UNIVERSITY OF AMSTERDAM

Assignment 2 Report

Luca Simonetto - 11413522
Edgar Schönfeld - 11398272
Computer Vision 2

May 22, 2017

1 Fundamental Matrix

Eight-Point Algorithm

First, given two images, we extract SIFT descriptors using the *vl_sift* function of the VLFeat library. The *vl_ubcmatch* function is then used to find matches between the features of two consecutive images. The matches are shown in figure 1. Given these points, the matrix A is created such that $A = [(x_1 * x_2)^T, (x_1 * y_2)^T, x_1^T, y_2^T, (y_1 * y_2)^T, y_1^T, x_2^T, y_2^T]$, where $*$ denotes an element wise multiplication. The matrix F is then constructed from the last column of V of the singular value decomposition of A. The singularity of fundamental matrix is then enforced as described in the problem description. Figure 1 shows two consecutive images where the extracted SIFT descriptors are highlighted.

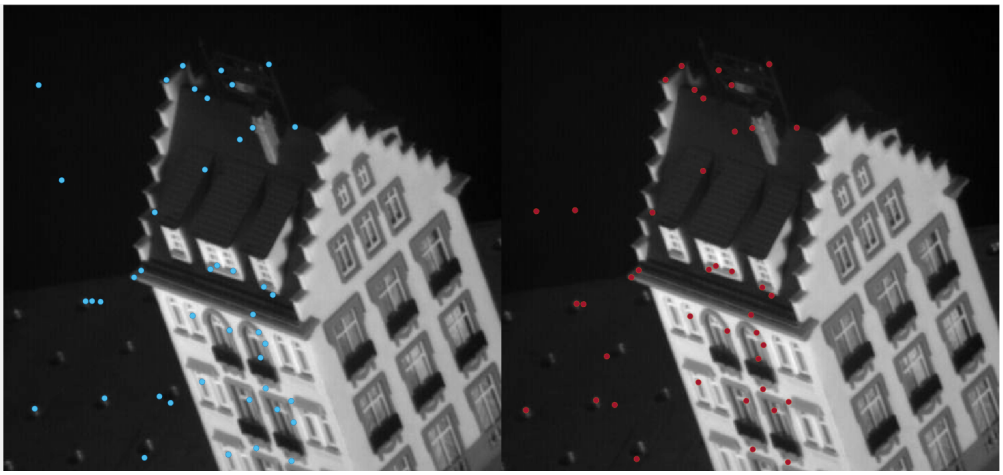


Figure 1: Matched SIFT points in two consecutive images.

Normalized Eight-Point Algorithm

This approach differs to the previous one in the following way: the points are normalized by the dot product $T \cdot p$ as described in the problem description. The two transformation matrices T_1 and T_2 are then used to recompute F according to $F_{new} = T_2^T \cdot F_{old} \cdot T_1$.

Normalized Eight-Point Algorithm with RANSAC

In this approach, after matching points are found, the F matrix is calculated along with the matching inliers that are found with the RANSAC algorithm. RANSAC is implemented as follows:

```
for max iterations do
    random sampling of 8 points
    find the F matrix
    calculate the Sampson distance of the points
    for every point do
        if the distance is less than a threshold then
            inliers ++
            save the point to a list of inliers
        end
    end
end
```

Algorithm 1: Pseudo code of the RANSAC implementation

Epipolar lines

In order to visualize the epipolar lines for two given images the following steps are undertaken:a SVD of F is initially computed. The epipole is then the rightmost column of V, divided by the rightmost singular value of V. The epipolar lines then stretch from the epipole to the inlier points determined by RANSAC. Figure 2 shows the epipolar lines calculated on two consecutive images.

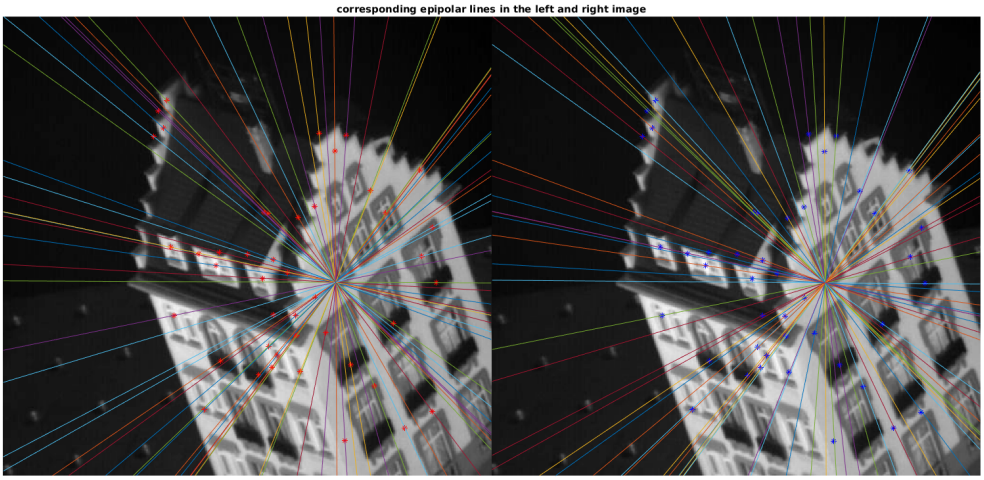


Figure 2: Epipolar lines and the points found in RANSAC

2 Chaining

After having implemented the required algorithm for fundamental matrix estimation and points matching, the pipeline for the creation of the point-view matrix has been created. This structure is defined as a binary valued $N \times P$ matrix, where N is the number of views (in our case 49) and P is the total number of points extracted from each view. In order to fill the matrix with the required data, each pair of consecutive images has been analyzed to match the frames of each picture, following 5 consecutive steps:

- **Descriptors matching**: in this phase, the descriptors of the two images ($image_1$ and $image_2$) have been matched, using the *vl_ubcmatch* function of the VLFeat library. This function matches two descriptors if their distance multiplied by a threshold is not greater than the distance of the first descriptor to all other descriptors [1]. The threshold value has been set to 2 (from the default 1.5), in order to extract only the matches with a high correspondence.
- **RANSAC on the matchings**: after having determined which pairs of points (sift frame) match, the RANSAC algorithm is used in order to find the best transformation that allows the highest number of matches to be preserved. A match is preserved if the Sampson distance between the transformed points and the target points is less than a specified threshold, that has to be discriminative enough to reject bad matches. The pairs that pass this condition form the inliers of that specific RANSAC iteration, and the algorithm completes when the highest number of inliers is achieved or the iterations have reached a set limit. At the end of the algorithm iterations, the best transformation matrix is discarded, as the important output for this section is the set of matchings.
- **Match check**: the returned matches consist of the points that are to be considered the same across the two images, meaning that their position is just changed according to the match. In order to correctly create the point-view matrix in this section, a point occurring in multiple views has to be correctly added in the same column of the matrix. In order to keep track of the various positions assumed by a particular point through the views a list of matrices has been created, where each matrix stores all the coordinates assumed by a certain point as well as the index of the view from which the point generated from. A sample of the data structure is the following:

$$P^1 = \begin{bmatrix} P^1_{1,x} & P^1_{2,x} & P^1_{3,x} \\ P^1_{1,y} & P^1_{2,y} & P^1_{3,y} \\ P^1_{1,z} & P^1_{2,z} & P^1_{3,z} \\ P^1_{1,i} & P^1_{2,i} & P^1_{3,i} \end{bmatrix} \Rightarrow P^2 = \begin{bmatrix} P^2_{1,x} & P^2_{2,x} \\ P^2_{1,y} & P^2_{2,y} \\ P^2_{1,z} & P^2_{2,z} \\ P^2_{1,i} & P^2_{2,i} \end{bmatrix} \Rightarrow \dots$$

  where P^1 is the first point, $P^1_{3,x}$ is the x coordinate of the third occurrence of that point (same with $P^1_{3,y}$ and $P^1_{3,z}$) and $P^1_{3,i}$ is the index of the image from which P^1_3 has been generated. In this way each point can be tracked in all views, and every newly discovered point is just a new matrix in the list. The view index added for each column will greatly ease the point-view matrix creation, as described next.
- **Point-view matrix creation**: after every pair of images has been analyzed (1-2, 2-3, ..., 48-49 along with 49-1), the final matrix can be created, by traversing the list of matrices, checking each view index and putting the corresponding data in the matrix, as described next. Each matrix contains all the coordinates assumed by the same point, so this data will form a single column in the final point view-matrix: the value of $P^1_{1,i}$ will dictate the row of the matrix, and a *True* value is putted in the corresponding location.
- **Second matrix creation** along with the first matrix a second, more complete matrix is created having a $2N \times P$ structure, in order to store the x and y coordinates of the various points. This matrix will be needed in the next section.

Figure 3 and 4 show a snapshot of the resulting matrices, where only the first 1000 points are displayed. It can be seen how most of the points are not common to all views, while a small number of them can be seen

[1] `http://www.vlfeat.org/matlab/vl_ubcmatch.html`

at any given time. As the list of matrices is analyzed from the start every time, the points with the highest number of occurrences are mostly at the start of the matrix.

Figure 3: Binary point-view matrix of the first 1000 points

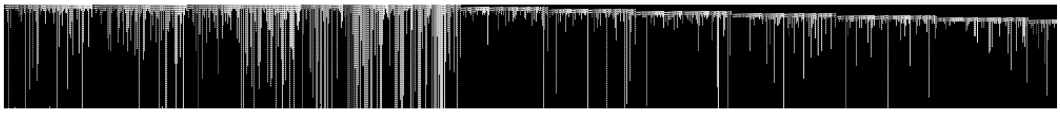
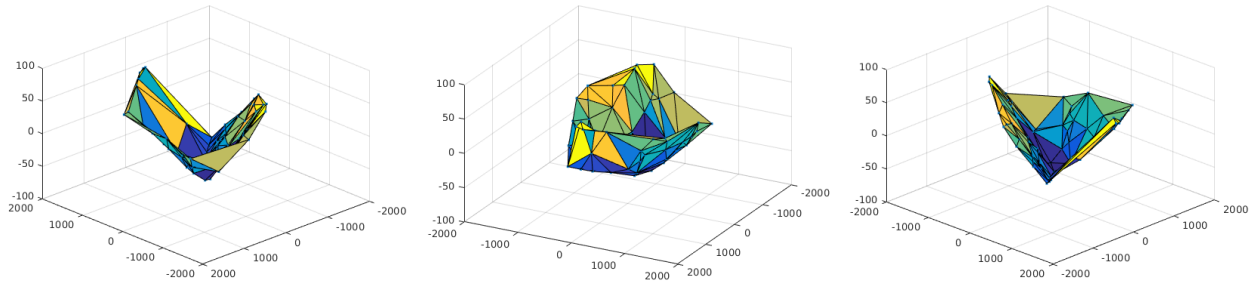


Figure 4: Complete point-view matrix of the first 1000 points

3 Structure from motion

With the data provided by the point-view matrix, it is possible to estimate a 3D reconstruction of some of the points that are common to multiple views. The points need to be seen from different angles, in order to allow an estimation of their position in the space. The structure from motion algorithm follows the guidelines from the assignment instructions so the implementation will not be discussed. For the experimentation part, the biggest dense block extractable from the computed matrix has been used, namely containing only the points common to all views and resulting in a 98×64 matrix. In order to better understand the results, a 3d surface has been created from the cloud of points, as shown in Figure 3: every triangle intersection determines the location of a point in space.



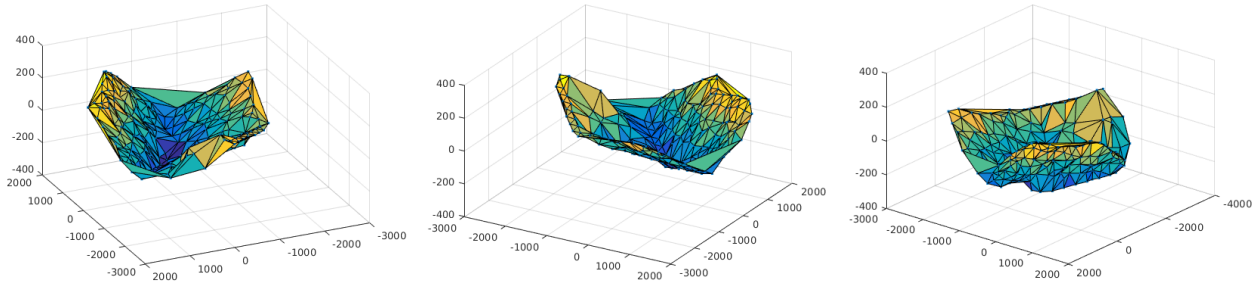
(a) (b) (c)

Figure 5: Views of the 3d surface computed on the reconstruction of the dense block.

In order to compare the results with a baseline, the plots of the 3d surface emerging from the provided matrix is also presented, in Figure 3.

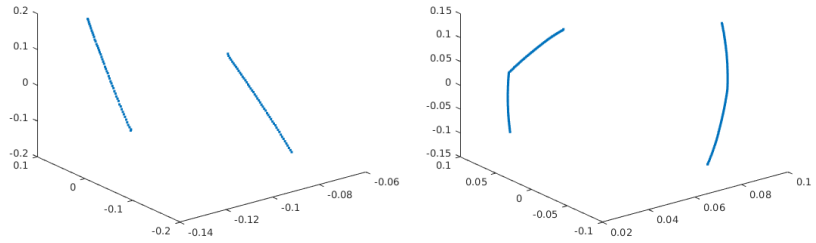
Along with the surface plot, the camera positions are also presented for both matrices in Figure 7, namely by plotting the M matrix resulting from the algorithm.

We can see that the reconstructed surface lacks a lot of details, and this is because only the frontal side of the house is recorded in this dense blocks. The experimental results seem to be in line with the expected



(a) (b) (c)

Figure 6: Views of the 3d surface computed on the reconstruction of the dense block.



(a) (b)

Figure 7: View of the camera positions extracted from the M matrix.

result of the given dense block, as the general shape and contour matches, although with less detail. In order to retrieve a complete representation of the given model a complete scanning is required, and the final result can be obtained by stitching multiple 3d surfaces using, for instance, the ICP algorithm. Looking at the camera positions plot it can be seen how the extracted dense block is created mostly from a rigid movement of the camera, while the provided one seems to originate from a more fluid movement.

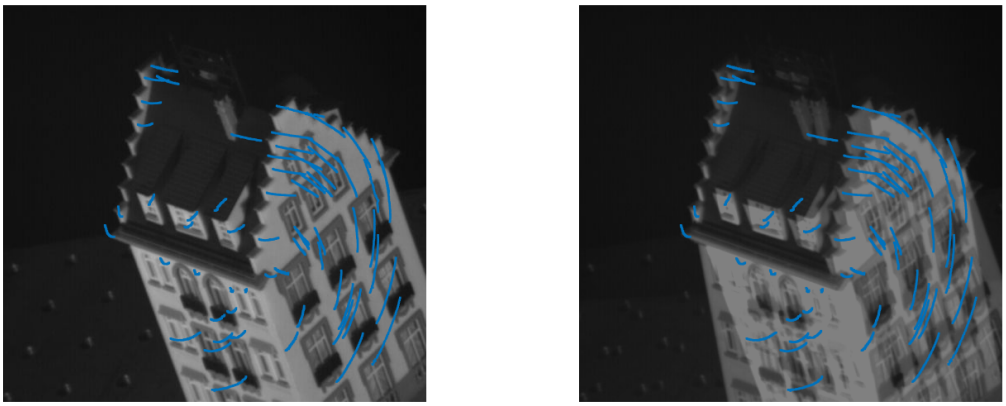
4 Tweaks and improvement

Here are listed a number of tweaks to the algorithms and other non required features that have been tested and implemented:

- **Automatic RANSAC stopping**: In order to allow the RANSAC algorithm to explore the space of the transformations in an adaptive way, an automatic iteration number has been implemented: the algorithm runs for an indefinite number of iterations and stops only when the number of inliers has not increased in the last 50 iterations, allowing more cycles when steady improvements are made and quick stopping when no better solutions are found. This optimization increased the efficiency of the project and the quality of the solutions.
- **Automatic RANSAC threshold**: Some tests have been done regarding the choice of an automatic threshold for the RANSAC algorithm, namely by determining the average Sampson distance between all the input points and scaling the threshold from that value. This implementation has not been used in this project, as no major improvements have been noticed.
- **Improved points matching**: As discussed in section 2, the descriptors matching has been done

by using the *vl_ubcmatch* function from the VLFeat library, as it is robust and optimized, reducing the workload during the initial implementation. Increasing the threshold increased the quality of the results, giving only good matches as an output.

- **Analysis of the dense block**: In order to better understand the execution of the project, extra work has been done towards plotting the movements of different points in the image. Figure 8 shows the change in position of each match.



(a) 1st image

(b) 1st and 20th images

Figure 8: Movement of the matches in various views. The Right image overlays the first and 20th image, in order to better understand the motion.

It can be seen that the computed point-view matrix is indeed correct, and the dense block of points lies only in the model, discarding matches in the background.

5 Conclusions

This project illustrates how SIFT descriptors, coupled with ad hoc algorithms, are able to give a concrete framework that can be used to reconstruct 3D points, given multiple views of the same object. As it is not possible to have a 360° view of an object in a single point in time, stitching of multiple dense block reconstructions (for example by using ICP) can be done, resulting in a full virtual 3D object. The final precision of the 3D reconstruction will be bounded by the precision, quality and quantity of the extracted points, but a relatively small number of them (~ 90 in this case) can already give good results.