University of Amsterdam

Assignment 2

Structure from Motion

May 13, 2022

Students:
Gijs van Meer
12152706

Irene Papadopoulou
13869337

Marius Strampel
12066664

Lecturer:
prof. dr. T. Gevers

Course:
Computer Vision 2

Course code:
52042COV6Y

1 Introduction

This report shows our attempts at creating 3D representations using motion. We first implement the Eight-point Algorithm, followed by constructing the Point-view matrix (PVM) of the frames of an object. We then stitch dense blocks from the PVM together to create a 3D representation of the object. Finally we experiment with COLMAP and suggest improvements for this tool.

2 Fundamental Matrix

In this part of the assignment we explore three different methods to calculate the fundamental matrix, which can be described as a projective transformation between corresponding points in two views. In particular, we have implemented the eight-point algorithm, the normalized eight-point algorithm and finally the normalized eight-point algorithm with RANSAC.

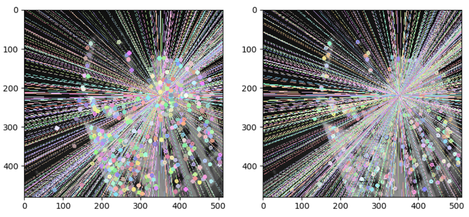
To compute the fundamental matrix with any of the mentioned methods, we follow a few prepossessing steps. First, we need to detect the interest points in each image and characterize the local appearance of the regions around them. We do so by using the OpenCV SIFT key point descriptors. Next, we need to find a set of corresponding matches between the key points of the two images. We do this by using the OpenCV's implementation of the BFMatcher and use a distance ratio of 0.7 (LoweDavid, 2004) to keep the most important matches. Now we are ready to estimate the fundamental matrix with the three different methods and analyse the results in the following sections.

2.1 Eight-point Algorithm

In order to estimate the fundamental matrix with the eight-point algorithm (Hartley, 1997), we first need to construct the nx9 matrix A, where n is the number of matches we have between the two views (images). As can be seen in 1, points x_i, y_i are points that lie in first image and points x'_i, y'_i are the corresponding matches of these points in the second image.

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

Next, we find the Singular Value Decomposition (SVD) of A ($A = UDV^T$) and take the components of the column of V that correspond to the smallest singular value (last entry of D). This is then reshaped to get the (3x3) estimated fundamental matrix F. We know that the fundamental matrix F is singular and has a rank of 2 which is something that we need to enforce by correcting the entries of F. To do this, we find the SVD of F ($F = U_f D_f V_f^T$), then we set the smallest value in D_f to 0 to obtain the corrected matrix D_f' and finally recompute $F = U_f D_f' V_f^T$. To test our implementation, we plot the epipolar lines, meaning the projections of lines on which the points in other image could have originated from. Results are shown in Figure 1.


(a)

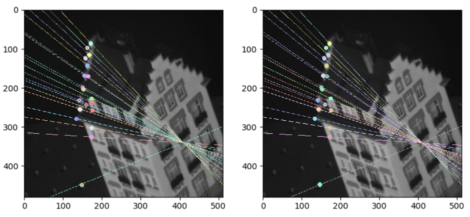

(b)

Figure 1: Epipolar lines of the eight-point algorithm with (a) all points, (b) 20 points. The lines on the left image correspond to the points in the right image and vice versa. We use two consecutive frames (1 and 2) as input.

2.2 Normalized Eight-point Algorithm

For this method, we first normalize all the points p_i in the first image and their corresponding matches $p_i{}'$ so that they have a mean of 0 and their average distance to the mean is $\sqrt{2}$. To do this we construct the similarity transformation T for points p_i and T' for $p_i{}'$ and get the normalized points by with $\hat{p}_i = Tp_i$ and $\hat{p}_i{}' = T'p_i'$. To test our normalization, we calculate the average distance of the matches to their mean and we indeed get 1.414 (=$\sqrt{2}$) which is a big difference from the previous 451.300 (points from frame1) and 451.718 (points from frame2) we get without normalizing. From here, we follow the exact same steps as before to construct the fundamental matrix $\hat{F}'$, now using the normalized points. At the end, we denormalize $\hat{F}'$, by $F = T'^T \hat{F}' T$. We show the epipolar lines we get with this method in Figure 2.

2.3 Normalized Eight-point Algorithm with RANSAC

Finally, to further improve the algorithm, we now estimate the matrix with RANSAC. In particular, we randomly pick 8 point correspondences from the normalized set $\{\hat{p}_i \leftrightarrow \widehat{p}_i'\}$ and we calculate the fundamental matrix F by running the normalized eight-point algorithm, with those 8 points as input. Next, we count the number of inliers from the matches, by computing the Sampson distance as shown in 2. If this distance is smaller than a threshold of 0.3 in our case, then we

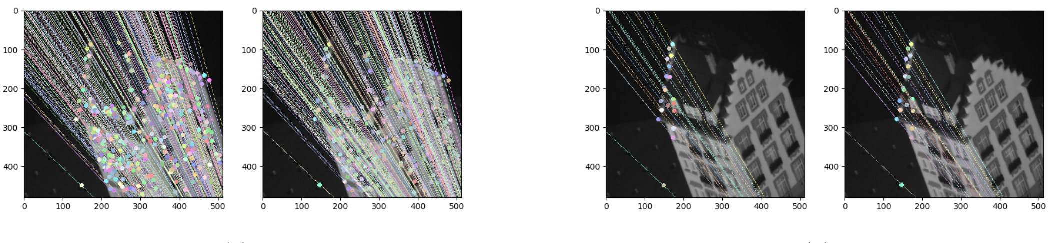

(a) (b)

Figure 2: Epipolar lines of the normalized eight-point algorithm with (a) all points, (b) 20 points. The lines on the left image correspond to the points in the right image and vice versa. We use two consecutive frames (1 and 2) as input.

count this match as an inlier, else we count it as an outlier. We repeat this process 300 times and calculate the set with the most inliers. Finally, we use this set as input to the normalized eight-point algorithm and get an estimation of the fundamental matrix. To check if this estimation is correct, we plot the corresponding epipolar lines in Figure 3.

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

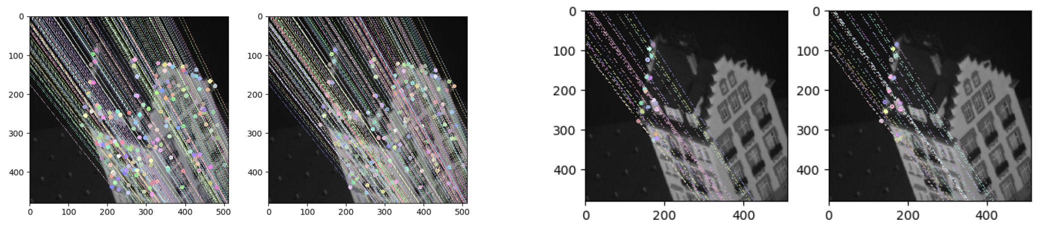

(a) (b)

Figure 3: Epipolar lines of the normalized eight-point algorithm with RANSAC (a) all points, (b) 20 points. The lines on the left image correspond to the points in the right image and vice versa. We use two consecutive frames (1 and 2) as input.

2.4 Analysis

By observing the results in Figures 1, 2 and 3 it is clear that the lines we get improve from non normalized to normalized and then even more with RANSAC. In particular, we observe that normalizing the points leads to the epipolar lines not intersecting. In addition, the use of RANSAC leads to fewer outliers, as all the points and corresponding lines are shown to lie on top of the house and not in the background.

The lines look plausible in all three cases. This is something that can be more clearly observed if we look at the right sub-figure in every method, which shows only 20 of the points and corresponding lines we get. The colours of points and lines match, so we confidently say that the lines are plausible. To calculate the epipolar constraint ($AF = 0$) we compute the mean of AF (all matches) which should result to 0. The results we get are shown in table 1. The table shows that most of the results are very close to 0 and they indeed improve between the three methods. We get a big improvement when we normalize the points. Removing the outliers by using RANSAC also helps but this is mostly dependent on the distance ratio we use. We experiment with different distance ratios, concluding that the lower ratio we use the less but more accurate matches we get as can be seen in Table 1.

Table 1: Epipolar constraint with the three different methods. We use frames 1 and 2 to compute the contraint

dist_ratio	**num_matches**	**EPA**	**Normalized EPA**	**Normalized EPA with RANSAC**
0.2	197	0.717	0.00902	0.000290
0.3	281	0.699	0.00310	0.00960
0.5	362	0.0588	0.00519	0.00996
0.7	425	0.0443	0.0633	0.00374

3 Chaining

In order to calculate the structure from motion we first need to determine the Point-view matrix (Rothganger et al., 2006), containing rows of points in frames and in the columns the matching points to each other. This will result in a 2NxM matrix of N frames and M points. To calculate the points to add in the matrix we do the following. We take 2 frames, and find their corresponding matches. We then check for each matching point in the first frame whether it is already part of the PVM. If it is then we add the matching point from the second frame to the column containing that point, if it does not we simply add a new column and add the points to the new column. We do not ever check whether a new column might possibly have matches with other, older frames. In order to indicate that these points in the matrix do not have a match we simply fill in -1 as the value, to show that there is no coordinate within the image where a match for this frame with the points of this column exists. The results of applying this to the first 49 frames can be found in Figure 4. We do see that each frame in the chain is able to find some matching key points between

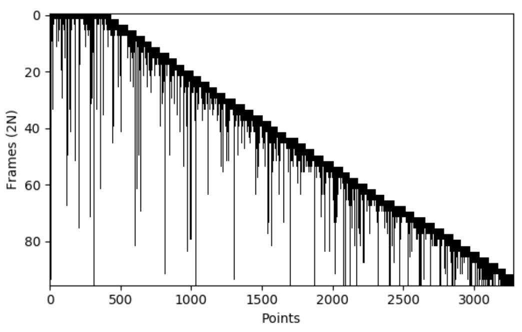


Figure 4: PVM with overlapping points. Each row is the x values of points of a given frame followed by the y values.

the previous and next frame, with some even flowing through all frames. The final frame has the least amount of key points between the two frames, but this makes sense as this is between the last and first image used even though our frames are not part of a closed loop. In total we end up with 3277 columns, with a total of 11 columns which appear in every image. The matrix is rather sparse, with most points not appearing in every image. With the image itself rotating and moving parts out of view this somewhat makes sense, however one would expect more points to be visible from all angles. A possible option to make this matrix more dense is to allow more points to be classified as similar as long as they are within a distance threshold to one another. This, however, should lead to a lot of mismatching between points which might not truly be connected making this a rather ineffective option to improve the density of the matrix. Another way to possibly improve the matching of the point view matrix is to find matches between a frame and all frames before,

but for those frames more than 1 frame away only checking if we can find the points contained within those frames. This could potentially add points again which were previously lost due to either a miscalculation or a specific rotation. Implementing this should improve the density of the PVM, or keep it the same density at worst. The results of this method are shown and discussed in section 5.
The PVM itself looks plausible, as between each frame we see most of the overlap between the previous and next frame, indicating these frames are closely connected. Our overall density is much worse than provided PVMs, where we can find 215 common points between more frames of the same data we used, indicating that there are more effective methods than our current implementation to find a good PVM.

4 Structure from Motion (SfM)

Finally we want to use the found PVM to obtain the 3D representation of the object in the images. However at the moment our PVM is sparse. To work around this we instead create dense blocks from our frames. We first select 3 frames, and find all columns in our PVM which are shared between these 3 frames. We then normalise these blocks by translating them to the mean of the points in the block. We then apply SVD to this block D to obtain the required matrices to calculate our structure (S) and motion (M) matrices. We display these calculations in 3. Here the $_3$ means that we take 3 columns from matrices U and V and the upper left 3x3 block of W.

$$D = U \cdot W \cdot V^T \ , S = W_3^{\frac{1}{2}} \cdot V_3^T \text{ and } M = U_3 \cdot W_3^{\frac{1}{2}} \tag{3}$$

Finally, we execute this for each block to find its structure and motion matrices. We then stitch these together by doing Procrustes analysis to find the similarity between two data sets. Given the nature of the blocks, our blocks won't always have the same size. There are two different solutions to this: one is to simply pad the shorter block with zeroes until both blocks have the same length. The other method is finding the intersection between these two blocks, and using only those points to do Procrustes analysis, before using this to roto-translate the entire block. We would expect that using the intersection between the blocks would lead to better results, as we are making approximations with the closest related points. We also experiment with using blocks consisting of both 3 and 4 frames each, we expect the results of 3 frames being slightly better, as our common points will be larger within the block. Finally we also theorize on the effect of selecting blocks of only 1 frame ahead of the previous block vs never repeating a frame in another block. Here we would expect selecting blocks of only 1 frame ahead to achieve more consistent results, as by never repeating frames we risk having to skip frames that do not fit into the block size (e.g. having 4 frames with step size of 3, resulting in the 4th frame being ignored).

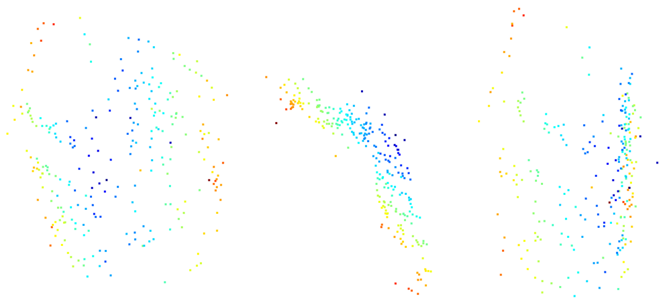

(a) front view (b) top view (c) side view

Figure 5: 3D point cloud obtained from **1 single dense block** using 3 consecutive frames

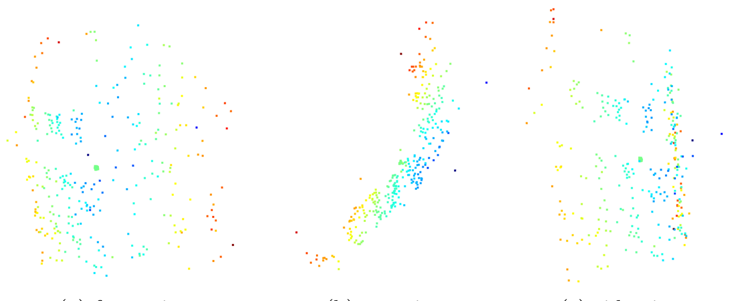

(a) front view (b) top view (c) side view

Figure 6: 3D point cloud obtained from **all blocks** using 3 consecutive frames

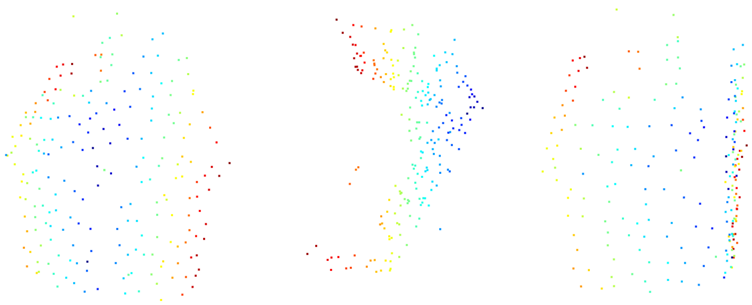

(a) front view (b) top view (c) side view

Figure 7: 3D point cloud obtained from **all blocks** with a **different structure-motion decomposition**

We show the results of using 3 or 4 consecutive frames on a single block in Figures 5 and 13. Both versions follow the same structure but we do observe a difference in the amount of points found. Using only 3 consecutive frames results in a more dense 3D point cloud, whereas with using 4 frames we see a slight decrease in the clarity of the edge around the house point cloud. Although the difference is subtle it seems to indicate that stitching 3 consecutive frames results in a more complete representation of each block. There is, however, always a trade-off for applying this, as having fewer frames per block results in more total blocks and thus more calculations to perform. This means there is a speed vs accuracy trade-off here.
When observing the results on the entire PVM (Figures 6 and 14) our results do not seem too different from the results achieved by using a single frame. However this is due to one part not working properly in the current implementation. At this point in time the scaling changes with every new point cloud added. This scaling change consistently decreases the size of the new point cloud we are adding to the complete cloud. We can observe this as a slightly bigger green mass of dots around the center of our cloud, although not shown in this report, zooming in on this green mass results in finding a representation of the next block followed by another mass being inside the center of this mass. This shows clearly that our scaling changes in between certain steps. In general the alignment of these masses seems to be correct, and only the scaling itself is off.
Using the PointViewMatrix.txt file (as shown in the Appendix, figure 12) makes little sense for factorize and stitch, as we already have equal, fully dense blocks simply obtaining the structure matrices and plotting them should be enough.
With this issue it is not possible to test the effect of using padding vs intersection in stitching. We can instead theorize about this method. We are unsure of the effect of adding zeroes in the Procrustes analysis, making it the more unsure method. If Procrustes were to ignore these we would expect this method to be better as it is finding a stitch that fits the entire data we want to

add.
Due to the point cloud not getting constructed correctly we are also unable to visually observe the difference between moving 1 frame each block and having no repeating frames between blocks. Thinking theoretically however, we can surmise that for early iterations moving 1 frame at a time will be better, however it will also result in a lot of cluttering with points as we move. This means that for larger datasets we might want to use the other method to keep the visualisation interpretable. Given the data we use having very few frames, we move 1 frame each block.

We also experiment with applying a different structure motion decomposition. In particular, we construct our structure S and motion M matrices by:

$$S = W_3 \cdot V_3^T \text{ and } M = U_3 \tag{4}$$

Figure 7 shows the results we get when using this decomposition, applied in all blocks and stitched iteratively. We can see that using this decomposition does lead to better results, particularly we can see that the mass of points in the middle of the image is no longer present and the results look very close to the baseline structure we get when we use the provided "PointViewMatrix.txt" shown in Figure 12.

5 Additional Improvements

5.1 Increasing PVM Density

In order to make our PVM more dense we apply the following method. We iterate over the PVM as normal, except when matching a new frame we check all previous frames more than 1 frame away as well, and find whether we can find more matches within the already existing columns of those frames. Either due to miscalculations or specific rotations we can often end up losing some points in some frames while they do reappear in later frames. This method tries to minimise the points we lose at each frame. The results are shown in Figure 8. We do see an improvement in

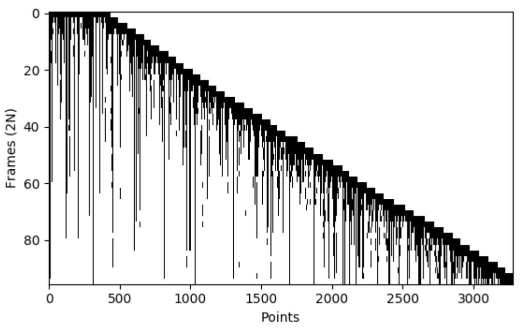


Figure 8: Denser point-view matrix.

density in comparison to the original PVM shown in Figure 4. There is no change in the amount of points found in total but this is more an implementation difference than anything, as we only add points after we have calculated the new matches, meaning they will always get added first. This method could be further improved by checking with all previous frames first and then adding the points between the frame with the previous one according to these results. This could potentially lead to a decreased number of points. An issue with attaining this denser PVM is time complexity however. Testing both methods 5 times the original method took 15.299(±0.186) seconds whereas the denser method took 178.342(±9.188) seconds. The issue that leads to this increased time complexity has to do with the fact that at each step we are increasing the amount of frames we calculate matches between, meaning just the last step of the dense method will have about the same time complexity as the sparse method from 3. For a more balanced method we could instead set a limit to how far back the frames we look at go, making an assumption that those older frames will have little to no relevance with the current frame. This assumption of course will be based on the data used and should be experimented with depending on the data as such. Results of running the structure from motion using the denser PVM are shown in the appendix in figure 15, the results seem similar to other results achieved in this paper but this could once again be due to the incorrect scaling in our implementation, although it could also be due to the fact that we are simply looking too far back at the moment in our frames, and choosing a shorter window (say 5 frames) might show more signs of improvement.

5.2 Structure Motion Decomposition

As mentioned in section 4, we experimented with different structure motion decompositions. Specifically, by applying W^x with $x = 1$ instead of $x = \frac{1}{2}$, our structure is given by 4. We went further and tried several different other powers. Table 2 shows the mean and standard deviations of 10 runs for each value of x. From table 2 we find that $x = 1$ is clearly the best option. Both increasing or decreasing x from $x = 1$ only increases disparity. We chose not to show comparisons of the stitches, as the resulting change is hardly noticeable due to the issues mentioned in section 4 and the improvement is more clearly reflected in the disparity values.

Table 2: Mean disparity for different powers of W component from dense block SVD.

W power	0.25	0.50	0.75	**1.00**	1.25	1.50	1.75	2.00
Mean Disparity	$2.058e^{-3}$	$2.122e^{-3}$	$3.564e^{-5}$	$\mathbf{7.983e^{-6}}$	$2.722e^{-5}$	$8.660e^{-5}$	$1.855e^{-4}$	$3.236e^{-4}$
Std. Disparity	$5.575e^{-4}$	$1.082e^{-4}$	$2.481e^{-5}$	$\mathbf{7.535e^{-7}}$	$2.511e^{-5}$	$9.896e^{-5}$	$2.216e^{-4}$	$3.928e^{-4}$

6 Real-world Data and Industry Tool

COLMAP (Schönberger & Frahm, 2016; Schönberger et al., 2016) is a general-use pipeline for Structure-from-Motion (SfM) and Multi-View-Stereo (MVS) reconstruction of 3D objects. We were not able to do MVS, as it was not given as an option in the dropdown menu COLMAP's documentation says it should be in. We eventually found where to do MVS, but it requires Nvidia GPUs (CUDA), which we do not have access to. As a result, we can not do MVS and thus the rest of this section will discuss SfM results only.

6.1 COLMAP Sparse Reconstruction

COLMAP's automatic reconstruction functionality was used to reconstruct the Sarphati monument. The Sarphati monument data[1] consists of 116 images taken from slightly different viewpoints, all of which were used to create the sparse reconstruction shown in figure 9. COLMAP does an excellent job reconstructing the monument itself, however it also finds a lot of background points. This is shown more clearly in figure 11a, which shows the same reconstruction as figure 9, but from an angle that shows more background points. Additionally it finds a lot of points around the base of the monument.

6.2 Improvements

There are two spaces in which to improve reconstruction quality: pre-reconstruction, and the reconstruction itself. Pre-reconstruction improvements are focused mainly on the images used for the reconstruction. Some obvious things can be done on this front, namely increase the number of images, make sure they are evenly spaced, make sure they are evenly lit, etc. But we can also get more creative. Fiducial markers are highly recognisable markers placed in a frame (either physically or digitally) to provide a reference point. The use of fiducial markers has been used

[1]https://surfdrive.surf.nl/files/index.php/s/keYiU9gfoenQV8N, accessed: 13/05/2022

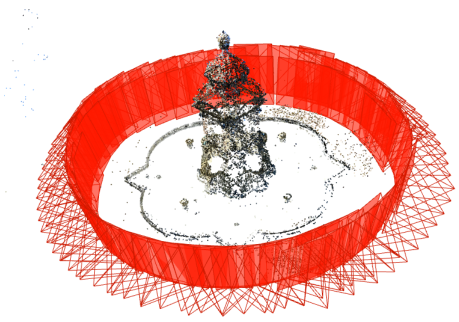


Figure 9: COLMAP SfM reconstruction of Sarphati monument.

in 3D reconstruction tasks (Klopschitz & Schmalstieg, 2007) with some success, but DeGol et al. (2018) have shown them to be greatly applicable to SfM. They provide an alternate SvM algorithm which increases performance when fiducial markers are available, while suffering no performance loss when they are not.
Regarding improvements in the SvM reconstruction process itself, a lot of research has been done into mitigating the influence of bad or incorrect matches. Most solutions presented in the literature try to find such wrong matches and remove them. Kataria et al. (2020) take a different approach: they do not remove faulty matches, but instead try to improve the reconstruction iterations that would normally be corrupted by such faulty matches.

6.3 Reducing Adjacent Frames

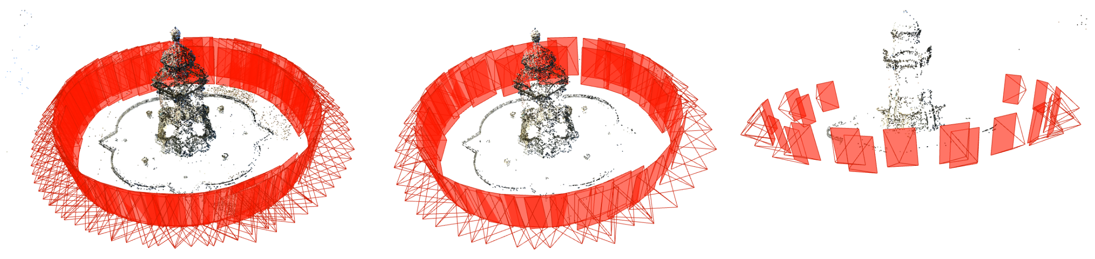
Figure 10 shows the automatic reconstructions resulting for feeding COLMAP all images, every other image, or every fourth image. This results in 116, 58 and 29 images respectively. Comparing figure 10 shows that using half the images still produces a very reasonable reconstruction. Using only a fourth of total images, however, results in COLMAP not being able to even provide a 360 degree reconstruction. As we systematically removed 3 out of every 4 images, and images are named counting up, we would expect subsequent images to be subsequent frames. If this were true, a full view would still be available, but as is indicated by the incomplete red ring of camera views in figure 10c, this was not the case.
Table 3 shows that the mean reprojection error decreases with the number of images used. This is likely due to the amount of total points drastically reducing as well, which is reflected in the mean number of points per image reducing along with the number of images. Additionally, using fewer images means fewer background points as well. The background have higher mean projection errors than the foreground points, the reason for which is discussed in section 6.4.

Table 3: Several reconstruction statistics for data differing used image ratios.

Data ratio	Mean track length	Mean observations/image	Mean reprojection error
1	6.02	1965	0.917
0.5	4.60	1237	0.867
0.25	3.60	613	0.783

6.4 Reducing Background Pixels

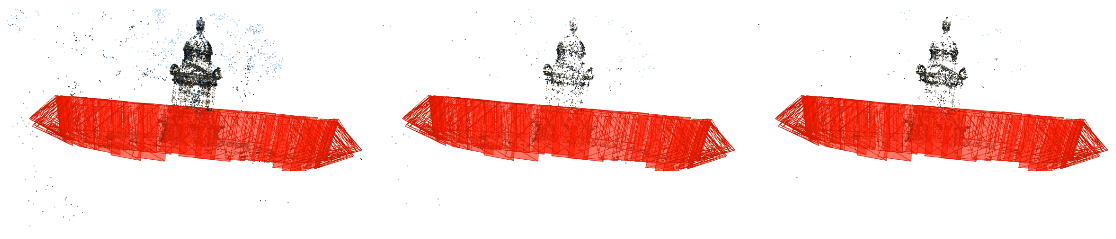
The amount of background pixels is quite large when using automatic reconstruction with no intervention (figure 11a). Just using the options presented in COLMAP gives us two options to reduce their occurrence. Firstly, we can decrease the maximum error it allows. as background points tend to have higher reprojection errors, these get filtered out more than foreground points. The effect of using a lower error threshold (0.6 instead of the default 2.0) is shown in figure 11b.



(a) (b) (c)

Figure 10: COLMAP sparse reconstruction of Sarphati monument on (a) all images, (b) every other image, (c) one every four images.

The value of 0.6 was found empirically. Although decreasing the maximum allowed error does remove most of the background points, it also removes a sizeable amount of foreground points.
Next, we tried increasing the minimum tracking length, which increases the minimum amount of frames a point should occur in to be kept. Since background points will not be visible from many camera angles, they will be more sensitive to the tracking length constraint than foreground points. Figure 11c shows the result of setting the minimum tracking length from 3 to 8. This method removes more background points than the previously mentioned method, whilst also keeping more foreground points intact.
Lastly, since COLMAP reconstructs the object at the origin, one could impose some constraint on the distance a point is allowed be away from the origin. We did not find a way to do this in COLMAP however. Furthermore, COLMAP allows to export the pointcloud, which means we could remove points too far away from the origin ourselves, then feed it back into COLMAP with the subset cloud. This is not possible, as COLMAP allows exporting of the pointcloud to both text or binary files. While we can read text files and perform our point culling, COLMAP only allows for the importing of binary point clouds. We are not able to read the points from the binary file however, and as such are not able to use this method.



(a) (b) (c)

Figure 11: COLMAP sparse reconstruction of Sarphati monument using (a) default parameters: me = 2.0 and mtl = 3, (b) me = 0.6 and mtl = 3, (c) me = 2.0 and mtl = 8, where me is the maximum reprojection error and mtl is the minimum tracking length.

7 Conclusion

In this report we tried to implement structure from motion. We were able to complete almost all steps but the stitching itself proved difficult. The scaling of the frames was off resulting in incorrect reconstruction. We were able to successfully use COLMAP to create a 3D reconstruction of the Sarphati monument and talked about possible improvements in both the pre-reconstruction and reconstruction.

References

DeGol, J., Bretl, T., & Hoiem, D. (2018). Improved structure from motion using fiducial marker matching. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision - ECCV 2018 - 15th european conference, munich, germany, september 8-14, 2018, proceedings, part III* (pp. 281–296). Springer. https://doi.org/10.1007/978-3-030-01219-9_17

Hartley, R. (1997). In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(6), 580–593. https://doi.org/10.1109/34.601246

Kataria, R., DeGol, J., & Hoiem, D. (2020). Improving structure from motion with reliable resectioning. In V. Struc & F. G. Fernández (Eds.), *8th international conference on 3d vision, 3dv 2020, virtual event, japan, november 25-28, 2020* (pp. 41–50). IEEE. https://doi.org/10.1109/3DV50981.2020.00014

Klopschitz, M., & Schmalstieg, D. (2007). Automatic reconstruction of wide-area fiducial marker models. *Sixth IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR 2007, 13-16 November 2007, Nara, Japan*, 71–74. https://doi.org/10.1109/ISMAR.2007.4538828

LoweDavid, G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision.*

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. https://doi.org/10.1007/s11263-005-3674-1

Schönberger, J. L., & Frahm, J.-M. (2016). Structure-from-motion revisited. *Conference on Computer Vision and Pattern Recognition (CVPR).*

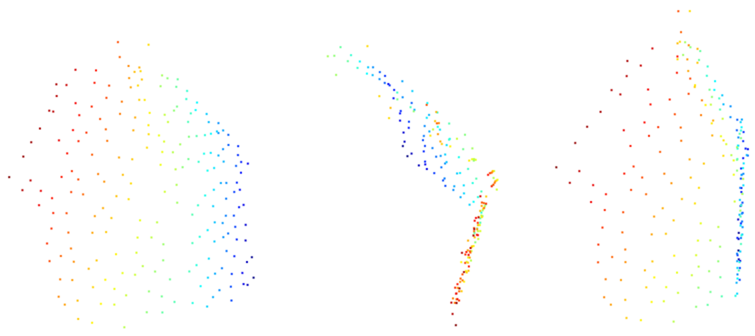
Schönberger, J. L., Zheng, E., Pollefeys, M., & Frahm, J.-M. (2016). Pixelwise view selection for unstructured multi-view stereo. *European Conference on Computer Vision (ECCV).*

Appendix

A Self-Evaluation

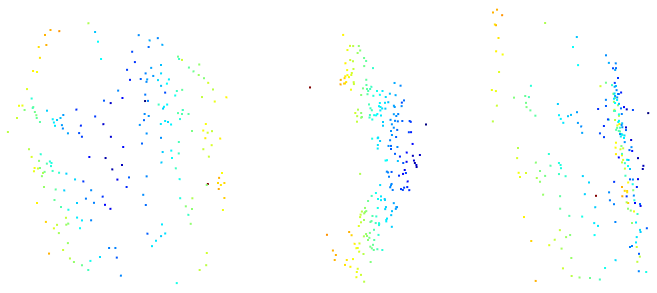
In this assignment Gijs worked on the chaining (section 4), the analysis of structure from motion (section 5.2) and the point-view density improvement (section 6). Irene implemented the fundamental matrix (section 3) and together with Marius implemented the factorize and stitch (section 5.1). Irene also created the graphs for section 5. Finally, Marius worked on the implementation of factorize and stitch together with Irene and also implemented the Real-World Data and Industry Tool (section 7). A large amount of time was spent by all 3 participants on the final step of section 5.1. We wrote the report together.

B Baseline SfM



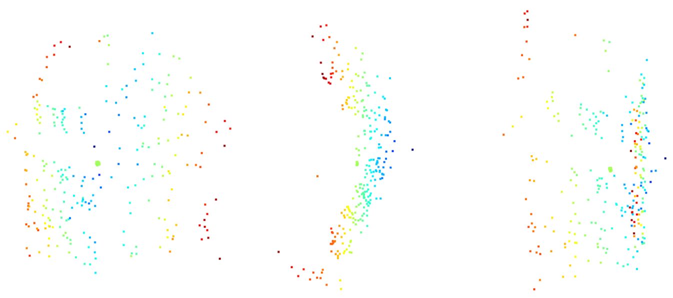
(a) front view (b) top view (c) side view

Figure 12: Baseline structure when using PointViewMatrix.txt



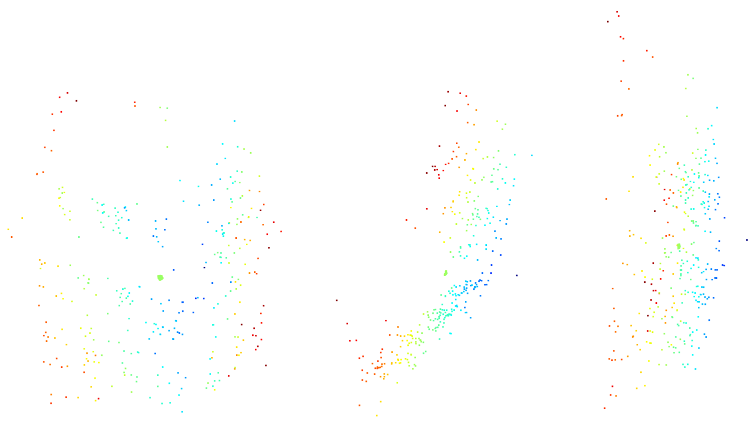
(a) front view (b) top view (c) side view

Figure 13: 3D point cloud obtained from **1 single dense block** using 4 consecutive frames



(a) front view (b) top view (c) side view

Figure 14: 3D point cloud obtained from **all blocks** using 4 consecutive frames



(a) front view (b) top view (c) side view

Figure 15: 3D point cloud obtained from **all blocks** using the denser PVM