# COMPUTER VISION 1
# ASSIGNMENT 3

MAURITS BLEEKER (10694439) & JÖRG SANDER (10881530)

6 March 2016

## 1 QUESTION 1 – HARRIS CORNER DETECTOR

Our implementation of the *Harris Corner Detector* resulted in a MATLAB function *HarrisCornerDetector*. The results for the delivered images *person toy* respectively *pingpong* are given in figure 2 and 3.

We tested our function initially with a "simple" checkerboard pattern in order to verify the detected corner points (please see figure 1). Frankly we think the results for the checkerboard were of a better quality than the results we keep getting for the other two images. With the *bare eye* it is difficult to see why especially in the *pingpong* images some points are detected as corners. Presumably these are due to *noise* and could have been avoided if more smoothing would have been applied to the images.

We experimented quite a while with different scales ($\sigma$'s), thresholds (for non maximal suppression) and derivatives. Finally we convolved the images with a two dimensional first order Gaussian derivative. We implemented the kernel length as a function of the scale ($\sigma$). For details please see the MATLAB code. We discarded detected corner points that were too close (half of the kernel size) to the images boundaries.

Zooming into the images one gets the idea that the detected corner points are just slightly positioned next to the "corner" where one would expect them. We surmise that this effect is due to the performed smoothing (especially the size of the scale) and non maximal suppression.

## 2 QUESTION 2 – LUCAS-KANADE ALGORITHM

Our implementation of the *Lucas-Kanade Algorithm* resulted in a MATLAB function *LucasKanadeAlgorithm*.

For the Lucas Kanade Algorithm we used a window size of 15 as stated in the exercise.

For the calculation of $I_x$, $I_y$ and $I_t$ we used two different kind of derivatives. We stared off with a 1-D Gaussian derivative (one for the x and one for the y direction), with a sigma of 1. For $I_t$ we used the absolute difference in pixel value per point in image 1 an 2. However, the optical flow results when using this Gaussian where not optimal (See figure 4 and 5). The directions of the velocity field are too divers for the presumed motion. We experimented a lot with different sigmas but none of them resulted in (significantly) better results.

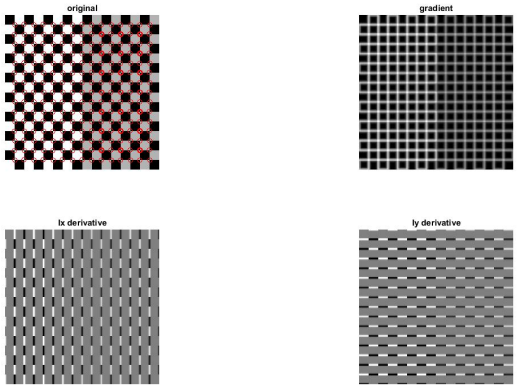


**Figure 1:** Test of Harris Corner Detector on simple checkerboard pattern

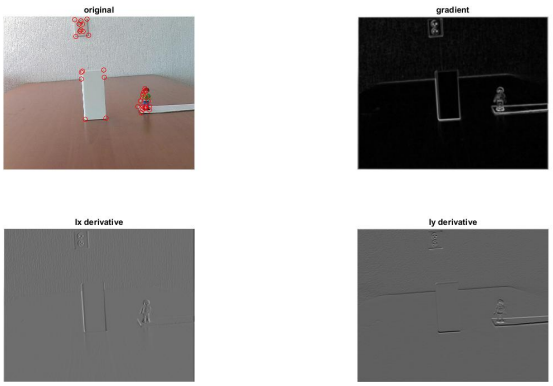


**Figure 2:** Detected Corner points with Harris Corner Detector $\sigma = 1.4$, window size= 15, threshold= 3.5

Therefor we tried another variation for the intensity derivatives. We calculated $I_x, I_y and I_t$ using the following filters respectively.

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{3}$$

In 7 and 6 the optical flow is visualised when using these type of filters instead of the Gaussian derivatives. Finally the velocity vectors have a homogeneous direction.

It is not obvious to us why these latter derivative filters result in a much better optical flow then using Gaussian derivatives. In our opinion Gaussian derivatives must work as well. Unfortunately we cannot find a proper explanation for this at the moment.

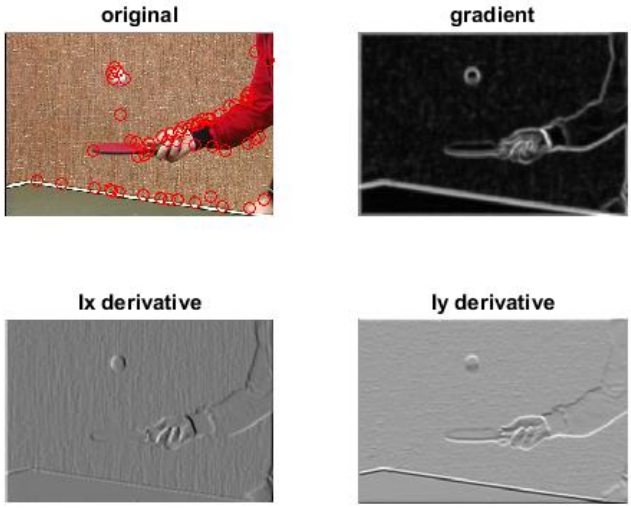


**Figure 3:** Detected Corner points with Harris Corner Detector $\sigma = 1.4$, window size= 15, threshold= 3.5

## 3 QUESTION 3 – TRACKING

We reused the previous developed *Harris Corner Detector* to identify the points of interest (aka *corner points*) that were tracked through the series of images for the *pingpong* and *person toy* examples.

The corner points were computed once for the first image. In each subsequent step the displacement vector was computed for each corner point by means of the Lucas-Kanade algorithm using the previous and "current" image (main MATLAB function *createOpticalFlow*).

In order to detect the corners we used the following parameter values: $\sigma = 1.4$, window/patch size $= 15$, threshold (non-maximal suppression $= 3.5$. For each corner point we created a patch window of pixel size 15x15 in order to calculate the displacement vector V by means of a least square solution. We have to admit that the results visualized in the attached video's are not optimal.

We visualized the subsequent velocity fields on top of the original images in order to analyze the calculated vectors in the neighborhood of the corner points. We tried different smoothing methods and derivative filters unfortunately without any significant success. For some of the interest points it seemed to be obvious that they were hard to track due to a lack of contrast with the background.

Especially for the *pingpong* images the tracking algorithm delivered poor results w.r.t. the tracking of the *ball*. We presume that this is due to the fact that the movements of the ball are fast and likewise changes between the subsequent images are rather big which makes them difficult to track with the *optical flow* method.

The results for the *person toy* images are slightly better. But some of the corner points are left behind during the movement of the person toy (this means that after a while some of the corner points are not moving along with the person toy anymore). This is due the fact that the optical flow vectors are really small for some of the the corner points while the person

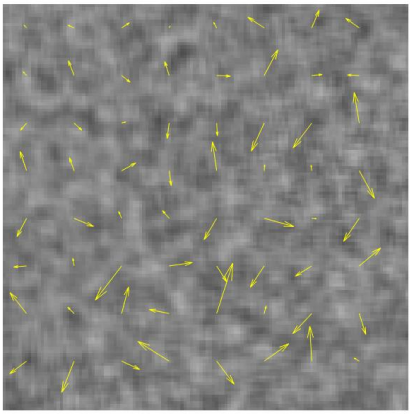

**Figure 4:** Result of calculated displacement vectors with Lucas-Kanade Algorithm using a Gaussian filter

toy is moving quite a lot. Therefor these corner points get behind the *original* points and no further movements will be detected for that point. We were not able to solve this problem.

We think that this phenomena is due to 'noise' in the pixel values. Even though the *person toy* is moving it is not possible to detect this in the pixel values of the two images. Hence some the corner points get *behind* the moving person toy. See the attached videos for detailed results.

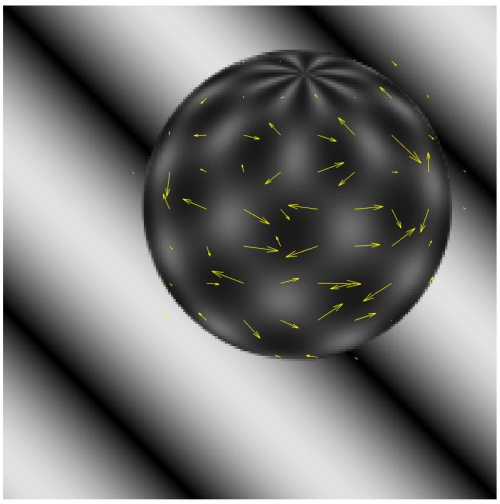

**Figure 5:** Result of calculated displacement vectors with Lucas-Kanade Algorithm using a Gaussian filter

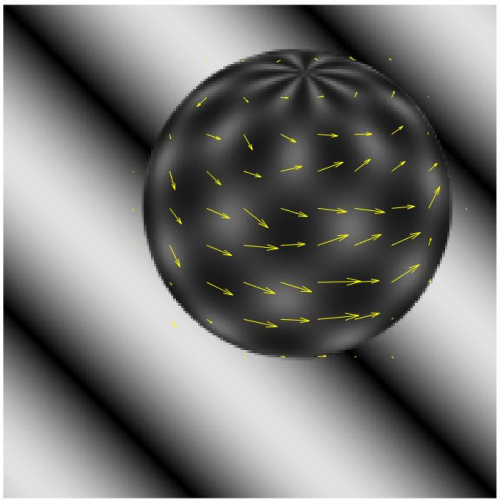

**Figure 6:** Result of calculated displacement vectors with Lucas-Kanade Algorithm

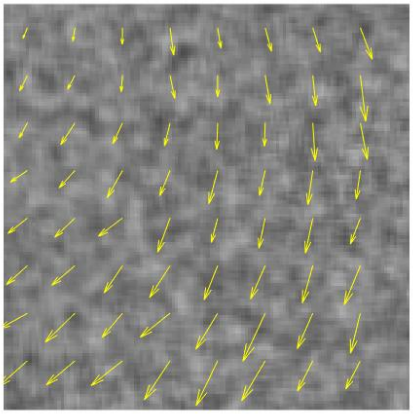

**Figure 7:** Result of calculated displacement vectors with Lucas-Kanade Algorithm