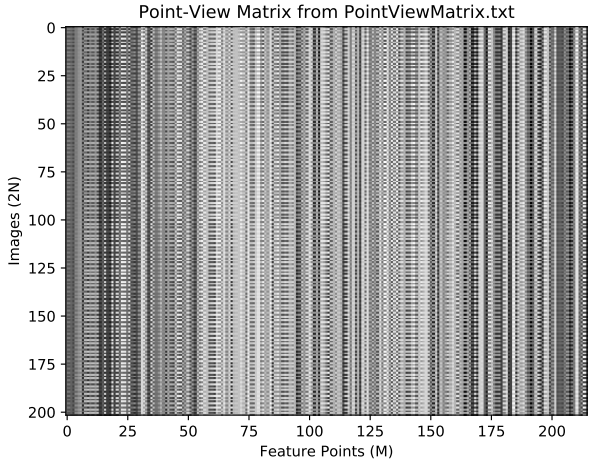


Figure 9: Representation of the point-view matrix from PointViewMatrix.txt

5 STRUCTURE FROM MOTION

For this part, we used the point-view matrix generated with Brute Force matching, using 50% of all found matches, and a distance threshold of 10.

In order to generate a point cloud, we take a dense point-view matrix and first normalize it by subtracting the mean and then computing a construction and motion matrix. The construction matrix is then visualized using Opend3D. For analysis purposes, we first used the entire provided benchmark point-view matrix as a dense matrix, resulting in the point cloud in figure 10b.

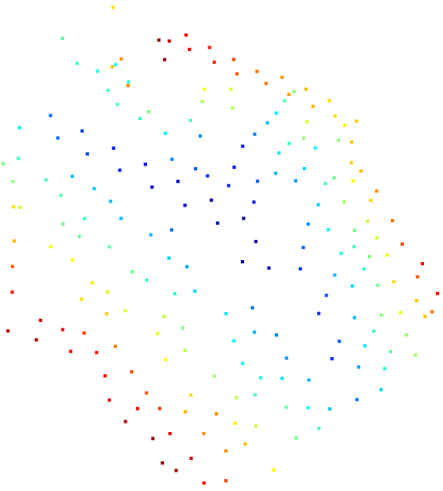
For our own point-view matrix, we obtain dense matrices by iterating over all rows and columns in the point-view matrix and for each 3 consecutive frames, taking all the consecutive columns containing values as the dense matrix. When using the first dense matrix of shape (6,62), we obtain the point cloud in figure 10a. Given the relative small size of this dense matrix, the result is much sparser then the benchmark point cloud.

We did not have time to construct the entire point cloud by stitching all frames. However, if we would have, we would first have obtained the dense matrices for all 3 consecutive frames and stored them in a list. Then, we would use the first matrix as the starting point for the complete point cloud. After, starting from the first two matrices, we would compute the construction matrices and subsequently compute the rotation and translation between the two using the ICP algorithm. This rotation and translation will then be used to transform the complete point cloud, upon which the second frame is stacked. This process is repeated until all frames are stacked and transformed to the same view point, after which the final complete point cloud is visualized.

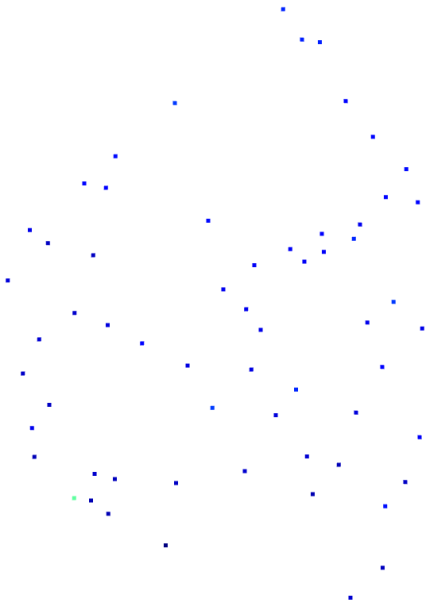
An alternative method of stitching the frames is to use procrustes analysis. For this approach, we first store all the calculated structure information in a list and save the indices in the point-view matrix from which the corresponding dense blocks are constructed. Then, we iterate over the list with structure information and for each consecutive point, find the overlapping indices from the corresponding

dense blocks. Procrustus analysis is then performed on these two overlapping entries.

From this procrustus analysis we would get a translation matrix with which we can translate a single structure entry. We would then stack this with previous transformed entries in order to construct a complete 3D reconstruction. However, due to the time constraint we were unable to finish the complete stitching implementation. Instead we constructed the point cloud for the first 3 consecutive, which can be seen in Figure 11.



(a) benchmark point-view matrix



(b) first 3 frames of constructed point-view matrix

Figure 10: Point cloud based on the benchmark point-view matrix (a) and the first 3 frames of the constructed point-view matrix using ICP (b).

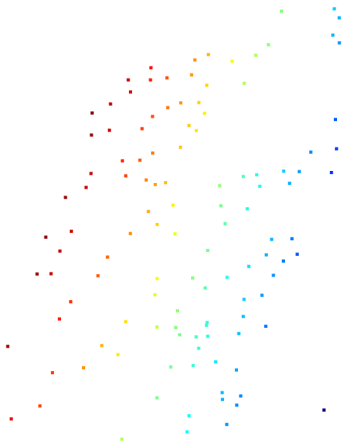
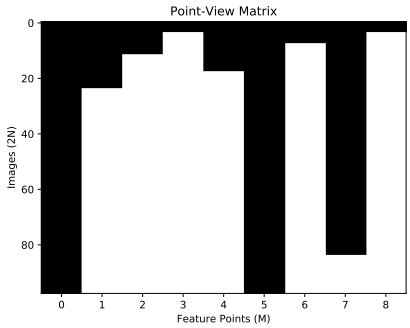


Figure 11: Point cloud based on the first 3 frames of the constructed point-view matrix using Procrustes analysis.

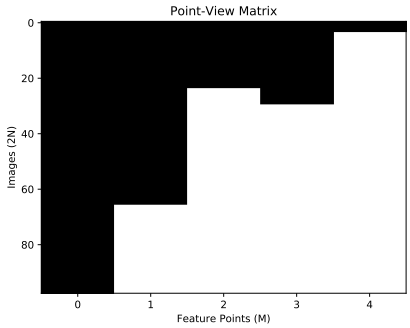
6 ADDITIONAL IMPROVEMENTS

A more dense point-view matrix can result in easier and more accurate performance of the structure from motion algorithm. As mentioned before, we attempted to increase the density of the point-view matrix by using a different method for finding matches. More specifically, we used the OpenCV Flann feature matching algorithm and to determine whether two points are the same, we used our default threshold of 10. Furthermore, we also experimented with using only a percentage of the matches. Results can be found in Figure 12.

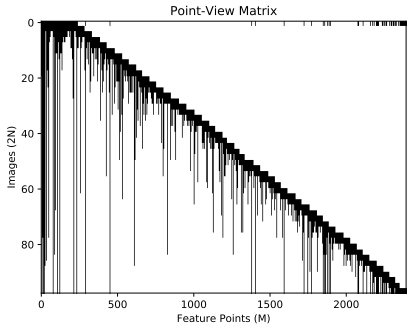
The figure clearly shows that, when using less than 100% of matches, the point-view matrix does not have the shape one would expect. When using all matches on the other hand, the matrix becomes more dense than any matrix generated with Brute Force matching. However, once again, note that using all matches might result in a poor 3D reconstruction using the structure from motion algorithm due to the added background bias.



(a) Point-view matrix generated with brute forced L1 matching using 25% of Flann matches



(b) Point-view matrix generated with brute forced L1 matching using 50% of Flann matches



(c) Point-view matrix generated with brute forced L1 matching using 100% of Flann matches

Figure 12: Point-view matrices generated with Flann matching and different percentages of matches used.

7 CONCLUSION

To conclude, the Eight-Point algorithm worked best when normalizing the points and filtering out the outliers using RANSAC. When constructing the point-view matrix, we see that the result is always very sparse. In order to increase the density we experimented with several approaches, of which using an alternative feature matching algorithm proved most beneficial. For structure to motion, we used a dense block from the constructed sparse point-view matrix, which resulted in a very sparse point cloud, where nevertheless a slight outlines of the house model is visible.

A SELF-EVALUATION

We divided the work fairly evenly among the two of us. Tim worked mainly on the fundamental matrices, and Emma on the chaining. For the structure to motion, we mainly worked together.

REFERENCES