Computer Vision

Assignment 7: Structure from Motion

Student: Ivan Alberico

The first step in this assignment is about computing the epipolar geometry between two images. To do so, I consider the first and the last images provided in the code framework, since they have a baseline which is large enough to have a good triangulation of the feature points.

Then, I extract the SIFT features in both images (with VLFeat) and I match them. After the matches have been computed, I run the 8-point RANSAC algorithm (by means of the function provided in the code framework) to filter out the outliers from this set of matches. I then compute the essential matrix ($E = K'F\,K$), and I decompose it to obtain the projection matrices. The matched points, after being calibrated (by pre-multiplying the points with K^{-1}), are then triangulated in order to obtain the corresponding 3D points, by means of the function `linearTriangulation.m` already provided.

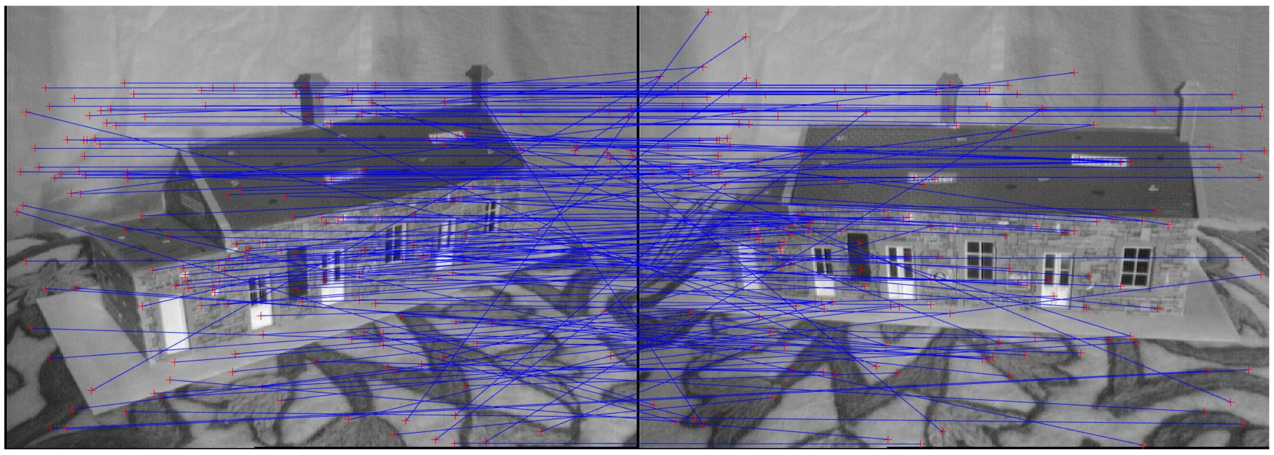


Figure 1 — All feature matches obtained with the VLFeat toolbox (images 0 and 4)

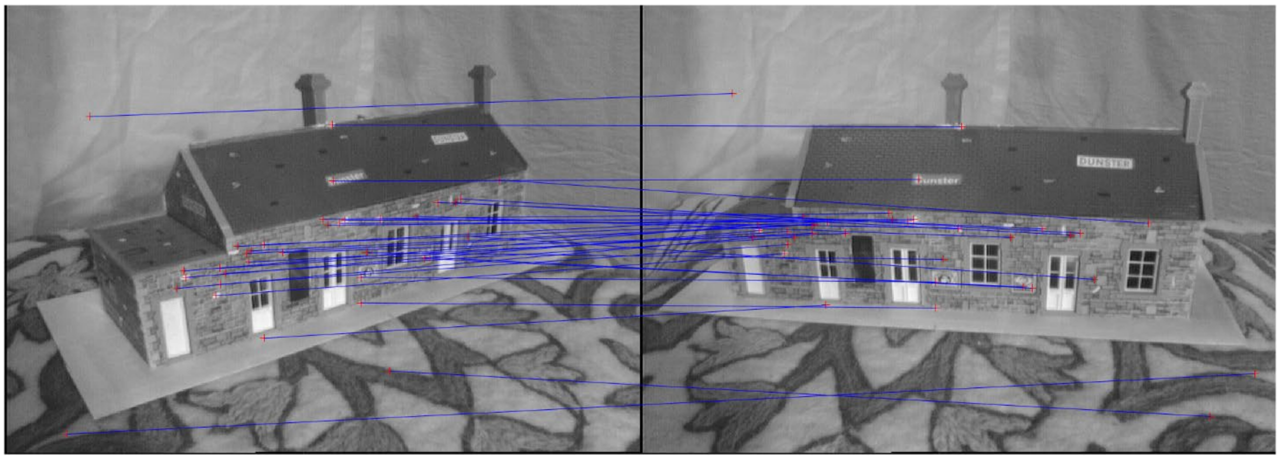


Figure 2 — Inlier matches obtained after 8-point RANSAC (images 0 and 4)

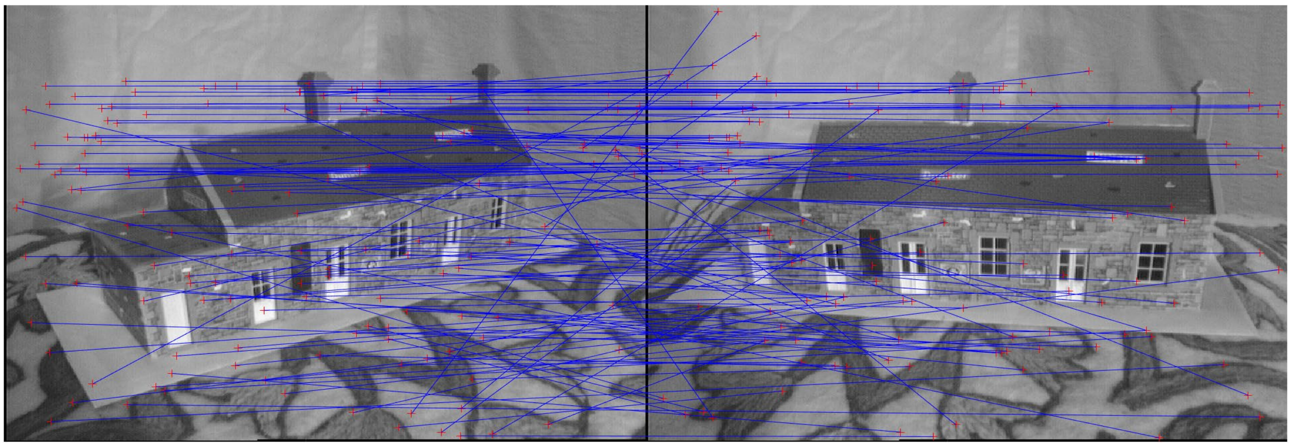


Figure 3 — Outlier matches obtained after 8-point RANSAC (images 0 and 4)

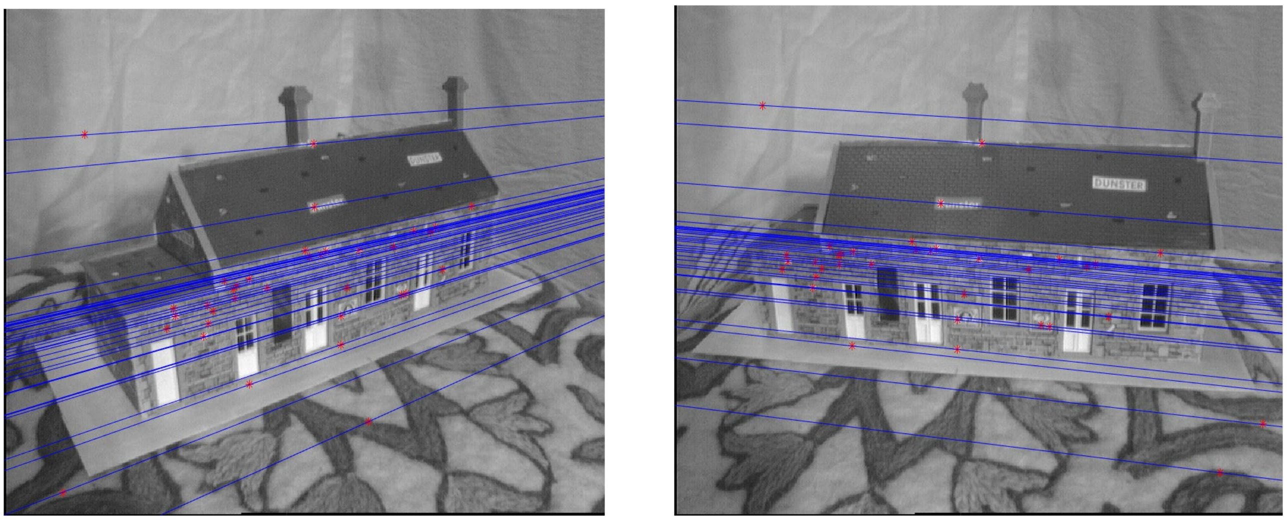


Figure 4 — Epipolar lines on image 0 and image 4.

In order to add more views, we need to match the 2D features of a new image with the features of an existing view (the first image), which had been already triangulated. After that, I run the 6-point RANSAC algorithm and triangulate the calibrated points.

However, before triangulating, a check on the projection matrix should be performed: it is important to make sure that the determinant of the rotation matrix R is positive. In case its determinant is negative, I should flip the sign of both R and t in the projection matrix.

For the assignment I added 3 more views, as it was the number of additional images provided in the code framework (the views of image 1, image 2 and image 3).

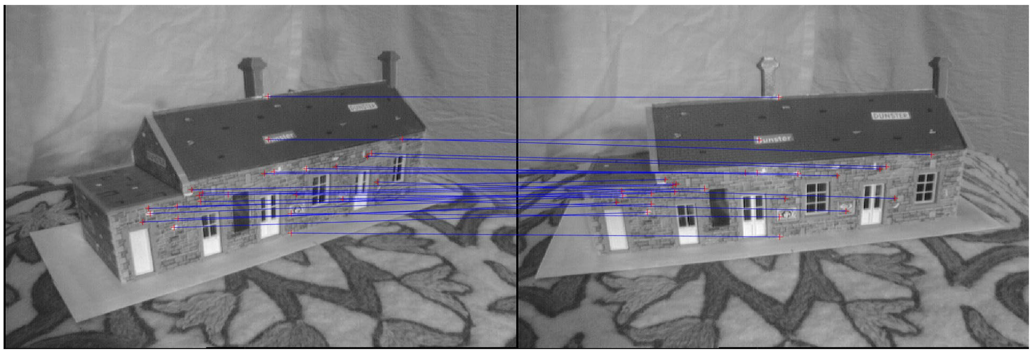


Figure 5 — Inlier matches obtained after 6-point RANSAC (images 0 and 1)

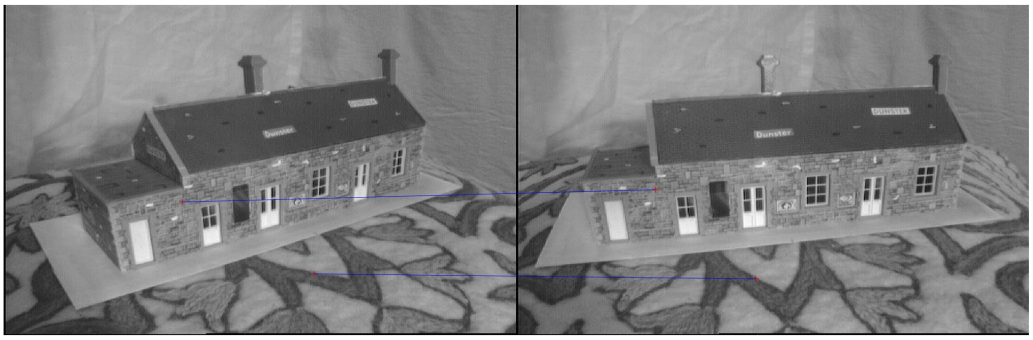


Figure 6 — Outlier matches obtained after 6-point RANSAC (images 0 and 1)

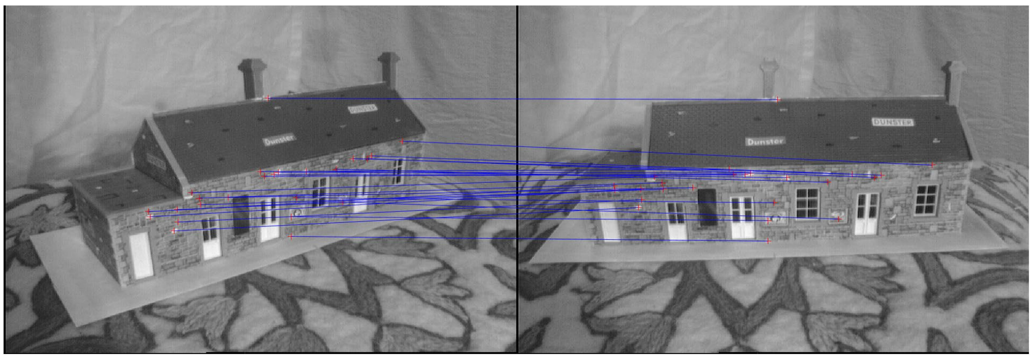


Figure 7 — Inlier matches obtained after 6-point RANSAC (images 0 and 2)

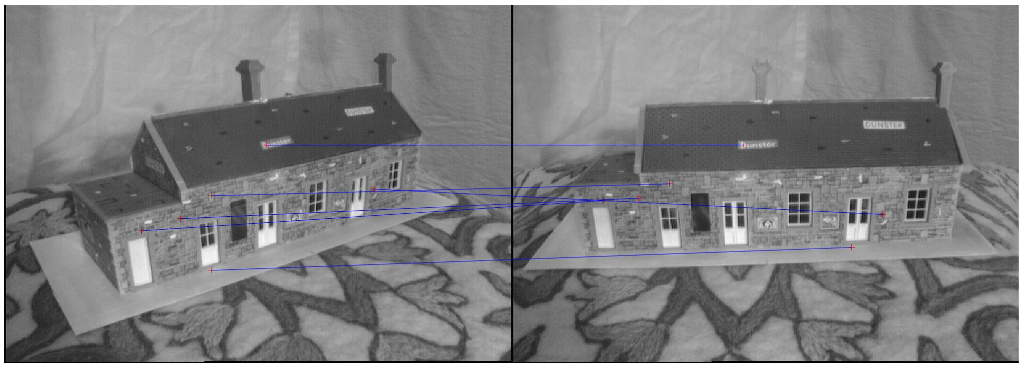


Figure 8 — Outlier matches obtained after 6-point RANSAC (images 0 and 2)

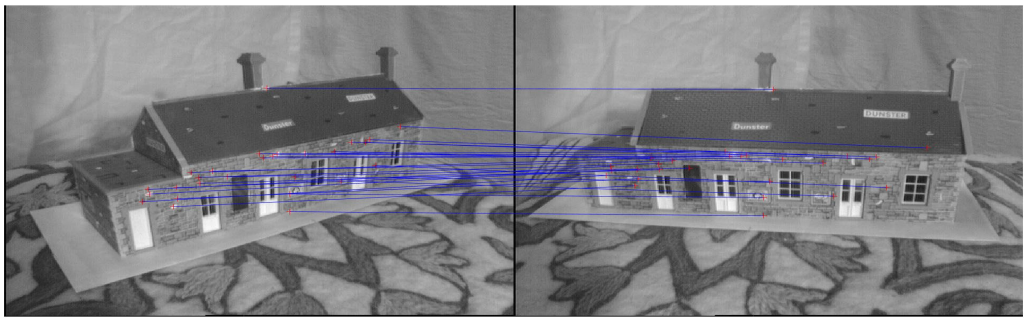


Figure 9 — Inlier matches obtained after 6-point RANSAC (images 0 and 3)

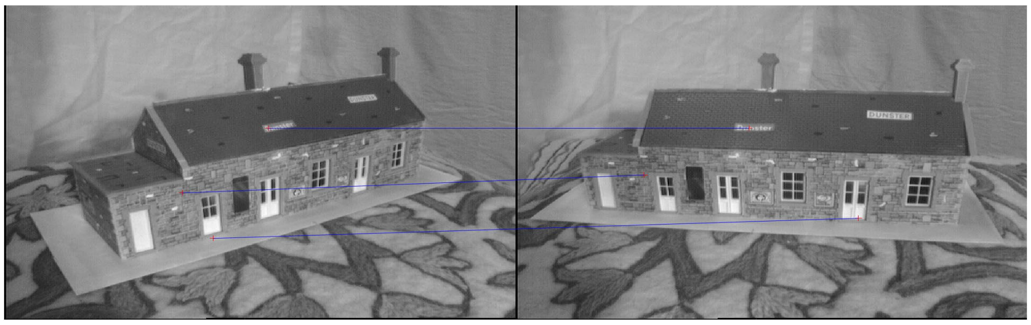
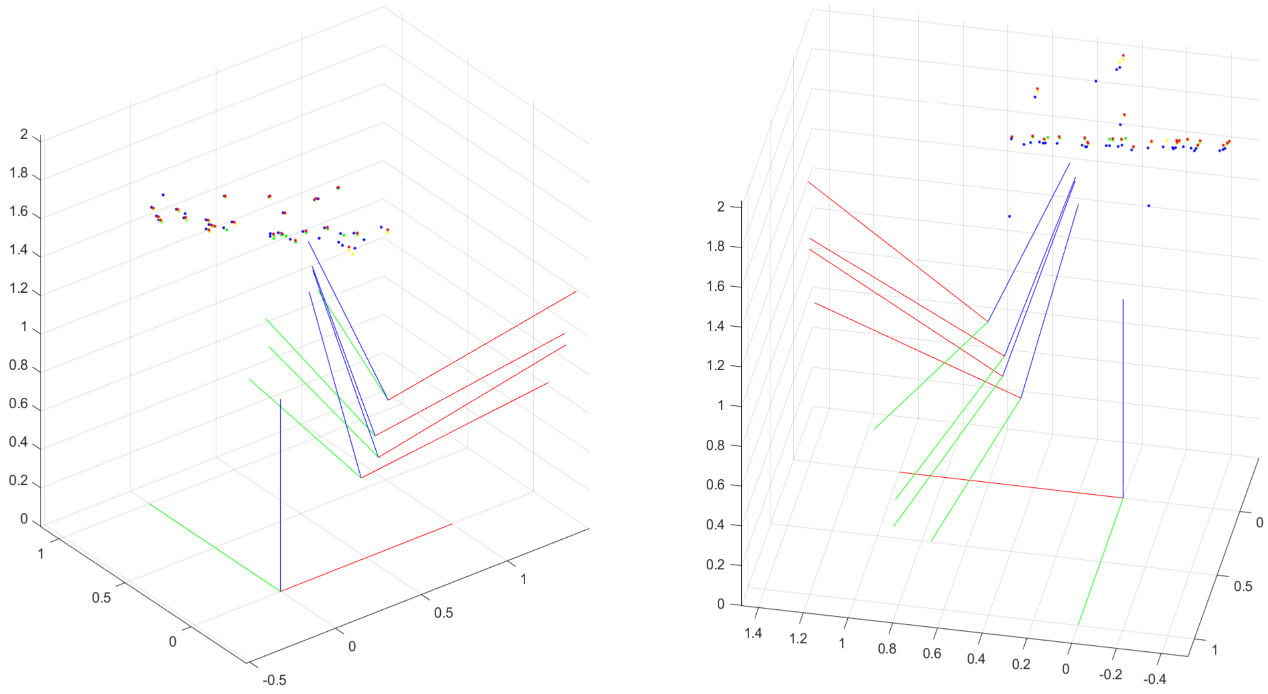


Figure 10 — Outlier matches obtained after 6-point RANSAC (images 0 and 3)

Very important is also the choice of proper values for the threshold in both the 8-point and 6-point RANSAC algorithm, in order to properly filter out the outliers. A too high value for the threshold would take in more outliers, leading to a worse estimation of the camera poses and to an increasing number of sparse points in our 3D reconstruction. Moreover, since RANSAC is a random algorithm, it is very likely that running the code several times will bring different results.

The final result shows the camera poses for every view as well as all the triangulated 3D points. Different colours are used to distinguish the triangulated points among the different views: blue refers to the points from the initialization, red refers to the points of the first additional image, yellow to those of the second additional image and green to those of the third additional image.



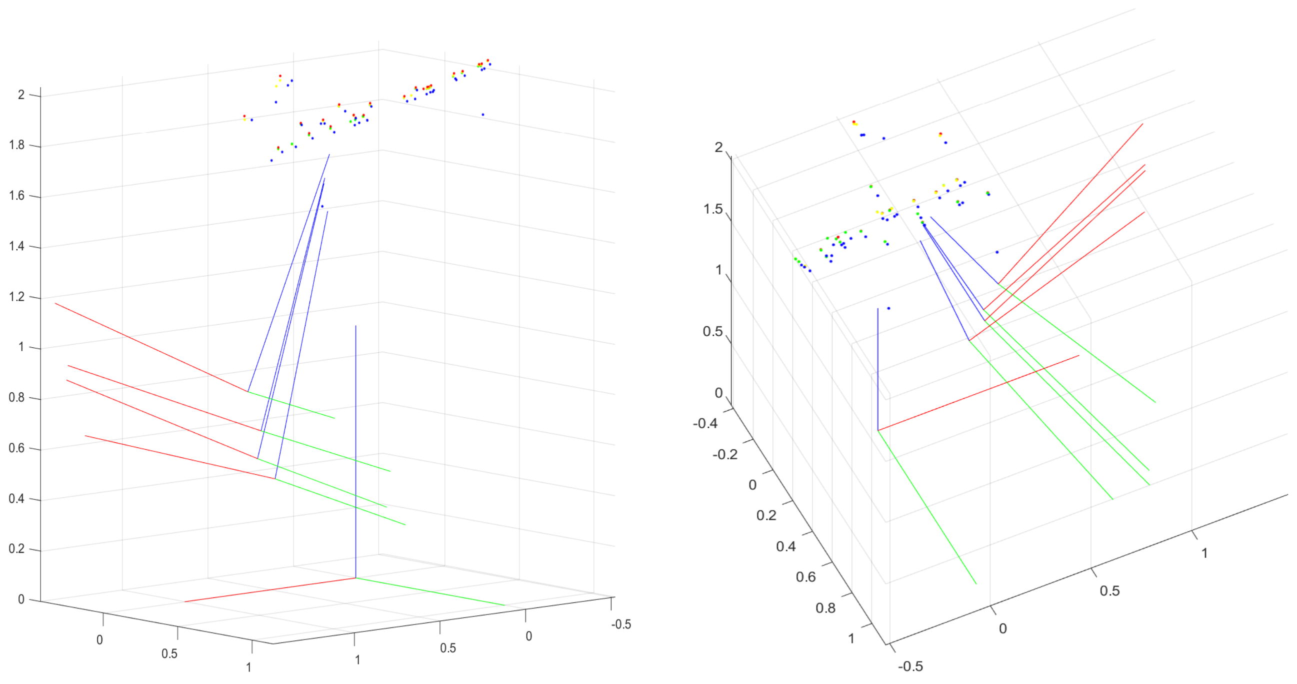
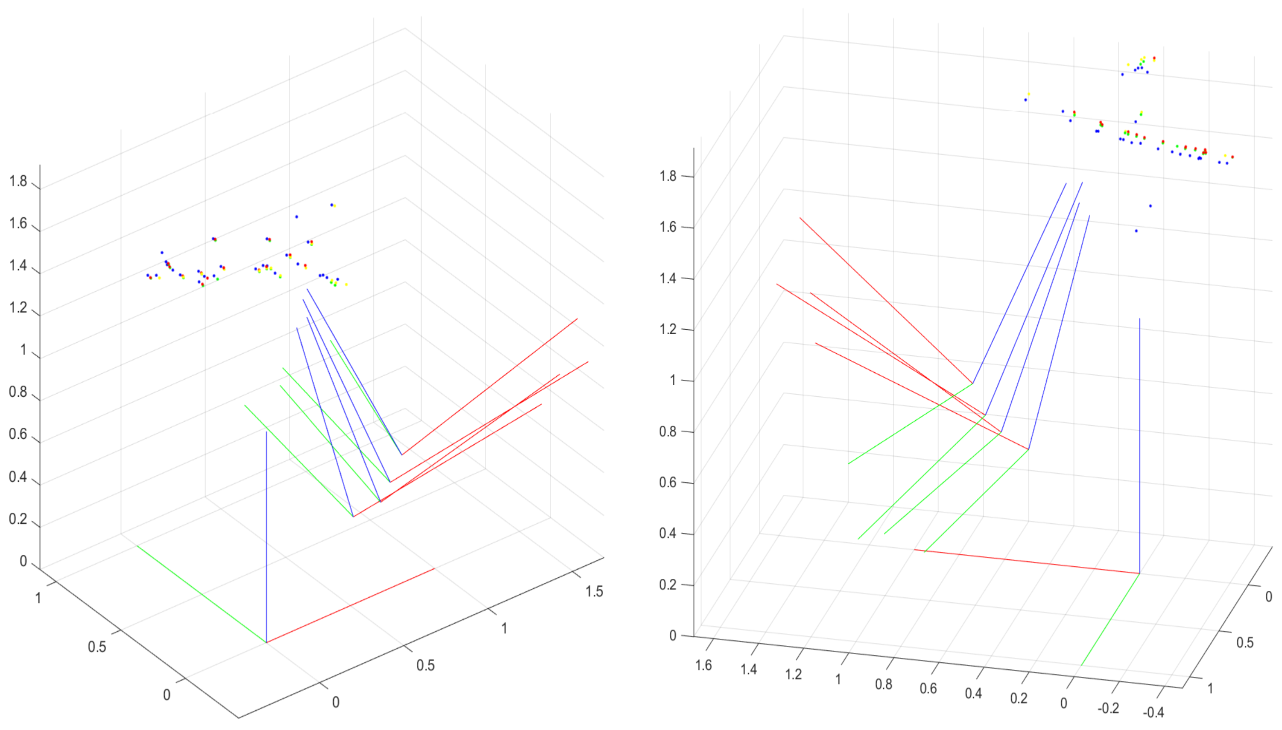


Figure 11 — Different viewpoints of the triangulated points and camera poses plotted in 3D. In this case, the threshold value of the 8-point RANSAC algorithm is set to 0.00001, while that of the 6-point RANSAC is set to 0.01.



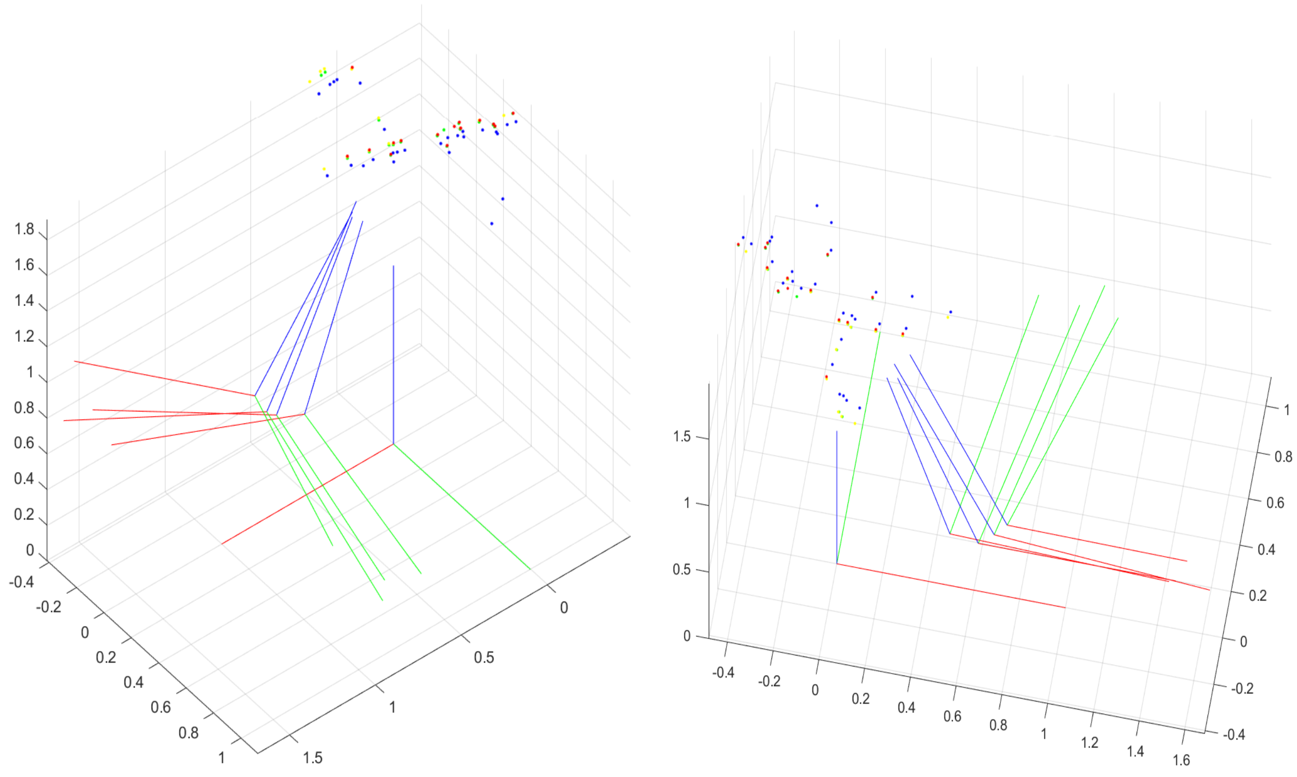


Figure 12 — Different viewpoints of the triangulated points and camera poses plotted in 3D. In this case, the threshold value of the 8-point RANSAC algorithm is set to 0.00001, while that of the 6-point RANSAC is set to 0.001.

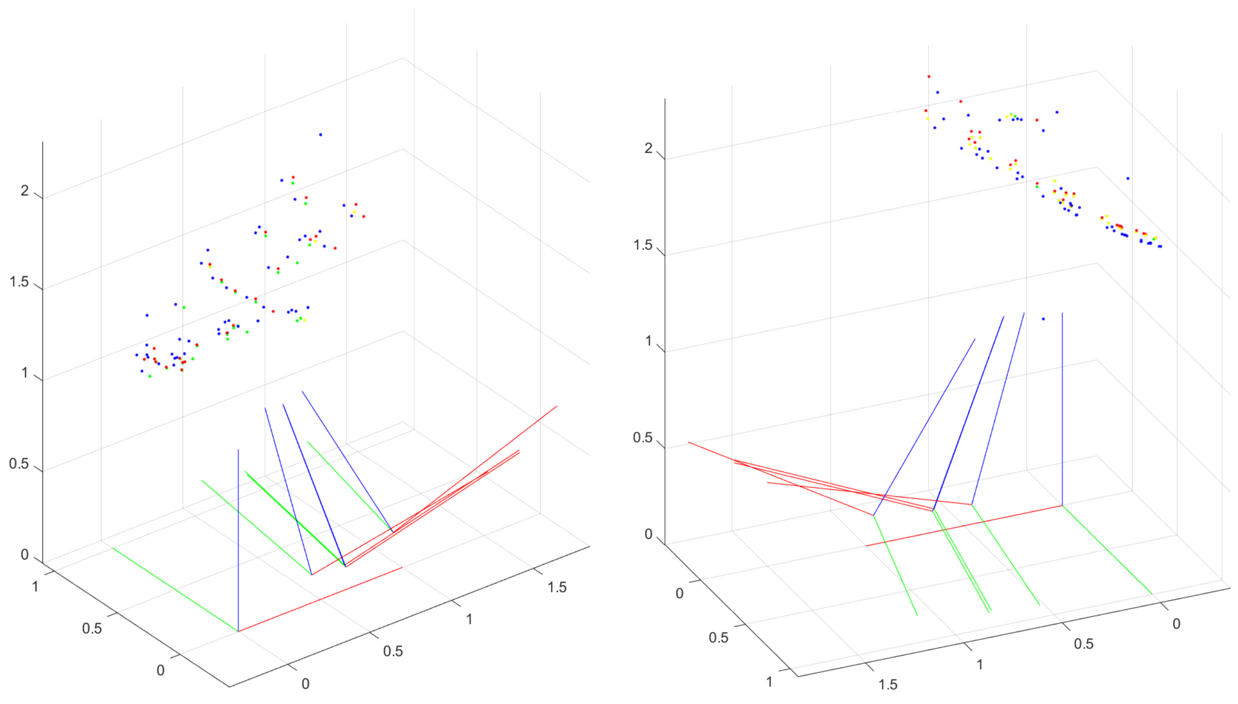


Figure 13 — Different viewpoints of the triangulated points and camera poses plotted in 3D. In this case, the threshold value of the 8-point RANSAC algorithm is set to 0.00007, while that of the 6-point RANSAC is set to 0.01.

We can see that by taking higher values of the threshold for the RANSAC algorithm, we have more triangulated points in the 3D plot, however the overall result seems to become slightly less accurate, with more sparse points.

In general, the estimated camera poses seem to follow a trajectory which is consistent and somehow similar to what I expected by only looking at the different images. Moreover, the triangulated points obtained with respect to the different views are almost overlapped with each other, following similar patterns and lines, and this is a positive outcome.

On the other hand, results could be further improved by performing bundle adjustment, which optimizes the positions of the features and of the cameras.

• Dense reconstruction

In order to achieve a dense reconstruction, I first perform the stereo matching between two views, in particular between image 0 and image 1, using the functions I implemented in Assignment 6. Firstly, I rectify the two images and then I apply the Graph-cut algorithm in order to compute the disparity map of this stereo problem. A textured 3D model is then generated and visualized on MeshLab.

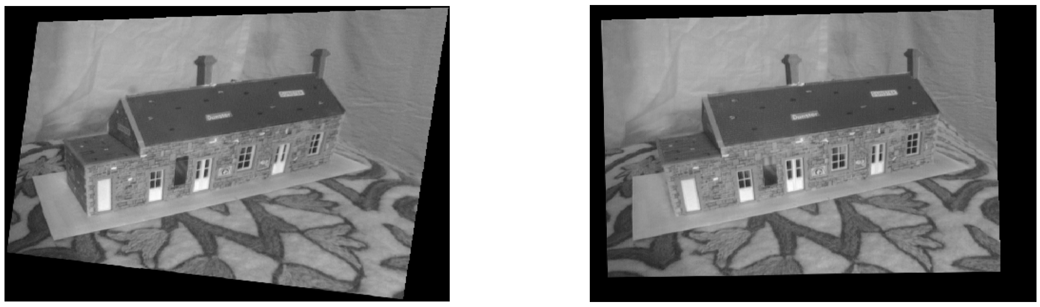


Figure 14 — Rectified images for stereo matching (respectively image 0 and 1)

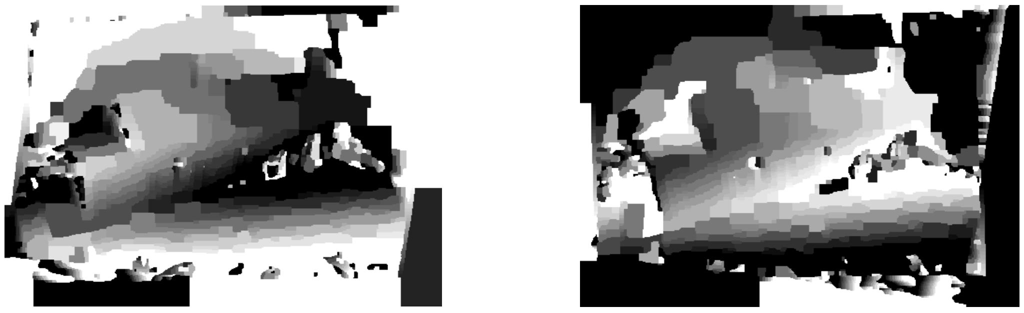


Figure 15 — Disparity map computed with the Graph-cut algorithm

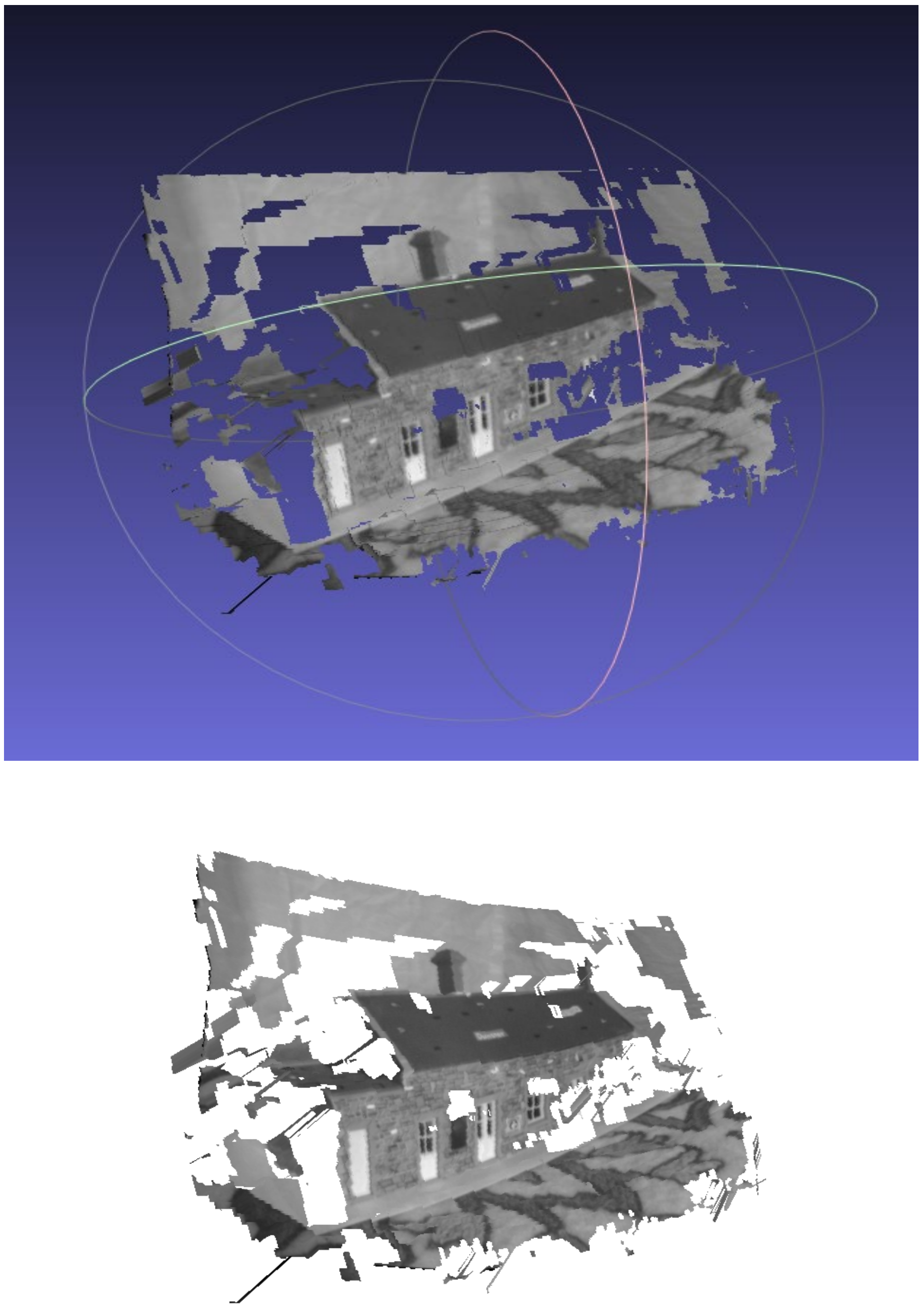


Figure 16 — Textured 3D model obtained with the Graph-cut algorithm and visualized on MeshLab

The depth map is computed from the disparity map, the length of the baseline (which is nothing but the difference of the camera positions of the two views) and the focal length (values in the diagonal of the calibration matrix), according to the following relation:

$$depth = \frac{baseline\ \times focal\ length}{disparity} \quad \rightarrow \quad z = \frac{b\,f}{d} = \frac{b\,f}{(x'_L - x'_R)}$$

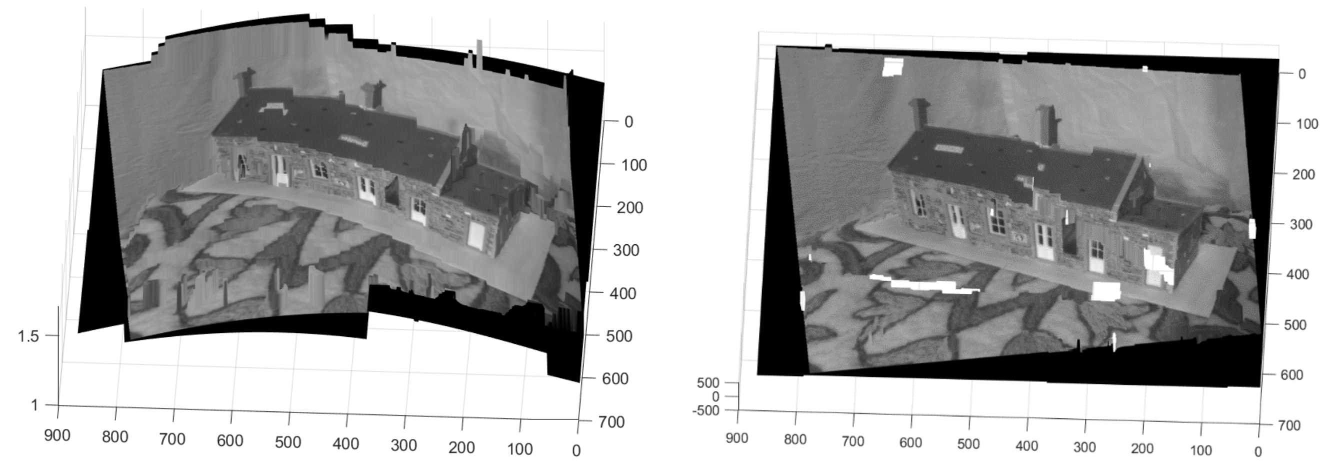


Figure 17 — Dense reconstruction of the rectified images using the function `create3DModel.m`. The depth map is computed from the disparity map previously obtained with the Graph-cut algorithm.

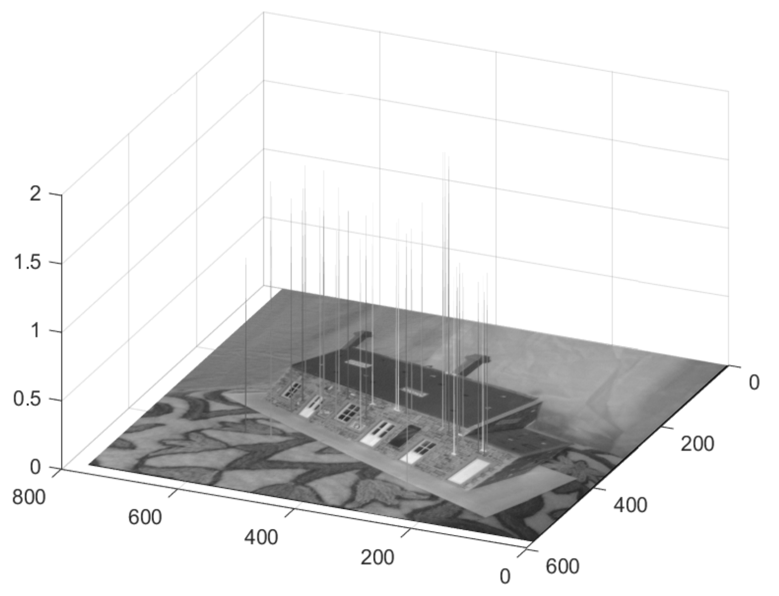


Figure 18 — Depth map of the triangulated 3D points (obtained from the previous steps), directly computed from the 3rd coordinate of these 3D points. The height of each peak represents the depth of the corresponding feature.