Computer Vision 1 - Lab 4

Pauliuc, Andrei-Sebastian
11596619

van Slooten, Renzo
11129778

1 Introduction

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision [1]. In this assignment, the final purpose is to stitch two images that present common points. In order for this process to work, we first have to align the right image with the left one. This is done by finding the affine transformation between them. VLFeat library contains functions that find and match features between pair of images. With the matched features, random sample consensus (RANSAC) computes the transformation parameters that map the features of one image to the other. These parameters are finally used to get the correct orientation of the right image. The left and transformed right image are precisely placed on two canvases which are subsequently combined into a single stitched image.

2 Image Alignment

Question - 1

The first step in image alignment is the keypoint detection and matching. Using VLFeat, we can detect features and descriptors from pairs of images and match those features based on their local appearances (**keypoint_matching.m** function). With the extracted matched features, we can perform random sample consensus (RANSAC) to extract transformation parameters which map the features from the first image to the features of the second image. An intermediary step is to show the initial images together with the matched points to observe their distribution and to detect the presence of any outliers. Figure 1 displays images *boat1.pgm* and *boat2.pgm* side-by-side, highlighting 50 random feature points with red and green dots, and showing the matched points with lines. While most of the points in this image are correctly matched, we can still observe that some of them have been wrongly detected as matching features; such points are considered to be outliers to our analysis.

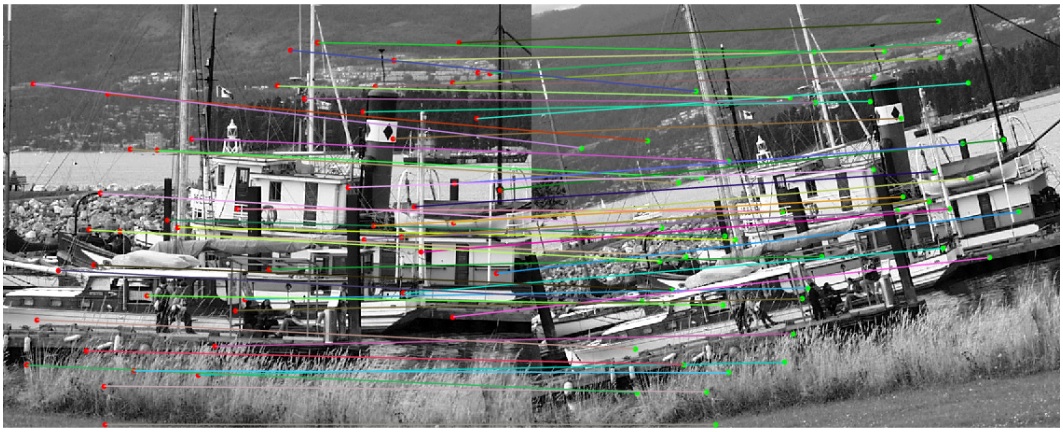

Figure 1: 50 random matched keypoints between boat1.pgm and boat2.pgm

In **RANSAC.m** function, P matched points are selected randomly from the whole set, and an approximation of the transformation parameters are found. These parameters are then used to compute the transformations of all feature points from the first image. We can thus detect which points are inliers; in this case, inliers are the transformed features that lie at a maximum distance of 10 pixels from the matching feature in the second image. Repeating this procedure N times, we ensure that the parameter approximation is performed on different sets of points, which might perform better than other sets. At each repetition, we check whether the best number of inliers has increased or not; if it did increase, we save the parameters as the best set of parameters. After N repetitions, we perform again Least Squares on the whole set of inliers. This ensures that our solution is robust and is computed from a bigger collection of features.

After we got the best transformation parameters from RANSAC, we transform the first image according to the transformation parameters. This transformation can be done in two ways: 1) standard MATLAB functions, like *imtransform* and *maketform*, or 2) own implementation, which can be found in function **transform_image.m**. In our own implementation, we first transform all pairs of (x, y) coordinates using the resulting parameters, while also padding the image because rotations might lead to negative coordinates. After padding and computing the new width and height of the image, we fill in pixel values from the corresponding pixels in the first image. Worth mentioning is that we also use some kind of nearest-neighbour interpolation to fix the black points resulting from rounding the transformed coordinates (rounding leads to skipped pixels). For the black points in the new image, this interpolation is made by picking the median value in a 3×3 window. The reason behind choosing the median is because this allows us to preserve the edges of the image and the black background surrounding it, while also correcting all bad pixels.

The results of the transformations (boat1 $\rightarrow$ boat2) and (boat2 $\rightarrow$ boat1) can be seen in figure 2. For easier visualisation of the results, we have placed the input images at the top; below image *boat1* lie the resulting transformations (boat2 $\rightarrow$ boat1), while the images below *boat2* are the results of the transformations (boat1 $\rightarrow$ boat2). After running the RANSAC algorithm with $P = 3$ and $N = 100$, we transform the images using the aforementioned implementations. The transformation (boat1 $\rightarrow$ boat2) using our own implementation can be seen in figure 2d, while figure 2f shows the same transformation with MATLAB functions. The other transformation, (boat2 $\rightarrow$ boat1), is found in figures 2c and 2e for the own implementation and the transformation with MATLAB functions, respectively.

The results show that the transformation is done in a correct way because the transformed images have the same orientation as the ones at the top. An interesting observation is to see that the images using with our own transformation present some irregularities caused by the correction done with nearest-neighbour interpolation. This is especially visible in the transformation (boat2 $\rightarrow$ boat1) (figure 2c). The irregularities can be seen mostly in areas where edges are close to each other or pixels have very different values. We consider that with a better interpolation algorithm, these images can be adjusted and finely tuned.

The demo script to run image alignment can be found in file **image_alignment_demo.m**.

(a) Image boat1.pgm

(b) Image boat2.pgm

(c) boat2 to boat1 - own implementation

(d) boat1 to boat2 - own implementation

(e) boat2 to boat1 - MATLAB functions

(f) boat1 to boat2 - MATLAB functions

Figure 2: Boat images transformations

Question - 2

Minimum number of matches In order to solve the affine transformation between two images, we need to use at least 3 matches. For each picked match, 2 equations are created; in vector notation, they are written as

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{1}$$

Since the transformation vector is the same for all points, we can stack the coordinate matrices vertically, the resulting equation being

$$Ax = b$$

$$\text{where } A = \begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix}, x = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix}, b = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix}$$

This gives us a linear system of 6 equations with 6 unknowns $(m_1, m_2, m_3, m_4, t_1, t_2)$. If we increase the number of matches used, the number of equations increase as well; the new system becomes overdetermined and can be solved using the least squares method. Thus, the minimum number of matches used to determine transformation parameters needs to be 3.

Number of iterations On a theoretical level, the number of iterations needed is dependent on how many matches have been found in the pair and how many matches are used in RANSAC. Let's denote the total of matches with H and the number of matches used with P. Assuming the matches are picked randomly from the set using uniform distribution, we would prefer to have $N = H/P$ iterations. In this case, all matches have an equal probability of being picked, which means that after $(N + 1)$ iterations, we have picked each match at least once. Since the randomisation step of the matches is being done at each step, of course this might not happen. In theory though, we would expect this formula to work well. An equal influencing factor is the number of outliers (wrongly detected matches), which might badly influence the outcome of the algorithm. If the outliers are present in a very big proportion in the dataset, we would prefer to choose a small set of matches (small P), thus increasing the number of iterations to ensure that we have computed the parameters using mostly inliers.

Besides the theoretical explanation given above, we have also conducted a small practical experiment where we tested the RANSAC method with different values for N and P. The idea was to see how many iterations it needs to get to the best number of inliers. Across all the tests, the average number of inliers is 905, the predominant count being 905. The average number of iterations needed for the count of inliers to reach the maximum is 18.29. Table 1 below summarises the average number of iterations across 15 RANSAC tests for each setup of parameters N and P.

N	50	100	300	50	100	50	100	10	50	10	50
P	3	3	3	10	10	25	25	50	50	100	100
Avg. inliers	13	48.2	44.6	8	20.2	9.4	7.6	3.4	18.2	4.6	24

Table 1: Average inliers based on different N and P values, across 15 tests

From the table, we can observe the average number of inliers increasing almost always when N increases. Although RANSAC ensures we get the highest number of inliers using the approximation based on P points, our results might indicate that it takes more iterations to get to 906 inliers compared to 905 inliers found 20 iterations before. The conclusion from this analysis is that an average number of iterations cannot be computed, because the selection of matches is purely random, and running the algorithm with the same parameters twice will surely result in the best count of inliers being

computed at different iterations. Nevertheless, our rule of thumb ($N = H/P$) holds, and, as seen is the results part, the algorithms give satisfactory results.

3 Image Stitching

Several steps are taken in order to stitch the two images seen in figure 3 together:

1) Detecting SIFT features
2) Matching features between the two images
3) Transforming 2^{nd} the right image using RANSAC
4) Canvas creation and placement
5) Combining the two images

The first 3 steps are explained in the section Image Alignment. To create the identical sized canvases for the two images, the size of the stitched image need to be calculated. The stitched image needs to fit the maximum x and y coordinate of the left and transformed image. The maximum of the left image is simply equal to the width and height of the image. For the transformed image, the maximums of the transformed corner coordinates are needed. These coordinates are calculated with (1).

Now that the size of the canvases is determined, both images are placed in these black canvases. Since we only had to focus on our specific case, the left image is simply placed on the top right of one of the canvases. The placement of the transformed right image is based on the position of the lower right corner. This position is found by using the maximum x and y of the transformed corner coordinates.

We used black canvases, appended with the left and transformed right images, to make it easier to combine them. The RGB values of two canvases are summed using **imadd()**. Due to the summation with zero's, non-overlapping pixels will get the correct RGB values. But this also means that overlapping pixels will become lighter then desired. That's why first imsubtract() is used to clear out the overlapping pixels in one of the canvases before adding them together. The resulting stitched image can be seen in figure 4.

The stitching demo script is in file **stitch_demo.m**.

Figure 3: Original left and right image respectively

Figure 4: Stitched image

4 Conclusion

After using the RANSAC algorithm to simply transform images and to align them for stitching, we have observed how it helps us in modifying and playing with images. The image alignment algorithm provides good transformation parameters, which, if applied correctly, will transform the input image into a robust output. An equally interesting observation is how the interpolation method can improve the quality of the transformed image. When no interpolation is done and the transformed coordinates are rounded, the output image presents many black pixels. Using the image alignment algorithm for a more advanced purpose, we have seen how stitching two images works. The result of the stitched images gives an visually pleasing result. In our specific case, the best result is obtained when the position of the right image is based on the lower right corner. When based on another corner, the image is shifted up by one pixel, which causes a visible seam. This is explained by the rounding of the transformed coordinates. A variety of techniques have been developed to eliminate these visible seams [1].

References

[1] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.