# Computer Vision 1 - Lab 3 Report

**Harris Corner Detector  Optical Flow**

**Michelle Appel (10170359)**

**Nils Hulzebosch (10749411)**

GitHub repository: `https://github.com/MichelleAppel/CV1_ass3`

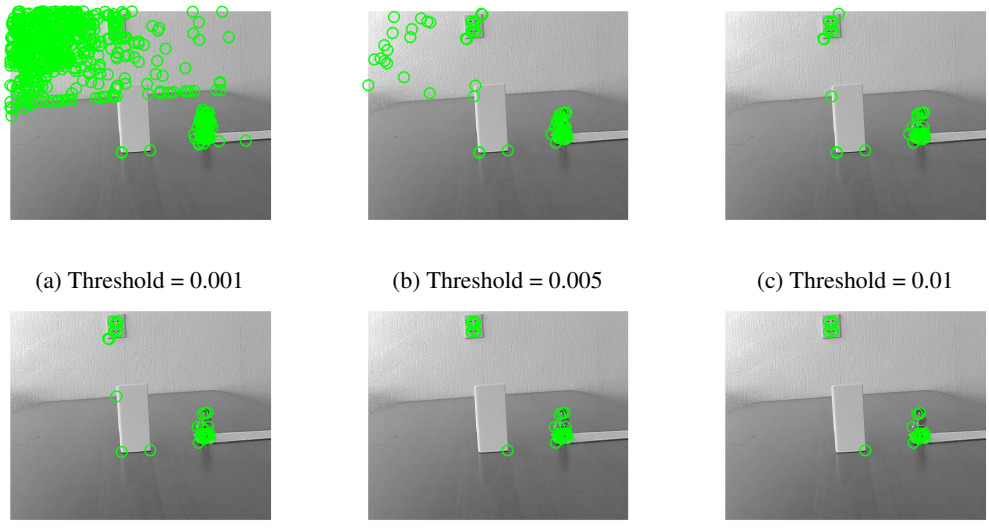
## 1 Introduction

In this report, two algorithms are examined: the Harris Corner Detector, used for detecting corners in an image, and the Lucas-Kanade Algorithm, used for detecting optical flow (movement) between images. Lastly, these methods are combined to track the movement of features (interest points) in a series of images.

## 2 Harris corner detection

### 2.1 Question 1

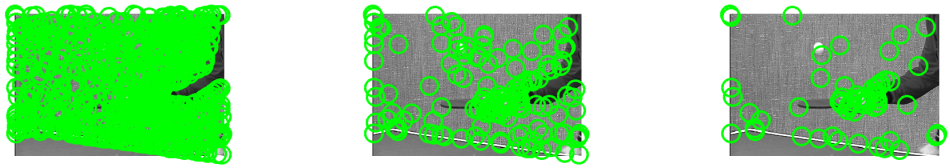
In this section, the results of the Harris Corner Detection algorithm are discussed. Figure 1 shows the result for sample image *person_toy/00000001* and Figure 2 for sample image *pingpong/0000*. Both images are tested with different threshold values, and different standard deviations for the Gaussian kernel.



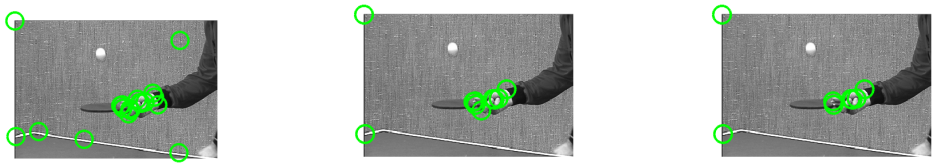
(a) Threshold = 0.001 (b) Threshold = 0.005 (c) Threshold = 0.01

(d) Threshold = 0.02 (e) Threshold = 0.03 (f) Threshold = 0.05

Figure 1: Detected corners for the person toy image using different threshold values.



(a) Threshold = 0.001 (b) Threshold = 0.005 (c) Threshold = 0.01



(d) Threshold = 0.02 (e) Threshold = 0.03 (f) Threshold = 0.05

Figure 2: Detected corners for the pingpong image using different threshold values.

In the following part, the robustness of the algorithm is tested by randomly rotating an image and applying the algorithm. The results for this are shown in Figure 3, using sample image *person_toy/00000001* and six random rotations. The threshold value is set to 0.01, since this seems the best value according to the results in Figure 1.

Furthermore, for the rotation three different methods are used: nearest, bilinear, and bicubic. As discussed in the documentation[1], the methods are the following:

- nearest: take the nearest pixel value
- bilinear: take the weighted average of pixel values in a 2x2 neighbourhood
- bicubic: take the weighted average of pixel values in a 4x4 neighbourhood

When comparing the detected corners of the rotated images and the normal images, it is clear that the algorithm is not invariant to rotation. The amount and location of detected corners differ between rotated images. Furthermore, they also differ between rotation method. As shown, the bilinear and bicubic methods greatly outperform the nearest method, which is intuitive, because the output pixels are smoothed, which results in less abrupt transitions, which results in less false positives (detected corners).

The variance of the algorithm could be explained by the fact that the gradients in the x and y direction become weaker when the image is rotated, because the angle is not parallel to the axes. Ideally, this should be captured by the diagonals in the H matrix, but these appear to be imperfect for this task.

### 2.2 Question 2

The cornerness, as defined by Shi & Tomasi (1993), is the following:

$$H = min(\lambda_1, \lambda_2) \tag{1}$$

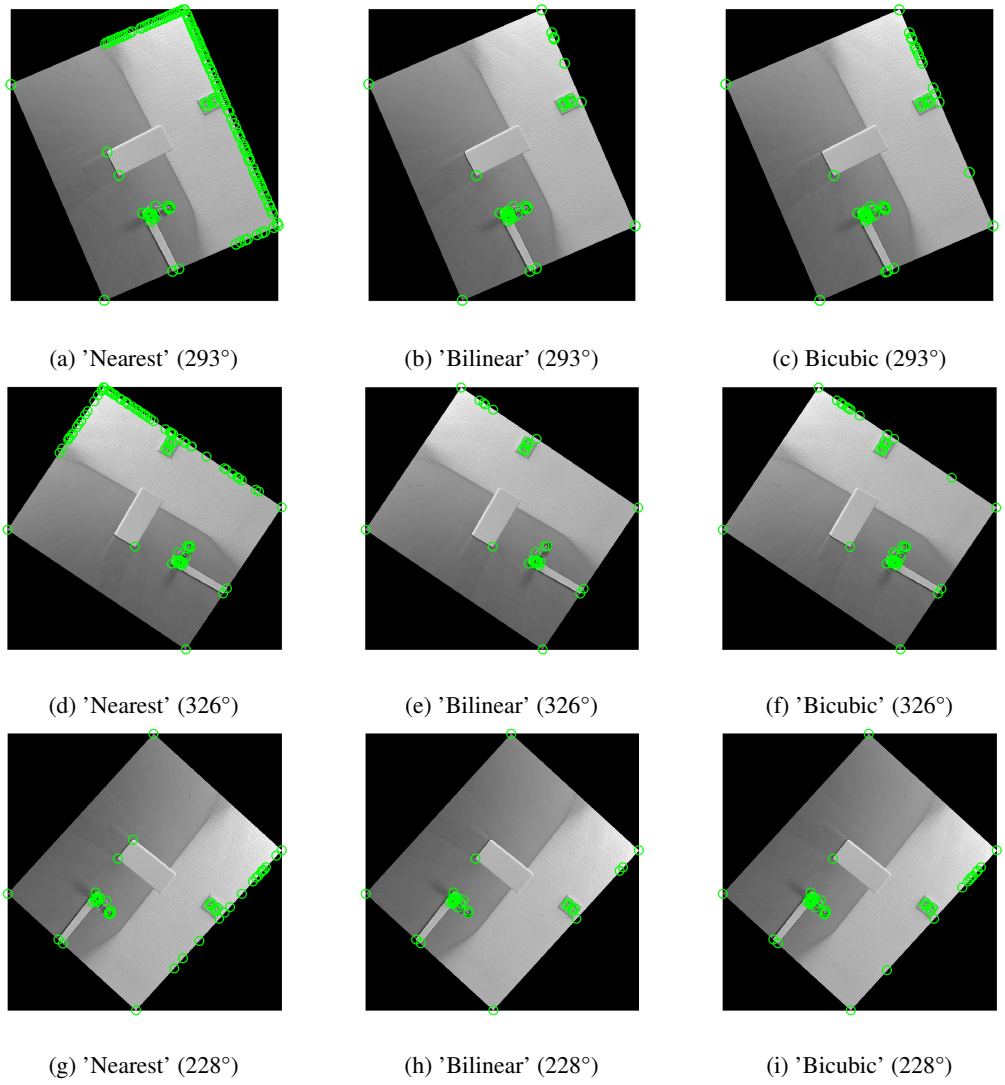
Here, $\lambda_1$ and $\lambda_2$ are the two eigenvalues of the matrix Q. Again, if the value H is greater than a given threshold, it is identified as a corner.

It is required to calculate the eigendecomposition of the patches, and use it to calculate the local maxima of the patches to determine the corner in that patch.

For three test scenario's, the relative cornerness values are given, using equation 1:

- Scenario 1: both eigenvalues are near 0 (intuitively speaking, there are no edges detected). This means that the resulting cornerness is also near 0, which is probably lower than the defined treshold, meaning it is not identified as corner.

[1] `https://nl.mathworks.com/help/images/ref/imrotate.html`



(a) 'Nearest' (293°) (b) 'Bilinear' (293°) (c) Bicubic (293°)

(d) 'Nearest' (326°) (e) 'Bilinear' (326°) (f) 'Bicubic' (326°)

(g) 'Nearest' (228°) (h) 'Bilinear' (228°) (i) 'Bicubic' (228°)

Figure 3: Detected corners for the person toy image using different angles and rotation methods.

- Scenario 2: one eigenvalue is big and the other is near zero (intuitively speaking, there is one edge detected). This means that the resulting cornerness is also near 0, which is probably lower than the defined treshold, meaning it is not identified as corner.
- Scenario 3: both eigenvalues are big (intuitively speaking, there are two edges detected). This means that the resulting cornerness is also big, which is probably higher than the defined treshold, meaning it is identified as corner.

## 3 Optical Flow with Lucas-Kanade Algorithm

The motion flow of some point of interest within an image can be estimated using the Lucas-Kanade algorithm, where some region around the point is taken, the gradients within x and y direction and the gradient over time of that region are computed, and used it to solve the motion vector.

### 3.1 Question 1

The algorithm divides the image in non-overlapping regions which have a default height and width of 15 pixels. For each region of the first image, the gradient in x and y direction is computed. Also, the gradient of t is computed, which is the difference between the two corresponding regions of the two images. Then, all gradients are multiplied by a Gaussian kernel, so that the center of the region weighs heavier for the calculation than the edges. Then the matrix $\mathbf{A}$ and vector $b$ are computed, and used to solve the motion vector $v = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T b$.
The use of different deviations for the Gaussian kernel that is used to multiply the gradients with has led to difference in performance. As can be seen in the figures 4 and 5, the larger the deviation, the more uniform behaviour the motion vectors show. This different from the expectation, because a sigma that goes to infinity approximates a equally distributed kernel, which would indicate that multiplying with a Gaussian kernel is redundant. However, when skipping the multiplication, the motion vectors seem to behave capricious. This indicates that the improvement in performance is a result of the normalization that is being done by the multiplication, as all elements of the Gaussian kernel sum to one.

### 3.2 Question 2

The Horn-Schunck method assumes a global smoothness in the flow over the whole image, where capriciousness of the vectors is being minimized to some extend. On flat regions, they perform well, since there is a high density of flow vectors.

## 4 Feature Tracking

In this section, the two methods are combined to detect the movement of feature points over time.

### 4.1 Question 1

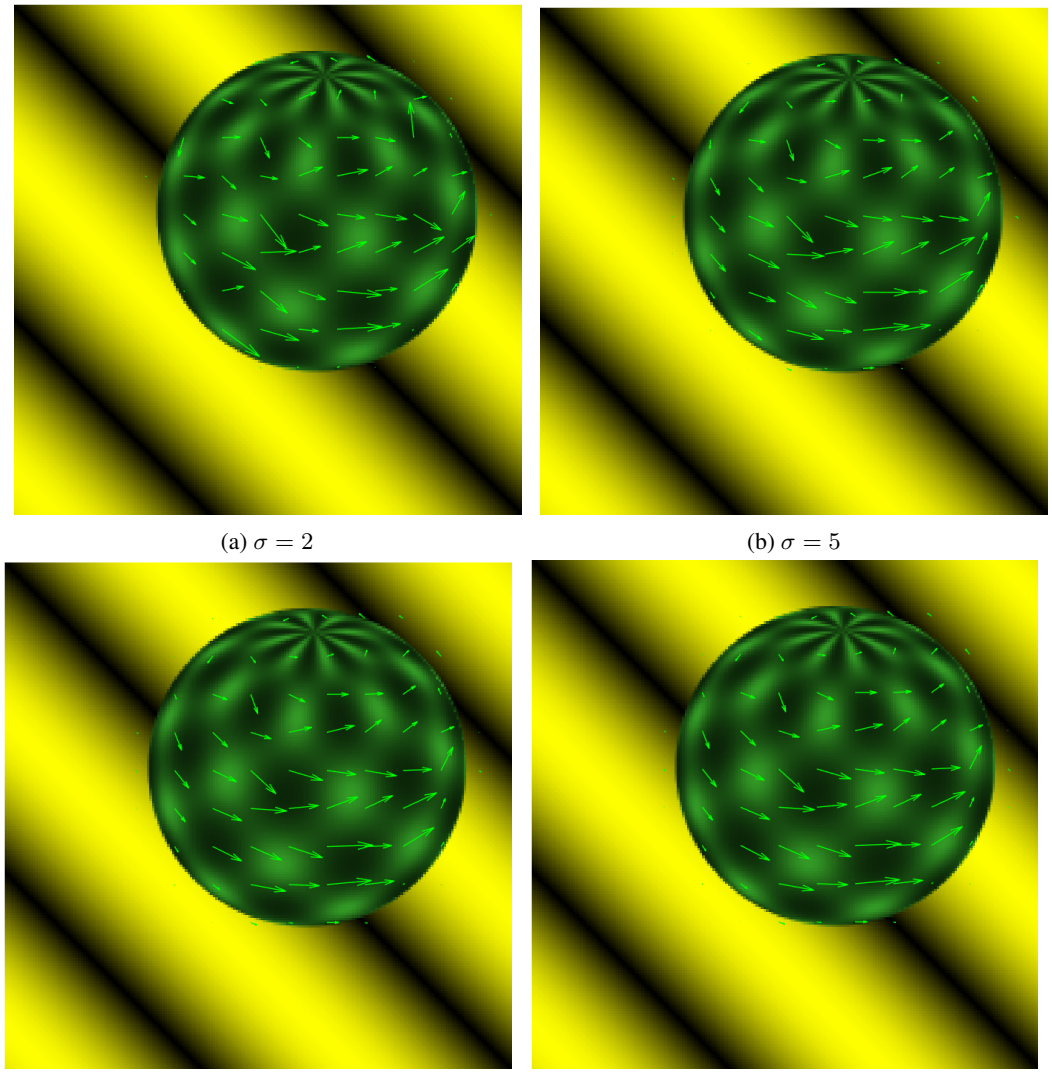
The points of interest are determined by searching for corners using the Harris corner detection algorithm, where the assumption is made of corners as interesting feature points as corners often enclose object.

Subsequently, the Lucas-Kanade algorithm is used to keep track of the feature points. The feature point coordinates are updated each frame and are the result of the flow vector added to the previous feature point coordinate. When just adding the flow vector, the indicated feature points seemed to get behind.

At first, an indication of the underlying problem seemed that the points are moving too fast to keep track of as they might move outside of the region within one frame, so a solution would be to enlarge the regions around the points of interest. However, this did not led to better performance and the points of interest still where behind. An less desirable solution is used, which is an empirically determined multiplication of 11 with the flow vector to compensate for the points of interest moving too slow.

### 4.2 Question 2

Feature tracking is used to keep track of movement of points of interest which are initially determined. Detecting features for each and every frame does not provide links between the features in the frames, giving no information about the movement of feature points as it is unknown which feature point



(a) $\sigma = 2$ (b) $\sigma = 5$

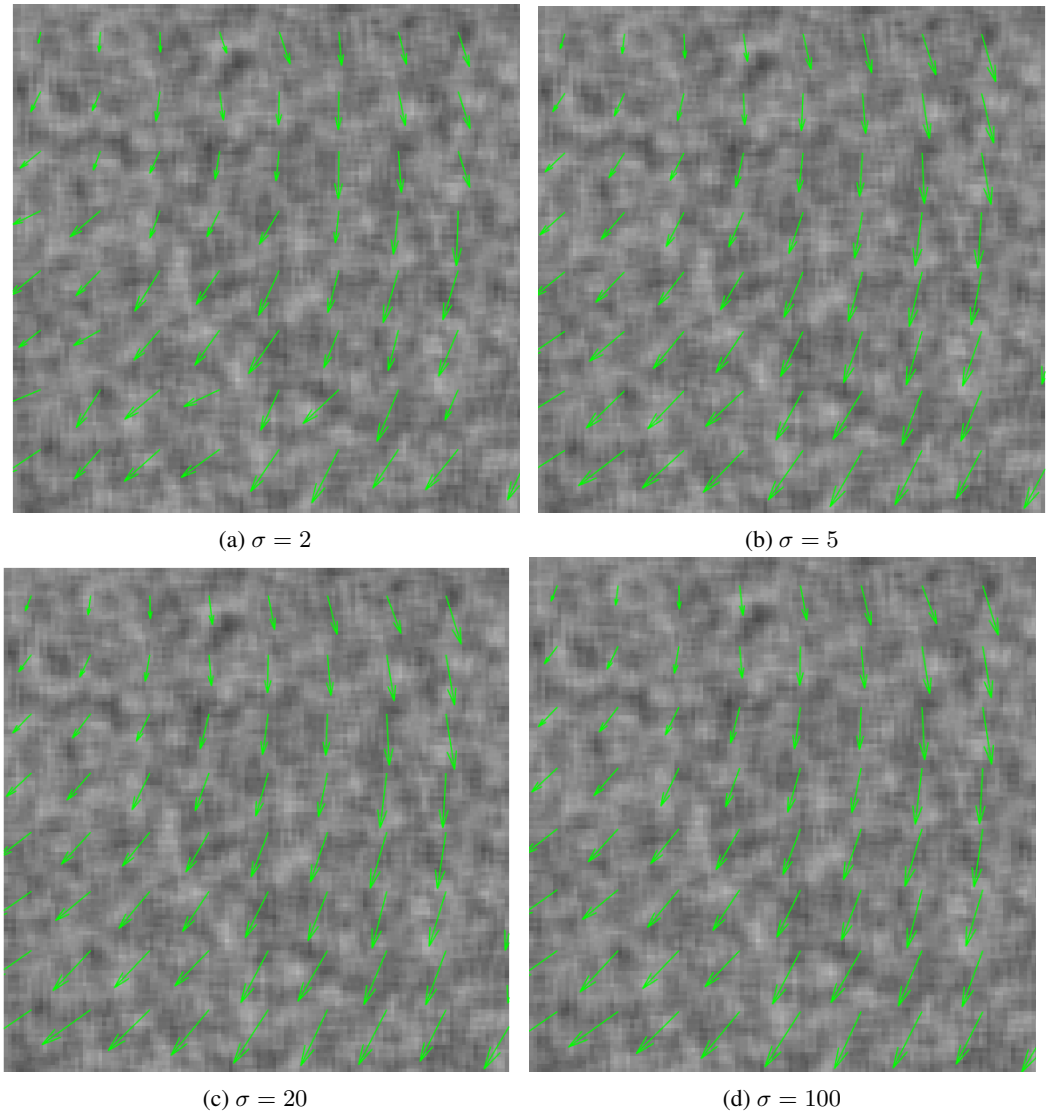
(c) $\sigma = 20$ (d) $\sigma = 100$

Figure 4: Computed motion vectors from the *sphere1.ppm* and *sphere2.ppm* images, using different deviation for the Gaussian filter.

correspond with which other feature point in the next frame. Also, more feature points might appear within later frames.

The result of the person_toy image set can be viewed at `https://github.com/MichelleAppel/CV1_ass3/blob/master/output/tracking/person_toy.gif`.

The result of the pingpong image set can be viewed at `https://github.com/MichelleAppel/CV1_ass3/blob/master/output/tracking/pingpong.gif`.



(a) $\sigma = 2$ (b) $\sigma = 5$

(c) $\sigma = 20$ (d) $\sigma = 100$

Figure 5: Computed motion vectors from the *synth1.pgm* and *synth2.pgm* images, using different deviation for the Gaussian filter.

## 5 Conclusion

As discussed in this report, Harris Corner Detection can be used to detected corners in images. The sensitivity of detection was measured using several threshold values. For two sample images, six values were tested. For the toy example, a value of $0.01$ seemed to be best, while for the pingpong example, a value of 0.02 seemed to be best. As shown, the algorithm is variant to rotation, which is probably due to the fact that the diagonals of the H matrix cannot detect the corners perfectly. For rotation, methods that take the weighted average of a neighbourhood for a pixel value outperform the method that takes only the nearest pixel value.

In the second part, optical flow was examined. Using the Lucas-Kanade algorithm, the optical flow was measured by looking at the differences in the x direction, y direction, and time (between two images), along with applying a multiplication with a Gaussian kernel. As shown, the larger the standard deviation of the Gaussian kernel, the better the performance, which indicates that actually equal normalization of the gradients leads to the improved results.

The final section describes a method for feature tracking, which combines the first method of Harris corner detection and the second method of Lucas-Kanade motion flow determination. The new coordinates of the feature points are a product of the motion vector added to the previous feature point, however this led to the feature point getting behind. This is solved by multiplying the motion vectors by 11, which is an empirically determined value.

## 6 Literature

Horn, B.K.P. & Schunck, B.G. (1981). Determining optical flow. Artificial Intelligence, vol 17, 185–203.

Shi, J. (1994, June). Good features to track. In Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on (pp. 593-600). IEEE.