Structure from Motion
Computer Vision 2

Nichita Diaconu (11737980), Radu Sibechi(11808527),
Ruben-Laurentiu Grab(11609923)

May 18, 2018

1 Introduction

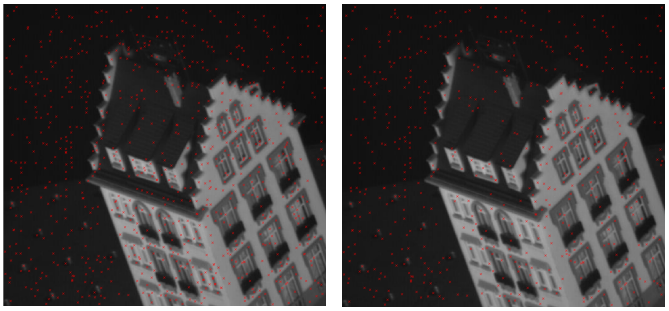
Structure from Motion is a technique that tries to reconstruct the 3D coordinates of objects that undergo various motions and are captured as 2D images from a moving camera. For this assignment we first implemented different variations of the Eight-Point Algorithm. Secondly, we represent the output of the Eight-Point Algorithm into a sparse patch-view matrix whose columns represent surface patches and rows represent the images they appear in. Finally, we tried to reconstruct the 3D coordinates of the points in each view based on the Structure from Motion algorithm.

2

3 Fundamental Matrix

3.1 Selecting points

In order to extract the fundamental matrix between two corresponding clouds of points, we first need to extract a correct set of points from each image. For this purpose, we used *SIFT* descriptors. We extracted them using the `vl_sift` method from the *VLFeat* MATALB library. We tried different setups of parameters in order to extract only points that are of interest for us, namely those that appear only on the house. We observed in our experiments with the `vl_sift` method that many points disappeared from the background for some setups. However, in the same time, many points on the house disappeared as well. Figure 1 shows the results we got when using `vl_sift` without any optoins, and when using it with options `EdgeThresh = 4` and `PeakThresh = 0.001`. We decided on extracting the descriptors without using any options, and instead found other ways of discarding the points on the background, as it can be seen in section 3.1.1.

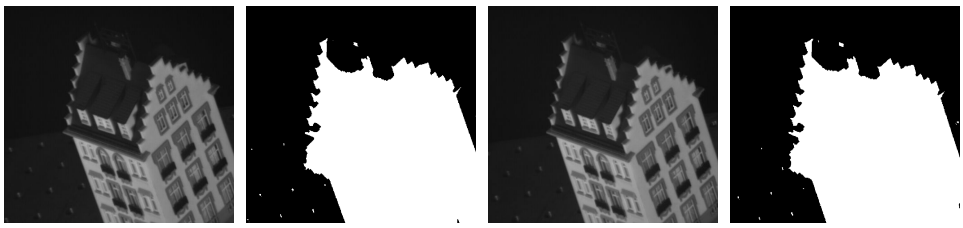


(a) `vl_sift` (b) `vl_sift` with options

In the second image one can see that some points from the background disappeared, however with the cost of some points on the foreground.

3.1.1 Background removal

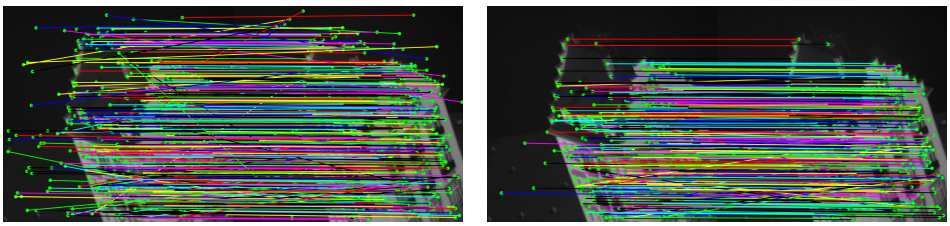
As stated above, we have noticed that besides the interest points detected on the house, the algorithm also detected multiple points on the background, or on areas of the roof that were very dark and could not be distinguished from the background. Although at matching time many of these points won't be matched, thus not taken into consideration, there are still some points in the background that are matched. In order to mitigate this aspect, we decided on running a simple background-foreground segmentation using the `activecontour` MATLAB function.



(a) Frame 1 (b) Frame 1 - mask (c) Frame 5 (d) Frame 5 - mask

Frames and their corresponding segmentation mask. The foreground is represented with white, while the background with black.

Although this segmentation is not close to human perception, as seen in figure 2, it helps in filtering most of the points on the background and in hard to distinguish areas, such as the roof. Figure 3 shows the difference in the quality of the selected pairs of points. One can notice that correspondences whose both points are in the foreground in their corresponding images are taken into consideration. However, because the masks shown in figures 2b and 2d are contain some misclassified patches, we see that in figure 3b there are still a few correspondences on the background. These will be taken care of, when we will introduce other techniques as described in the following sections.



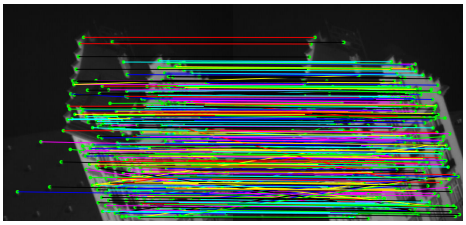
(a) Before background removal. (b) After background removal.

Comparison of the correspondences of the points found in frame 1 and 5, before and after discarding points that are not in the foreground. The matchings were made with `vl_ubcmatch` method called with the threshold parameter set to default(i.e. 1.5).

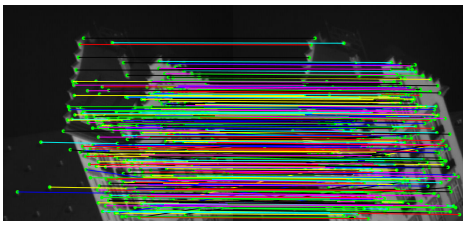
3.1.2 Sift matching threshold

Having the interest points selected, we used the method `vl_ubcmatch`. We have noticed that increasing the threshold parameter of this function, our results improve. In figure 4a we can see that the line of some matchings don't have the same orientation as the others. One can see a black line connecting two wrongly matched points, running from the bottom-left corner to the top-right corner. As we increase the threshold, we observe that many of the wrongly matched points, such as the line previously discussed and some background matches, disappear.In figure 4d we observe just a couple of background points. However, increasing this parameter comes with a cost, discarding some good matching. This is the reason why we chose to set it a value of 3 for the results displayed here.

(a) Threshold set to 1.



(b) Threshold set to 1.5(default).



(c) Threshold set to 3..

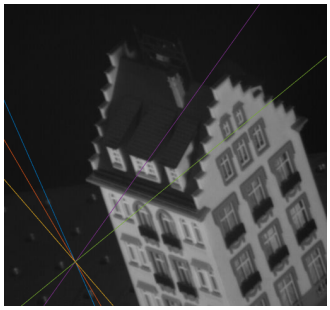
(d) Threshold set to 5.

Comparison of the correspondences of the points found in frame 1 and 5, after discarding points that are not in the foreground, for different thresholds.

3.2 Eight-Point Algorithm

Having an appropriate set of matching points, we ran the Eight-Point-Algorithm on them. The results can be seen in figure 5. In figure 5b we observe that the epipolar lines do not meet the epipolar constraint, as they do not pass through the corresponding points given in the first image.

(a) Points in 1^{st} frame.

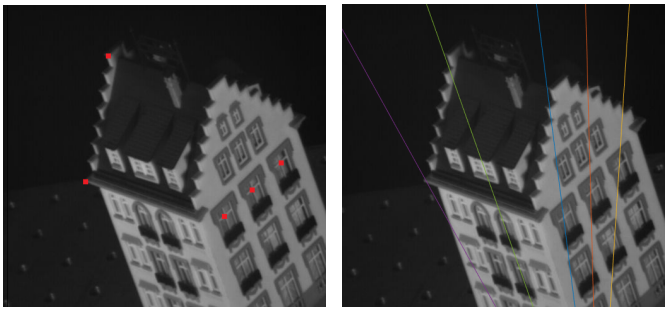


(b) Epipolar Lines in 5^{th} frame.

Points in the first frame, and corresponding epipolar lines in the fifth frame, using the standard Eight-Point Algorithm. The results are not satisfying.

3.3 Normalised Eight-Point Algorithm

We also implemented another version of the algorithm, in which we first normalised the points. The results we got after we ran the algorithm on the points can be seen in figure 6. Compared to the standard algorithm, results are better: in figure 6b we observe that the epipolar lines meet the epipolar constraint, as they pass through the corresponding points given in the first image. One can observe that the Epipole, according to this fundamental matrix, is found outside the image, somewhere under it.

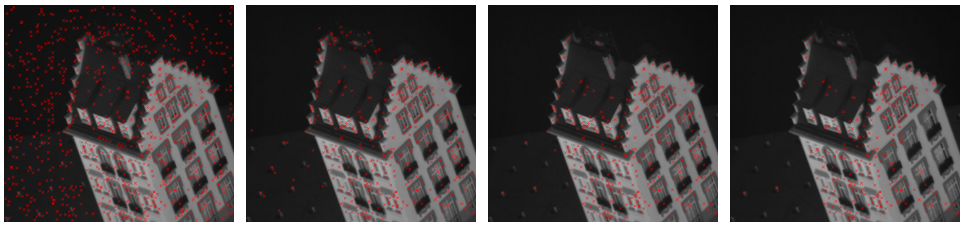


(a) Points in 1^{st} frame. (b) Epipolar Lines in 5^{th} frame.

Points in the first frame, and corresponding epipolar lines in the fifth frame, using the Normalised Eight-Point Algorithm.

3.4 Normalised Eight-Point Algorithm with RANSAC

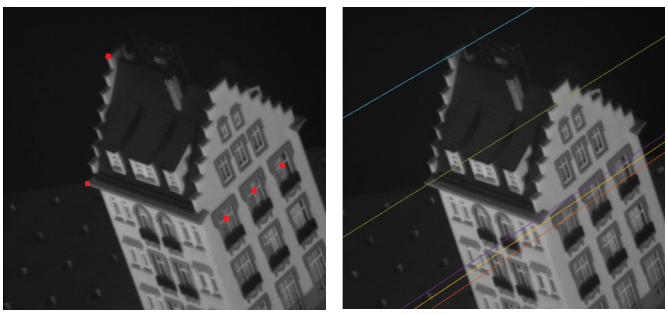
We then turned our attention to finding the fundamental matrix using RANSAC. We believed this will help us discard some unwanted points, thus having a more accurate fundamental matrix. Indeed, analyzing figure 7, we see how we gradually reject unwanted points. From all the sift interest points detected(fig.7a), we remain with only a part after we match them with another frame(fig.7b). We discard some more points after taking into consideration only the points in the foreground(fig.7c). Lastly, using RANSAC, we are able to come up with a smaller set of points(fig.7d).



(a) All points. (b) Matched points. (c) Foreground points (d) After RANSAC

Evolution of interest points taken into consideration after each process.

The results we got after we ran this version of the algorithm on the points can be seen in figure 8. These results seem to model the best the position from where the two frames were shot, as they are almost parallel, which is to be expected for two images shot form almost the same position. In figure 8b we observe that the epipolar lines, indeed, meet the epipolar constraint, as they pass through the corresponding points given in the first image.

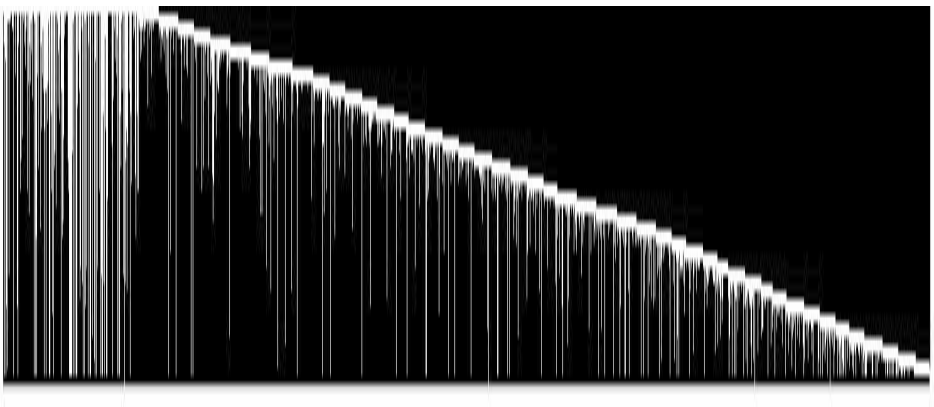


(a) Points in 1^{st} frame. (b) Epipolar Lines in 5^{th} frame.

Points in the first frame, and corresponding epipolar lines in the fifth frame, using the Normalised Eight-Point Algorithm with RANSAC.

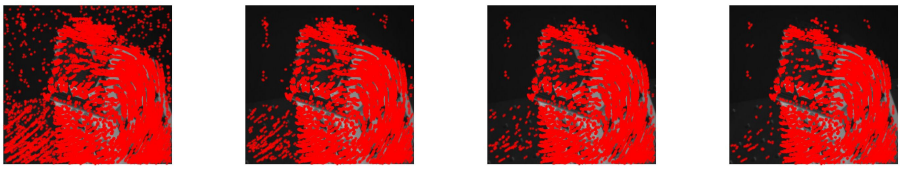
4 Chaining

The matching process described in the previous section outputs affine regions matched across pairs of views. These matches can be represented in a single match graph structure, where each vertex corresponds to an affine region, labeled by the image where it was found while arcs link matched pairs of regions between images. Intuitively, the set of views of the same surface patch forms a connected component of the match graph, which can in turn be used to form a sparse patch-view matrix whose columns represent surface patches and rows represent the images they appear in. In order to construct the point-view matrix, we start from any two consecutive image matches and add a new column to the point-view matrix for each newly introduced point. If a point is found which is already introduced in the point-view matrix, we add it to the point-view matrix using the previously defined point column.

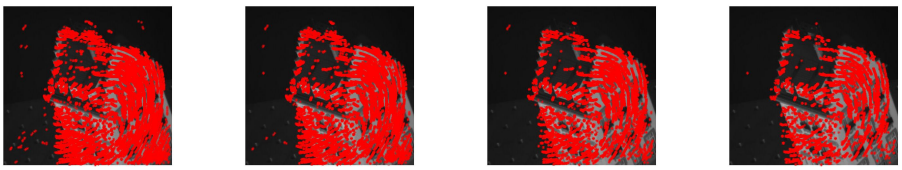


PVM visualization (background removed, threshold 1.5). 0 values are black, non-zero are white

We experimented with different thresholds for the `vl_ubcmatch` function. In figures 10 and 11 we plot the matched points from the point-view matrix over all the 49 images. As we can see, increasing the threshold reduces the number of points, and more specifically highly decreases the number of background matches.



(a) Threshold 1.5 (b) Threshold 5 (c) Threshold 10 (d) Threshold 15

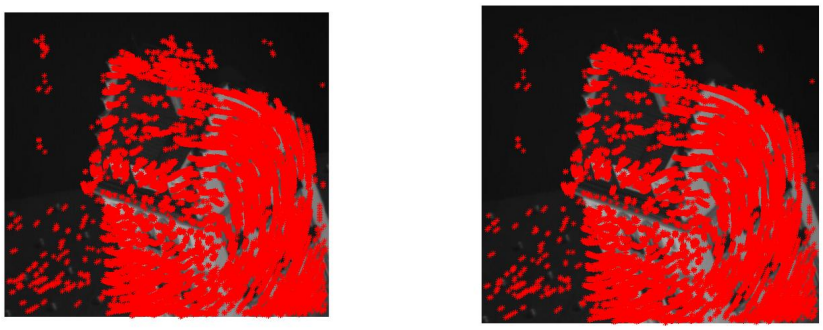


(a) Threshold 20 (b) Threshold 40 (c) Threshold 100 (d) Threshold 200

Points obtained from the PV matrix for different threshold values

As we can observe from figure 12, we can see that there is no much difference in background noise removal if we are using F to remove outliers or not. The reason

for this is that the background points are not actually outliers and they should not have been used.

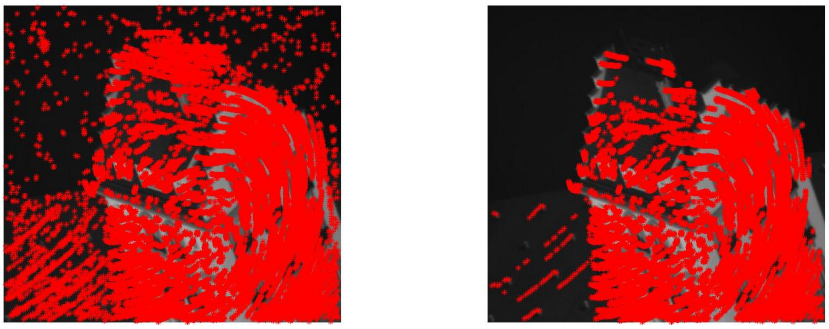


(a) No outliers removed

(b) Outliers removed

Difference in points obtained from pv matrix without and with removing outliers using F

To tackle this issue, we have also experimented with manually removing background sift in order to improve the performance of the algorithm, by only dropping background points and not foreground points. A mask of the house image is created and matches outside the mask are dropped. This approach has the disadvantage of being more computationally expensive and as such the algorithm becomes approx 10x slower. The results can be seen in figure 13



(a) Threshold 1.5 no background removed

(b) Threshold 1.5 background removed

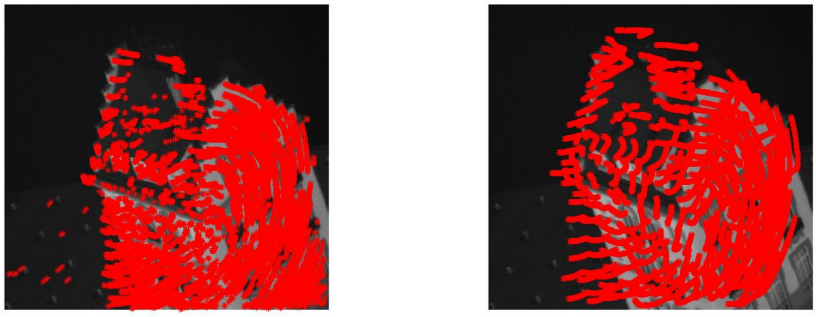
Difference in points obtained from pv matrix without and with removing background points

One small issue we bumped into is the fact that if we couple background removal with a **very** high threshold, the points that we are left with for doing the eight point algorithm are less than 8. In this case we just skip these images.

We can observe from 9 that our matrix is quite sparse and that the top right corner is full of 0's. The bottom part of our matrix is quite sparse as well, although it is not completely empty. We can see that the matrix resembles the shape of a diagonal matrix.

When comparing our matrix with the `PointViewMatrix.txt`, the first thing we observe is the density of the matrix, which has no non-zero values. It's worth noting, that their matrix contains less points, 200, when compared to ours, which contains 3000 points. Furthermore, their matrix contains two times more images than ours. Although the non-zero values are somewhat similar and make sense between the point-view matrices, the very high difference in density makes our matrix look less plausible. However, when comparing the points in the point view matrices between

them, as we have done in fig 14, we can see that indeed our matrix does look plausible. Furthermore, we can see that although our background points are farther from the house, they seem to be less than in their case. This is interesting, since we have **much more** points than they have in their matrix. An interesting observation is that the top of our roof has less points than in theirs.



(a) PV Matrix points from ours

(b) PV Matrix points from provided

Comparison in points obtained from our point view matrix and the provided one

One improvement to the Point-view matrix could be to have more data, such as the full dataset. Furthermore, we could try to match points between more than 2 consecutive images, such as 3, 4 or 5. This would ensure that we only keep the points that are shared across multiple views, leading to less 0's in our matrix, and as such dealing with the sparseness issue.

5 Structure from Motion

The structure from motion algorithm is following [1]. Based on the Rank theorem: Without noise, the registered measurement matrix is at most of rank three. The registered measurement matrix is the measurement matrix with the normalized points per view, the mean of each view is subtracted from the entire view. As a result, the origin of the world coordinates is placed at the centroid of the object points. The problem is that our measurement matrix also contains noise and therefore the rank is more than 3. By decomposing the matrix, into orthogonal matrices and singular values using SVD and selecting only the sub-matrices of the highest 3 singular values, we get the best possible shape and rotation estimates. Following Figure 5.1, the M matrix is the rotation and S matrix is the shape.

The true points coordinates are then recovered from S. Another problem that we face is that not all points are present in all views and therefore we cannot apply this method directly to the measurement matrix. Therefore, we divide the matrix into dense sublocks, apply SVD on the sublocks and merge the sublocks based on their common points by aligning them to the main view. This iterative manner of stitching uses Procrustes analysis or Iterative Closest Point. The initial results are not very satisfying:

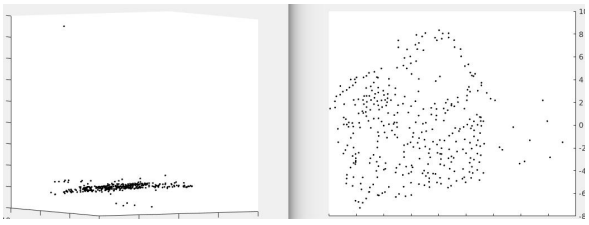


Figure 5.2 Initial results.

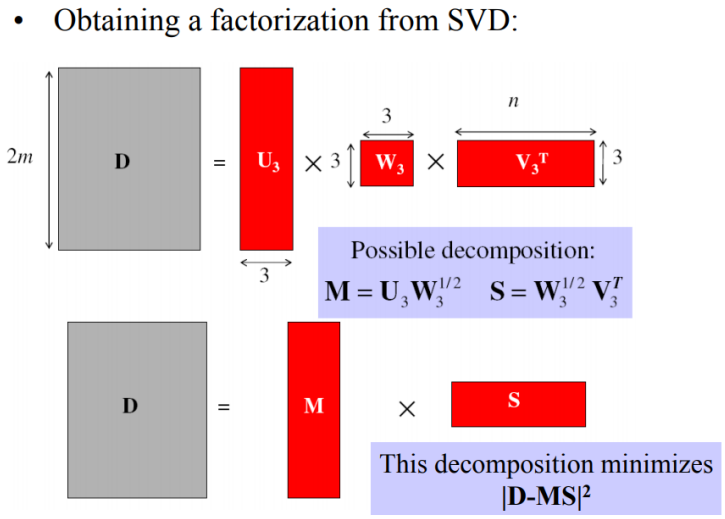
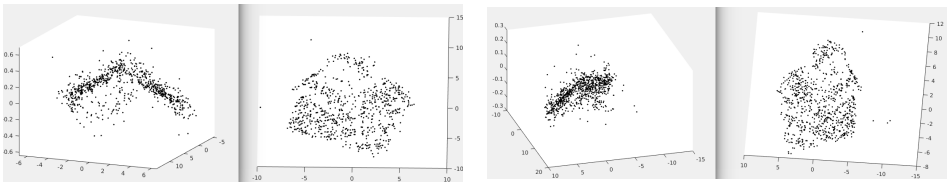


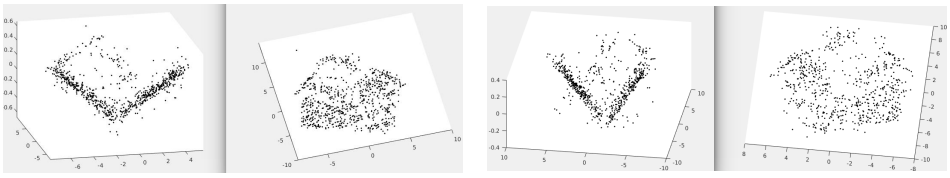
Figure 5.1 SVD decomposition after keeping only the sub-matrices corresponding to the highest 3 singular values.

Nonetheless, this is because of the points in the background, which are very far away, and make the depth of the house relatively small. Our solution to this is to remove the points which are reconstructed and are very far away. We tried to do this in 2 ways, either by removing them at each SfM iteration and reaplying SVD decomposition on the inlier points, either by removing them only after all the points were reconstructed. The former method works well in both cases, but is more demanding, while the latter method works only when there are few points in the background, but is less demanding. The results are shown below, we tried to depict the top view that minimizes the variance of the walls and a front view of the house:



(a) Few outliers (b) Many outliers

Figure 5.3 Comparison reconstructed view from matrices with few outliers(left) and many outliers(right), given that we remove them only after reconstructing the whole house



(a) Few outliers (b) Many outliers

Figure 5.4 Comparison between matching points with procrustes analysis(left) and ICP(right)

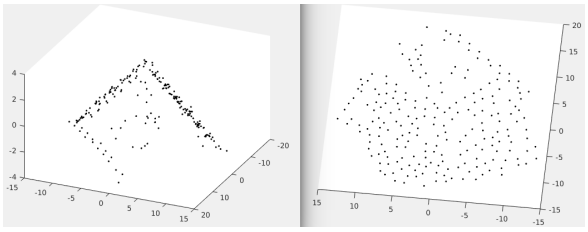


Figure 5.5 Reconstruction from the provided matrix PointViewMatrix.txt

In the end we can see that the most successful method is applying procrustes analysis and eliminating points in background after each dense block reconstruction, before stitching the denseblocks together Figure 5.5 left. The results from the factorization of only one dense block from our measurement matrix:

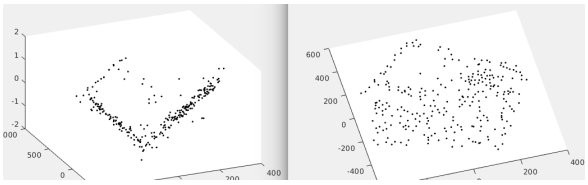


Figure 5.6 Reconstruction of only one dense block, the first dense block, using the best approach.

6 Additional improvements

From part 5 and Figure 5.1, M and S are linear transformations of the true rotation and shape estimates. More specifically,

$$D = MQ * Q^{-1}S$$

. We can impose the vectors in the motion matrix to be orthonormal resulting in orthographic constraints image axes are perpendicular and scale is 1. Solving this affine ambiguity renders the following improvements:

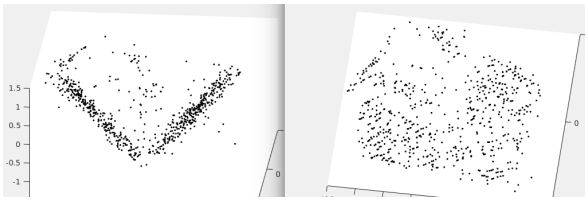


Figure 6.1 Reconstruction using Affine Ambiguity

This seems to not add a lot to our previous results and we believe this is the case because the big number of common points between views which made the results plausible in the first place. We hypothesize that there would have been a bigger difference if there were less points on the house or less noise.

Nonetheless, there is still a big local variance that is not dealt with, applying some smoothing method or further constraining the SVD could help.

7 Self-Evaluation

We divided out work in such a way so that we could work independently, and in the end to combine the work done by each of us. We also tried to divide it equally, and in the cases where that was not possible, we agreed to be flexible and to work together. Ruben-Laurentiu Grab took care of the implementation and all aspects

from section 3(Fundamental Matrix). Radu Sibechi, implemented all aspects from section 4(Chaining). Nichita Diaconu worked on the last part of the assignment, namely section 5(Structure from Motion) and 6(Affine ambiguity). However, due to the fact that all parts needed to be combined and work together, we collaborated and helped each other. In consequence, we feel that the work has been fairly shared among ourselves.

8 Conclusion

Working on this assignment, we learned about an interesting Computer Vision topic. We were encouraged to experiment and to see what fits best on our data. We learned the importance of fine-tuning and finding the appropriate hyper-parameters. Although we are satisfied with our findings, we believe that there are still improvements that could be made to the whole algorithm, as we stated in section 6(Additional improvements).

9 References

[1] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. IJCV, 9(2):137-154, November 1992