Computer Vision

Assignment 4: Model fitting

Student: Ivan Alberico

1. Line fitting

The first part of the assignment is about fitting a line through the RANSAC algorithm. The dataset of points used for this first task is randomly generated by the function `genRansacTestPoints()`. At each iteration, two points are randomly picked from the dataset. Then, we calculate the coefficients of the line passing through these two points with the function polyfit() and after that, we compute the distance of all points from this line. For each point, if the distance is below a certain specified threshold, then the point can be considered as an inlier. For each iteration we count the number of inliers we get from the fitting, and in the end, the best solution is the one that counts the highest number of inliers. From the best solution, we compute the best fitting line of the specific set of inliers, and we store the coefficients of this line.

The results of this fitting through the RANSAC algorithm (red line in Figure 1) are compared with the least square fitting (green line in Figure 1) and the real model of the line (black line in Figure 1). For each of the 3 cases, we calculate the squared error based on the true inliers. By comparing the results obtained, we can see that the RANSAC line fitting is a very good approximation of the real line, respectively having $error_{RANSAC} = 40.8345$ and $error_{real} = 40.1634$. On the contrary, least square fitting poorly approximates the real line, showing a very high error, $error_{least-square} = 198.4748$.

For the testing, I used the default values `iter = 300` and `thresh_dist = 3`. However, since the whole process is randomized, to check whether the algorithm is actually implemented well, it is better to make more than one simulation. Averaged results obtained on 10 different simulations are the following: $error_{RANSAC} = 42.8206$, $error_{real} = 40.2145$ and $error_{least-square} = 143.3682$.

By increasing the number of iteration for the RANSAC algorithm, it is more likely to find a solution that approximates the real line even better.

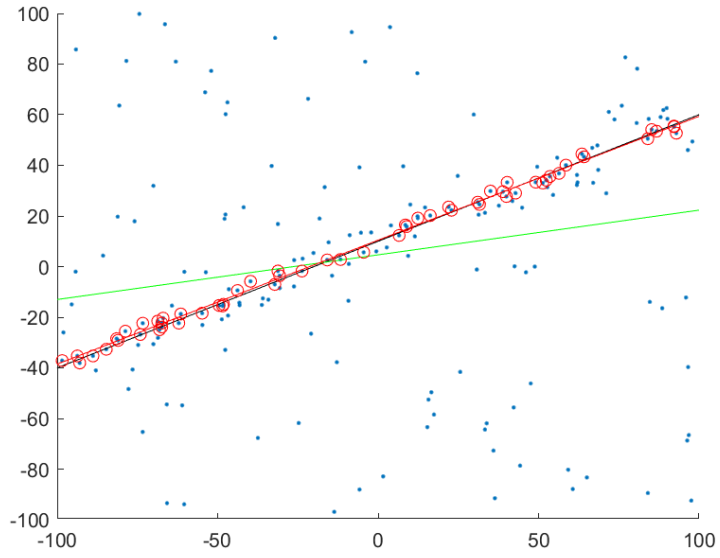

Figure 1 — Comparison between the RANSAC fitting line (in red), the least square fitting line (in green) and the real line (in black)

2. Fundamental matrix estimation

In the second part, we want estimate the fundamental matrix F for each image pair with the eight-point algorithm. To implement this algorithm we need at least 8 points' correspondences between the two images (the more correspondences, the better is the estimation). After the points have been normalized (this is essential for numerical stability, in order to avoid having different orders of magnitude among the elements of the matrix), we compute the homogeneous least-square solution of the following:

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

$$(xx', xy', x, yx', yy', y, x', y', 1)\, vec(F) = 0$$

It is important that the normalization is removed afterwards.

In general, when drawing the epipolar lines with the non-singular fundamental matrix F, I noticed that, in all the three cases, all the lines were perfectly passing through the selected points correspondences, but not intersecting in a unique point. On the other hand, by enforcing the singularity constraint on matrix F, we are forcing all the epipolar lines to intersect at the epipole. However, in this second case the epipolar lines are no longer passing exactly through the selected points, but they are slightly shifted.

At first glance, by looking at the values of F and $\hat{F}$, it seemed that in all the three cases, the matrices were numerically equivalent. However, by showing also the least significant digits it was possible to spot slight differences between them in the order of $10^{-6} - 10^{-9}$ and below. (this was done by typing "format long" on Matlab, which shows all the numerical values in the *long* format). The points correspondences which were used to test the results were saved in .mat files and uploaded in the zip file.

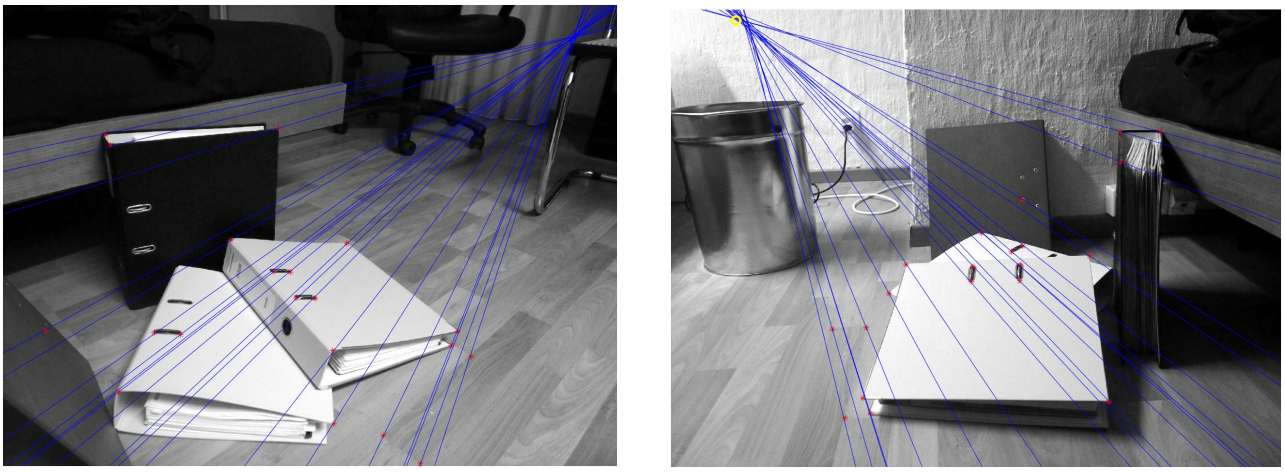

Figure 2 — Epipolar lines of the "rect" image pair considering the non-singular fundamental matrix.

$$\mathrm{F_{rect}} = \begin{pmatrix} 0.0000000577 & 0.0000016428 & 0.0000904334 \\ 0.0000016229 & 0.0000000722 & 0.0003505537 \\ 0.0000157642 & 0.0030402825 & 0.1160532887 \end{pmatrix}$$

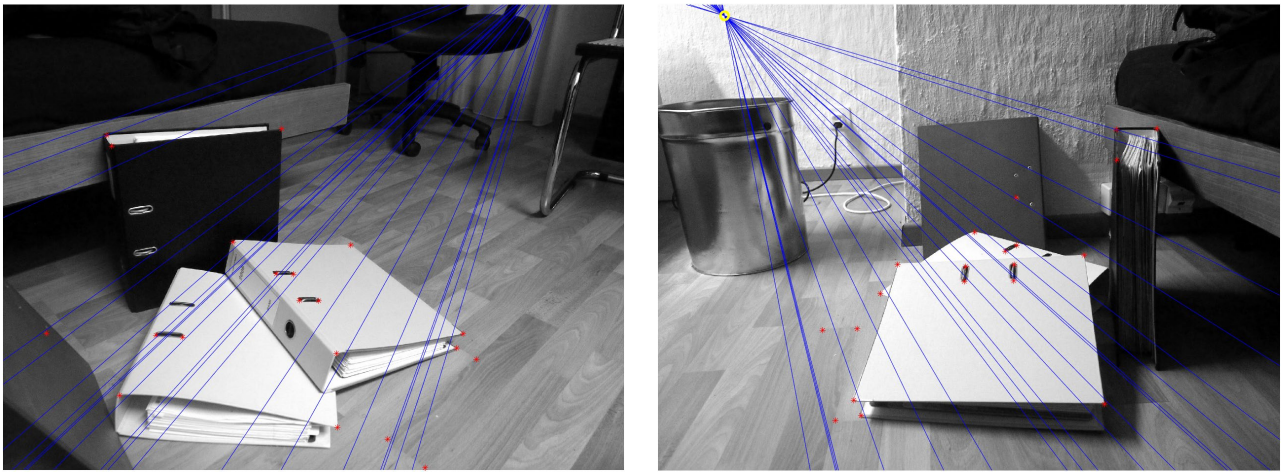

Figure 3 — Epipolar lines of the "rect" image pair considering the singular fundamental matrix.

$$\hat{\mathrm{F}}_{\mathrm{rect}} = \begin{pmatrix} 0.0000001311 & 0.0000016751 & 0.0000904325 \\ 0.0000016086 & 0.0000000697 & 0.0003505538 \\ 0.0000157641 & 0.0030402825 & 0.1160532887 \end{pmatrix}$$

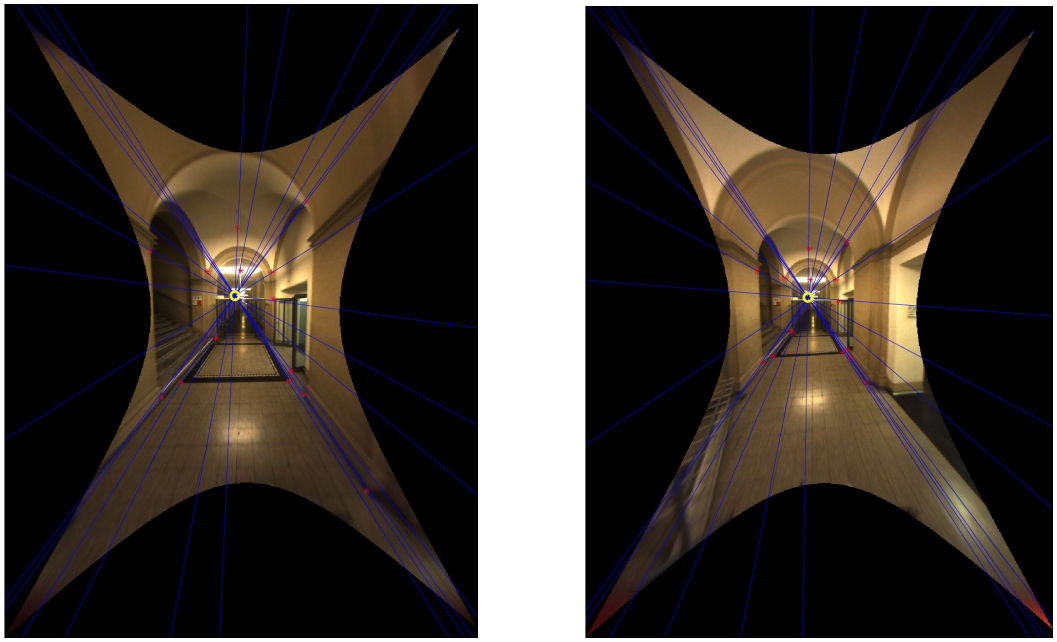

Figure 4 — Epipolar lines of the "ladybug" image pair considering the non-singular fundamental matrix.

$$\mathrm{F}_{\mathrm{ladybug}} = \begin{pmatrix} 0.0000019143 & 0.0000849111 & 0.0351433267 \\ 0.0000836449 & 0.0000008333 & 0.0392012953 \\ 0.0321151886 & 0.0403857996 & 1.1021414067 \end{pmatrix}$$

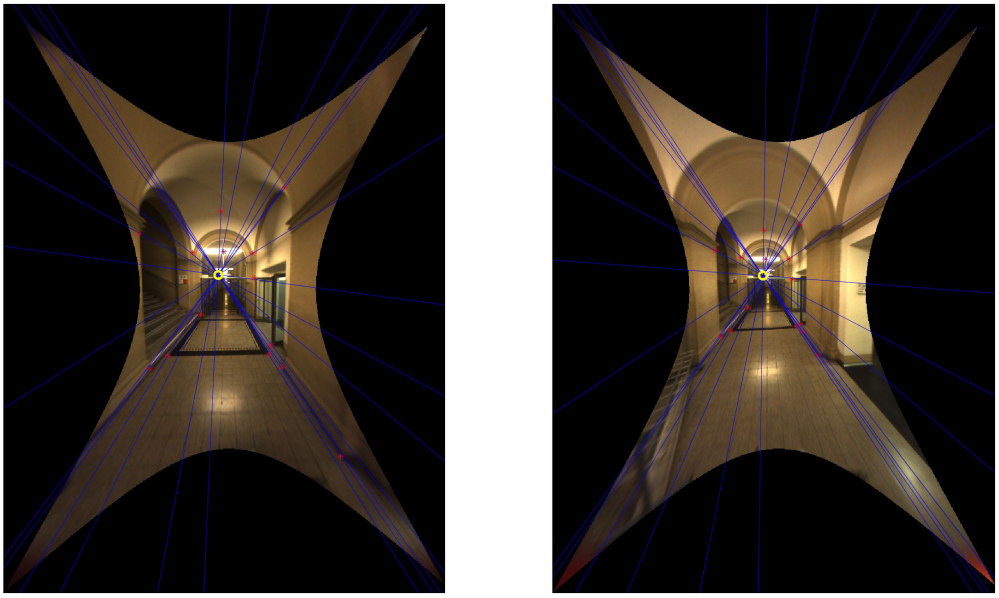

Figure 5 — Epipolar lines of the "ladybug" image pair considering the singular fundamental matrix.

$$\hat{F}_{\text{ladybug}} = \begin{pmatrix} 0.0000019177 & 0.0000849139 & 0.0351433267 \\ 0.0000836417 & 0.0000008357 & 0.0351433267 \\ 0.0321151886 & 0.0403857996 & 1.1021414067 \end{pmatrix}$$

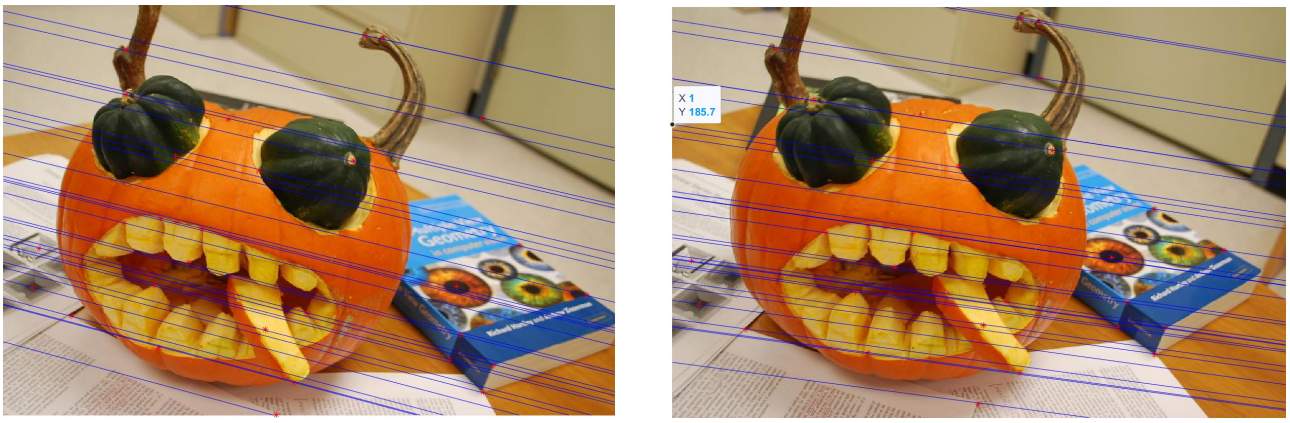

Figure 6 — Epipolar lines of the "pumpkin" image pair considering the non-singular fundamental matrix.

$$F_{\text{pumpkin}} = \begin{pmatrix} 0.0000000936 & 0.0000003076 & 0.0008901782 \\ 0.0000001618 & 0.0000001765 & 0.0040202349 \\ 0.0005998342 & 0.0037768604 & 0.0385361319 \end{pmatrix}$$

Figure 7 — Epipolar lines of the "pumpkin" image pair considering the singular fundamental matrix.

$$\hat{F}_{\text{pumpkin}} = \begin{pmatrix} 0.0000000912 & 0.0000003080 & 0.0008901782 \\ 0.0000001624 & 0.0000001766 & 0.0040202349 \\ 0.0005998342 & 0.0037768604 & 0.0385361319 \end{pmatrix}$$

3. Feature extraction and matching

By using VLFeat toolbox, I was able to extract and match the SIFT features for each image pair. However, in all the three cases, the matchings obtained were not always correct, due to the presence several outliers.

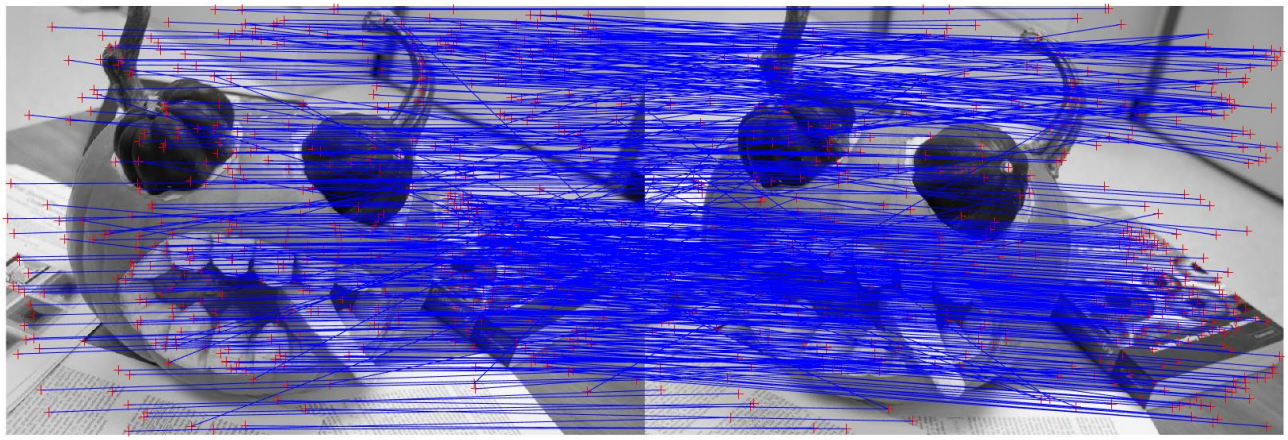

Figure 8 — SIFT matching features of the "pumpkin" image pair obtained with the VLFeat toolbox.

4. Eight-point RANSAC

In order to filter out the outliers from the SIFT features previously extracted, 8-point RANSAC algorithm was implemented. The inliers and outliers were chosen thresholding on the following distance measure:

$$\varepsilon(x.x') = \frac{d(x', Fx) + d(x, F^T x')}{2}$$

Different tests were performed by varying the error threshold. I will now show the different results obtained from the pumpkin image pair and I will compare the percentage of inliers and their mean error at different values of error threshold. As stated in part 1, since the RANSAC algorithm is somehow randomized, all the results that will be presented are averaged over 10 simulations (with the same parameters).

By setting the threshold to 5, I obtained an average percentage of inliers (on the total number of SIFT features) of 65.24% and a mean error of inliers of 1.161. For lower values of the error threshold, as expected, lower percentages of inliers were obtained, as well as lower mean error of inliers (it is somehow reasonable that we obtain a lower value of the mean error, since the number of inliers is lower and so less is the probability to have a wrong match of features). For a threshold value of 2, on average I had 55.58% of inliers with 0.6994 mean error, while for a threshold of 1, 45.99% of inliers and 0.3776 of mean error. On the other side, by increasing the threshold, a reverse trend was observed. 67.85% of inliers with a mean error of 1.302 was recorded for a threshold value of 7.

Overall, from the results above, we can say that there seems to be a linear dependency of the threshold value with the number of inliers obtained.

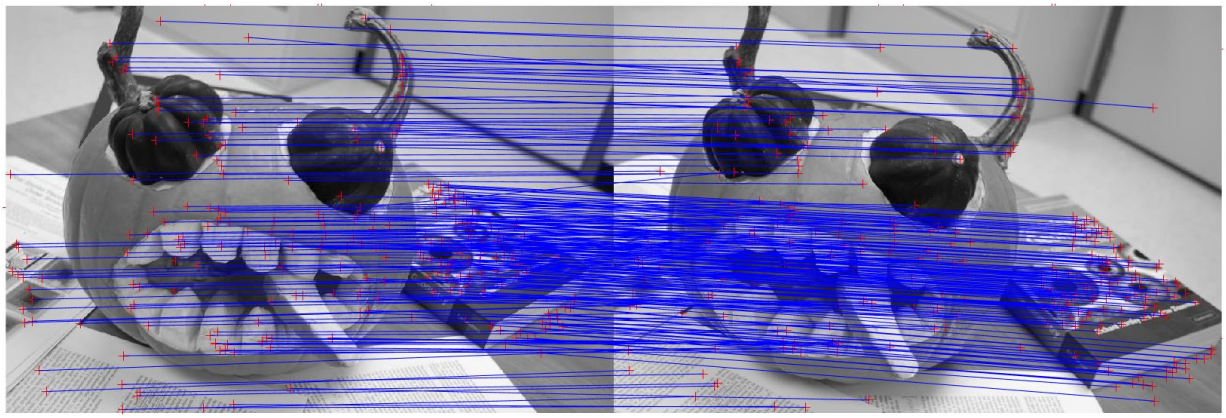

Figure 9 — 8-point RANSAC with "pumpkin" image pair with threshold = 2. Results obtained are a percentage of inliers = 56.03% and a mean error of inliers of 0.5971.

Figure 10 — 8-point RANSAC with "rect" image pair with threshold = 1. Results obtained are a percentage of inliers = 16.87% and a mean error of inliers of 29.73.

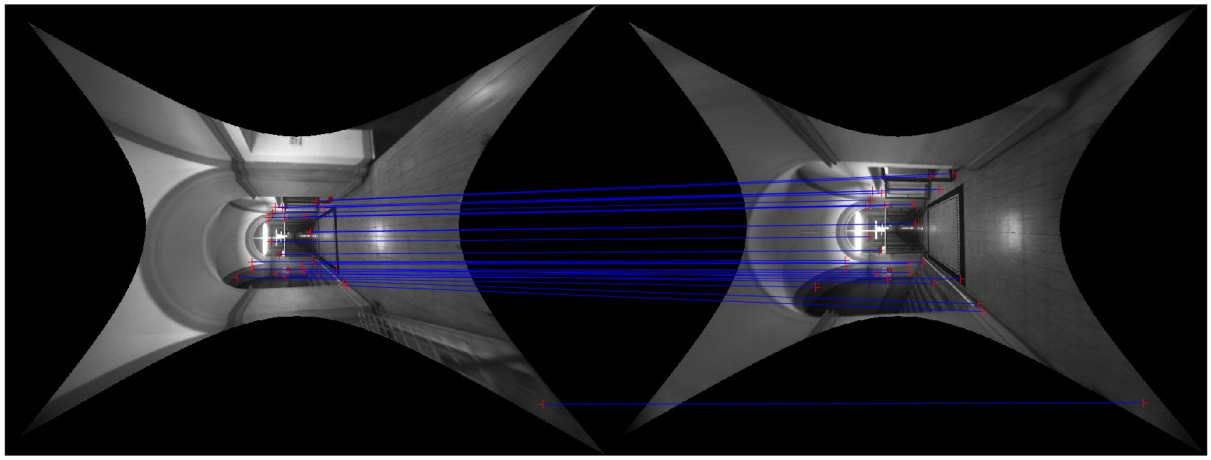

Figure 11 — 8-point RANSAC with "pumpkin" image pair with threshold = 1. Results obtained are a percentage of inliers = 28.71% and a mean error of inliers of 8.51.

From the images, it can be observed that RANSAC is pretty efficient. In fact, RANSAC seems to correctly select the good matches (inliers) between each pair of images.

4.2. Adaptive RANSAC

RANSAC can be further improved in terms of computational cost, by early stopping the algorithm once that an optimal set of inliers is found. Adaptive RANSAC, in fact, terminates the whole process after M trials, when the following equation is satisfied:

$$p = 1 - (1 - r^N)^M$$

where N is the number of samples drawn, r is the inlier ratio and p is the probability that at least one of the random samples is free from outliers.

It was observed that the number of iteration M at which the algorithm stops depends on the error threshold, the probability p, but also on the set of images chosen. For examples, with the pumpkin image pair, by setting $p = 0.99$, different values of M were obtained at different error threshold (for each threshold value, I ran the algorithm 10 times and averaged over these 10 different simulations). On average, for $T = 5$, the algorithm stops at the 476th iteration, for $T = 2$ at the 2337th, for $T = 1$ at the 4932th and for $T = 7$ at the 160th (the maximum number of iterations was set to 20.000). Hence, it was observed that as the error threshold decreased, a higher number of iterations was necessary to find a subset of inliers that satisfies the previous equation.