Computer Vision - Theoretical Assignment 3

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

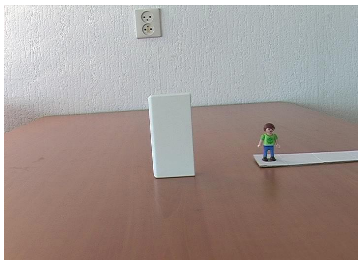
Introduction

This assignment discusses the task of finding unique points in images and tracking them over time. For this we first explore the Harris Corner Detection algorithm and apply it to sample images. In the course of this we discuss different properties of Harris Corner Detection, which we compare with a similar algorithm by Shi and Tomasi. Following, we get to know the Lucas-Kanade algorithm for estimating the optical flow between two given images. Finally, we combine corner detection and optical flow estimation in one system, to perform feature tracking.

1 Harris Corner Detector

1.1 Question 1

In this section we implement a corner detection method based on the algorithm of Chris Harris and Mike Stephens [3]. We will test the Harris Corner Detector on the following two real-world photographs (see figure 1) and answer questions about its properties and performance.


(a)

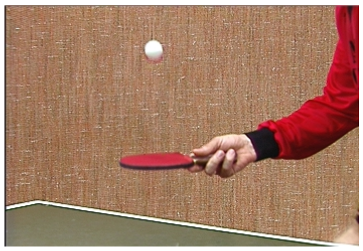

(b)

Figure 1: Images used for the Harris Corner Detection.

(1 & 2) In order to detect corners in a image we first calculate the image gradient in both the X and Y direction (I_x and I_y). To prevent our image gradient from being influenced by noise, we first smooth the image with a Gaussian filter. After obtaining I_x and I_y, we calculate I_x^2, I_y^2 and I_xI_y each of which we weight (convolve) with a window function. Here we use the same Gaussian filter as before. With that we arrive at the smoothed gradient matrices $A = g(I_x^2)$, $B = g(I_xI_y)$ and $C = g(I_y^2)$. We can use these to determine H, which gives us the *cornerness* for each point (x, y) in the image. We define,

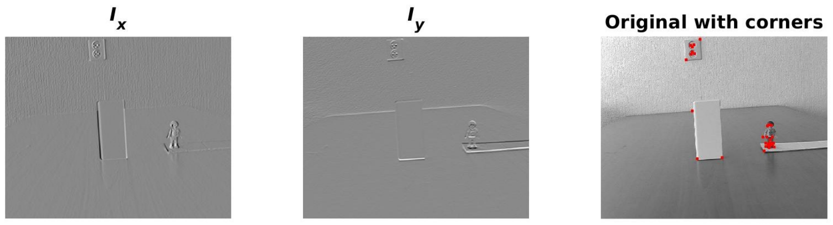
$$H = (AC - B^2) - 0.04(A + C)^2$$

where 0.04 is an empirical constant. The rationale behind this equation is that we use the gradient information in I_x^2, I_y^2 and I_xI_y to assess the magnitude of change around each pixel in the X and Y direction. This magnitude can be quantified by the eigenvalues of the following matrix:

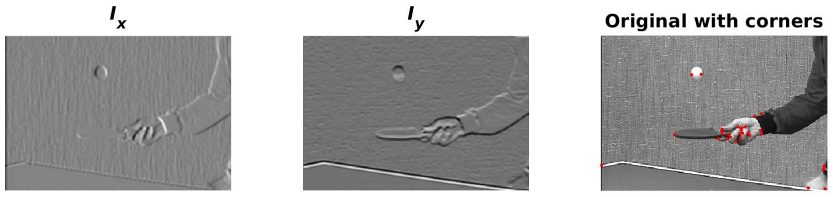
$$Q(x, y) = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

where we have such a matrix for each pixel (x, y). If both eigenvalues of $Q(x, y)$ are large, we have a change in both directions around this pixel and thus detect a possible corner. If only one eigenvalue is large, we assume an edge and if both eigenvalues are small, we assume a flat surface.

The local maxima of the matrix H that are greater than a given predetermined threshold are regarded as corner candidates. In order to determine unique corners, we want to ensure that the corner candidate is greater than its neighboring pixels (Non-maxima suppression). For this we define a neighborhood window of 3×3 pixels where the corner candidate is located in the middle. If a corner candidate is larger than each of its neighbors we consider it a corner.



(a)



(b)

Figure 2: The Harris Corner Detection algorithm applied on the sample images. For both images we see the image gradient in the X direction I_x and in the Y direction I_y. Finally, we see a grayscale version of the original image with the detected corners marked with red dots.

We then apply this procedure to our sample images (see figure 2). To obtain these results we empirically determine the parameters of the Gaussian filters and the value of the corner candidate threshold. After testing different configurations, we found that $\sigma = 1.5$ and a kernel size of 5×5 for both Gaussian filters works best for our purposes. Further, we use a corner candidate threshold of 0.005.

If we look at the values I_x and I_y for both images (figure 2a and 2b), we see that corner points at pixels are only detected if there is change in both the X and Y direction. We also note that in the image with the person playing ping pong it is harder to detect corners, as the background has a hatched structure. If we do not do enough Gaussian smoothing or set the threshold of the corner candidate to a value that is too small, we will recognize corners in the background even though it is a flat surface. This can be observed in figure 3. This is a good example of how confounding factors in the background can violate an algorithm's assumptions and thus negatively influence its performance.

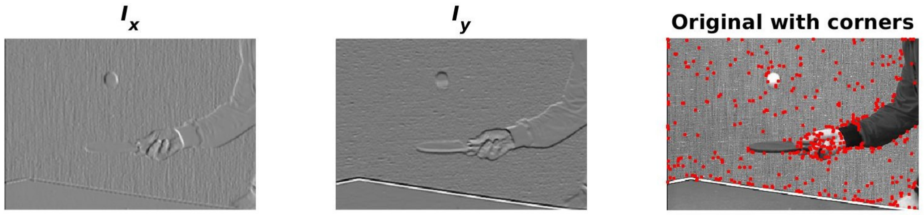


Figure 3: Illustration of falsely detected corners due to the hatched structure in the image background. The plot was created with a $\sigma = 1$, a kernel size of 3×3 and a corner candidate threshold of 0.01.

(3) To answer the question if Harris Corner Detection is rotation invariant, we rotate the image with the toy (see figure 1a) 45 and 90 degrees and apply the algorithm afterwards. When comparing the results in figure 4, we notice that in both images the detected corners are nearly at the same geometric locations. Although the exact number of corner points may not necessarily match those of the original image, we do not observe any additional noise or significant displacement of the corner points in the rotated images. These observations strongly suggest that Harris Corner Detection is invariant to rotation. Additionally, these observations are coherent with the rationale that changes in intensity around a pixel are not affected by the image's orientation. Therefore, Harris Corner Detection is able to detect corners regardless of how they are rotated. Another perspective is that the eigenvalues of each pixel's matrix Q is not affected by rotation, since the orientation is defined by the eigenvectors.

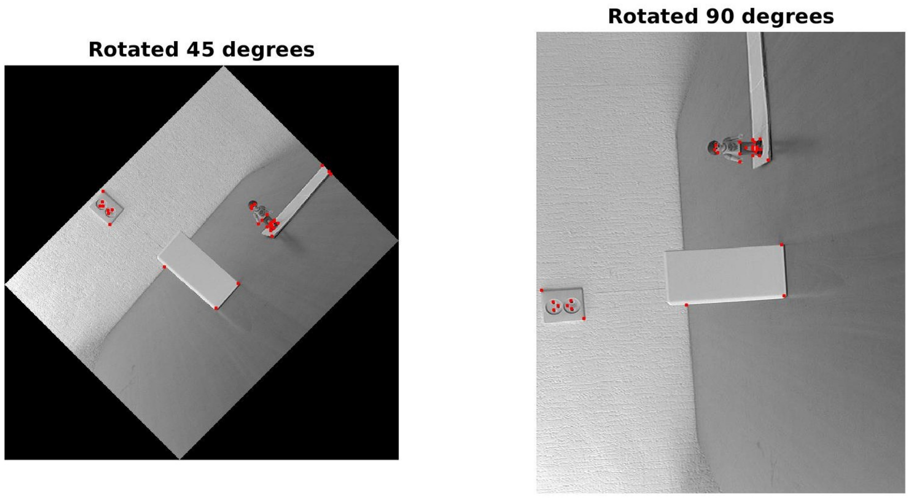


Figure 4: Depiction that the Harris Corner Detector rotation invariant. The left picture shows the detected corners after rotating the image 45 degrees counterclockwise. The right image shows the detected points after a rotation of 90 degrees counterclockwise. We observe that both images have corners detected at approximately the same locations.

1.2 Question 2

In this section we will compare the the Harris algorithm to a small modification called the Shi-Tomasi algorithm.

- The cornerness is defined by the smaller of both Eigenvalues for this point. The computation is simpler than the Harris cornerness. A corner is detected when a the value is above a certain threshold [5] [2].

$$\mathrm{H}_{\mathrm{ShiTomasi}} = \min\left(\lambda_1, \lambda_2\right)$$

$$\mathrm{H}_{\mathrm{Harris}} = \lambda_1 \lambda_2 - 0.04\left(\lambda_1 + \lambda_2\right)^2$$

- For the Harris cornerness we do not need to compute the Eigenvalues for each patch/pixel/point. Using the Shi-Tomasi modification we do need it. In cases where the first Eigenvalue is already below the threshold we do not need to compute the second Eigenvalue.

- Case 1: If both Eigenvalues are near 0 the smaller one will determine the cornerness and depending on the threshold it will probably be below and not be classified as a cornerpoint.
  Case 2: If one Eigenvalue is big and the second Eigenvalue is near zero, the cornerness is determined by the second one. Again depending on the threshold it will probably not be classified as cornerpoint. This would be the case of a line where a sharp change in direction of one axis is detected.
  Case 3: If both Eigenvalues are big it indicates a corner and if the smaller of both is above the threshold a cornerpoint will be detected. This is the case of a corner or edge where sharp changes occur in both axis.

2 Lucas-Kanade Algorithm

2.1 Question 1

(1 & 2) We implement the Lucas-Kanade algorithm as given by the task description. Therefore, we are operating on grayscale images. For calculating the derivatives I_x and I_y, we made use of Prewitt filters [6], which are a standard way to calculate a gradient with smoothing. These filters are defined as

$$p_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad p_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

and lead to the gradient estimates $I_x = p_x * I$ and $I_y = p_y * I$, where $*$ denotes a convolution. For estimating the temporal gradient I_t, we consider from assignment 1 that

$$I_t = \frac{\partial I(x,y,t)}{\partial t} = I(x,y,t+1) - I(x,y,t)$$

and hence we just subtract the second from the first image. With all the derivatives defined, we simply followed the task description and calculated the optical flow v with the least-squared estimate for each 15×15 pixels region.

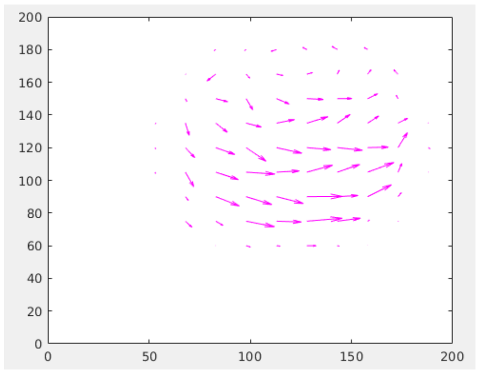
(3) The figures 5 and 6 display our results for the given test images. As demanded in the task description, we used the MatLab function *quiver* to visualize the optical flow.

2.2 Question 2

(1) The Horn-Schunck algorithm [4] is a method for estimating optical flow by performing a *global* optimization that takes all pixels of an image into account. The function that is optimized is defined as

$$\int\int (I_x V_x + I_y V_y + I_t)^2 + \lambda(||\nabla V_x||^2 + ||\nabla V_y||^2)\; dx\; dy \quad ,$$

where λ is a constant of the regularization (and therefore of the smoothness constraint). The first term is referred to as the *brightness constraint* (we also have that in Lucas-Kanade) and the second as the *smoothness constraint*, which causes the estimated flow to be as small as possible. Therefore, this algorithm finds the smoothest (i.e. smallest) flow over a whole image, that can account for the changes between two given images. In contrast, Lucas-Kanade operates on a *local* level. For estimating the optical flow within a pixel, it does only consider the changes in brightness of pixels within the surrounding window.


(a)

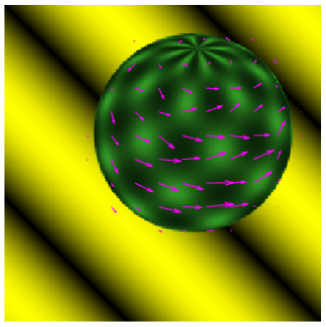
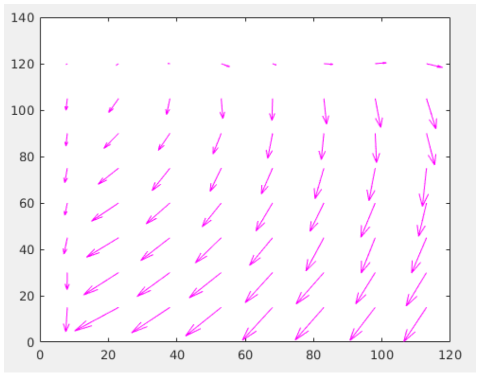

(b)

Figure 5: Optical Flow between the images *sphere1.ppm* and *sphere2.ppm*, estimated by the Lucas-Kanade algorithm. Figure 5a shows the estimated flow per image position and figure shows the flow within the input image.


(a)

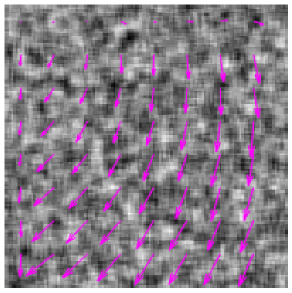

(b)

Figure 6: Optical Flow between the images *synth1.pgm* and *synth2.pgm*, estimated by the Lucas-Kanade algorithm. Figure 6a shows the estimated flow per image position and figure shows the flow within the input image.

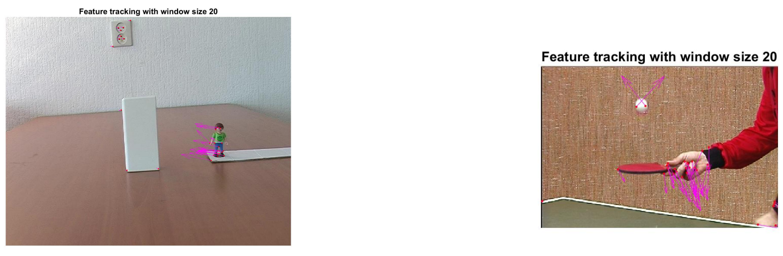
(2) The Lucas-Kanade algorithm fails to estimate the optical flow within flat regions, because it only considers a window of neighbouring pixels when estimating the optical flow in a point. If the whole window is on a flat surface, i.e. no pixel changes even though there is movement in the overall image, the image gradients vanish and the algorithm assigns no movement to the region [1]. Note that depending on the exact procedure of estimating I_x, I_y and I_t, all of their values can be 0 for such a flat region window. This would mean that Lucas-Kanade would actually fail, because the inverse of A is not specified. In Contrast, the Horn-Schunck method fills the flow information in flat regions from close regions that contain motion. It therefore results a dense flow field, but is also more sensitive to noise within an image and may display flow where actually none is present [4].

3 Feature Tracking

In this section we implemented a feature tracking algorithm which makes use of both Harris Corner Detection and the Lucas-Kanade Optical Flows.

The idea of this feature tracking algorithm, is that initially the corners detected with Harris Corner Detection. Over the next frames these initial features are followed using the Lucas-Kanade Optical

Flows. For each flow detected, the initial points are updated according a vector that represents the movement / flow of that feature. For each of these features the flow is calculated by taking a square around the position of the feature. The size of this square is defined by a variable window size, just as in the normal Lucas-Kanade method.


Feature tracking with window size 20
Feature tracking with window size 20

(a) (b)

Figure 7: First frames of tracking.

In figure 7 we can see the two initial images and their first flow directions. We can see the red dots, which are the initial discovered features, and in pink the enlarged directional vectors. We also created 2 GIF files[1] [2] which can be found in the results folder of our ZIP file. In these GIF files we can see that some points lose track of the actual corners they are following. For example the table tennis ball, moves too fast and is therefore difficult to follow.

We found out that the vectors that are produced were always very small, making the points barely move. To compensate for this, we multiplied these vectors with a value of 8. A lower value would make the points follow slow movements better, while a higher value made it possible to follow fast movements better. For the value of 8 we found a nice balance between these 2. For the case of ping-pong, a higher multiplying value would make it possible to track the table tennis ball better, but the points around the hand would not make sense. With the value of 8 the features on the moving hands seemed to follow its movements properly, however the tennis-ball was lost after a couple of frames.

As we mentioned earlier, the window size parameter can be chosen. We found that a window size of 20 pixels gave us the best results given the multiplier value of 8.

Overall one could also calculate for each frame new features given the Harris Corner Detection. However this process is computationally quite expensive, and fine-tuning for a specific picture often needs human observations. Also it's more difficult to connect new corners to the old ones, as they might not represent the same actual corner. With this just described feature tracking method, it do is possible to track movements of specific parts of the images over time. To reduce the computational load and human interference, one can also use the feature tracking method, in which at ones features are extracted and then followed using Lucas Kanade Optical Flows. However, as we showed, not always points are properly followed, and some points can get 'lost' overtime. Therefore we would suggest to perhaps find a combination of the two in which every now and then the features are recalculated to prevent losing all the features over time.

Conclusion

In part one we used the Harris algorithm to detect corners in images. We rebuild the algorithm and fine-tuned it to find the optimal threshold for corner-detection. Our result reaffirm that this algorithm is invariant to rotation. Further we investigated on a modification of this algorithm by Shi and Tomasi. It performs better but requires to compute the Eigenvalue decomposition of the image. Following, we estimated optical flow between two images by using the Lucas-Kanade algorithm. In comparison to the

[1] `https://github.com/3lLobo/ComputerVisionLab3/blob/master/CV1_A3/results/pingpong.gif`
[2] `https://github.com/3lLobo/ComputerVisionLab3/blob/master/CV1_A3/results/person_toy.gif`

Horn-Schunck algorithm, we saw that Lucas-Kanade operates only locally and may not assign optical flow to movement in flat regions.

We then introduced a feature tracking method which initially extracts features using the Harris corner detection, and then follows these features throughout following images using Lucas-Kanades Optical Flow method. We showed that in some cases these can properly follow corners over a picture, making it possible to follow objects in videos. However, as soon as there are both fast and very slow moving objects, some features can get 'lost' on the long-term. We would suggest to perhaps find a way to every now and then perform a new corner detection to keep the objects up-to date with the actual imagery.

References

[1] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005.

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

[3] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[4] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[5] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.

[6] Judith MS Prewitt. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970.