Computer Vision 2, Assignment 2

Davide Belli
11887532
davide.belli@student.uva.nl

Gabriele Cesa
11887524
gabriele.cesa@student.uva.nl

Lukas Jelínek
11896493
lukas.jelinek1@gmail.com

Introduction

In this assignment we are going to implement Structure-from-Motion algorithm with the goal to reconstruct a 3D-structure from a set of images picturing the same object. At first we are going to implement Eight-point algorithm to compute the Fundamental matrix describing the 3D transformation from a view to the next one. To construct this algorithm, we start from a basic version of it, improving it successively with Normalization of data and applying RANSAC to find the best transformation. Afterwards, we are going to iteratively match pairs of views in the dataset in order to create a single chain structure connecting all the views. The match graph created in this way is represented as a sparse point-view matrix. This matrix will be compared with the ground-truth dense matrix representing the transformation of the whole point cloud. Finally, we will use these matrices for affine Structure-from-Motion. To solve the problem with the sparse matrix, we are going to work with dense blocks in the matrix. By computing once again SVD composition, the matrices representing Structure and Motion can be derived. Some improvements among which affine ambiguity removal will be discussed and implemented to conclude this assignment.

1 Fundamental Matrix

In this first part of the assignment, we are going to implement different approaches to Eight-point algorithm in order to find the Fundamental matrix representing a 3D transformation from one image to another. In all of the three versions we are going to use, the steps to select features and create point matches are shared.

The first step in the algorithm consists in detecting interest points in both images to be considered. Interest points can be efficiently described using SIFT features based on HoG. In our code, we use the implementation provided in VLFeat library[1]. An important parameter allows us to modify the peak-threshold for feature acceptance. This thresholding makes it possible to remove misleading points such as background ones, and will be described in more details in Sec. 1.3. After feature detection is completed, we use another function from VLFeat which tries to find the best set of matches between the two image described by SIFT features. The coordinates of the point pairs are then transformed in homogeneous coordinate.

The following steps now differ among the basic version of Eight-point Algorithm, the Normalized one and the Normalized version using RANSAC to find the best transformation. We will describe the main differences and analyze experiment results in the following sections.

[1] http://www.vlfeat.org/overview/sift.html

1.1 Different approaches to Eight-point Algorithm

Starting from the representation of point descriptor in homogeneous coordinates, the Fundamental matrix we aim to recover is defined by the equation:

$$\mathbf{x'}^T\mathbf{F}\mathbf{x} = 0$$

which can be also expressed as:

$$\mathbf{A}\mathbf{F} = 0$$

where **A** is the outer product between **x'** and **x** reshaped into a vector. From this form, the Eight-point algorithm recovers the Fundamental matrix **F** as the column with smaller eigenvalue from the matrix **V** obtained by SVD decomposition of $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. The additional constraint of making it singular is then manually enforced.

The Normalized Eight-point algorithm follows the same idea, with the difference of a preprocessing of the vectors **x'** and **x** to normalize the data to a mean of zero and an average distance from the mean of $\sqrt{2}$. The corresponding denormalizing transformations are applied in the end to the recover the Fundamental matrix:

$$\mathbf{F} = \mathbf{T'}^T\hat{\mathbf{F}}\mathbf{T}$$

Finally, the Normalized Eight-point algorithm with RANSAC introduces an iteration over the selection of 8 point matches from which the Fundamental matrix is computed. The accuracy of this transformation with respect to the whole set of point matches can be computed using the Sampson distance. Each point pair is then classified as inlier or outlier comparing the distance value to a threshold empirically set to $10\mathrm{e}{-5}$. The Fundamental matrix producing the higher number of inliers is then selected as the optimal one.

The geometrical interpretation of the Fundamental matrix is that it represents the mapping from one point in one image plane to the corresponding epipolar line in the second image plane. If the epipolar constraint is satisfied, the matched point in the second image plane will lay on this epipolar line. In the experiments reported in the following section, we will see how this constraint is satisfied for accurate Fundamental matrices.

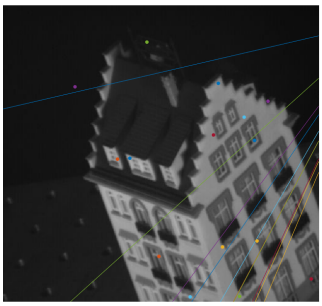
1.2 Experiment results

After implementing the three versions of Eight-point algorithm we discussed, we ran some experiments to verify ad evaluate performances on the provided dataset. In theory, since the approaches without RANSAC include outliers (bad matches) in the computation of the Fundamental matrix, their results should be worse than the approach with RANSAC. In addition, Hartley [1] shows how the basic version of the algorithm is very unstable due to noise. He then proposes a simple normalization of the data to significantly improves the stability of the algorithm.

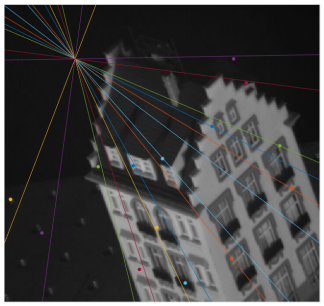
These differences are confirmed from the results of our experiments, as shown in Fig. 1. In these images we check for the epipolar constraint to be satisfied by taking into consideration a subset of matches points. Ideally, every colored point on the transformed image should lay on the epipolar line representing the constraint enforced by the corresponding point in the original image plane.

In the first three subfigures we remove the peak-threshold ($\tau = 0$) to emphasize differences in the three variants of the algorithm. The first subfigure makes it evident that the Basic version of Eight-point algorithm is highly unreliable and sensitive to outliers. Indeed, all of the points are very distant from the corresponding epipolar line. Improvements can be noticed in the second subfigure thanks to the introduction of data normalization. The points are now closer to the epipolar lines but they still do not lay on them perfectly. In the third subfigure we see how RANSAC allows the removal of the outliers, or wrong matches, when computing the Fundamental matrix. In this case, most of the points perfectly lay on the corresponding epipolar line.

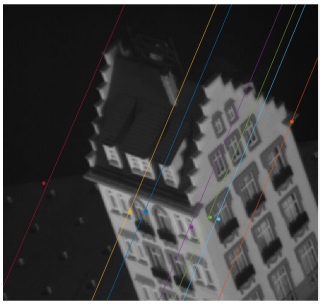
In the last subfigure we show how increasing the threshold value to $\tau = 10$ drastically improves the accuracy in reconstructing the 3D transformation. In this case we are plotting all of the matching points (49) found from SIFT features. Notice how all of them perfectly lay on the epipolar lines, thanks to the removal of misleading point descriptors (e.g. the ones from the background) found without threshold. We will analyze in more details the effects of peak-threshold on performances of Eight-point algorithm in the next section.



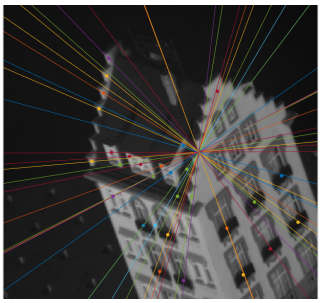
(a) Basic Eight-point ($\tau = 0$)



(b) Normalized ($\tau = 0$)



(c) Normalized + RANSAC ($\tau = 0$)



(d) Normalized + RANSAC ($\tau = 10$)

Figure 1: Performances of different version of the algorithm, with and without threshold τ.

1.3 Removing misleading points by tuning SIFT threshold

After comparing the results obtained with these different versions of Eight-point Algorithm, we tried to further improve the accuracy levels we obtained by tuning the peak-threshold parameter. Thinking at the idea behind RANSAC algorithm, we realize how it heavily relies on the random selection of matches subsets to find good suboptimal approximations of the real transformation. Indeed, the higher is the number of matches found, the lower is the chance to find the best subset among those point pairs. For this reason, we decided to experiment with the peak-threshold defining the acceptance criteria for feature selection. Decreasing the number of 'low-quality' matches (such as background ones) significantly increases the probability to find subsets in point pairs corresponding to high-quality transformations. This thresholding can be done by tuning the corresponding parameter in *vl_sift*. In Table 1 we show how RANSAC performances changes when varying the threshold value.

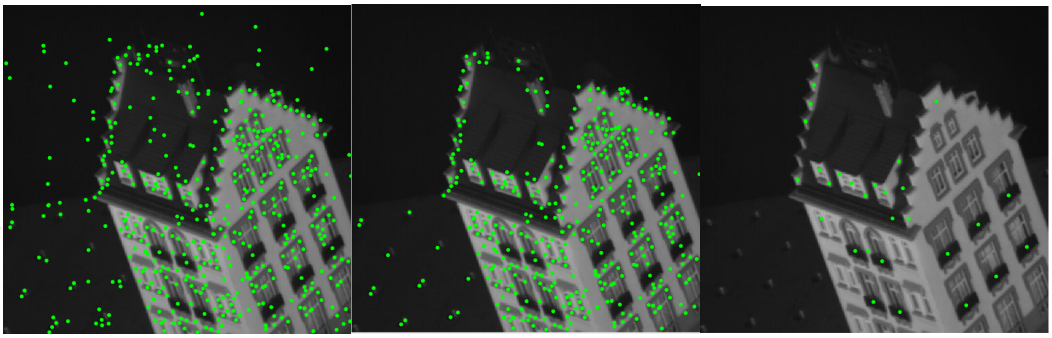
It is evident that higher thresholds reduce the total number of matches found, since fewer features are used to describe the image pairs. Also, the ratio of inlier points (again using as criteria Sampson distance ≤ 0.00001) significantly increases from *81.3%* to *94.7%*. Notice that increasing the threshold value over 12.5 results in RANSAC failing from not having enough matches (less than 8). Ideally, the best transformation possible could be obtained without thresholding by trying all possible subsets of matches. Since this would result at least in exponential time complexity, we show that thresholding is a very effective way to find almost optimal solutions with very few iterations. The average number of iterations to converge was found to be 11 for the higher threshold tested (12.5).

Table 1: Performances of RANSAC for Eight-point algorithm changing peak-threshold in Feature Selection.

Peak-Threshold	Inliers Ratio	Matches found	Iterations to Convergence
0	0.813	530	28
0.5	0.884	520	25
1	0.911	453	26
5	0.928	293	26
10	**0.947**	49	22
12.5	0.943	14	**11**

Finally, we test performances of the versions of Eight-point Algorithm with and without normalization using $\tau = 10$ and we notice how matches are as good as the ones found using RANSAC, with no outliers and with all of the epipolar constraints satisfied.

In image 2 we show that increasing the threshold parameters prunes less meaningful feature points. In the last subfigure, we notice how point descriptors are placed in meaningful positions to accurately describe characteristic shapes in the walls and windows of the building.



(a) Threshold = 0 (b) Threshold = 1 (c) Threshold = 10

Figure 2: SIFT feature points found using different peak-threshold values.

2 Chaining

We have implemented the basic version of chaining algorithm which takes feature points between two images and compares its descriptors. The result of this comparison is saved in the Point-View matrix 3. Every image represents one row of output matrix while each column represents one unique feature point. The elements of this matrix are feature points in an image local camera coordinate frame. The matrix is sparse and tends to not have values above the diagonal. This is expected because we match only against previously seen feature points. An exception is if we wrap around dataset and get to the first image row. Furthermore, not every point seen in the n-th image can be observed in all previous n-1 images. This makes the matrix sparse. We also assume that if we loose track of some feature point in the n-th image we cannot re-observe it again in the (n+1)-th image. This makes no holes in columns of the matrix. It also helps again matching two different points on similar location in an image with similar descriptors which are from different real locations.

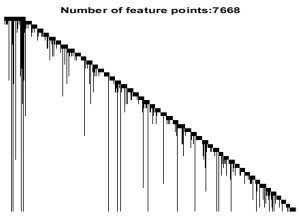
We have experimented with different parameters. The number of unique features is dependent on SIFT peak-threshold values 2. The higher threshold causes less points in the image which makes resulting matrix denser. It also makes result more robust because feature points are usually farther apart (only best key point in area is selected) which makes descriptor matching more precise.

In order to make the matrix more robust to outliers, we implemented two types of match filtering.

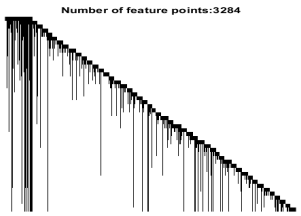
First, we use the distance between points to reject two points which have similar descriptor but location far apart. On one side, this makes the matrix even sparser as can be seen in Figure 3 c). On the other hand it is more certain that points in the same column are really representing the same point in 3D space.

Secondly, we use RANSAC Eight-point algorithm described in the previous section to insert only inliers into the Point-View matrix. This improvement doesn't change density of matrix significantly Figure 3 d). However, it redistributes matches more evenly across the matrix. This should result in removing mistakes from too dense areas of the matrix.

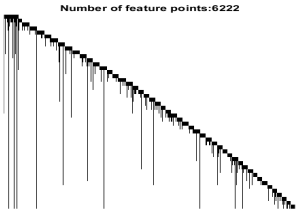
Lastly, we have also implemented an additional column filtering to make the matrix denser by removing less informative columns (those whose points appear in less views than a threshold). The results are shown in 2.1



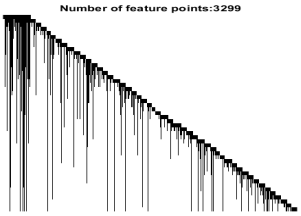
(a) Peak-threshold = 1



(b) Peak-threshold = 5



(c) Distance=1.3, RANSAC = false



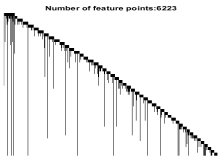
(d) Distance=false, RANSAC = true

Figure 3: Density and number of points inside of Point-view matrix with different parameters.The first row represents different setting for SIFT as shown in 2. The second row has this threshold set to 5 and turns on Distance match filtering and RANSAC match filtering. The distance parameter represents euclidean distance between feature points.

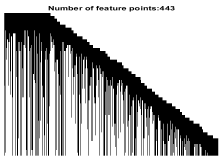
2.1 Point-View matrix reduction

Since it has been used in all the experiments we will talk about later, we present here the first improvement we implemented.

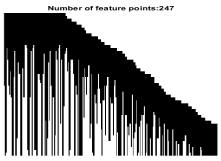
One of the possible improvements to structure from motion algorithm is to use denser Point-View matrix. If the matrix is denser the algorithm finds more points inside the biggest dense block and is able to better estimate the real point location. The initial matrix is really sparse with only a couple of columns representing the same points.The majority of the columns has less than 5 matching images. Every real 3D point which is reconstructed from a small number of rows has a high chance of introducing noise or outliers into reconstruction. Therefore, we remove these columns. The results of this operation is shown in Figure 4. We experiment with different thresholds to decide whether to drop a column depending on the number of rows in it. The main disadvantage of this approach is that a smaller number of columns reconstructs a smaller amount of points which leads to less points in the final structure we generate.



(a) Drop = 0



(b) Drop < 5



(c) Drop < 10

Figure 4: Density of Point-View matrix with different parameter of column dropping.

We also tried to build the Point-View matrix both with and without matching the last view to the first one (as required in the assignment). Actually, the result of this matching is that we retrieve very few matches (we had to decrease the threshold values to get enough matches to run RANSAC over them); as a result, most of the points of the first view are added as new columns in the matrix, which are later discarded by the column filtering described above (whichever threshold greater than 1 will discard them). We tried using very low thresholds for matching the descriptors and removing the filtering by distance on this pair of views. Again, the only matches we could get were with a few points which

were already matched (i.e. points whose columns included all the views), which means this match is not adding any new information. For this reason, we have almost always used the matrix without this last match for our experiments.

3 Structure from Motion

If a dense *Point-View Matrix* is available, we can reconstruct the *point cloud* using the *factorization* algorithm over it. However, this is not always the case since it requires every point to be seen from all points of view. For a general sparse matrix we can still exploit the *factorization* algorithm by decomposing the sparse matrix in a set of (overlapping) dense sub-blocks and by *factorizing* each of them independently. The outputs of *factorization* are the projection matrices of each of the views and the reconstructed 3-D points cloud. Since the dense blocks share some subset of the points and we know which point in a cloud corresponds to which point in another, we can merge those clouds together estimating a transformation between the coordinates in the two clouds for the shared points and, then, applying it to the not shared ones. The idea is to start from a good dense block, use its reconstruction as the base one and, then, gradually merging all other blocks in it.

So, what is left to define is how to compute the dense blocks and in which order merge them.

In this section we are going to explain the basic version of *Structure from Motion* we implemented and the results we obtained. In this basic version, the set of blocks is built by taking each consecutive N views and all the points they share. As required in the assignment we experimented with $N = 3$ and $N = 4$. Then, blocks are parsed and merged in the same order as they are built, i.e. following the order of the views. Therefore, the block containing the first N views will be used as the base. The resulting process is equivalent to iteratively add a new view (we consider the next dense block which will contain the new view), compute the transformation for the points from the new block, adding as many new points as possible (the ones shared by all the views in this new dense block) and applying this transformation to them.

3.1 Results

Now we present the results found with this setting.

Before experimenting with the matrix we built from *Chaining*, we tried the *Point-View Matrix* provided. Actually, this matrix is completely dense and, so, it represents the ideal case. For this reason, we can directly run the *factorization* algorithm over it and we don't need to deal with missing points. Some views of the reconstructed cloud can be seen in Fig 5.

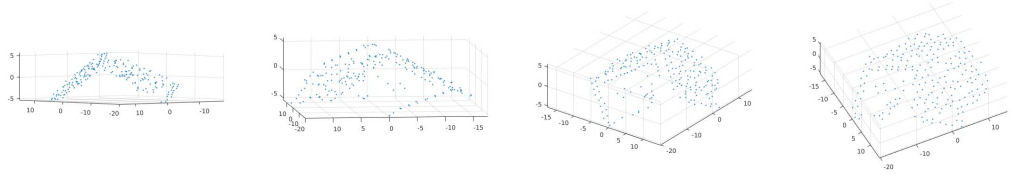
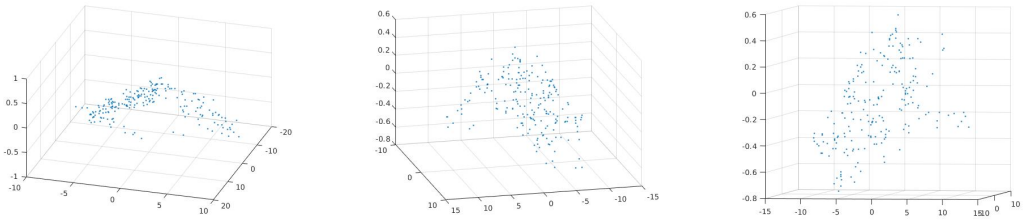


Figure 5: Some views of the reconstructed cloud from the factorization of the provided data

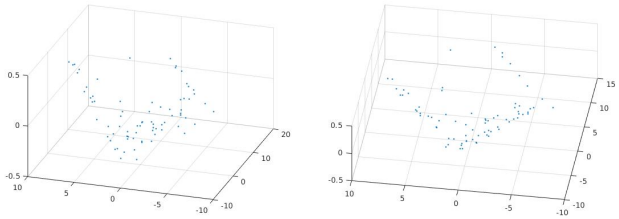
Although the cloud seems a bit flattened, it is possible to recognize the shape of the house.

In order to try our *Structure from Motion* implementation we used the output of the *Chaining* task. In the following experiments we used the *Point-View Matrix* generated using *RANSAC*, a minimum number of 10 views for each point, both a *SIFT* threshold and *matching* threshold of 5 and a *distance* threshold of 1.3. As required in the assignment we experimented with $N = 3$ (see Fig 6) and $N = 4$ (see Fig 7). Since we don't have any objective measure to evaluate our results and we don't even have a ground truth to compare with, we are can only stick to a visual evaluation. From this point of view, both the reconstructions seem good, though the reconstruction with step 4 seems to have a bit more noise.

We can also compare these results with the ones obtained using the provided data (Fig. 5). As expected, by factorizing the provided dense matrix we found visually better results as the shape of

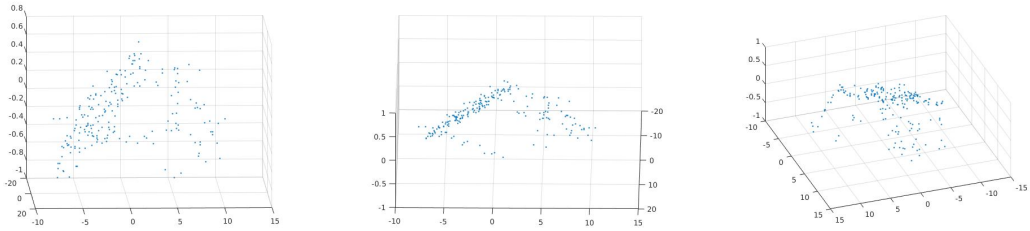


(a) Different views of the final reconstructed cloud

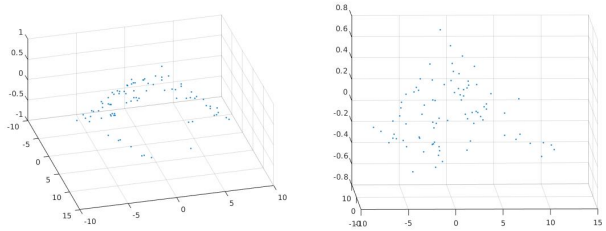


(b) Different views of the cloud reconstructed using one of the dense blocks

Figure 6: Experiment with step 3



(a) Different views of the final reconstructed cloud



(b) Different views of the cloud reconstructed using one of the dense blocks

Figure 7: Experiment with step 4

the cloud is neater, the surfaces are less noisy and the edges are sharper. Anyway, the reconstructions from our point-view matrix are still able to recover the shape of the house.

4 Additional Improvements

As explained in Sec. 2.1, the first improvement we implemented was to make the Point-View matrix more dense. Now, we will explain the other improvements implemented for *Structure from Motion*.

4.1 Structure From Motion: Affine Ambiguity Removal

In order to improve our results, we implemented *Affine Ambiguity Removal*. The idea is to add *orthographic* constraints to the factorization algorithm. In Fig 8 we show the results applying this improvement to the experiment with step 3 shown in Sec 3. Comparing them with the ones in Fig. 6,

they seem very similar. However, it is important to notice the different scale on the Z axis: indeed, without the affine ambiguity removal, the reconstruction is deformed and compressed (the visual result is similar because Matlab rescaled the axis for better visualization).

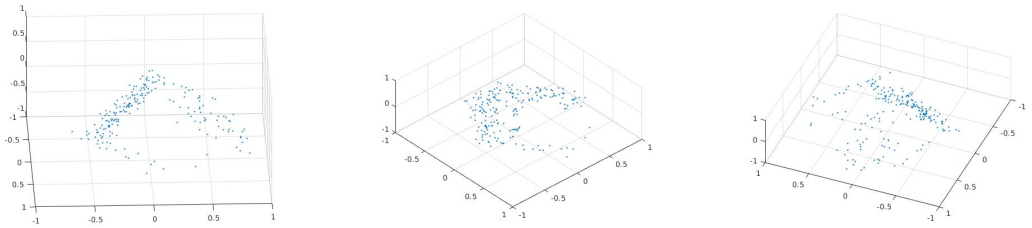


Figure 8: Experiment with Affine Ambiguity Removal

We also tried affine ambiguity removal on the dense matrix provided. The results can be seen in Fig 9.

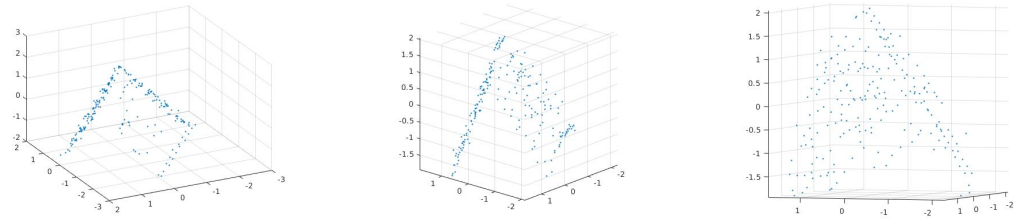


Figure 9: Experiment with Affine Ambiguity Removal on the dense matrix provided

4.2 Structure From Motion: Average of Reconstructions

As another improvement, we tried to change the *Structure from Motion* pipeline described in Sec 3 by updating the estimations of all points at each iteration. More precisely, after processing a dense block and computing its transformation to the coordinates system of the point cloud, instead of adding to the cloud only the new points we also update the estimation of the other points (the ones used to compute the transformation). The estimation of a point is a weighted average among the estimations from all the dense blocks containing that point, where the weights are proportional to the number of views in a dense block (the idea is that more views a block contains, more reliable its reconstructions are). In Fig 10 we show the results with this improvement (the other settings are the same as in Sec. 4.1).

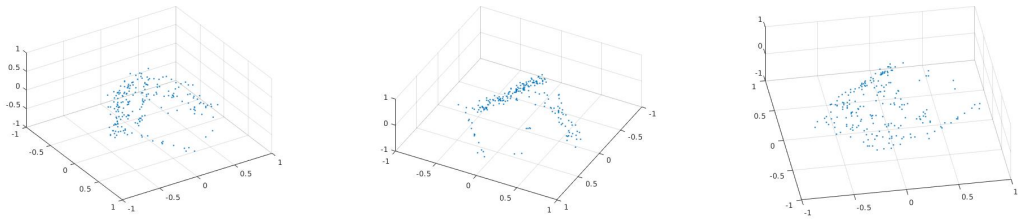


Figure 10: Experiment with Average of Reconstructions

In this case we don't see relevant differences with respect to the results without this improvement. However, plotting the reconstructed points cloud during the iterations of the algorithm, we see that the points do not drift much due to the updates; they rather tend to move around some fixed point. We interpret this an indication of the robustness of the reconstructions with respect to the set of views (i.e. the dense block) used. Indeed, this means that the spatial arrangements of the points with respect to the ones reconstructed from different dense blocks are roughly the same.

4.3 Structure From Motion: Maximum Spanning Tree over the Dense Blocks

As a further improvement, we tried to implement a different type of ordering of the dense blocks in the *Structure from Motion* pipeline. For this purpose, we draw inspiration from the *stitch graph* in [2]. We first compute a set of dense blocks to use. Afterwards, we build a graph where the nodes are these dense blocks and we add an edge between each pair of nodes with a weight proportional to the number of points the two blocks share. Then, we take as *root* the block with the largest number of cells (number of points times number of views) and compute the *Maximum Spanning Tree* rooted on this node. Finally, orienting the edges from the root to the leaves, we order the blocks according to the topological sort of the directed tree (i.e. first the root, then its children and so on until the leaves). The idea is that we want to maximize the number of points already known among the ones in a block when processing it in order to have an estimation of the transformation as accurate as possible (if only few points are know whereas most of the points of the block are new, the transformation will have high uncertainty and we would add to the cloud many points whose estimation is not very accurate). Notice that we actually did not implement exactly the same procedure suggested in [2], as they used Dijkstra's algorithm on the graph to find for each block the path which maximizes the points shared in its blocks. However, the problem of searching the path with maximum weight is equivalent to the problem of the shortest path with negative weight, which can't be solved by the standard Dijkstra's algorithm. Anyway, this approach performs "local" optimizations (the path for the single blocks are optimized independently). For this reason, we have thought that a *Maximum Spanning Tree* is a more appropriate solution, as it optimizes for all the blocks together.

In Fig 11we show the results using this improvement, while the other setting of the experiment are the same as in Sec. 4.2. In this case, we computed the set of dense blocks as before (each 4 consecutive views), but instead of parsing them from the first to the last one, we parse them according to the topological sort of the maximum spanning tree.

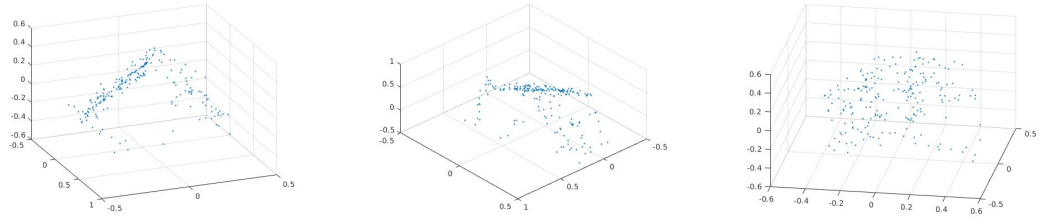


Figure 11: Experiment with Maximum Spanning Tree over the Dense Blocks

Unfortunately, we don't see any relevant visual improvement in these results with respect to the basic implementation. It is interesting to note that, in this case, the *average of estimations* improvement led to better and less noisy results. Our interpretation is that the *tree* approach can link dense blocks that don't share any views. Although we noticed in Sec 4.2 that the reconstructions are rather robust with respect to the set of views used, some reconstruction can still be a bit noisy or two blocks containing not very close views can produce slightly different reconstructions. While in the basic implementation we parse blocks in order (and so we consider consecutive views), in this case when a block is processed it is possible that no blocks nearby have been processed yet. Therefore, sometimes the estimated transformations might be not accurate with a resulting noisy reconstruction. Conversely, using the average between the estimations seems to reduce this noise.

Due to the page limit we can't include the results for this experiment (the one without the *average*) here, but this experiment as well as many others can be found in the "output/StructureFromMotion" folder in our submission.

4.4 Structure From Motion: Maximal Cliques

Finally, we implemented two other algorithms to find the dense blocks. More precisely, we implemented the method suggested in [2] and the method described in the appendix of [3] to find (a subset of) the maximal cliques in the connectivity matrix. Of course, these two methods work only with the tree-based ordering described above, since there is no natural ordering of the blocks.

Again, we saw a behavior similar to the one explained in 4.3 with the *average of estimations*, but we haven't been able to find any relevant visual improvements. The results found using the method from [2] and [3] are shown respectively in Fig 12 and Fig 13.

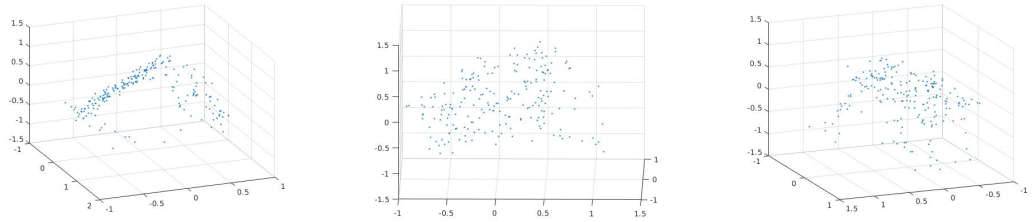


Figure 12: Experiment with Maximal Cliques Method from [2]

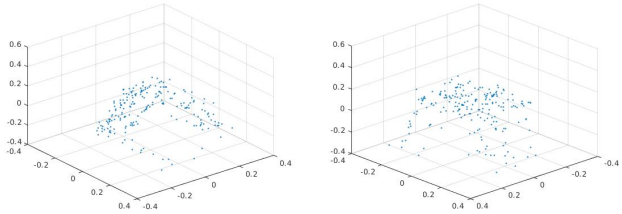


Figure 13: Experiment with Maximal Cliques Method from [3]

In conclusion, unfortunately we have to notice that most of the improvements we implemented for *Structure from Motion* did not have significant impact on the results. However, because of the lack of quantitative measures and a ground truth, our evaluation is unlikely to be objective. Moreover, the problem we have been working with is a toy problem and only small number of views were available; we expect our improvements to have larger impact in case of more complex problems, with many more views or with a less regular Point-View matrix.

Conclusion

Working at this assignment we had the chance to experiment and understand more deeply concepts related to Epipolar Geometry, 3D affine transformations and Structure from Motion. After implementing Eight-point algorithm, we have seen how Normalization and RANSAC are key-points to good performances. This evaluation was done by visualizing the satisfaction of the epipolar constraint using the computed Fundamental matrix. For chaining, we have seen how the matrix computed on real data is always very sparse. We have discussed some ways to improve on this problem, for example removing columns from the matrix which are not informative or performing match filtering. In Structure from Motion, we discussed how results change when including different sets of camera views to reconstruct the matrices. We have also seen differences from the reconstructions created from the ground-truth dense matrix or from the sparse matrix we generated. Finally, we have visually presented the results, noticing how meaningful reconstructions can be obtained but also how they are not highly detailed nor realistic.

References

[1] R. I. Hartley. In defense of the eight-point algorithm. *TPAMI*, 1997.

[2] F. Rothganger and S. Lazebnik. 3D Object Modeling and Recognition Using Local Affine Invariant Image Descriptors and Multi-View Spatial Constraints. 2004.

[3] F. Rothganger and S. Lazebnik. Segmenting, Modeling, and Matching Video Clips Containing Multiple Moving Objects. 2006.