Computer Vision - Theoretical Assignment 4

Wolf, Florian - 12393339 (UvA)
flocwolf@gmail.com

Biertimpel, David - 12324418 (UvA)
david.biertimpel@protonmail.com

Lindt, Alexandra - 12230642 (UvA)
alex.lindt@protonmail.com

Fijen, Lucas - 10813268 (UvA)
lucas.fijen@gmail.com

October 5, 2019

Introduction

In this assignment, we explore how to combine images of the same scenery taken from different viewpoints. For this purpose, we first explore key point matching between two images using the *SIFT* method. We then implement the *RANSAC* algorithm for estimating a transformation between two images from given key point pairs between them. Finally, we incorporate both methods into an algorithm that stitches two images of the same scenery together into one image.

1 Image Alignment

In this section we will align two images using keypoint matching. Keypoint are points of interest in an image. By matching these points between two images we can calculate the rotation matrix M and translation vector t. To find and match the key points we use the `VLfeat` library [4].

1.1 Question 1

For this exercise two pictures of boats are analyzed. Both images mostly show the same scenery, only the 2nd image is clearly rotated to the left.

(a) (b)

Figure 1: Original pictures of the boats

Our function `keypoint_matching.m` takes image 1 and the rotated image 2, localizes points of interest using the function `vl_sift.m` and then matches the keypoints using `vl_ubcmatch.m`. The following plot shows an example of a subset of 10 interest points in each image.

Figure 2: A sub-set of 10 interest points on the image.

Once these keypoints are matched, we plot a connection line between every pair. Each line uses a unique color to make distinction easier. However, quite often not all matches are correct.

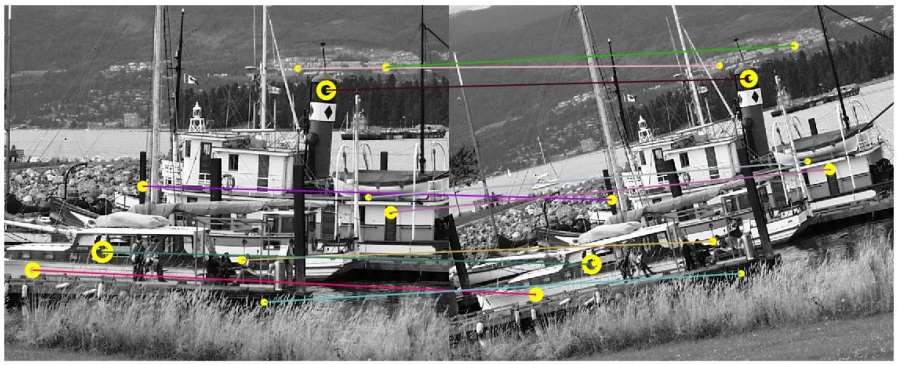


Figure 3: A sub-set of 10 interest points connected from the original image to the transformed image.

To align the second image to the first one, we wrote the `RANSAC.m` function. The algorithm bases on the following matrix calculation to calculate the rotation and displacement of an image compared to the other:

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

This equation is solved for a randomly selected subset of size P keypoint matches. The resulting transformation matrix is then applied to the image and for every matched keypoint the inliers within a 10 pixel radius are counted. These steps are iterated over for a chosen N times. The transformation with the lowest amount of inlier is selected as the resulting transformation matrix.

In our function we performed this process the other way around than one might expect, so if we wanted to transform image1 to image2, we calculated the linear transformation to go from image2 to image 1. With this transformation, we calculated for each pixel in the image using the transformation matrix

the position in the original matrix that best corresponds to that pixel. If the rounded transformed position points to an actual position in the matrix, the corresponding pixel is placed in the new image. By doing so for every pixel in the new image, we can reconstruct an image without any missing pixels within the image.

(a) (b)

Figure 4: Transformed picture compared to the target picture using $N = 10, P = 10$

(a) (b)

Figure 5: Transformed picture compared to the target picture using $N = 10, P = 10$

As we can see in Figure 4 and Figure 5 the results for these . Occasionally a bad transformation might happen due to the fact that only 10 times 10 random matches were selected and used. However most of the cases this method seems to be showing a correct transformation of the images. As the dimensions of the image get changed, the composition of the image might seem a bit odd, and small jumps in lines can occur due to the rounding of positions of pixels from the transformation.

We also compared our results with the built-in function of MatLab: `rotate.m`. This function uses the by default the 'nearest-neighbour', bilinear, or bicubic interpolation. The left image shows our plot and the right image was generated by MatLabs default function [3].

For better comparison with the original image, the aligned image has been cropped to the according size. Even tough the images vary slightly in pixel definition, the transformation was clearly successful.

(a) (b)

Figure 6: With RANSEC.m translated image compared to MatLab imwarp.m translated image.

(a) (b)

Figure 7: Cropped translated image compared to original image.

1.2 Question 2

The theoretical minimal number of matches would be three to make any kind of transformation possible. We have an equation with 6 unknown variables. Three matches result in 6 equations and the equations system can be solved. With less, some of the dimensions would become free variables in the linear transformations, and it would therefore be a lucky shot if a good result would have been found. However, as we know from before, there also are some incorrect matches between the matches. If you would narrow your amount of matches down to 3, and one of the three turns out to be an incorrect match, an incorrect transformation would be found. A higher amount of matches, will reduce the impact of having an incorrect match in the subset. We found that using 10 matches gave us stable results.

We also found that good results can be achieved with 10 iterations in the RANSAC. Less iterations might every now and then result into a good transformation, but also increases the chance that a poor transformation is found. A concrete number can not be given with respect to the question. As this part of the code is not that computationally expensive, 100 iterations could be chosen if one needs more stable results. When a certain level of significance would be needed, we would suggest to use the Student's T test or similar tests [1].

2 Image Stitching

With the previously described functions `ransac` and `keypoint_matching`, we implement a procedure for image stitching that combines images of the same scenery from different perspectives into one image. In the following, we go through the image stitching process with the two images shown in figure 8, where we plan to stitch the left image (I_{left}) and the right image (I_{right}) together.

(a) (b)

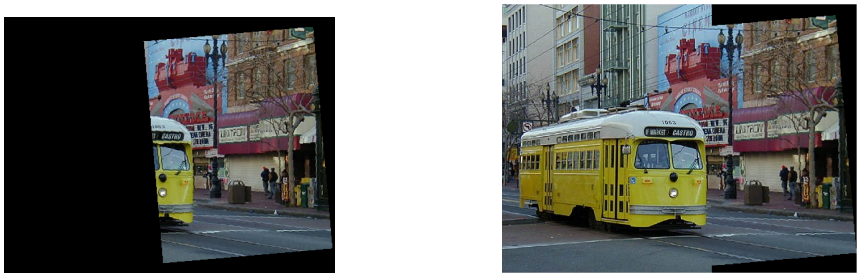
Figure 8: Images used for the stitching process.

In order to be able to stitch two images together, we must first align them correctly. For this purpose, we use the matched keypoints of both images (obtained from `keypoint_matching`) to transform I_{right} into the coordinate space of I_{left}. This is achieved with the transformation matrix $\boldsymbol{M} \in \mathbb{R}^{2\times 2}$ and translation vector $\boldsymbol{t} \in \mathbb{R}^2$ that we receive from the `ransac` function with our keypoints.

To perform the transformation, we first create a new black image I_{trans} whose size is determined by a scaling factor and the size of the original image I_{right}. I_{trans} represents I_{right} transformed by the `ransac` parameters. To fill I_{trans} with the respective pixels from I_{right}, we iterate over I_{trans} and transform every pixel p'_{xy} back to its original position p_{xy} in I_{right} with

$$\begin{aligned} p_{xy} &= \boldsymbol{M} \cdot p'_{xy} + \boldsymbol{t} \\ &= \boldsymbol{M} \begin{bmatrix} x' \\ y' \end{bmatrix} + \boldsymbol{t} \end{aligned}$$

where x and y are the coordinates of the pixel position. If p_{xy} is defined in I_{right}, we assign its color value to the corresponding p'_{xy}. If not, p'_{xy} stays black in I_{trans}. By using this procedure, we avoid holes in the transformed image. The resulting I_{trans} is shown in figure 9a. Since the `ransac` function includes randomness, the outcomes of the described procedure may vary slightly.



(a) (b)

Figure 9: Illustration of I_{trans} after the transformation from I_{right} (a) and the final stitched image where I_{left} and I_{right} are combined (b).

After transforming I_{right} we want to combine it with I_{left} to obtain the stitched image. In order to

do this, we stitch I_{left} and I_{trans} together by filling the black pixel values of I_{trans} with the respective pixels from I_{left}. Following, we cut away the completely black bottom rows and the completely black right columns. This procedure results in the final stitched image we observe in figure 9b. For the results shown in figure 9 we executed `ransac` with $N = 50$ iterations and $P = 10$ random matches.

Conclusion

In this homework, we have explored image transformation and image stitching. We showed how *RANSAC* and *SIFT* can be used to allign images by matching points of interest. In this method in iteration small subsets of these matching points were alligned and the resulting transformation matrices were used to transform the entire picture. We showed that actually quite low amounts of iterations ($N = 10$) and a small subset ($P = 10$) can already result in stable and proper results. Further we used this technique to perform image stitching on real world examples. We used two images of a tram in a city which had a small shared area. We showed that using *RANSAC* and *SIFt* we were able to find proper allignments of the two images and were able to accurately stitch the two images together into one larger image. If one would expand on this subject, these methods could be expanded into the alignment of videos. This would be an interesting field to study further as this would create a 3 dimensional space that requires alignment [2].

References

[1] Geoff Cumming. *Understanding the new statistics: Effect sizes, confidence intervals, and meta-analysis*. Routledge, 2013.

[2] E Roy Davies. *Computer and machine vision: theory, algorithms, practicalities*. Academic Press, 2012.

[3] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[4] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.