Computer Vision 1 - Assignment 4

Ruben-Laurentiu Grab
ruben.grab@student.uva.nl 11609923

Andrei Marius Sili
andrei.sili@student.uva.nl 11659416

Introduction

In this assignment we experiment with the SIFT method for detecting and matching key points between images. RANSAC is used for bootstrapping a robust set of matching points afterwards. We use these methods to perform image alignment and image stitching.

1 Image Alignment

Question 1

1.1 The procedure follows the one outlined in the assignment instructions. We use *vl_sift* to extract feature points and descriptors around those points. We then match the feature points between the 2 images based on their descriptors.

1.2 We can visualize the correspondence between the key points by plotting the two images and linking the matching points. Figure 1 shows a random sample of 50 key points matched between the two images.

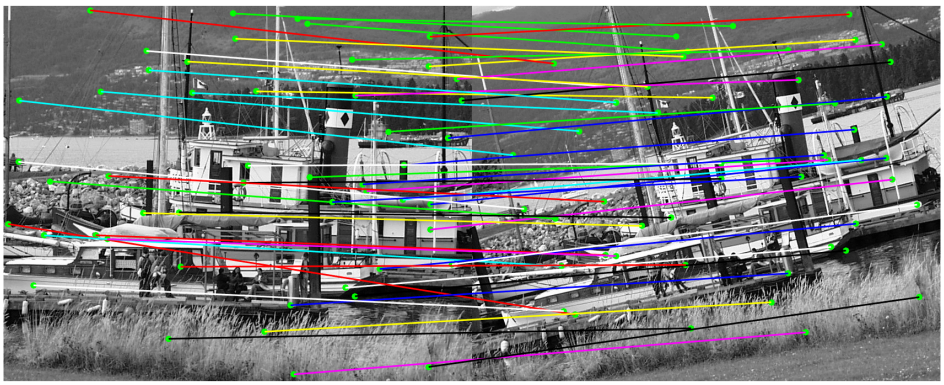


Figure 1: **Keypoint Matching.** Random sample of 50 matched keypoints between images *boat1.pgm* and *boat2.pgm*. Connecting lines are plotted with random colour for better visualization. Even for a random sample, almost all keypoints are well matched in between the images, with very little error. This is likely due to the high similarity between the two images, as one is almost just a rotated version of the other. However, some outliers are present, and we will use RANSAC to weed those out for further applications.

1.3 To find the best transformation between the 2 images, we use random sample consensus. We use a sample size of 3 and we calculate the number of trials depending on the confidence level that

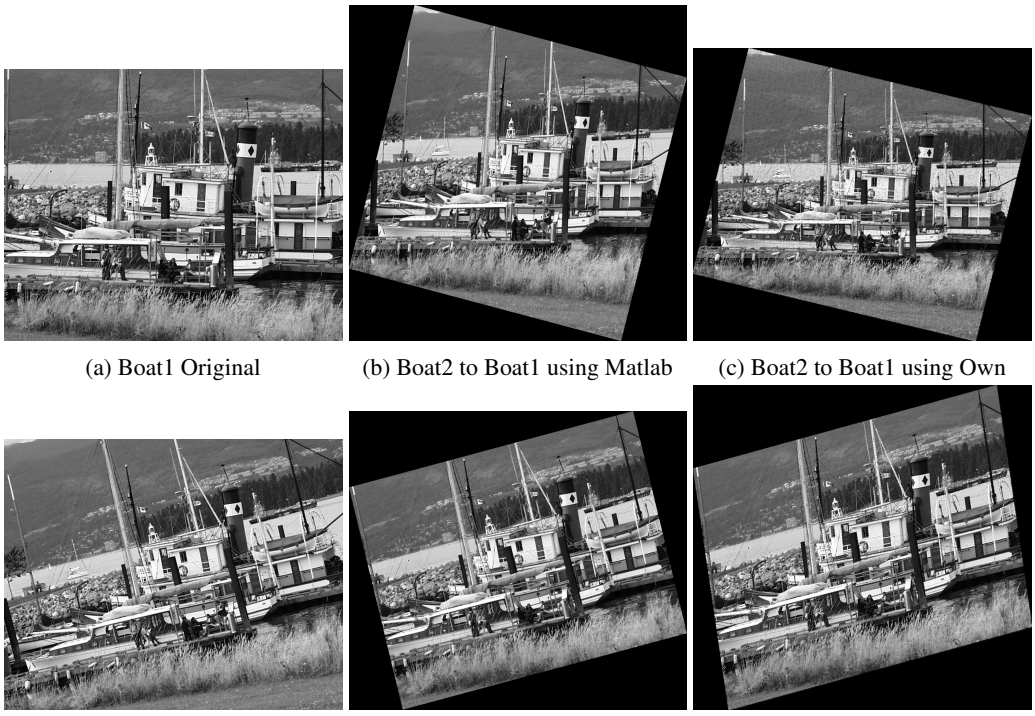
29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

we want to obtain. Figure 2 shows tuples formed of the original image, the transformed image using MATLAB functionality with 'bilinear' interpolation, and the transformed image using our implementation with 'nearest-neighbour' interpolation.

Comparing the two implementations, it can be seen that the MATLAB transformation achieves a smoother image, while our implementation suffers from a sort of salt and pepper noise. This noise is generated because there are no pixels mapped from the the source image to the destination image. This is likely to happen since we are interpolating by rounding to the nearest integer coordinates, so some points in the destination image might be left without a correspondent. To mitigate this aspect, for very pixel that that doesn't have a correspondent after rounding, we average the surrounding pixels. This, indeed, helps in giving better results.

In figure 2, one can see that the images don't have the same size or transformation. This is due to the random part of our RANSAC algorithm: at each run we might get different transformations, depending on the chosen inliers. However, most importantly, the pose of the boats in the two transformed images align for the 2 implementations.

Another difference is the size of the output images. In our implementation, we calculate the offset of of the pixels for the x and y coordinates in the destination image as the minimum value of the coordinate between the four corners. If the offset is positive, then the value is subtracted from the destination coordinate. This prevents extra black padding around the warped image. If the offset is negative, then the value is added to the destination coordinate. This prevents out of bounds errors when generating the image. We are not sure what the differences are between our implementation and the MATLAB one, but the differences are small enough that they can be neglected.



(a) Boat1 Original (b) Boat2 to Boat1 using Matlab (c) Boat2 to Boat1 using Own

(d) Boat2 Original (e) Boat1 to Boat2 using Matlab (f) Boat1 to Boat2 using Own

Figure 2: Image Transformations

Question 2

2.1 The affine transformation is defined as

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t1 \\ t2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

with $m_1, m_2, m_3, m_4, t_1, t_2$ being the unknown variables. Thus there are 6 unknowns, so we need at least six equations for a well posed problem. Since each match gives 2 equations, one for each coordinate, we need at least 3 matches to be able to solve for the parameters of the affine transformation. Naturally, since this is only an estimation, a 6 equation system will typically be incompatible, so more points would be needed to find a best fit solution, typically using least squares. The more matches, the better in general, as long as they are not spurious (i.e. there are no outliers).

2.2 The number of trials required for a good fit is dependent on the desired confidence that at least one model will contain only inliers. This can be expressed as

$$p = 1 - (1 - (1 - e)^s)^N)$$

so N, the number of trials can be extracted as

$$N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

where p is the desired confidence level, e is the probability of choosing an outlier if we assume uniform probability of outliers vs inliers, and s is the minimum sample size required to fit the affine transform. In our case, there are 947 matches, and the minimum number of matches needed to estimate sample size is 3, as detailed earlier. This gives $e = 3/974 = .003$. So, for a confidence level of .9999, N should be set to 2.

To test this claim, we setup an experiment in the following form. We range N from 1 to 5 and for each of those we run 10 RANSAC procedures, collecting the inlier count of the best transformation. We then compare the number of inliers between the procedures with varying number of trials. The results can be observed in figure 3. This shows that for a number of trials of 2 or higher, the majority of transformations obtained are of high quality, as indicted by the high number of inliers present. For example, for $N = 5$, out of 10 trials, 8 yielded a high number of inliers, whereas only 2 gave poor results. This contrasts with the results obtained when we used $N = 1$: the majority of trials yielded poor results: more than 75% trials with under 200 inliers(compared to 900 inliers for the rest of the trials.

2 Image stitching

Question 1

1.1 Using the RANSAC function that we implemented earlier and also a translation function, we created a script that stitches images based on common feature points.We followed the guidelines given to us in the assignment and took into consideration all the intermediary steps and hints. In the end, our implementation of the algorithm yields the stitched version of the two images given to us. However, we encountered a small impediment that we didn't succeed to solve in an elegant manner. We were able to get the correct rotation of the second image in order to match the first one, but with respect to the translation of the *right.png*, the computed values were erroneous. We managed to tune the position of *right.png* using a formula involving *t1* and *t2*, thus the algorithm yielding the correct stitching. We suspect the issue lies within the RANSAC algorithm, the part when we use the *least squares* method, but we still need time to investigate further in order to confirm out speculations. and to solve the issue.

1.2 Figure 4 helps us to visualize the intermediary steps suffered by the images in order to be stitched together. Sub-figures (a) and (b) shows us the gray-scales of *left.png* and *right.png* respectively. In sub-figure (c) we ca observe the transformed *right.png* using the affine transformation computed by our implementation of RANSAC algorithm. We can observe how the orientation of the tram in *right.png* aligns with the orientation from *left.png*. Finally, the stitched version is presented in sub-figure (d).

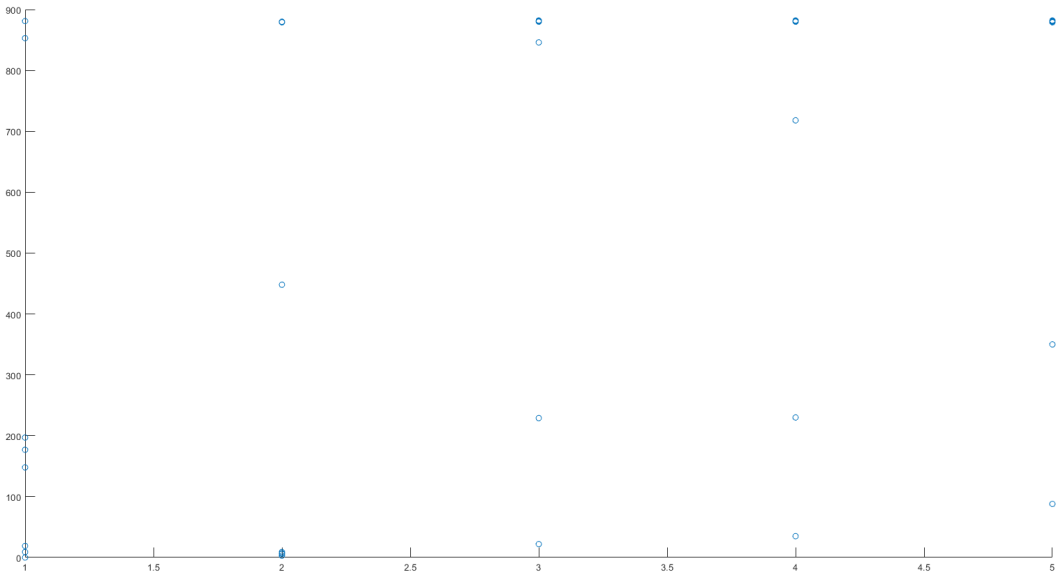


Figure 3: Number of inliers for different setups of N. For each N, we run 10 RANSAC trials

Conclusion

By working at this assignment, we gained a further understanding of the importance of choosing feature points invariant of scale, rotation, and also learned about the RANSAC method. We learned about other practical applications of this method and about the proper hyper-parameters to be used in order to maximize its performance(of course, empirically chosen). Conceptually we understand the reasoning behind basic stitching algorithms. We admit that some issues we encountered on the way were caused by not having a solid understanding of the coordinates system used by images and plots in Matlab(they are different), but the ones regarding the computer vision theory challenged us in a constructive way.

(a) Left image

(b) Left image

(c) Transformed right image

(d) Stitched images

Figure 4: Image stitching