Computer Vision 1 - Lab 3

Pauliuc, Andrei-Sebastian
11596619

van Slooten, Renzo
11129778

1 Introduction

An equally important objective in computer vision is the ability to correctly detect corners, feature points, and furthermore, track these along multiple frames using optical flow analysis. The corner detection part is helpful in identifying points worth to be tracked and points important for image stitching, while the optical flow methods provide indication of how objects have moved in two consecutive images. This assignment starts with a corner detection algorithm using the famous Harris Corner Detector. After that, we learn how to detect the optical flow of pairs of two images using the Lucas-Kanade algorithm. In the end, these two techniques are brought together and used in identifying and tracking feature points in two sequences of images.

2 Harris Corner Detector

Question - 1

The Harris Corner Detector function outputs the cornerness matrix H(x,y), which holds values for the amount of cornerness of each pixel. The H matrix can be obtained by the auto-correlation matrix Q. This can be obtained by convolution of the image with the horizontal and vertical derivatives of a Gaussian [1] (see figure 1).

$$Q(x,y) = \sum_{(u,v)\in W(x,y)} \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

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

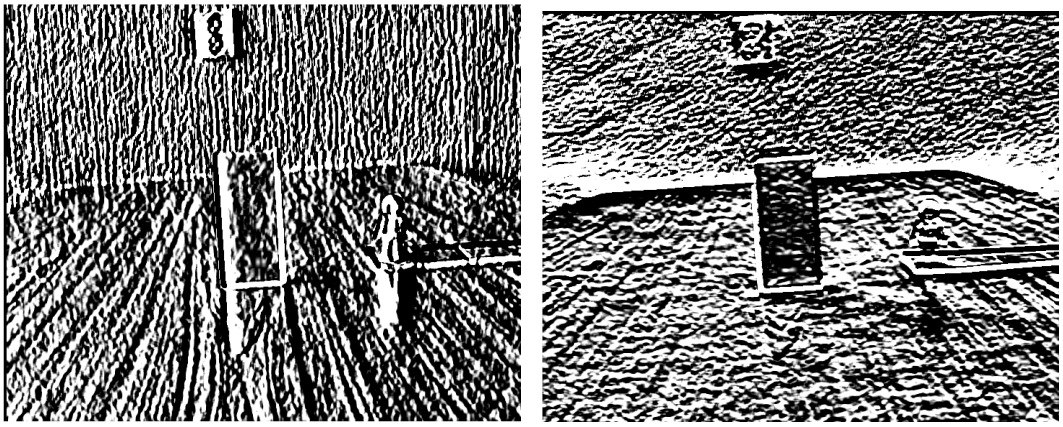

Figure 1: Harris Corner Detector: (left) image derivative Ix, (right) image derivative Iy

Since only the local maxima of H are corner points, the H matrix needs to be checked if it is greater than all its neighbours (in an n x n window centered around this point). This is done with non-max

suppression. Values which are below the user-defined threshold should not be seen as corners, so also these values are suppressed.

The result can be tuned by changing the sigma of the Gaussian kernel, the size of the non-max suppression window or the threshold.
Sigma: only the sharper corner will be detected using a smaller sigma.
Window size: the smaller the window size the more corners are allowed in a region.
Threshold: only well defined corners are detected using a bigger threshold.

Playing with these parameter with the person toy and ping pong images, we found the following parameters to be (visually) the best (see table 1, left image in figure 2, and figure 3).

	Sigma	Window size	threshold
person toy	3	14	16000
ping pong	2	10	35000

Table 1: Final parameters selection

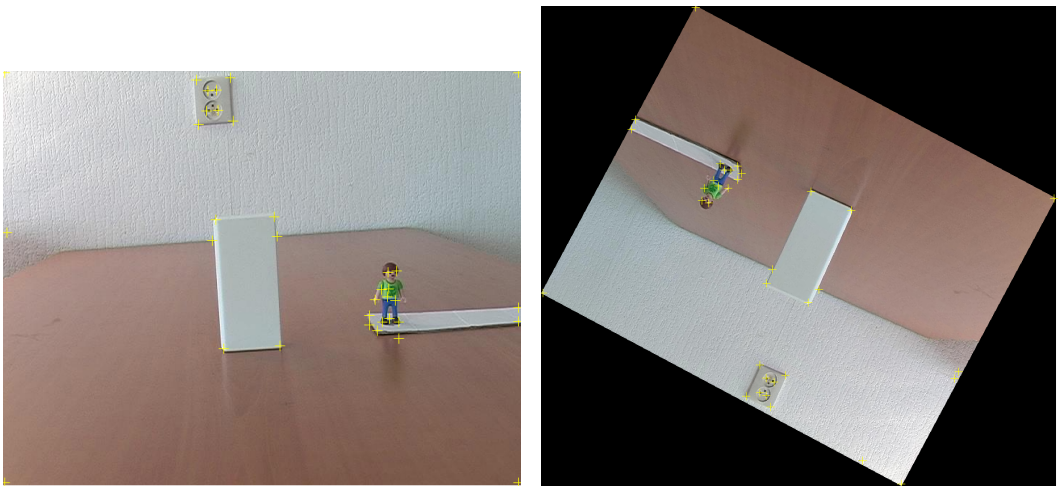

Figure 2: Images with corner points: (left) Original, (right) Rotated

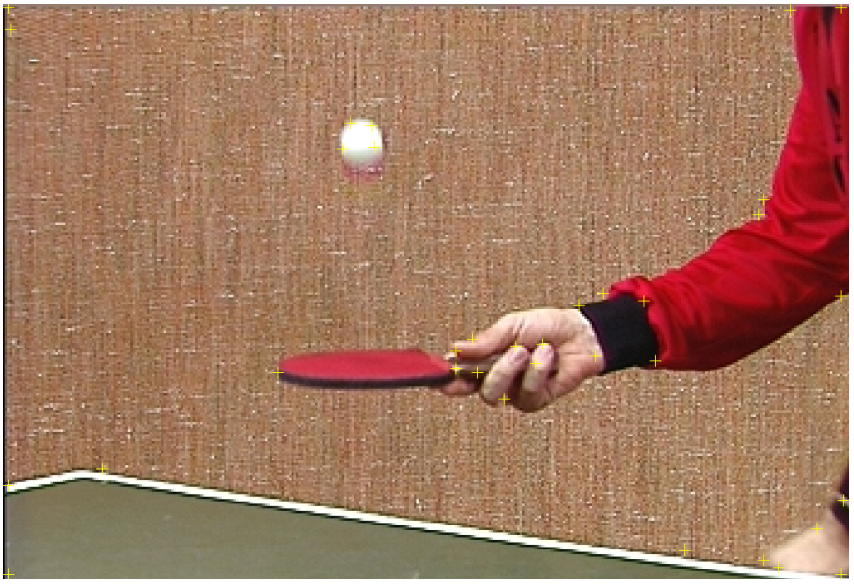

Figure 3: Harris Corner Detector: Original ping pong image with corner points

We can visualise Q as an ellipse with lengths and rotations of axis following from the eigenvalue analysis of the auto-correlation matrix. If the image rotates, the lengths of the axis remains the same, but the ellipse rotates in the same way as the image. This means that the cornerness matrix H is invariant to image rotation. Looking at the right image in figure 2, we see that this holds.

Question - 2

Shi and Tomasi look at another way to find keypoints. Instead of only looking at the minimum eigenvalue λ_0, they proposed a feature selection criterion that is optimal by construction because it is based on how the tracker works. By minimising the dissimilarity function and linearizing the resulting system they get to following formula: T**z = a** where **z** collects the entries of the deformation, **a** the error function and T a 6x6 matrix:

$$T = \int \int_w \begin{bmatrix} U & V \\ V^T & H \end{bmatrix} w d\mathbf{x}$$

$$\text{where } H = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The eigenvalues of the H matrix say something about the textures within a window. So you need to calculate the eigen decomposition of the patches. If both eigenvalues are near 0, the intensity within a window is roughly constant. If one eigenvalue is big and the other is near zero, the texture pattern in a window is unidirectional (lines instead of corners). If both eigenvalues are large, the window represents corners, salt-and-pepper textures, or any other pattern that can be tracked reliably [2]. So in conclusion, if both eigenvalues of Z are bigger than a predefined threshold, the window is accepted.

3 Optical Flow with Lucas-Kanade Algorithm

Question - 1

The Lucas-Kanade algorithm for optical flow is implemented in the file **lucas_kanade.m**; a demo script is included in the file **lucas_kanade_demo.m** which displays the optical flow for the two given pairs of images. Using the steps provided as guidance, the input images are first split into blocks of 15×15 pixels, where we register the (x,y) coordinates of the central pixels. After computing the I_x, I_y and I_t derivatives of the input images corresponding to the (x, y, t) directions, we compute the variables A, A^T and b for each region. These variables use all the pixels in a region. Following the Lucas-Kanade method, the optical flow of each region is computed using equation 1.

$$v = \left(A^T A\right)^{-1} A^T b \tag{1}$$

Running the algorithm on the sphere and synth pairs of images, the resulting optical flows are displayed with red arrows in figure 4.

Question - 2

Operating scale of optical flow algorithms

Working only with information from the neighbouring pixels in a region, the Lucas-Kanade method operates on a purely local scale. This makes it more robust to noise than other point-wise methods, because a larger window diminishes the effect of noise. Because of this local analysis of pixels, Lucas-Kanade method works under the assumption that the motion is small. However, the method has some downsides. The assumption of small motion is violated in cases where objects move fast. The effect of this violation is observed in one of the examples given for the feature tracking algorithm, discussion which takes place in section 4. Another issue of the algorithm is presented by the uniform (flat) regions; this problem is discussed in the following section.

Unlike Lucas-Kanade method, the Horn–Schunck method is estimating the optical flow using global solutions. One of the assumptions of this method is that flow in images is continuous, putting more

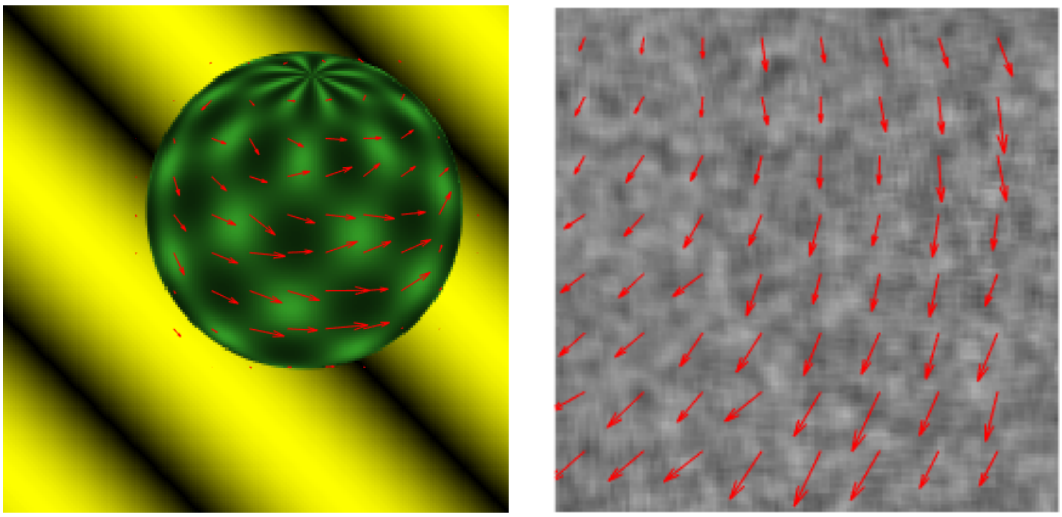

Figure 4: Optical flows using Lucas-Kanade method: (left) sphere image, (right) synth image.

focus on smoothness. In this method, the main objective is to minimise a "global energy function", defined by equation 2.

$$E = \iint \left[(I_x u + I_y v + I_t)^2 + \alpha^2 (\|\nabla u\|^2 + \|\nabla v\|^2) \right] \mathrm{d}x\mathrm{d}y \tag{2}$$

On the one hand, the Horn-Schunck method provides advantages over Lucas-Kanade in solving the aperture problem because of the global character of the method. The aperture problem, as seen in figure 5, represents a problem when trying to detect the direction of motion of a contour. If we look only at the circle surrounded by the white box, the perception of motion in all three situations is the same. The correct way to determine the optical flow of the lines is only when using information based on the whole image. On the other hand, the disadvantage posed by it is the sensitiveness to noise.

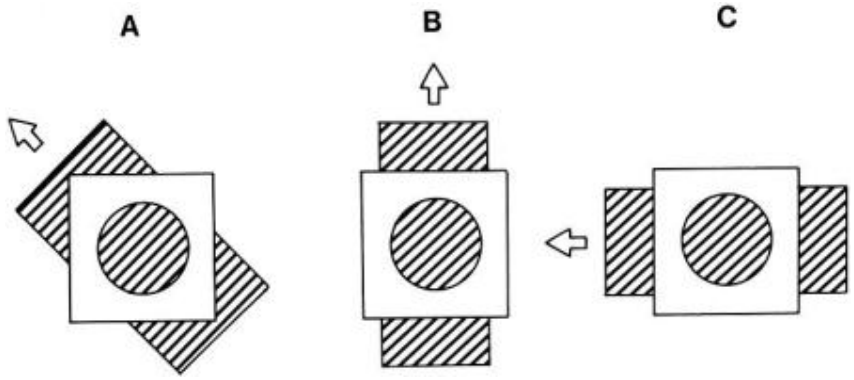

Figure 5: Illustration of the aperture problem

Behaviour on flat regions

Lucas-Kanade method is prone to errors on the flat regions of an image. The algorithm solely relies on the image derivatives in the (x, y, t) directions, and thus, it cannot detect flow in regions where the derivatives are zero. Since the flat regions present zero derivatives, they pose problems in detecting optical flow. The Horn–Schunck method provides a solution to the flat regions precisely because it works on global scale. The flat regions are not analysed separately from the rest of the image, which provides an advantage to detecting optical flow in them.

4 Feature Tracking

Question - 1

The feature tracking algorithm is implemented as a function in file **tracking.m**. The code reads the sequence of images given in a folder, and processes them as follows. Using the first image, the coordinates of the feature points are found using the Harris Corner Detector method implemented before. Starting with the second picture, the Lucas-Kanade algorithm is used to detect optical flow around the feature points, while also updating their (x, y) coordinates. The initial corners and each optical flow between two consecutive images is displayed on screen. We also save the results to a GIF file for easy visualisation afterwards. The results of the algorithm can be found in folder **results_tracking**: the file *pingpong.gif* for the *pingpong* sample, *person_toy.gif* for the *person_toy* sample. A demo script is also included, which can be found in the file **tracking_demo.m**.

An interesting observation to be made regarding the feature tracking algorithm is the result of the *pingpong* example. As we can see in the resulting frames, the algorithm is not really precise at tracking the ball itself. While the hand, the person and the whole background are correctly tracked to some extent, there are big problems regarding the position and the optical flow for the ball. We believe that this example is a clear violation of the assumption of the Lucas-Kanade method, which states that the movement in the images should be small. Since the ball moves fast between frames and changes positions drastically (up-down-up-down), the algorithm cannot detect the real motion and is stuck at finding the right results. A possible fix for this would be to reduce the resolution of the image (through blurring or rescaling) to a certain point where the movement of the ball between frames becomes small enough, and then apply the Lucas-Kanade algorithm.

Question - 2

Using feature tracking (and mainly optical flow), we can detect the movement of a specific patch of pixels (or points), and can be sure that the object we are tracking has moved indeed. Of course we can detect feature points for every frame, but the problem is that there is no way to map those points of interest between them. Furthermore, if the second image presents us with more feature points, we do not know where they come from; if it has less feature points, how do we know which objects left the frame or what movement caused this result? The only way to be partially aware of what objects have moved, left or entered the frame, we would need to use feature tracking algorithms.

5 Conclusion

The conclusion is that, if it is used with the right parameters, the Harris Corner Detector method is a good algorithm to detect feature points such as corners. The results and math show that the method is invariant to image rotations. Shi and Tomasi [2] extended on the method, which allowed for affine motion fields and even better tracking of features using the tracker itself to find the feature windows which can be tracked reliably. The optical flow analysis using Lucas-Kanade method presents us with a good way of detecting motion using local information. As we have seen, while it can correctly compute flow in pairs of two images, there are some problems when using multiple frames, with varied motion speed. Also, fast movement and big changes between frames cannot be correctly tracked with this method. Combining the two aforementioned algorithms, we have developed the ability to identify interest points and track them in consecutive frames, taking a step further in the vast domain of computer vision.

References

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

[2] Jianbo Shi. Good features to track jianbo shi carlo tomasi computer science department computer science department cornell university stanford university ithaca, ny 14853 stanford, ca 94305. *Computer Science Department Cornell University and Stanford University*, 1994.