Computer Vision 2: Structure from Motion

Gabriele Bani
11640758
gabriele.bani@student.uva.nl

Andrii Skliar
11636785
andrii.skliar@student.uva.nl

Andreas Hadjipieris
11730064
andreas.hadjipieri@student.uva.nl

1 Introduction

In this report, we will explore the capabilities of the Structure-from-Motion (SFM) algorithm. The SFM algorithm will try to find the best representation of an object in multiple images in 3 dimensions. This procedure makes a point-view matrix which dense blocks of this binary matrix represent surfaces of the object. A spectrum of versions of the algorithm were implemented ranging from changing features like SIFT to more advanced features like AKAZE. Furthermore we have tried to dynamically choose dense blocks in the point-view matrix to further improve the performance due to the errors from creating the point-view matrix. The current task is to reconstruct a house from 50 images of this object from different angles.

2 Fundamental Matrix

2.1 Implementation

The plan is to implement different versions of the eight-point algorithm to represent all the images to a 3D projection using the fundamental matrix. The improvements for his specific algorithm was to use AKAZE features instead of SIFT. We choose there features because we wanted to see the effect in performance from different kinds of features. Moreover we have used "cross-check" when matching the descriptors with k-Nearest-Neighbors since we wanted one-to-one matching between descriptors from the two images.Cross-check helped a lot on the speed of the overall algorithm since no more loops or checks were needed to get the non matched descriptors.

2.2 Eight-Points Algorithm

We start by representing descriptors in homogeneous coordinates, we get the fundamental matrix with equation proposed in [1].

$$x'^T F x = 0$$

From this equation the algorithm recovers the Fundamental matrix F as the column with the smallest eigenvalue from the matrix V which we got when we used SVD decomposition of $A = UDV^T$.

The normalized version of Eight-point algorithm basically does the same but with the difference that we normalize the data to a mean of zero and $\sigma = \sqrt{2}$, as suggested in [1]. Then we use the reversed transformations to obtain the Fundamental matrix. This improvement reduced the error of the transformation by a factor of 10, while not making the overall algorithm significantly slower. This suggests that there is no reason not to use the normalized version of the algorithm.

We iteratevely select 8 points with RANSAC to reconstruct the Fundamental matrix, which the error of this transformation was computed using the Sampson distance. With the help of a threshold, which we set to $10e-4$, we differentiate pairs between inliers and outliers. The matrix with the highest number of inliers is returned as the result of this algorithm. The fundamental matrix serves as a mapping between one point in an image to its epipolar line in the target image plane. we need to satisfy the constrain that a matched point in the target image will be on an epipolar line.

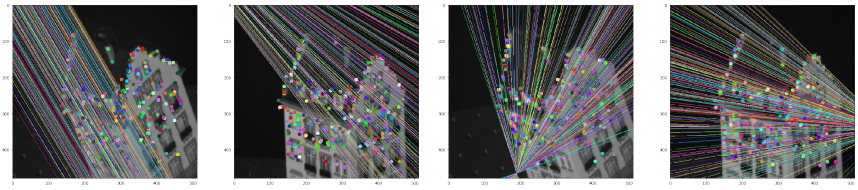
2.3 Results

We run the three algorithm and qualitatively compare their results. The eight-point algorithm and its normalized variant require as input a sample size. We expect that both eight-point algorithm and its normalized version will perform much worse than RANSAC due to the fact that they are very biased towards noise and outliers and therefore can't produce reasonable values for the fundamental matrix. As you can observe in fig. 1, it is indeed the case. If we would satisfy epipolar constraints, we expect epipolar lines to be coherent with the constraints specified by the corresponding points in the original image.

Also, it is important to notice that RANSAC is capable of removing outliers and thus can obtain more accurate results. Regarding the number of iterations of the algorithm, we decide to fix the maximum number of iterations to 100, since empirically the results were already very satisfying for such number. Of course its been clearly shown than with larger number of iterations the result is more accurate.

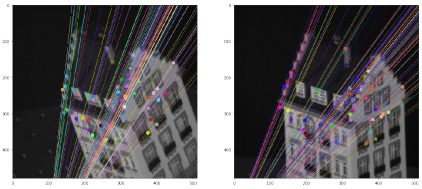
However, while testing matching procedure between images (you can see results in fig. 2), we found out, that different features produce different results and therefore they have to be tuned. It can be clearly seen, that SIFT descriptors capture interest points in the background, which adds noise to our final result. Therefore, we have manually tweaked contrast threshold, which allowed us to get rid of the keypoints in the background therefore only leaving keypoints, which would be meaningful for further structure from motion.

One more thing to notice is that we have tried using cross-checking procedure, which ensures that features found are meaningful and might serve as a good replacement for the original procedure of distance testing, suggested by D.Lowe in his original paper [2]. This procedure will be explained later in section 3.1.



(a) Eight-point

(b) Normalized eight-point

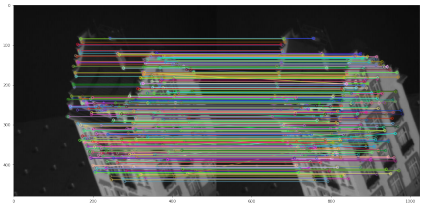


(c) RANSAC with normalized eight-point

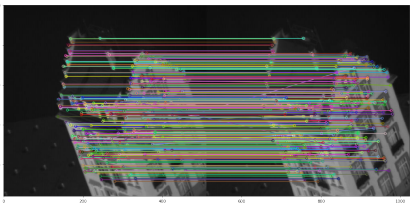
Figure 1: Epipolar lines using various algorithms

3 Chaining

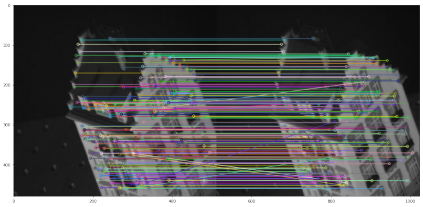
This part turned out to be one of the most complicated parts of the assignment due to the various ways point-view matrix can be constructed. Also, this part is very crucial to Structure from Motion procedure, therefore, finding good point-view matrix was one of the most important tasks.



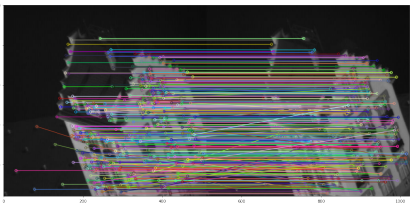
(a) AKAZE features without cross-check



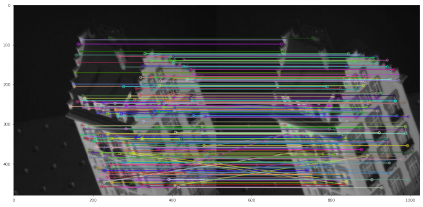
(b) AKAZE features with cross-check



(c) SIFT features with cross-check and threshold



(d) SIFT features with cross-check and without threshold



(e) SIFT features without cross-check and with threshold

Figure 2: Example of matches for various algorithms with and without threshold and cross-checking

In order to do the chaining, we have implemented an algorithm, which matches feature points between two images. This algorithm produces point-view matrix [3], which will then be used in Structure from Motion algorithm. Rows of the point-view matrix represent images, while columns represent single feature keypoint. Each cell of the resulting matrix contains location of the feature keypoint in a corresponding image.

Important and expected property of a matrix is that it is sparse and only has elements below the diagonal. This is due to a way matrix is constructed: as we are only matching features between $image_i$ and $image_{i-1}$, we can only match features, which have been previously found in $image_{i-1}$, but there might be some new points in $image_i$, which, appended, form a diagonal form of a matrix. Due to the way structure from motion works, we also need to make sure, that the matrix doesn't have holes in the columns, meaning that if some point was seen in views from i to n, it won't be seen in view $n + 5$. Comparing current image with only previous image ensures this property.

Note, however, that if we do comparison with some fixed step, i.e. match every 3 images, we might have holes in our matrix, which can be afterwards filled using interpolation of coordinates of previous and next elements in a column. This might be a possible improvement, however, we decided not to implement it in this work.

However, the main issue of such basic approach is that having too many descriptors, we have a risk of having many outliers in our point-view matrix. To fight this issue, we have used two approaches for filtering matched keypoints (you can see resulting point-view matrices for various combinations of these approaches in fig. 3):

1. First approach relies on a fact that images within provided dataset don't change much between two consecutive images, therefore, we can assume that similar points will be close to each other in those. Using this assumption, we consider matched keypoints, which have

distance larger than a given threshold between two consecutive images, as outliers and remove them from a set of matched keypoints.

2. Next, we have used RANSAC for filtering out the outliers. This guarantees that the points found will indeed be corresponding to a meaninglful features of the final object, therefore improving performance of structure from motion.

Note, also, that we have tried matching the last view with the first one, as mentioned in the description of the assignment and found out, that it has no effect on the final point-view matrix. This is due to the fact that because images are very different, results produced by RANSAC are not of any good, returning only few inliers, especially if we filter out the outliers using distance between them in two consecutive images. This results in adding a lot of new columns, which are removed afterwards using dropout procedure are described in section 3.1.

3.1 Improvements

Here, we would like to introduce improvements, that we have used to both increase density of the point-view matrix and find better keypoints, we have tried several more approaches. As our further results highly depend on these improvements, we would like to describe them in separate subsection.

One idea that we have used for making matrix more dense, was that if some points are encountered in less than some fixed number of views, they are not informative and therefore can be discarded. It makes sense as if we reconstruct 3D points from the point, which was encountered in only few views, it is highly possible that this point will only add noise instead of making reconstruction better. This improvement greatly increased the performance of our system. However, it has an obvious drawback of having less points than the original point-view matrix and therefore results in less points in a final point cloud thus having less structure. You can see results for different minimum number of view fig. 5.

Also, to improve our results, we tried using KAZE features instead of SIFT features. The main reason for choosing KAZE features was that they preserve all invariances of SIFT features while also achieving faster speed and better results in terms of repeatability, distinctiveness and robustness, which leads to better performance, as reported in the original paper. As you can see, using AKAZE features significantly increases density of the final matrix.

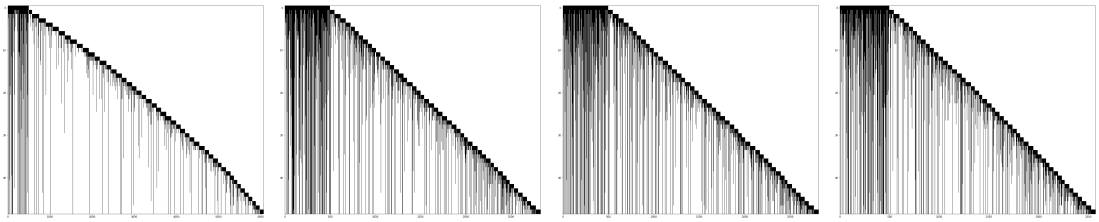
One final improvement that we have was that normally, features are being matched using K-Nearest neighbours, which doesn't provide us with one-to-one correspondence between features as some of the features of train image can have multiple query features as closest. This issue is addressed in the original D.Lowe paper on SIFT, where he uses distance test in order to remove points, which might be corresponding to two points, as they are considered non-informative. However, we have also tried doing cross-checking, which would only match points if they have each other as closest neighbour. This makes final matrix more sparse, however, after removing points encountered in few number of views, we can see that the final matrix gets more dense and, also, we can expect features found to be more meaningful and descriptive of a corresponding image. You can see results for different features and testing methods (cross-check or distance test) in fig. 4.

4 Structure From Motion

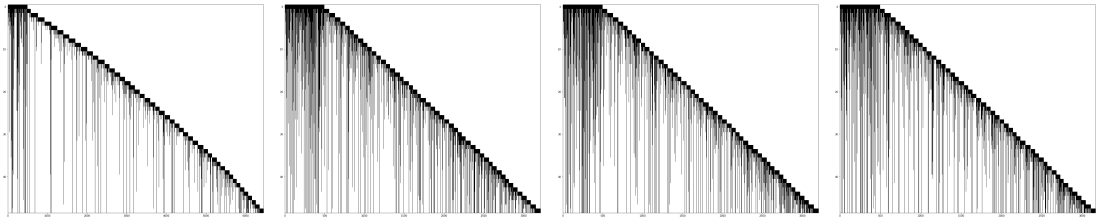
4.1 Implementation

In order to run the algorithm, we have to first find dense blocks in the point-view matrix. To do so, we use a very simple strategy. We choose a block size as parameter, and then iterate over the images in order to understand which keypoints to add to each block. If we have a batch size B, the number of blocks composed by B consecutive views is $M - B + 1$, where M is the number of views. The i^{th} block is composed by the keypoints that are present in all the views $i \dots i + B$. This strategy is very simple, but allows algorithm to construct a reasonable 3d point cloud.

Now, we follow the procedure indicated in the assignment. We keep track of the points that we process and save them cumulatively, together with their indices in the point view matrix. For simplicity, we decide to use the first dense block as a starting block, and add all its keypoints to the cumulative array. We can then proceed to iteratively select the next dense block, create a $2M \times N$ measurement



(a) distance threshold = 1 (b) distance threshold = 3 (c) distance threshold = 7 (d) distance threshold = None



(e) distance threshold = 1 (f) distance threshold = 3 (g) distance threshold = 7 (h) distance threshold = None

Figure 3: Point-view Matrices with various approaches for filtering out points

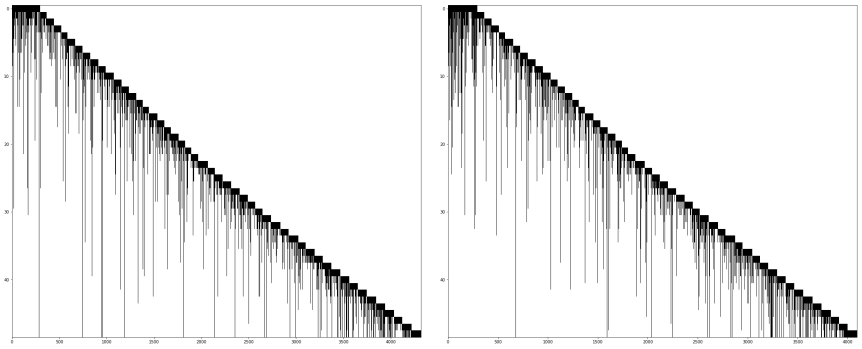
matrix and normalize it by row (view). Then, retrieve the structure and motion matrices using the SVD decomposition. Finally, we find the common keypoints between the current block and the first block, and use the Procrustes function obtain a translation vector t, rotation matrix R and scale value s that allow the projection of the current block keypoints Y to the cumulative points X by computing $Y' = s * YR + t$. Note that we stitch to the cumulative point cloud only the keypoints that are present in the current block but are not already in the point cloud. We do this in order to avoid redundancy of points in the final point cloud.

4.2 Results

First of all, we visualize one single dense block as a point cloud, which shows the result of one single step for the structure from motion algorithm. We visualize the first block obtained as described in previous sections, the result can be seen in **??**. We can see that the results is visually correct. We would like to note the fact that even with a small block size, the recostruction is already satisfying, and this is due to the fact that the image is very simple. Of course, when also projecting the other views, we can get a more complete representation of the object.

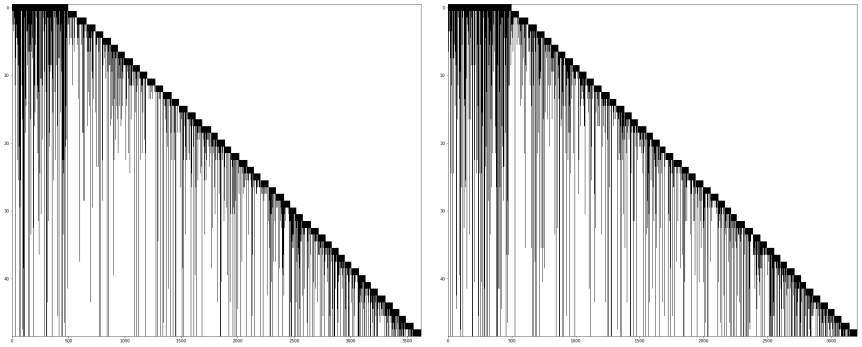
In **??** we can see the result of the full structure from motion algorithm. Although an interactive 3d visualization clearly allows for better qualitative analysis, in this report we show one 2d view of the 3d point cloud produced as output by the algorithm. For a more clear insight, refer to the code. First of all, the shape of the object is distinguishable even without any particular improvements, which means that we successfully managed to merge the information contained in different 2d images to reconstruct a three dimensional structure. However, a notable result is the presence of outliers. We can see in fact many points that are far away from the surface points. As explained in previous section, more advanced use of features for detecting keypoints and can eliminate much of those outliers. Also, our results have a very large number of features, which clearly introduced not needed points.

Finally, we use the given full dense matrix, which contains information for all keypoints for all the views. This has two main uses: firstly, it greatly simplifies the problem and allows to find the output 3d point cloud by just one usage of the structure from motion decomposition. Secondly, it allows us to have a reliable baseline solution. In general however, this a full dense matrix of keypoints and views cannot be built, and one cannot rely on this method. Furthermore, when the number of points and views becomes very large, it might be impractical to run an SVD given its high computational cost, and it is preferred to run an iterative process as described. A visualization is shown in fig. 8. The result is much better than the ones obtained by our reconstructions, and this is becasue we start from a matrix with much more complete information. In particular, we see that there are no outliers, and the number of keypoints is much lower, so the visualization is much more neat.



(a) SIFT, No Cross-Checking (using distance test)

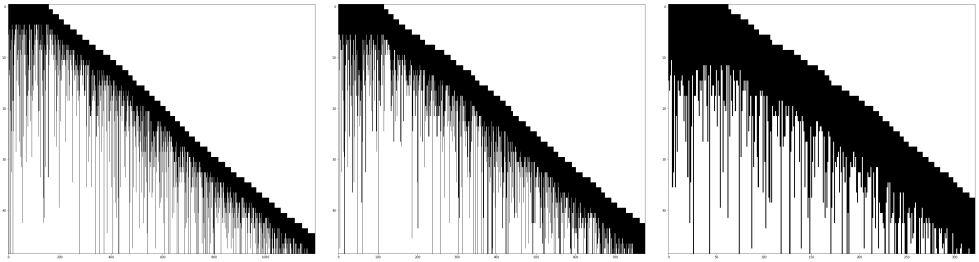
(b) SIFT, Cross-Checking



(c) AKAZE, No Cross-Checking (using distance test)

(d) AKAZE, Cross-Checking

Figure 4: Results for AKAZE and SIFT features



(a) Drop-rate = 3

(b) Drop-rate = 5

(c) Drop-rate = 10

Figure 5: Results for various dropping rates (number of views in which point should appear not be removed)

4.3 Additional Improvements

5 Affine Ambiguity Removal

We want to impose the constraint $MLM^T = I$, where M is the concatenation of all the motion matrices of the blocks. In order to solve the linear system we keep track of all the motion matrices and concatenate them vertically (the number of columns is fixed to 3 in our case). We can now find the least squares solution to the equation $M_c L M_c = I$, as a unique single solution does not exist in general, given the dimension of the matrices. Then, we compute the Cholesky decomposition of $L = CC^T$ and compute the corrected structure and motion matrices by calculating $S = C^{-1}S$, $M = MC$.

Figure 6: Result of the structure decomposition of the first dense block.

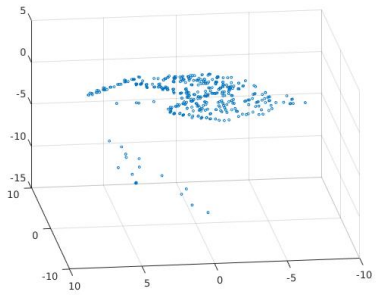
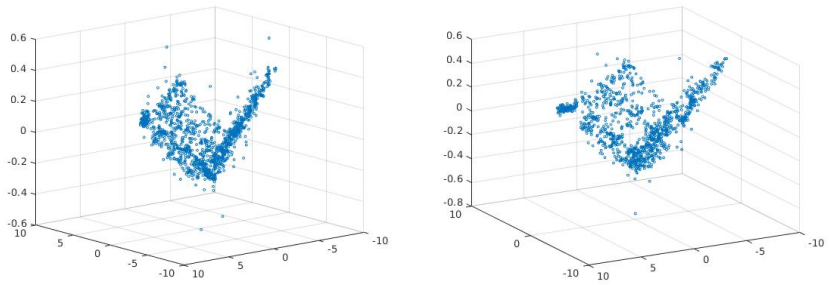


Figure 7: Result of the structure from motion algorithm, under different 2d projections. In the first column, we show result for block size of 3, in the second column of size 4



From fig. 10, you can see that the result seems visually almost the same. The only key difference is in the range of values is in the range of the axes.

6 Conclusion

We have implemented the SFM algorithm and different improvements for it, and discussed its strong and weak points, along to other possible improvements. We have run different version of the algorithm, and discovered that in our case, factors such as sample size were of little importance for the convergence of the algorithm although a smoother result on the point-view matrix was observed. Surprisingly, both quantitative measures such as the approximation errors and qualitative ones such as the shape of the surfaces of the blocks, showed almost no difference while changing sample size. The most impressive improvement was the dynamic selection of dense blocks in the point-view matrix, and with the assumption that that matrix is never going to be perfect, the fact that we had a threshold for how dense a block should be helped to deal with this huge problem.

7 Self Evaluation

Andrii Skliar:
Implemented: Eight-Point, Chaining,SFM
Wrote about: Everything

Gabriele Bani:
Implemented: SFM
Wrote about: Everything

Andreas Hadjipieris:

Figure 8: Result of the structure decomposition of the full dense matrix given.

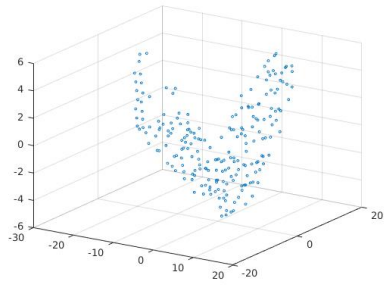
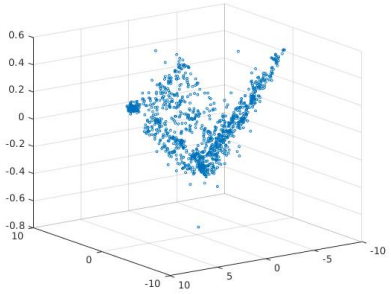


Figure 9: Results when applying the densification of point-view the matrix.



Implemented: Normalized Eight point,Chaining Wrote about: Fundamental Matrix, images

References

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

[2] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[3] Fred Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 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, 2006.

Figure 10: Results when using affine ambiguity removal