Computer Vision 2 Assignment 2

Emma Quist 11706724, Haohui Zhang 14124262, Faye Raaijmakers 11848669

May 13, 2022

1 Introduction

This report will focus on the implementation of Structure from Motion (SfM). SfM is the technique used in computer vision to construct 3D structures from 2D images. We were provided with a dataset of 49 images of a house, all portrayed from a slightly different angle. The first part of this report will focus on implementing the 8-point algorithm to compute the fundamental matrix between two images. Following, we will construct the point-view matrix, which represents the matches for all 49 consecutive pairs of images in the dataset. Then, we will use this point-view matrix for affine SfM. Finally, we will use the COLMAP application which allowed us to explore SfM on real-world data.

2 Fundamental Matrix

The following parts will focus on constructing the fundamental matrix between two images. We do so by implementing the Eight-point Algorithm and two adaptions on this algorithm; the Normalized Eight-point Algorithm and the Normalized Eight-point Algorithm with RANSAC.

2.1 Eight-point Algorithm

First of all, we constructed the fundamental matrix F by implementing the Eight-point Algorithm. The pseudo-algorithm is as follows:

1. Detect keypoints p_i and p_i' in image X and image X'
2. Find matches between p_i and p_i'.
3. Sample 8 matches
4. Construct matrix A (see figure 1)
5. Find SVD of A, $A = UDV^T$
6. The entries of F are the components of the column of V corresponding to the smallest singular value.
7. Find the SVD of F, $F = U_f D_f V_f^T$
8. Set the smallest singular value in the diagonal matrix D_f to zero in order to obtain the corrected matrix D_f'.
9. Recompute F, $F = U_f D_f' V_f^T$

$$\underbrace{\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_nx_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix}}_{A}$$

Figure 1: Matrix A composition.

Following this algorithm, we obtain $F \in \mathbb{R}^{3\times3}$.

2.2 Normalized Eight-point Algorithm

The normalized Eight-point Algorithm uses similar steps as stated in 2.1. However, this time we make sure that the input matches are all normalized and thus have a zero mean and a standard deviation of $\sqrt{2}$. We do so by constructing the following matrix:

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

where m_x and m_y are the respective means of the matches in x and y direction and

$$d = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(x_i - m_x)^2 + (y_i - m_y)^2}. \tag{2}$$

Following we obtain our normalized points by $\hat{p_i} = Tp_i$ and $\hat{p_i'} = T'p_i$. From here on, we can continue the Eight-point algorithm as described in section 2.1, starting at point 3. Following the algorithm, we obtain $\hat{F}$, which we can denormalize by $F = T'^T\hat{F}'T$.

2.3 Normalized Eight-point Algorithm with RANSAC

Finally, we constructed the fundamental using the Normalized Eight-point algorithm as described in 2.2 in combination with the RANSAC algorithm. We did so by using the following algorithm:

- Sample 8 matches: $\hat{p_i} \leftrightarrow \hat{p_i'}$
- Following these 8 matches, construct fundamental matrix F
- For all other remaining matches, compute the agreement d_i of the match to F (see 3).
- If $d_i \ < \epsilon$, d_i is said to be an inlier
- Repeat this process K times and return F with the corresponding highest amount of inliers.

We set $\epsilon = 0.1$ and K = 10000 and d_i is calculated in the following way:

$$d_i = \frac{(p_i'^T F p_i)^2}{(Fp_i)_1^2 + (Fp_i)_2^2 + (F^T p_i')_1^2 + (F^T p_i')_2^2} \tag{3}$$

2.4 Analysis

The following part will analyse the previously described methods on their performance on constructing fundamental matrix F. Figure 2 and 3 show the results of the regular Eight-point algorithm on the frames T=1 and T=18. One thing that stands out immediately, is that the epipolar lines are not consistent. This is due to the random sampling of key-points in the algorithm, which influences the final result. We can observe that the epipolar constraints are not met, as the lines do not pass through the given keypoints. As a matter of fact, we see that the distance between the lines and the keypoints is significantly large in both figures, but especially in figure 2. However, the results in figure 3 do look plausible. This goes to show that the algorithm is dependent on the initial state of the sampled points. Therefore, there is room for improvement and optimization.

Following, the results of the normalized Eight-point algorithm are displayed in figure 4. Similarly to the regular Eight-point algorithm, the results again depended on the initial state of sampled key-points. The epipolar constraint is again not met, as the epipolar lines still do not pass through all the key-points. However, when comparing the results to the Eight-point algorithm, we see that the normalized Eight-point algorithm finds epipolar lines closer to its key-points. In addition to this, the results yielded by the normalized algorithm tend to be more plausible regardless of the sampled points. Results as shown in figure 2 occur way less often. Normalizing the key-points gets rid of the problem caused by the regular algorithm; orders of magnitude difference between column of matrix A. The magnitude difference causes the least-squares approach to yield poor results.

Finally. the results of the Eight-point algorithm combined with RANSAC are shown in figure 5. The first thing that stands out is that due to the RANSAC implementation, the algorithm consistently finds plausible epipolar lines. In addition to this, the epipolar lines now meet the epipolar constraints;

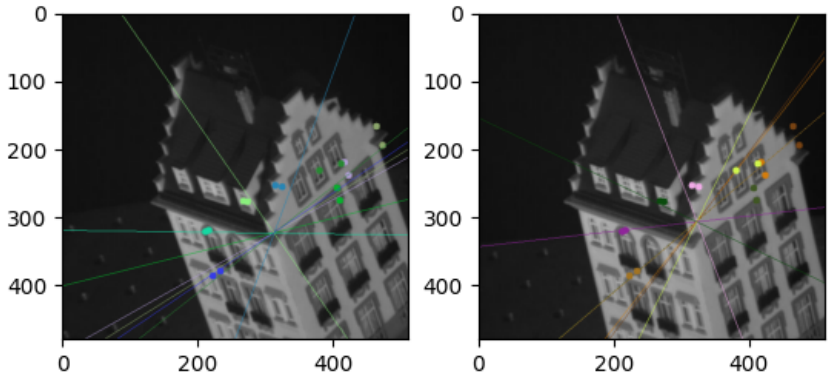


Figure 2: Eight-point algorithm results on T=1 and T=18

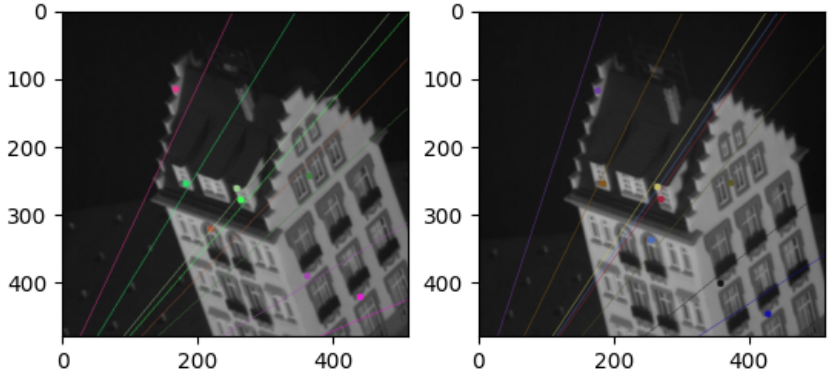


Figure 3: Eight-point algorithm results on T=1 and T=18

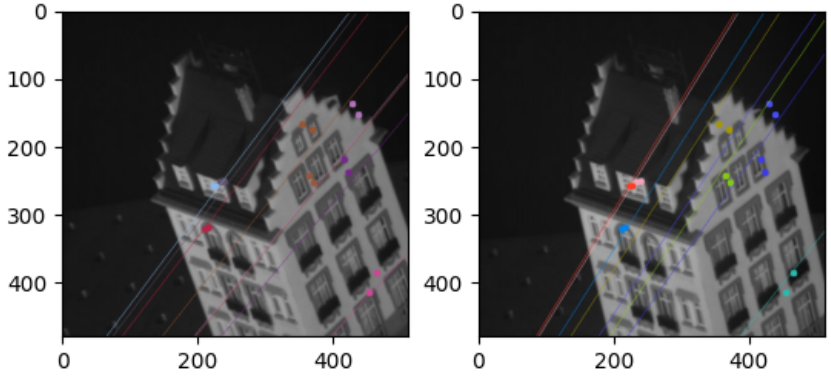


Figure 4: Normalized Eight-point algorithm results on T=1 and T=18

the lines pass through the key-points. This shows that including RANSAC optimizes the Eight-point algorithm.

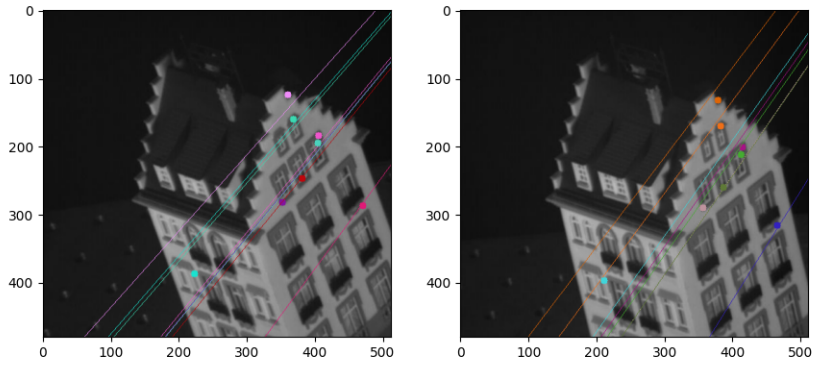


Figure 5: Normalized Eight-point + RANSAC algorithm results on T=1 and T=18

3 Chaining

The previous section focused on finding the fundamental matrix between images pairs. The following section will focus on representing these matches in a single match graph structure.

3.1 Point-view matrix

The goal of this part is to construct a point-view matrix which allows us for chaining all 49 images in the data-set together. We started by finding the matching key-points for the first two images i_1 and i_2. For each match, we added a new column in which we added the coordinates of the key-point. The x- and y-coordinates were added on separate rows. Therefore, our final matrix consisted of 98 rows (2 rows for each consecutive image pair and for the matches between the last and first image). Then, we repeated the same process for i_2 and i_3, while checking whether the newly found match was already present in the matrix. If not, we appended another column with the coordinate values. Otherwise we would just fill in the correct coordinates in the corresponding column. For all matches that were not present in an image, we filled in NaN. We repeated this process for all consecutive image pairs and for the pair of the final image i_{49} with the first image i_1.

A visualization of our point-view matrix is shown in figure 6. Note: our final matrix was of size 98 $\times$ 2892, and thus the proportions of the visualization are incorrect. As can be observed, the matrix is very sparse. This makes sense as we think about the fact that key-points found in image 1 will most likely not appear halfway in the data-set as a result of angle change in the images. In other words, we only match against previously seen matches. In addition to this, newly introduced images will sometimes contain key-points that have not been seen before, thus resulting in a sparse matrix. From here on, we can conclude that our final matrix contains very plausible results.

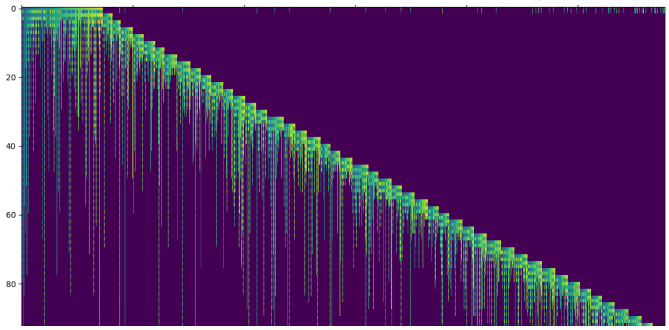


Figure 6: Visualization of the constructed point-view matrix

3.1.1 Improving the point-view matrix

Room for improvement in this matrix lays within the selection of key-points. Currently, we are adding a new column for every new key-point we find. However, the sparsity of the matrix troubles the SfM algorithm as described in the next section, as this algorithm selects dense blocks in the matrix. In addition to this, key-points that are only present in few frames are more likely to introduce noise to the reconstruction. Therefore, we adapted the matrices by deleting the key-points (columns) that were present in less than N frames. The results are shown in figure 7. As can be observed, the newly retrieved matrices are still sparse, but way less sparse than the original matrix. This can be observed by the decrease of the x-axis and the increase of width of the colored parts. Therefore, finding dense regions in the newly retrieved matrices will be easier. The disadvantage of this approach is that we could lose useful key-points.

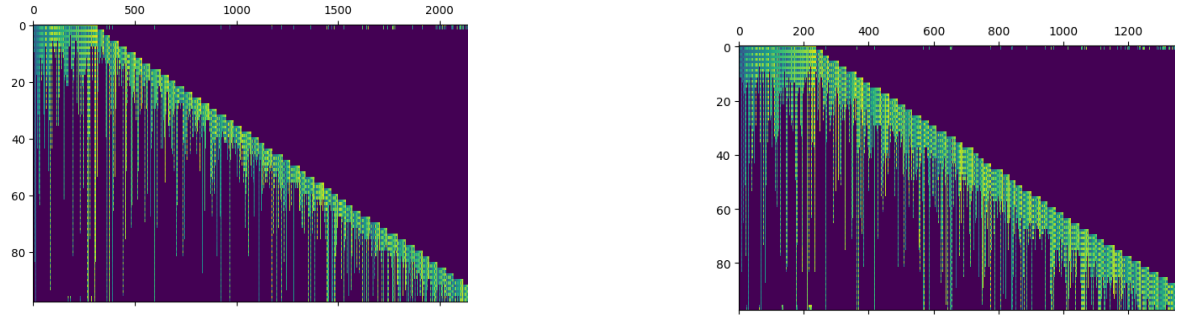


Figure 7: Adjusted point-view matrices for N = 5 (left) and N = 10 (right).

We were also provided with a sample point-view matrix. A visualization is shown in figure 8. The first thing that stands out is the difference in size. The provided matrix was of size 202 × 215. From here we can conclude that the provided matrix was based on a data-set of 99 images. In addition to this, the provided matrix is a completely dense matrix. From here on we can conclude that this matrix represents the ideal case.

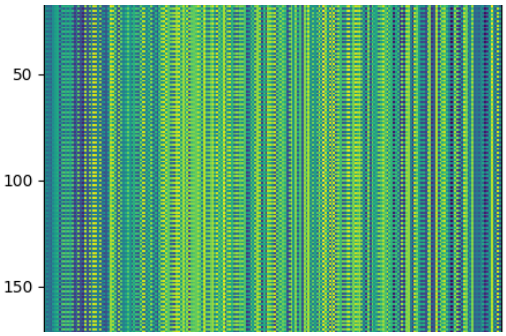


Figure 8: Visualization of the provided sample point-view matrix

4 Structure from Motion

In this part, we aimed on creating a 3D structure from the provided data-set.

4.1 Factorize and Stitch

The first approach was to select a dense block from our constructed point-view matrix (figure 6). From here on we followed the following algorithm:

1. Represent the selected block as measurement matrix D (see figure 9
2. Normalize the points in matrix D by substracting the centroid of the image points

3. Compute the SVD of D; $D = USV^T$
4. Define $M' = U_3(S_3)^(1/2)$ and $S' = (S_3)^(1/2)V_3^T$
5. Obtain L by the following constraints: $L = QQ^T$ and $AL = b$ using Cholseky
6. eliminate ambiguity to recover M and S from M' and S'
7. Stitch each 3D point set to the main view using the point correspondence.

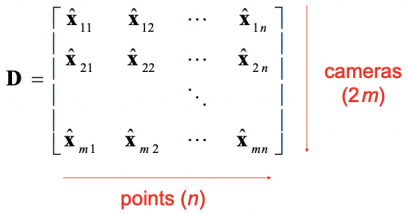
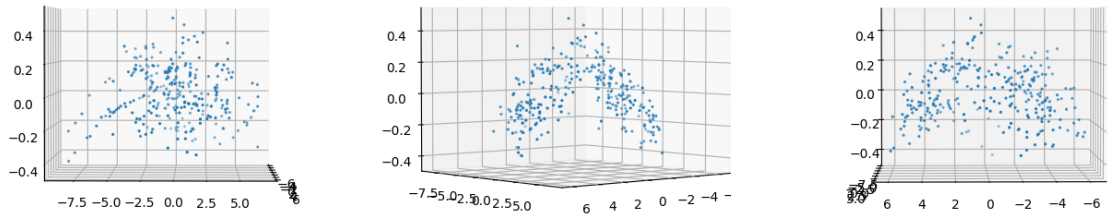


Figure 9: Matrix D reconstruction

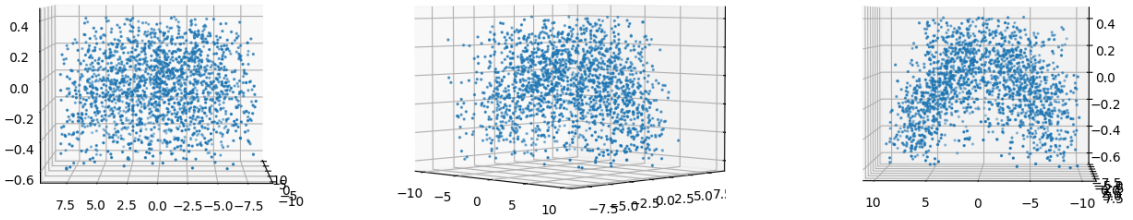
By following this algorithm, we obtained the results as shown in figure 10. The figure contains three differently orientated views. As can be best observed in figure 10b the method did succeed in capturing the shape of the house (roof on top). However, the first four images contained images in very similar angles, thus not offering a lot of depth. Therefore, the reconstructed 3D points lack detailed structure on the sides that were not captured in the frames (best visible in figure 10b).

In order to fix this problem, we implemented the same algorithm iteratively. For each N (N=3 and N=4) consecutive images, we followed the algorithm as described above, resulting in a several different 3D points sets. Subsequently, we stitched the different sets together, making use of Proscrutes analysis. The results of this approach for N=4 are shown in figure 11. The use of the entire matrix instead of one selected block increased the amount of points. In addition to this, the use of all the images in the data-set overcame the problem of the back structure not being captured as occurred in the previous approach. Al though hard to visualize through 2D images, the correct shape of the house was preserved, as best visible in figure 11c. In addition to that, despite the fact that the outliers have been removed, the result still looks very noisy. This is most likely due to the fact that the approach of constructing the point-view matrix doubles certain key-points, as the different angles of view do not differ too much. The result is multiple key-points in slightly different locations, that actually should have been just one key-point.



(a) (b) (c)

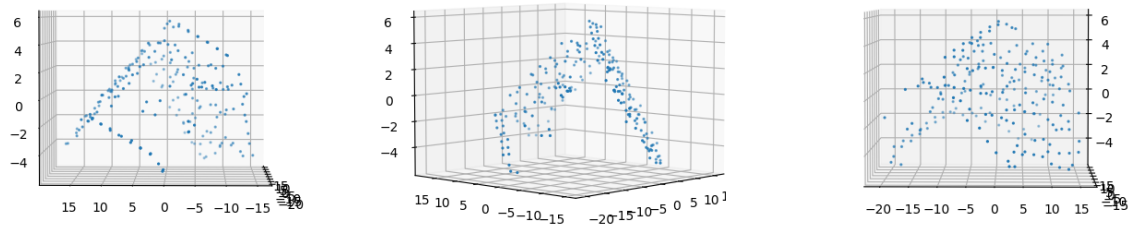
Figure 10: SfM visualization from selected dense block of sparse point-view matrix.



(a) (b) (c)

Figure 11: SfM visualization from multiple dense blocks of sparse point-view matrix.

Finally, we followed the algorithm using the provided dense point-view matrix. As the provided matrix was already dense, step 1 was skipped. The results are shown in figure 12. As already discussed, the provided dense matrix contained the ideal case. The 3D reconstruction therefore looks very accurate. The shape of the house was captured perfectly and no noise seems to be present.



(a) (b) (c)

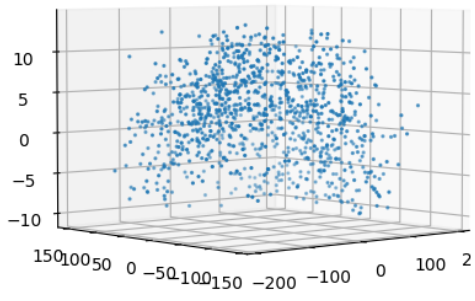
Figure 12: SfM visualization from selected dense point-view matrix.

5 Additional improvements

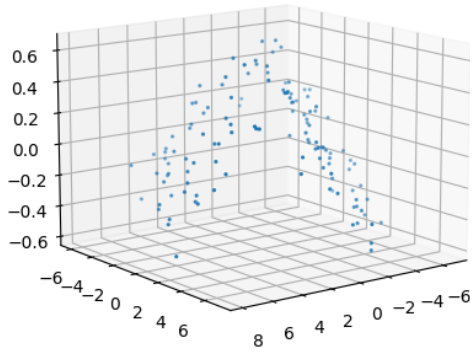
This section will focus on attempting to improve the results as obtained in section 4. Firstly, we tried to increase the density of the point-view matrix by the technique as explained in section 3.1.1. As already explained, we expect this technique to reduce the noise in the 3D reconstruction. Secondly, we removed the affine ambiguity of the point-view matrix. The decomposition of measurement matrix $D = MS$ is not unique. We namely also obtain D when using $M \to MC$ and $S \to C^{-1}S$ for any $C \in \mathbb{R}^{3\times3}$. This is due to the fact that the applied decomposition does not enforce any euclidean constraints (such as image axes being perpendicular). We recovered C using Cholesky decomposition: $L = CC^T$. Then finally, we updated our motion matrix M and structure matrix S in the following way: $M = MC$ and $S = C^{-1}S$. In practice, we used Newton's method to obtain the L.

The results of using the point-view matrix with increased density and removed affine ambiguity is shown in figure 13a. Even though we expected the result to increase in accuracy and look more like the ideal case in figure 12, this is clearly not the case. The constructed 3D point-cloud looks randomized regardless of the angle of view, meaning that the house shape is not derivable. We suspect the messy result is caused by our initial result (figure 11) which was already pretty noisy. We have tried to remove the noise by deleting outliers, however, this did not save the problem of many key-points being detected closely next to each other. Therefore, we expect density increasement and ambiguity removal to better work on a less noisy initial reconstruction.

Finally, we attempted to tackle the problem caused by selecting one dense block containing limited angles of view. Instead of selecting consecutive images to construct the 3D point-cloud, we randomly selected images N throughout the entire data-set that shared enough key-points together. The result is shown in figure 13b. When comparing it to our initial result (figure 10, we see that the newly obtained result contained way less noise. We suspect this is the case because frames that are far away from each other have less matches and thus have less key-points to plot. In addition to this, we see that the 3D structure now captures more details than in the previous method.



(a) Increased density + affine ambiguity removal



(b) Random sampling

Figure 13: Additional improvements on point-view matrix

All in all, we see that our implemented SfM approach was did capture the main idea, but was not perfect. We suspect that our implementation of stitching together the different point clouds was incorrect, as the final result was very noisy. However, regardless of the accuracy of the results, our implementation showed the core idea behind SfM.

6 Real-world Data and Industry Tool

In this section we will explore an industry tool that achieves the same result as with the Structure-from-Motion (SfM) algorithm that we implemented above. This tool, named COLMAP is a general-purpose Structure-from-Motion and Multi-View Stereo (MvS) pipeline with a graphical and command-line interface. For the following experiments the Sarphati monument dataset is used, containing 116 (sequential) images.

6.1 Tutorial

To obtain the results we followed the tutorial as explained in https://colmap.github.io/tutorial.html

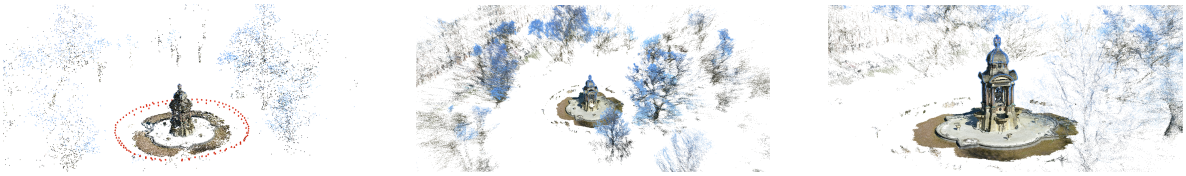
6.1.1 SfM

The first two step is feature extraction and matching to find sparse feature points in the image and describing them in appearance using a numerical descriptor. For the feature extraction it was recommended to share the camera intrinsics between all images when all images were captured by the same physical camera with identical zoom factor, therefore we choose this setting for our reconstruction as well as the 'simple pinhole' setting. Secondly, feature matching and geometric verification is done to find the correspondences between the feature points in the different images. Here, exhaustive matching is used, as the number of images in the used dataset is relatively low (up to several hundreds). Meaning that every image is matched against every other image. However, as the images in our dataset are acquired in a sequential order, we also explored the sequential matching. As the consecutive frames have visual overlap and therefore there is no need to match all image pairs exhaustively.

6.1.2 Reconstruction with MvS

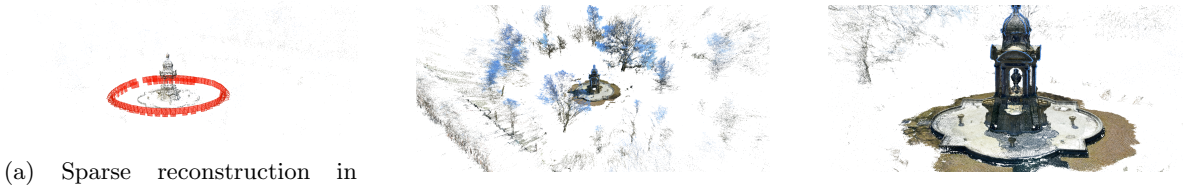
Now these steps are done and we've obtained a scene graph we can obtain a sparse and dense reconstruction.

For the sparse reconstruction COLMAP loads all the extracted data from the saved database (containing the extracted data) into memory and seeds the reconstructing from an initial image pair. After which the scene is incrementally extended by registering new images and triangulating new points. The exhaustive matched sparse reconstruction is visualized in results are visualized in figure: 14a. We've noticed that both the exhaustive and sequential method show similar results (especially



(a) Sparse Reconstruction (b) Fused point cloud (c) Fused point cloud closeup

Figure 14: Sparse and Dense reconstructions using exhaustive matching on all images in COLMAP.



(a) Sparse reconstruction in COLMAP (b) Fused point cloud (c) Fused point cloud closeup

Figure 15: Sparse and Dense reconstructions using sequential matching on all images in COLMAP

for the upcoming steps) when looking at the monument. However, the main difference in results lies in the background pixels. These are reduced in the sequential implementation, visualized in figure 15a.

After obtaining the sparse reconstruction of the scene and camera poses of the input images, Multi-View Stereo (MVS) can recover denser scene geometry. MVS takes the output of the SfM, i.e. the previous steps, to compute depth and/or normal information for every pixel in an image. For this, first, the images are undistrorted, second the depth and normal maps are computed using stereo and third these depth and normals maps are fused to a dense point cloud with normal information. The visualizations of this point cloud in COLMAP can be found in figures 14b and 14c, additionally the same results but for the sequential matched reconstructions are shown in figures 15b and 15c shows the same point cloud in MeshLab.

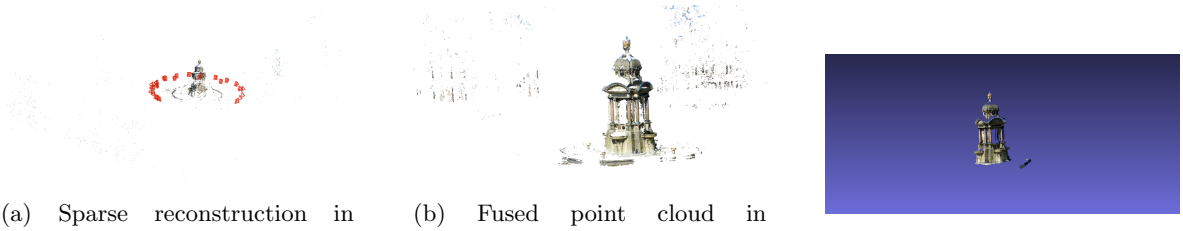
Lastly, a dense surface from the fused pointcloud is estimated using Poisson or Delaunay reconstruction. These reconstructions (for the sequential implementation) are visualized in figures 16a and 16b. Here we see that the Poisson reconstruction looks more smoothed but noisy as with the Delauny reconstruction. Which is more clean but not smoothed at all.

6.2 Further improve results

There are several ways to improve the obtained results, in this section some of those will be mentioned. First of all, the number of images could be increased, e.g. by taking frames of a video going around the monument. Secondly, the reason for the Poisson reconstruction being noisy and the Delauny reconstruction being not smooth, could be due to a incomplete point cloud. Therefore, either the patch size in the stereo step could be increased or the image resolution decreased to overcome.

(a) Poisson (b) Delauny

Figure 16: Dense Reconstructions in MeshLab using sequential matching on all images.

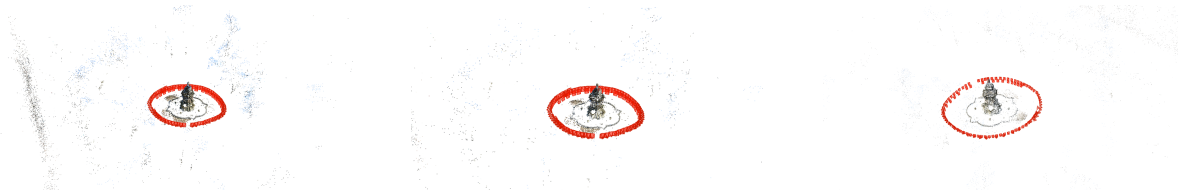


(a) Sparse reconstruction in COLMAP

(b) Fused point cloud in COLMAP

(c) Poisson dense reconstruction

Figure 17: Sparse and Dense reconstructions using sequential matching on 29 images



(a) Default parameters

(b) Lowered render parameter

(c) First SIFT octave is 2

Figure 18: Reducing background pixels Sparse reconstructions

6.3 Reduced number of adjacent frames

In order to investigate what happens when the number of (adjacent) frames is reduced, we used every fourth image in the dataset (resulting in 29 images) to perform the same analysis described above (using exhaustive matching). All results are shown in figure 17. Looking at figure 17a, we see that even though we used every fourth image meaning that the images cover the whole circle around the monument, the camera positions don't make a full circle. This becomes also visible when move the reconstruction around, as one side of the monument is not reconstructed properly. Besides that, the visualizations show that only the top part is somewhat well reconstructed.

6.4 Number background pixels reduced

Several different approaches reducing the number of background pixels have been executed, however, we have not find the correct way to do this. As our results were not as promising as expected. We noticed that by using sequential matching in stead of exhaustive matching, that the number of background pixels was already reduced. Besides, that for the sparse reconstruction, we lowered the render parameter called 'Point max. error' from two to 0,75, which also resulted in less background pixels. These result are visualized in figures 18a and 18b. This However this had no effect on the dense reconstruction. Lastly, we adjusted the first SIFT octave parameter in the feature extraction from -1 to 2, this had also some effect on the background pixels, this result is shown in figure 18c. It is important to note that all approaches also had effect on the quality of the reconstruction of the monument.

7 Self-Evaluation

Emma and Faye provided the code for part 3. Haohui provided the code for part 4, 5 and 6. Faye wrote the report all those sections. Emma implemented and wrote the report for part 7.