Assignment4 - Computer Vision

Tushar Nimbhorkar 11394110
Diede Rusticus 10909486

February 2017

Abstract

This week is the goal to get a better understanding of image alignment and image stitching.

1 Image Alignment

To align an image (*image1*) according to another image (*image2*) a mapping function needs to be created to transform every pixel to a new coordinate. This mapping function:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} + T \qquad M = \begin{bmatrix} m1 & m2 \\ m3 & m4 \end{bmatrix} \qquad T = \begin{bmatrix} t1 \\ t2 \end{bmatrix} \tag{1}$$

As you can see, M and T contain 6 parameters in total. These need to be computed so the following steps were taken:

1. A function is created which requires image1 and image2 as arguments.
2. The two images are put in grayscale and in single precision.
3. The keypoints (or *salient points*) and their descriptors are computed with the function `vl_sift` provided by the package VLfeat.
4. Likewise, the set of matched keypoints T is extracted, based on the descriptors of the two images with `vl_ubcmatch`.
5. We are going to compute the parameters based on P randomly picked matches.
6. The resulting parameters will enable us to transform all pixels in image1. But first, to check the accuracy of the parameters, we will check for all matches in T the Euclidean distance between the ground truth coordinate and the coordinate transformed based on the parameters.
7. If the point lies within a radius of 10 pixels, the point is count as an inlier.
8. Because the parameter-set is based on the randomly drawn P keypoints, we are repeating this process N times, and overwriting the set of parameters when a higher number of inliers is found.
9. We only need 3 matches to solve an affine transformation formulated as equation (1), because we have 6 unknown variables and to solve we need 6 equations. But having $(x, y) - (x', y')$ as a pair, we only need 3 set of pairs to solve the system of equations. We set N to 10, but we see it converges after 5 iterations. The result is a percentage of inliers of 95%.

10. Fifty random matched keypoints are plotted on the two images and a line is drawn between them to show which points are matched. Figure 1 shows this result.

We now have a set of parameters which enables us to transform all pixels and create the new image. To create the new image you can chose to either loop over the pixels of the new image, and fill in its value based on the first image, or you can loop over the pixels of the old image and calculate the new coordinate in the new image and fill this pixel with its value. We have chosen for the latter, however, this does requires us to do k-nearest-neighbours, to fill in the black gabs. Namely, the computed new x and y coordinates are floats rounded to integers, so some pixels will be skipped. Therefore, the resulting image will show little black dots.

1. We initialise the new image filled with zeros. The size of this image will be bigger than image1 due to the transformation. To calculate the correct size of this image we need to determine the coordinates of the corner points. First, we calculate the new position of the whole image by only transforming the corner points of image1. The height of the new image is the maximum row index minus the minimum row index. The width of the new image is the maximum column index minus the minimum column index.

2. In order to make sure the transformed image is exactly placed within this new image, a shift is calculated needed for all coordinates. This shift is defined as:

$$shift = \begin{bmatrix} column\_shift \\ row\_shift \end{bmatrix} = \begin{bmatrix} round(min(column\_index)) \\ round(min(row\_index)) \end{bmatrix}$$

3. We now loop over every pixel (x, y) in image1 and calculate the new coordinates (x', y') with the mapping function (1) and the best set of parameters.

4. We subtract the shift from the new coordinates.

5. We fill the pixel (x', y') with the pixel value of (x, y). The results are shown in Figure 2 and in 3. Also, the matlab inbuild method is used to compare our results with. It shows that our method is similar to the inbuild function.

Figure 1: Results of the SIFT function; find matched keypoints

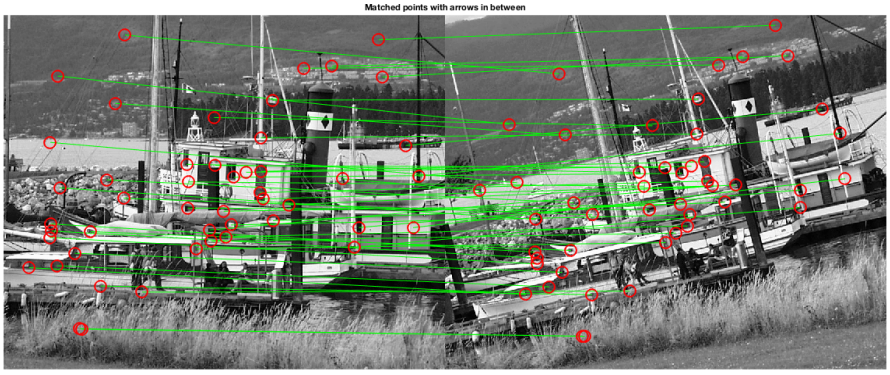


Figure 2: Results of the transformation function and k-nearest neighbours using boat1 as image1

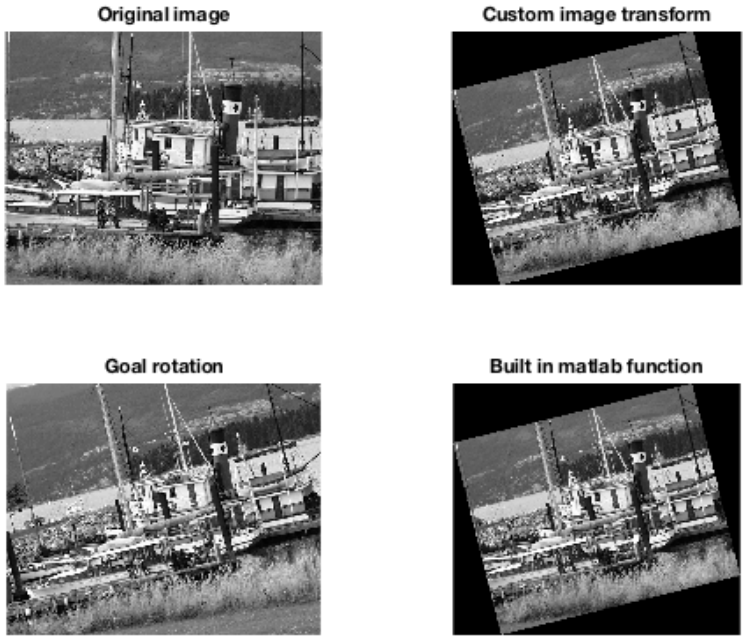
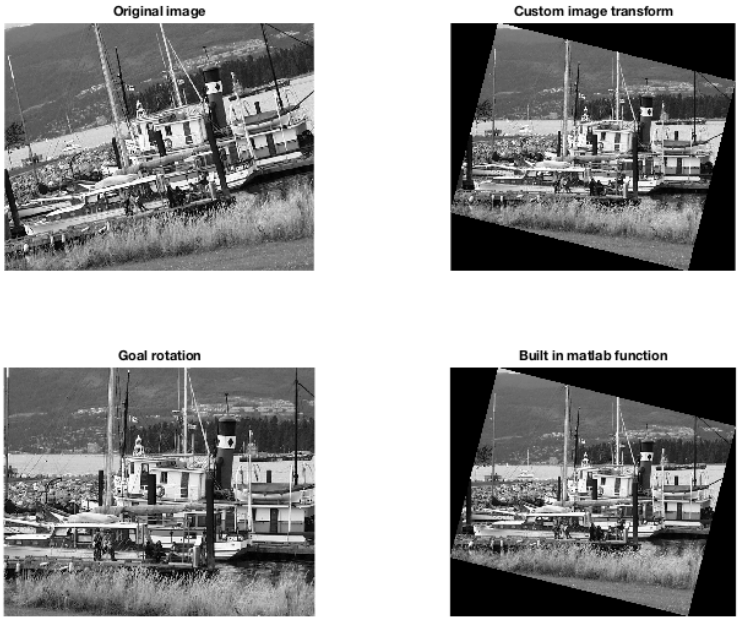


Figure 3: Results of the transformation function and k-nearest neighboursusing boat2 as image1



2 Image Stitching

To stitch two images (*left.jpg and right.jpg*). We first find the transformation parameters using RANSAC function. Having found the transformation parameter. We first find the transformation of the corner points. This will help us getting the linear transformation. At the same time we find the whole transformation of the image. We first calculate the linear shift based on the transformed corners and second, we apply the linear transformation. We use 4 conditions to find this shift. Below, we will refer to the left-image with *the original image* and to the transformation of the right-image with *transformed image*.

1. Left corner has a negative x-coordinate:
   There will be shift in the x-direction of the original image. This shift is the absolute value of the x-coordinate of the left corner, and adding +1 (because of the Matlab indexing)

2. Left corner has a positive x-coordinate:
   There will be a shift in the x-direction of the transformed image. This shift is the absolute value of the x-coordinate of the left corner, and adding -1.

3. Right corner has a negative y-coordinate:
   There will be shift in the y-direction of the original image. This shift is the absolute value of the right corner and adding +1.

4. Right corner has a positive y-coordinate:
   There will be shift in the y-direction of the transformed image. This shift is the absolute value of the right corner and adding -1.

Because of the image stitching, we will exceed the size of the original image. We will therefore have to pad the two images with zeros according to the size of the stitched image. This is done by following two conditions:

1. If the difference in height of the transformed image and the original image is greater than zero, we pad zeros to the original image in the y-direction. Otherwise, we pad with zeros to the transformed image in the y-direction. The number of zeros we pad is equal to the absolute difference in height.

2. If the difference in width of the transformed image and the original image is greater than zero, we pad zeros to the original image in the x-direction. Otherwise we pad zeros to the transformed image in the x-direction. The number of zeros we pad is equal to the absolute difference in width.

Finally, both images have the same size. To stitch the images together we simply add the two image matrices. However, we still have the problem with some pixels that are overlapping. If we just add the two images, the problem occurs that the overlapped area will be very bright. We could divide every pixel by 2 in order to avoid the bright overlapping part. However, the non-overlapping parts will be darker this way. This is shown left in Figure 4. Therefore, we decided to go take the maximum value for every pixel in the addition. This will not affect the non-overlapping parts, but a pixel in the overlapping part will just get either the value of the original image or the value of the transformed image. On the right in Figure 4 it shows our final result.

Figure 4: Result of image stitching