Comuputer Vision Lab 4

Maximilian Schlögel UvA
Anca-Diana Vicol VU
Benjamin Kolb UvA
Philipp Lintl UvA

1 Introduction

In this assignment, we explore the possibilities that come with matching keypoints in different images of the same scene. In the first part, we use matchings to compute the shift and rotation between images. This can then be used to transform and align different images of the same scene. In the second part, we take two images of the same scene but different views to obtain one stitched image without overlappings. To achieve that, functions from the first part are drawn on in order to determine keypoints between the two images.

2 Image Alignment

In the first section we will look into image alignment, specifically how we can take too images of the same scene and computes the corresponding affine transformation. We will make use of the toolbox 'VLFeat' for matlab to first find the matching points between those two images. Then we will apply the RANSAC method to compute the most fitting transformation (parameters) using these matching points. For actually applying the transformation we will need a method for dealing with the discreteness of pixels the real-valued transformation parameters. Here we first implement the nearest-neighbors method and compare it to the built-in 'imwarp'-method from matlab.

2.1 Question-1

To be able to perform image alignment and how to transform the images, we need to know what regions on both images correspond to each other. We used the toolbox 'VLFeat' for finding these regions. We can see in the figure 1 an example of how such a matching of regions/points would look like.

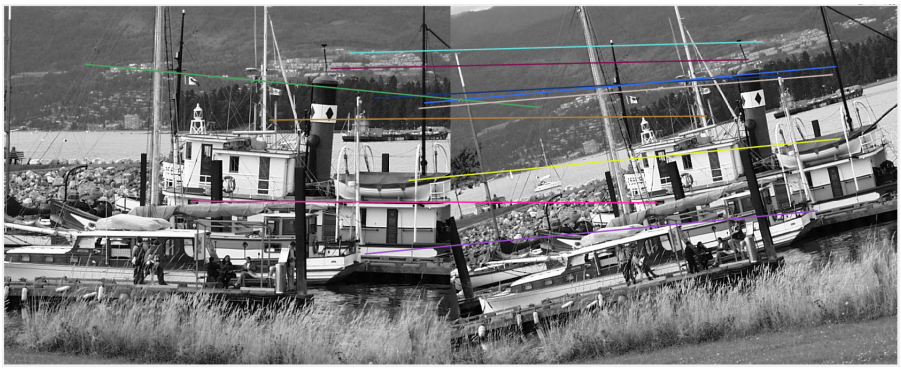

Figure 1: Sample of matched keypoints

Using these regions and the corresponding centre points, we applied RANSAC to find the transformation parameters. 250 times we took 25 randomly chosen points of one image and the matching points

Preprint. Work in progress.

from the other and used those to solve the following function for v. Here $(x_1, y_1), (x_2, y_2), ...,$ are the points of image 1 and $(x'_1, y'_1), (x'_2, y'_2), ...,$ are the points in image 2

$$Av = \begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_2 & 0 & 1 \\ \dots & & & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \dots \end{bmatrix} = b \tag{1}$$

Matlab this gives us the least-squared solution for this linear system. Here the $m_1, ..., m_4$ parameters in vector v are the parameters determining the rotation and stretching of the points and t_1, t_2 represent the translation along the x-, and y-axes.

After learning these parameters we know how to transform on image in such a way that it aligns with the other image. One issue we face when doing this transformation is the discrete nature of the pixel representation of images. The transformation parameter ar very likely real valued, that means, when transforming the discrete pixel values (done by multiplication with the parameters) we get real values, which cannot be easilty translated into a discrete representation again. There are different ways to deal with this issue. The one we are using is the nearest neighbor method, which assigns the transformed pixel value to the nearest pixel. For example a pixel value transformed to be at pixel coordinates $(10.3, 21, 7)$ is assigned to the pixel $(10, 22)$. Since this may lead to pixel, that have no assigned value (see figure 2), we use this method the other way around. We go through every pixel in the new image and use the inverse transformation, combined with nearest neighbor, to get the pixel value in the old image (the one we want to transform) that corresponds to the inverse transformed pixel. This way we make sure, we assign a pixel value to every pixel in the new image (see figure **??**). Matlab itself also offers a built-in function 'imwarp' that does the transformation for us. This method uses linear interpolation. Comparing both methods, displayed in figure 4 and 3, we can see that the nearest neighbor method we use is a lot more 'pixely' and the built-in imwarp function looks a lot smoother.

Figure 2: Nearest neighbor without taking the inverse transformation

Figure 3: Nearest neighbor method we use

Figure 4: Matlab's 'imwarp' function using linear interpolation

Now that we have dealt with this issue we can finally apply the alignment-transformation to the images. Looking at the images of the boats in figure 1 it is clear which one is the 'right way up' and which one needs to be aligned. But it still is possible to apply the transformation both ways and sometimes it either does not matter, which way we go or sometimes it is not clear which one is the

right way. Therefore it is also interesting to look at both possible transformations: The first image in figure 1 aligned with the second (this can be seen in figure 6) and vice versa (this can be seen in figure 5).

Figure 5: Transformation of image two to fit image one

Figure 6: Transformation of image two to fit image two

2.2 Question-2

1. how many matches do we need to solve an affine transformation which can be formulated as shown in figure 1? 2. How many iterations in average are needed to find good transformation parameters?

2.2.1 Question-2-1

How many matches do we need to solve the linear system for the affine transformation? Since equation (3) in the exercises has six unknowns but only two rows for one match, we need to stack it up for at least $P = 3$ matches in order to get a unique solution. But this only holds theoretically. In practical matters, we could get wrong matching regions and in this case, only three matches would lead to very bad results, since a large portion of the results are relying on the bad matching. Therefore it would be useful to use a lot more matchings and using the inbuilt linear system solver in Matlab then returns a good approximation by using the least-squared solution.

2.2.2 Question-2-2

How many iterations do we need to get good transformation parameters? In order to find out which number of iterations is required to get a sufficiently good results, we set $P = 25$ and ran our algorithm 10 times with 30 iterations each. In figure 7 we see the development of the average success rate, depending on the number of iterations. We can observe that around 6 iterations are needed to get a average success rate of around 96 %, which does not increase after that. So we conclude that with a set of 25 matching points, around 6 iterations are needed to get good transformation parameters.

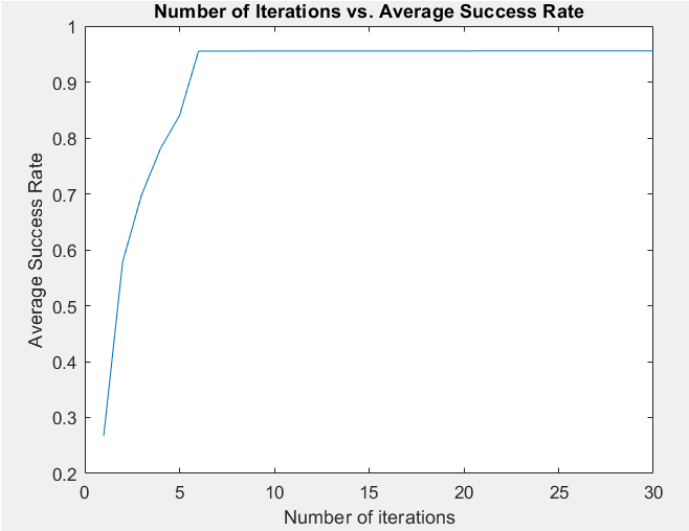

Figure 7: Average parameter quality after number of iterations

3 Image Stitching

In this section, we implemented a stiching mechanism, that draws from the functions of task 1. In it, two overlapping images of the same scene but different angles, `left` and `right`, as seen in Figure 8/9 are taken as input. The goal is to concatenate the two into one final image based on their matching keypoints without displaying the overlapping parts twice. In order to stitch the right image on the left, we firstly need to transform the latter to display it regarding the coordinates of the first image as its angle is different.

Figure 8: `left.jpg` image.

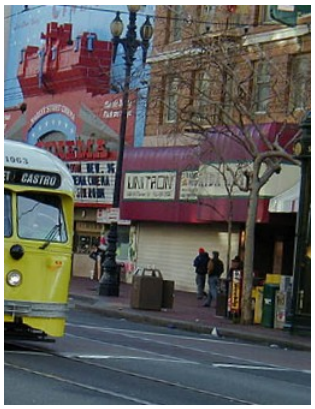

Figure 9: `right.jpg` image.

This is accomplished by our `transform_image.m` function, which transforms the right image in terms of the left. Again, we use our `Ransac.m` function together with `SIFT` in order to determine keypoints. This enables the representation of the right image regarding the coordinates of left and is necessary to stitch them together. The transformed image of `right` is seen in Figure 10.

Then, the new corners of the right image are obtained by applying Equation 1 with regard to the original corner points of the right image. After transforming the corners, we can calculate the size of the final, stitched image. Its width is the difference of the maximum of the new corner width and the width of the left image. The new height is obtained similarly, simply swapping the width with height of the respective images. Then, We create a new image with corresponding size and fill it with zeros. Finally, the stitched image is the left image up until the width of the original left image and

then rightwards filled with the transformed right image. In total, Figure 11 depicts the final stitched image, which had Figure 8 and Figure 10 as input.

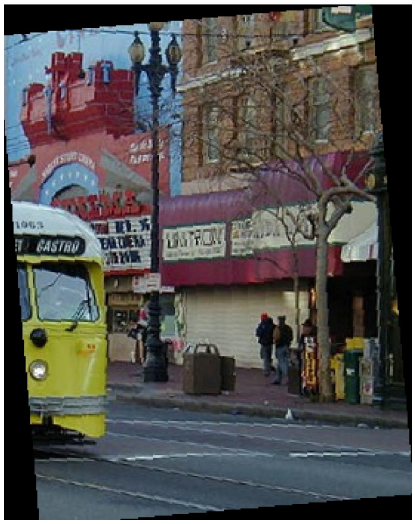

Figure 10: Transformed image of `right.jpg`

Figure 11: Final stitched image

When we assess the quality of the stitched image, we focus on the transition of the two images. Looking at Figure 11 shows that our stitching procedure worked. We draw this conclusion, as the signs on the floor, such as the rails and the white vertical line seem to be aligned. Furthermore, the red part of the wall does not indicate any transition, as well.

4 Conclusion

In the first part we were able to apply keypoint-matching and use this and RANSAC to apply image alignment. For the application of the approximated transformation we also implemented the nearest neighbor method. We compared this one to the built-in function 'imwarp' and could observe that 'imwarp' delivered smoother and better quality transformations than nearest neighbors. Lastly we investigated how many iterations are needed to get good transformation parameters, which turned out to be at least 6 iterations. The second part demonstrated, that two overlapping images can be stitched together to one image. We only need to represent them both regarding the same coordinates, match keypoints and then fill a new image based on the new image size.

Workload division:

Exercise 1 - Image Alignment: Benjamin Kolb, Maximilian Schlögel

Exercise 2 - Image Stitching: Anca-Diana Vicol, Philipp Lintl