Computer Vision 2 - Structure from Motion

Maurice Frank - 11650656 (UvA)
maurice.frank@posteo.de

David Biertimpel - 12324418 (UvA)
david.biertimpel@student.uva.nl

Leonardo Romor - 12261734 (UvA)
leonardo.romor@student.uva.nl
(All contributed equally)

May 10, 2019

Introduction

Structure from Motion [7] comprises a range of techniques that aim to reconstruct both the camera motion and the 3D shape of objects in a given scene. The original technique uses popular methods such as *SIFT*[5], *SURF*[1] or *ORB*[6] to extract key points in consecutive 2D images and associates them across the complete sequence. During this process, the affine transformation between each image pair is estimated and represented with the fundamental matrix $\boldsymbol{F}$. Finally the position of the points in 3D space is inferred. In the following we first explore the process of constructing $\boldsymbol{F}$ by using three different versions of the *eight-point algorithm*. Besides implementing the vanilla version, we further improve its performance by normalizing the input points and finally using *RANSAC* to draw preferably outlier-free samples. To realize the Structure from Motion algorithm, we first construct a *Point View Matrix*, which connects the found key points over successive frames and then implement the algorithm with steps based on [8]. Finally, we aim to further improve our results by, among other things, making the *Point View Matrix* denser and removing affine ambiguities.

Implementation: Over the course of the experiments we use *ORB* [6] and AKAZE [4] for finding keypoints and *OpenCV*'s *BFMatcher* for matching these key points. Our code is implemented in *Python* with *Numpy*, *OpenCV* and *Matplotlib* as our main dependencies.

1 Fundamental Matrix

The fundamental matrix $\boldsymbol{F}$ relates two points p and p' coming from two coordinate systems defined by two different camera perspectives. Considering the original point in the 3D world scene's coordinate system, $\boldsymbol{F}$ constraints the location of p and p' on their respective image planes. $\boldsymbol{F}p$ describes a line on p''s image plane along which p' must lie. The same applies simultaneously to p's image plane given $\boldsymbol{F}p'$. These lines are called *epipolar lines* and the constraint the *epipolar constraint*. This *epipolar constraint* leads to the equation $p'^T \boldsymbol{F} p = 0$, which should hold for every point on the two image planes and defines the fundamental matrix. Based on these definitions we will estimate the fundamental matrix $\boldsymbol{F}$ and find *epipolar lines* for a given image pair.

To estimate $\boldsymbol{F}$ we will explore different approaches involving the *eight point algorithm*. First, we use the plain *eight-point algorithm* as a baseline method. To improve performance, we then normalize the input points using a technique introduced by Richard Hartley [2]. Finally, we use *RANSAC* on this *normalized eight-point algorithm* to further improve and stabilize our results.

For assessing the *eight-point algorithm*'s performance, we use the *average error* from Hartley's paper [2], which is the average distance of a key point to its corresponding *epipolar line*. Note that here we consider all key points in the image. To get a more representative impression, we run the algorithm 100 times and calculate the *average error* each iteration. Afterward, we take the fundamental matrix $\boldsymbol{F}$ corresponding to the smallest error and plot the key point's *epipolar lines* in the respective images. The *average errors* can be seen in figure 1 (For better visibility, the error values are sorted in descending order) and the plots of the *epipolar lines* in figure 2. We discuss all of the obtained results at the end of this section (see 1.4). For the experiments we use the first and second image of the house data.

1.1 Eight-point Algorithm

In order to estimate the fundamental matrix $\boldsymbol{F}$ with the *eight-point algorithm*, we first find *ORB* key points in both images and find matches between them. After randomly sampling 8 matching key points, we construct the matrix $\boldsymbol{A} \in \mathbb{R}^{8\times 9}$ in which the i-th row has the structure

$$\boldsymbol{A}_{i:} = \begin{bmatrix} x_i x_i' & x_i y_i' & x_i & y_i x_i' & y_i y_i' & y_i & x_i' & y_i' & 1 \end{bmatrix}.$$

We can infer $\boldsymbol{F}$ by finding the *singular-value decomposition* (SVD) of $\boldsymbol{A}$ while ensuring that $\boldsymbol{F}$ is rank deficient.

1.2 Normalized Eight-point Algorithm

Richard Hartley found in his paper ([2]) that the performance of the *eight-point algorithm* can be considerably improved by normalizing the data. This normalization process consists of two parts. First we shift the data such that the centroid of the key points is at the the image's origin. Second, the data is scaled such that the average distance from the points to the origin is $\sqrt{2}$. This scaling factor is isotropic so that the coordinates of a point are scaled equally.

This normalization is achieved by constructing the matrix

$$\boldsymbol{T} = \begin{bmatrix} \sqrt{2}/d & 0 & -m_x\sqrt{2}/d \\ 0 & \sqrt{2}/d & -m_y\sqrt{2}/d \\ 0 & 0 & 1 \end{bmatrix},$$

where m_x is the mean of the points x component, m_y the mean of the y component of the points and d is the square root of the average combined distance of x and y to m_x and m_y respectively. We construct two matrices $\boldsymbol{T}$ and $\boldsymbol{T'}$ one for each image. After obtaining $\boldsymbol{T}/\boldsymbol{T'}$ we can normalize the key points with $\hat{p} = \boldsymbol{T}p$ for the first, and $\hat{p}' = \boldsymbol{T'}p'$ for the second image. Afterward, we apply the *eight-point algorithm* as before with the exception that $\boldsymbol{F}$ needs to be denormalized in the end.

1.3 Normalized Eight-point Algorithm with RANSAC

Although normalization is expected to significantly improve the *eight-point algorithm*, we can still draw an unfavorable sample containing outliers and get a distorted result. For this reason, we use the *RANSAC* algorithm, we already know from the fourth Computer Vision 1 assignment, to find a $\boldsymbol{F}$ coming from a preferably outlier free sample. The only major difference compared to the Computer Vision 1 assignment is the way we detect outliers. Instead of detecting outliers based on the accuracy of an affine transformation, we now use the Sampson distance

$$d_i = \frac{(\hat{p_i}'^T \boldsymbol{F} \hat{p_i})^2}{(\boldsymbol{F}\hat{p_i})_1^2 + (\boldsymbol{F}\hat{p_i})_2^2 + (\boldsymbol{F}\hat{p_i})_1^2 + (\boldsymbol{F}\hat{p_i})_2^2}$$

to see if two points $\hat{p_i}$ and $\hat{p_i}'$ agree on the fundamental matrix $\boldsymbol{F}$. We run *RANSAC* in total for 100 iterations and consider each matched point pair with a d_i value larger than 0.05 as an outlier. In the experiments we only evaluate the *RANSAC* approach with the *normalized eight-point algorithm*.

1.4 Results

Looking at the results for the *average error* in both Table 1 and Figure 1, we immediately notice that the *RANSAC* based approach achieves dramatically better and more consistent results. While the minimum of the *average error* of all approaches is on a comparable level, both approaches without using *RANSAC* have notably higher maximum values. This is reasonable as the eight input points for the algorithm are drawn randomly and the *eight-point algorithm* and its normalized version have no means to detect and cope with outliers. However, it should be noted that the *eight-point algorithm*'s maximum error is more than four times higher than that of the *normalized eight-point algorithm*, which underlines that the normalization of the input values considerably improves the algorithm's performance.

	Min	**Average**	**Max**
Eight-point Algorithm	1.9386	7.6322	227.1484
Normalized Eight-point Algorithm	2.0271	5.3183	55.1653
Norm. Eight-point Algorithm w. Ransac	1.7413	2.0937	2.5318

Table 1: Mininimum, average and maximum values of the *average error* across the different methods.

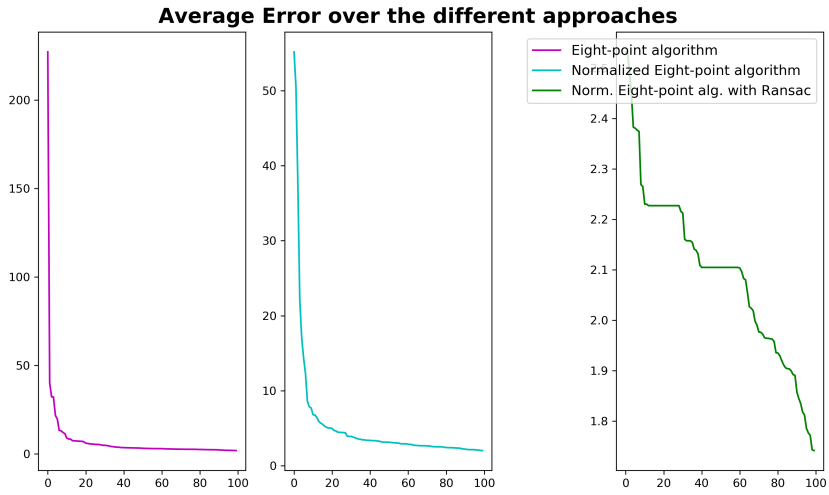

Figure 1: Visualization of the distribution of *average errors* produced by the three algorithms. It should be noted that the error values were sorted in descending order for better visualization. The x-axis therefore does not represent a temporal dimension.

For each approach, we now consider the *epipolar lines* estimated with the $\boldsymbol{F}$ corresponding to the lowest *average error* from the preceding evaluation process. The associated visualizations can be observed in Figure 2.

During our experiments we observed a high variance in the *epipolar lines* and corresponding *epipoles* both across different runs and the three methods (the *epipole* is randomly varying between 4-5 positions). In the following we distinguish between the correctness of $\boldsymbol{F}$ by considering the condition $p'^T \boldsymbol{F} p = 0$ and if the resulting position of the *epipolar lines* and *epipole* is actually reasonable.

The question whether the condition $p'^T \boldsymbol{F} p = 0$ holds is mostly correlates with the *average error* from before. If we draw a sample of matching points without any outliers, the values resulting from $p'^T \boldsymbol{F} p$ are constantly very close to 0. As before *RANSAC* is producing by far the best results.

When observing the *epipolar lines* in Figure 2, we notice that they do not necessarily reflect the transformation between the two images. For example, we would rather expect the results of the Figure 2b and 2c in case the camera moves towards the house, as illustrated in Figure *9.8* in *Multiple View Geometry in Computer Vision Second Edition* [3].

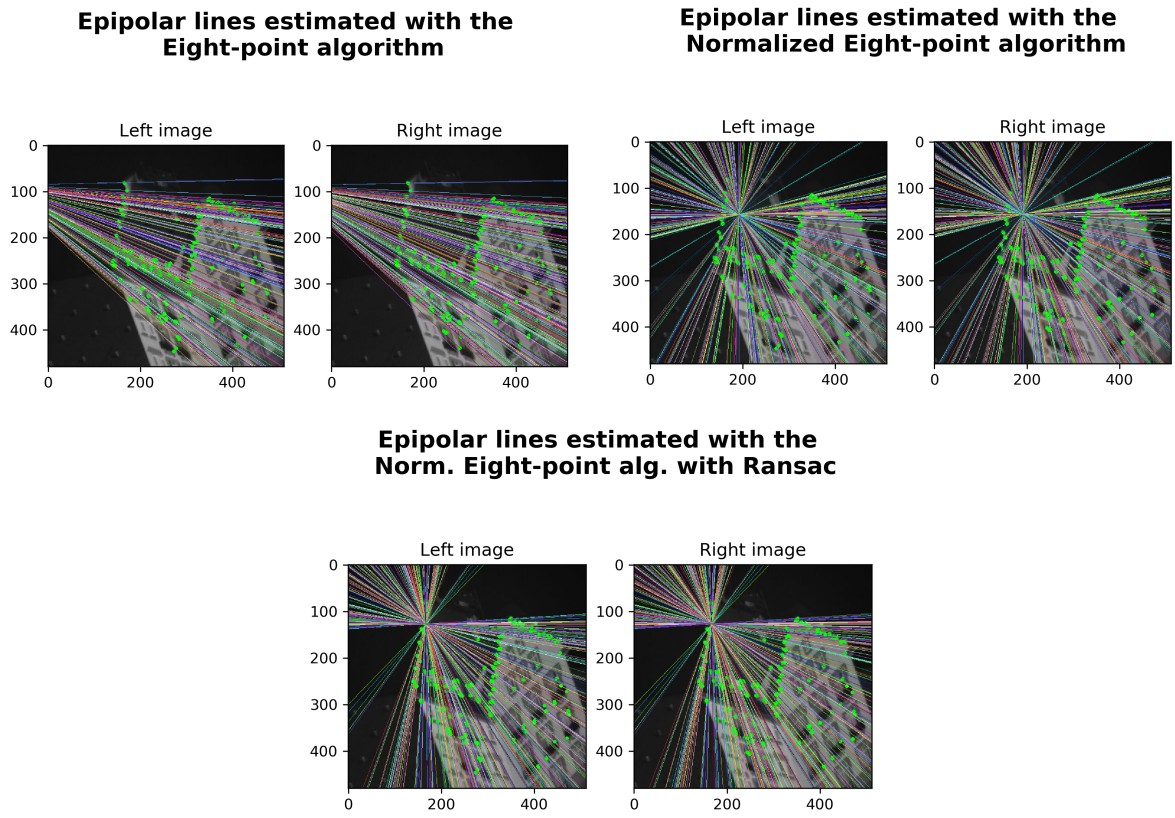

Figure 2: Illustration of the estimated *epipolar lines* by the three approaches.

Although we were still in an active discussion about the interpretation of the *epipolar lines* in this context, after closer examination of the found matches, we decided to explain the fluctuations in $\boldsymbol{F}$ as follows. Due to the fact that the two consecutive images are very similar, the underlying transformation is not strongly reflected by all key points. This small change between some key points leads to a situation in which, in case of a bad sample, the *eight-point algorithm* may underestimate the movement between the images. Furthermore, if the change between the key points is small, the estimated $\boldsymbol{F}$ is easily distorted by noise. This observation is underlined by looking at the *epipolar lines* obtained when comparing the first and 40^{th} image from the dataset (see Figure 3). Here the difference between the images is more pronounced and consequently we observe the estimated *epipolar lines* to be substantially more similar both across several runs and the three approaches.

2 Chaining

After finding the relationship between matching points in an image pair, we need to relate the key points across multiple views to estimate their underlying 3D structure. In order to do this, we create a sparse *Point View Matrix* by chaining matching key points across images and building up trajectories of key points. In our chaining method we go through all images and compute the keypoint and their respective descriptors. We use AKAZE [4] features in the chaining experiments. Compared to ORB they showed better performance and we still avoid non-FOSS software. Next we match those features with the ones from the previous frame. Here we add three levels of filters to filter out unwanted matches. Instead of just matching the closest point we do a K-Nearest Neighbour search ($k = 4$). From this quadruple we keep the match with the lowest spatial distance in the image space. As there are recurring image features like the windows one descriptor might be closely similar to multiple points. With this first filter we take similarity in the feature and image space into account at the same time. Further we have a general distance filter. Matches with an euclidean distance bigger the threshold are

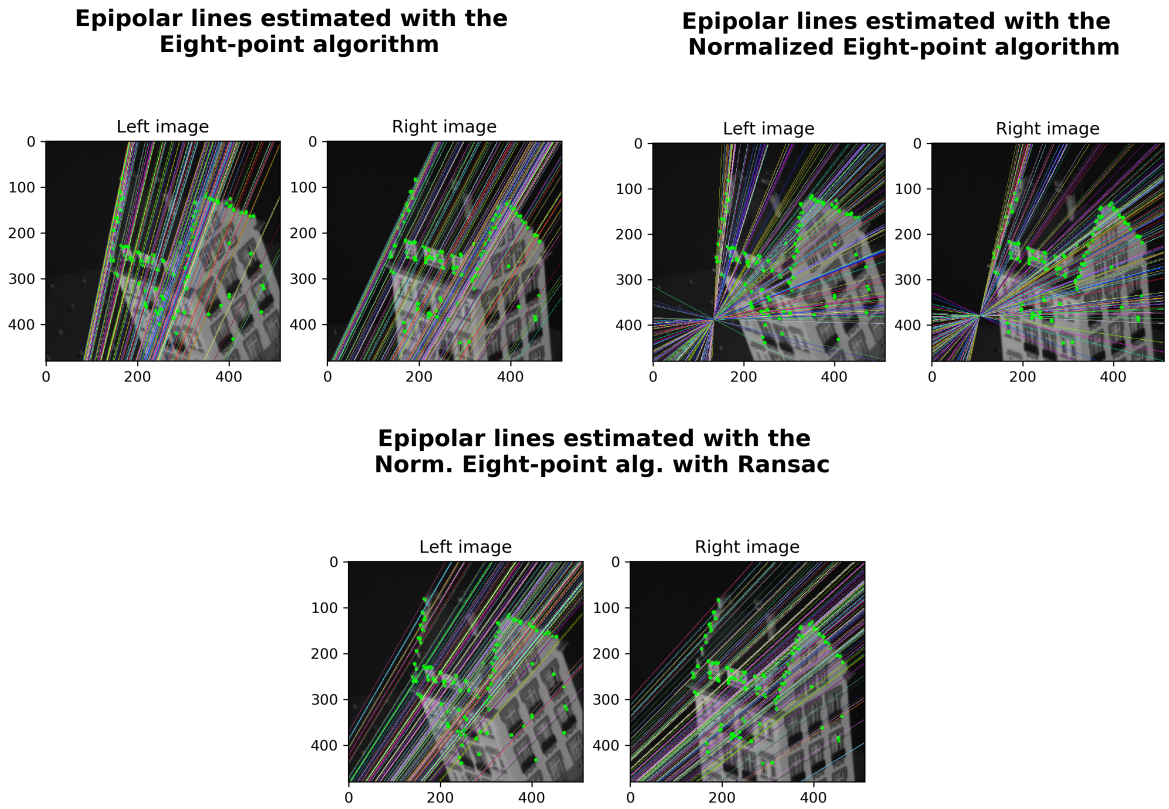

Figure 3: Illustration of the estimated *epipolar lines* by the three approaches using the first and 40^{th} image from the data.

removed ($\mathbf{thres}_2 = 20$ (Figure 4).

Finally we test to filter using RANSAC (Figure 7). For this we build the fundamental matrix using the two matching point clouds. We remove matches for whom:

$$p_{\text{right}}^T \boldsymbol{F} p_{\text{left}} < \text{thres}_2$$

With the right and left point being in the current and previous frame, respectively. We set $\text{thres}_2 = 0.01$. With the remaining matches we fill the point-view matrix. If point was set in the previous frame we add the position in the current frame to the column otherwise we add a new column. After building the matrix we use one last filter. We remove columns that have chains shorter than $\text{thres}_3 = 20$ meaning we remove chains that only survived for more than 20 frames (Figure 5).

The results show that our filtering methods do improve the tracking of the keypoints as anticipated. Filtering by chain length reduces the matrix to a more dense one. Filtering matches by distance keeps chains from jumping around too much. The results from RANSAC filtering are two-fold. On the one hand it does reduce the chains as such that all chains move clearly coherently and the set shows only one movement over time. On the other hand this does reduce the number of tracked keypoints a lot and thus actually reduces the quality of the reconstruction.

3 Structure from Motion

In terms of the implementation of the *Structure from Motion* method, we closely orientated ourselves to the formulas given in *Jan van Gemert*'s lecture. We iteratively find dense blocks $\boldsymbol{D}$ in the *Point View Matrix*, which we normalize first by subtracting the mean of each row from each element in

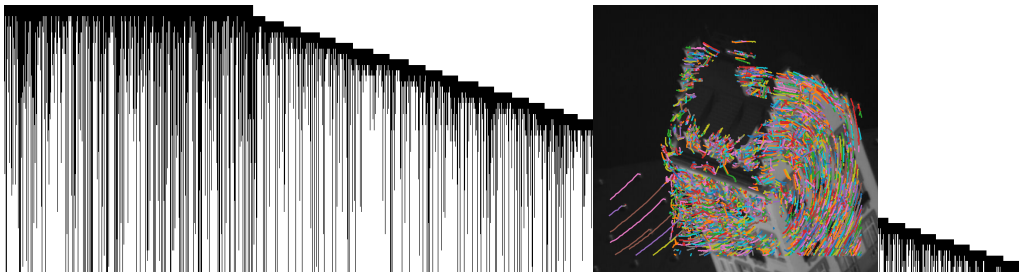

Figure 4: *Left:* Point-view matrix. The columns denote the keypoints and each row is one frame. *Right:* We superimpose the chains of tracked points on the first frame. Here we used the spatial distance filter but not the chain length filter. Also no RANSAC was used. Note how there is a lot of columns in the matrix but most chains are visibly really short.

Figure 5: Same as in Figure 4 but in here we do not use the spatial distance filter but instead the chain length filter with minimum length 20. We see that the matrix got dense and we only have long chains but many chains show jumping behaviour.

Figure 6: This shows the results when chain length filter and distance filter are combined. Note how the movement in the superimposed image of the chains is coherent.

Figure 7: Lastly we add RANSAC filtering. We use the normalized Eight Point algorithm as described before.

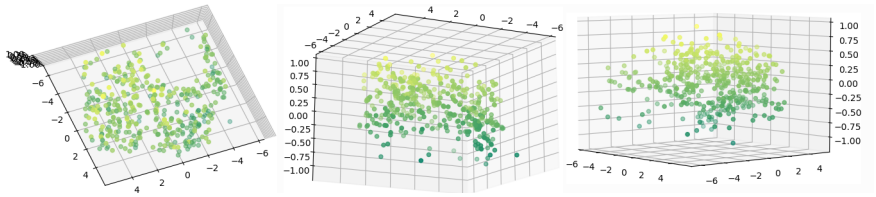

Figure 8: Results of *Structure from Motion* using only the first block from or own PVM. From *left* to *right*: Front, Top, Side.

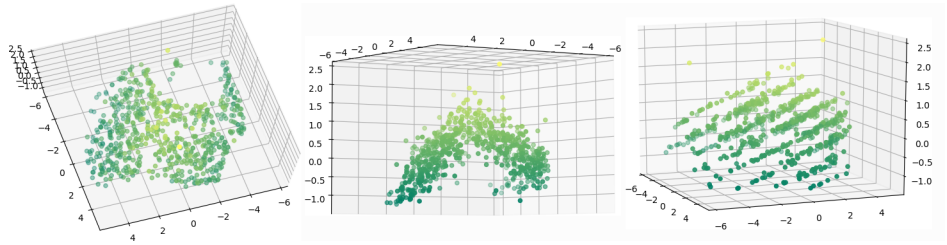

Figure 9: Results of *Structure from Motion* using our own PVM and using dense block of 3. From *left* to *right*: Front, Top, Side.

$\boldsymbol{D}$. Subsequently, we perform a SVD on $\boldsymbol{D}$ and receive $\boldsymbol{UWV^T} = \boldsymbol{D}$. To satisfy the constraint of $rank(\boldsymbol{D}) = 3$, we alter $\boldsymbol{U}, \boldsymbol{W}$ and $\boldsymbol{V}$ the following way:

$$\boldsymbol{U}_3 = \boldsymbol{U}_{:,:3},\ \boldsymbol{V}_3 = \boldsymbol{V}_{:,:3},\ \boldsymbol{W}_3 = \boldsymbol{W}_{:3,:3}$$

Afterward, we create the shape and motion matrices $\boldsymbol{S}$ and $\boldsymbol{M}$ by performing

$$\boldsymbol{M} = \boldsymbol{U}_3\sqrt{\boldsymbol{W}_3},\ \boldsymbol{S} = \sqrt{\boldsymbol{W}_3}\boldsymbol{V}_3^T.$$

Finally, we use either $\boldsymbol{S}$ or in the case of affine ambiguity removal both $\boldsymbol{S}$ and $\boldsymbol{M}$ to transform new points to the world coordinate system. For mapping the selected points in $\boldsymbol{S}$ to the world coordinate system, we use a Python version of *Matlab*'s *procrustes* function.

We now present and discuss the results from building a 3D model with our *Structure from Motion* approach. In Figure 11 we see the point cloud for the provided dense matrix. For our own chained key point trajectories, we use all improvements listed in Section 2 without the *RANSAC* as this gave us the best reconstruction visually. See the best estimated structure in Figure 10.

In both structures we can recognize the house from the image. The dense matrix shows a clearer structure with fewer points. In contrast, our own reconstruction is a more crowded point cloud that follows the general shape of the house, but is less sharp in details. If we use only the first dense block of 3 steps (Figure 8), we already see the general house shape, although it is quite noisy and we mostly miss the depth in which we are actually interested.

Comparing the final result between steps of 3 (see Figure 9) and 4 (see Figure 10), we observe that using the bigger block size does yield slightly better results. Interestingly, for steps of 3 we see on the z-axis that the points are not spread continuously, but lie on different levels. Unfortunately, we could not make out the reason for this.

4 Additional improvements

We already described several improvements to the chaining in section 2. There we firstly removed far apart matches base on euclidean distance in the image space, second used KNN matching to pick matches coincidentally on their distance in feature and image space, thirdly removed incoherent

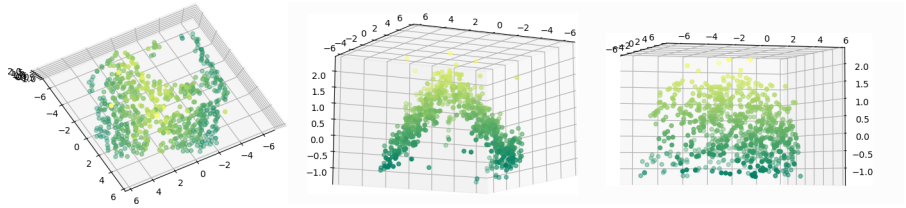

Figure 10: Results of *Structure from Motion* using our own PVM and using dense block of 4. From *left* to *right*: Front, Top, Side.

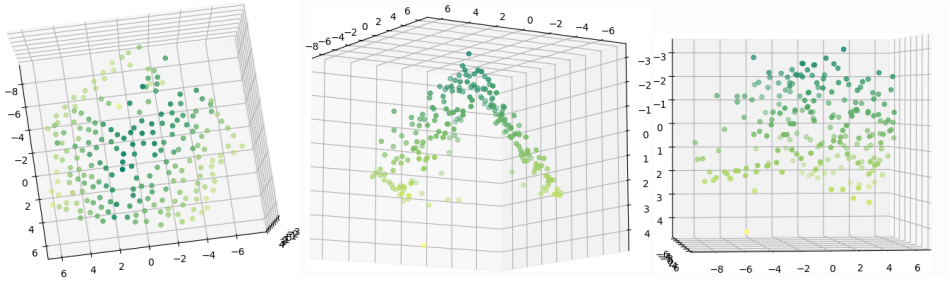

Figure 11: Result of *Structure from Motion* using the provided PVM. From *left* to *right*: Front, Top, Side. The color encodes the z-value.

matches using an estimation of the fundamental matrix and lastly we preprocessed our point-view matrix by removing short chains.

For the structure from motion we also implement the elimination of affine ambiguity. This we do by:

$$M = U \cdot \sqrt{W} \tag{1}$$

$$L = M^{\dagger} \cdot \mathcal{I} \cdot (M^T)^{\dagger} \tag{2}$$

$$C \cdot C^T = \text{cholesky}(L) \tag{3}$$

$$S = C^{-1} S \tag{4}$$

where U and W are from the SVD.

We add this improvement to our structure from motion method. In Figure 12 we see the new reconstructed structure with affine ambiguity removal for our previously best estimation. We clearly see the quality of the reconstruction actually degraded. On the x- and y-axis we have considerably larger scales and the structure shows less clearly the house.

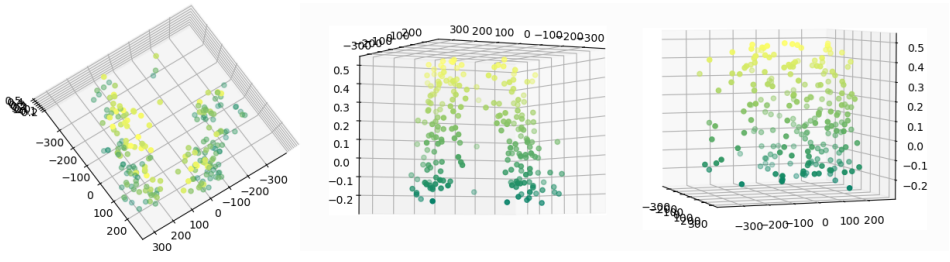

Figure 12: Reconstructed cloud point of our point-view matrix using elimination of affine ambiguity.

Conclusion

During this assignment we went through several stages on the way to implementing *Structure from Motion*. In the first part we estimated the fundamental matrix $\boldsymbol{F}$ using the *eight-point algorithm* and estimated *epipolar lines* based on $\boldsymbol{F}$. While the *normalized eight-point algorithm* performed better than its vanilla counterpart, both were susceptible to outliers and only the *RANSAC* based version achieved consistently good results. We further found that the position of the *epipole* and *epipolar lines* fluctuated over different runs, which we attributed to the high similarity between successive images and a resulting susceptibility to noise. As an additional preparatory step for realizing *Structure from Motion*, we implemented a chaining method that creates trajectories of matching key points to track their positions. There we saw that filtering is of utter importance. With several filters, we were able to create a set of long, coherently moving key point trajectories. Finally, we used these key point chains to reconstruct a 3D model of the object shown. While we were able to retrieve an fairly good shape, we saw that any noise quickly deteriorates performance.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] R. I. Hartley. In defence of the 8-point algorithm. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1064–1070, June 1995.

[3] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[4] Yuxuan Liu, Chaozhen Lan, Canhai Li, Fan Mo, and Huabin Wang. S-akaze: An effective point-based method for image matching. *Optik*, 127(14):5670–5681, 2016.

[5] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, Sep. 1999.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

[7] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, Nov 1992.

[8] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: A factorization method. *Int. J. Comput. Vision*, 9(2):137–154, November 1992.