Computer Vision 1
Assignment 3: Harris corner detector and Optical Flow

Minh Ngo 10897402[1], Riaan Zoetmulder[2] 6072909

March 6, 2016

University of Amsterdam
[1]minh.ngole@student.uva.nl, [2] riaanzoetmulder@gmail.com

1 Harris Corner Detector

1.1 Implementation Details

For the purpose of this exercise we have implemented a *Harris Corner Detector*. The first step in our implementation consisted of calculated the partial derivatives of the pixel intensity in the x and y direction. We convolved the image with a *Gaussian Derivative* kernel in each respective direction.

We proceeded to calculate components necessary in order to calculate the matrix H that defines the *Cornerness* of each pixel in the image. We did this by computing:

$$A = \sum_W I_x(x,y)^2 \tag{1}$$

$$B = \sum_W I_x(x,y) I_y(x,y) \tag{2}$$

$$C = \sum_W I_y(x,y)^2 \tag{3}$$

Afterward, the H matrix can be computed according to following formula:

$$H = (AC - B^2) - 0.04(A + C)^2 \tag{4}$$

The next step consisted of finding local optima in windows of pixels, if these optima also exceeded a certain predefined threshold they were classified as corners. We ran several experiments, with different window sizes, thresholds, sigmas and kernel lengths to find what configuration gave the best results.

1.2 Results

After experimenting with window sizes, thresholds, sigmas and kernel lengths we decided that we would use a window size of 25, a threshold of $1.66 * 10^{-10}$, a sigma of 5 and a kernel length of 15. This particular configuration yielded the results for the *toy-person* depicted in the Figure 1 and results for *pingpong* in the Figure 2.

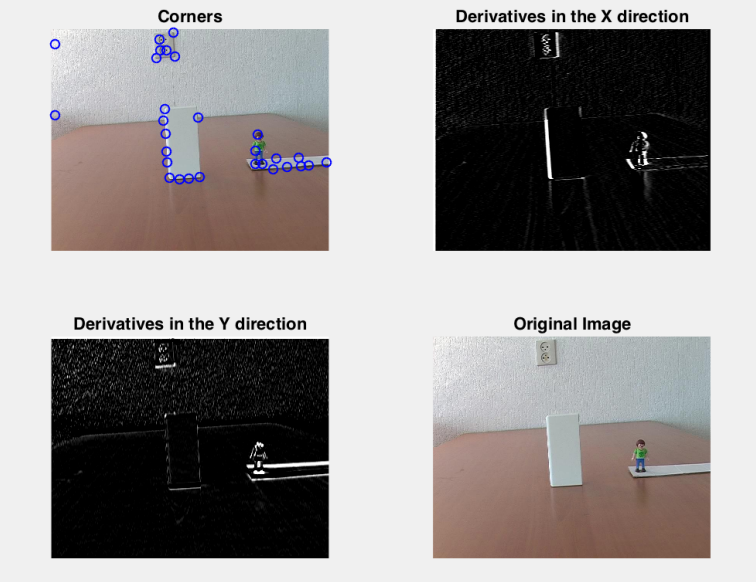

Figure 1: Detected Corners using the harris corner detection algorithm for the *toy-person*

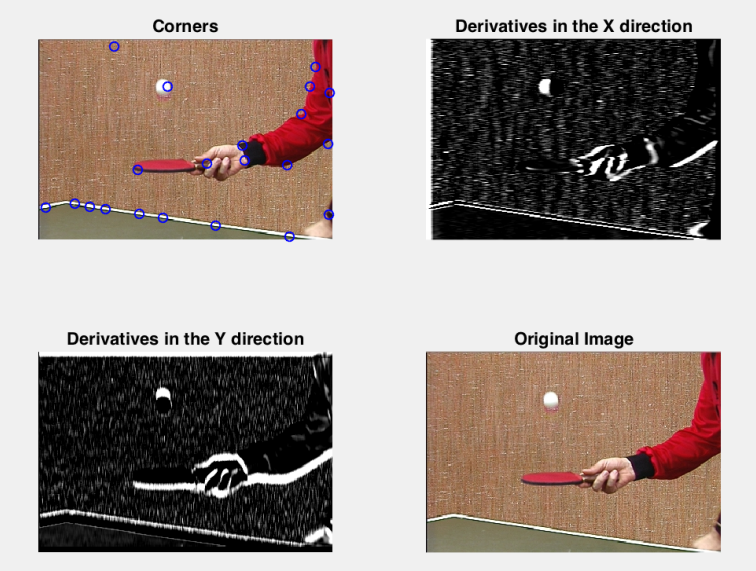

Figure 2: Detected Corners using the harris corner detection algorithm for the *pingpong*

We found that the smaller sigma, kernel length, and window size were and the lower the threshold was, the higher the amount of detected corners becomes. Furthermore, adding padding to the image in order to convolve it with the Gaussian Derivative kernel resulted in the corners of the image itself being detected, which was not the intention of the algorithm. In this case it can be noticed that the greatest change of color happens in the corners of the image because we are going from pixels that are 0 to pixels that have a value higher than 0. We therefore removed corners that were found in the corner of the image itself.

2 Optical Flow: The Lucas-Kanade Algorithm.

2.1 Implementation

In order to calculate the optical flow we calculated the Gaussian derivatives of the image in the x and y direction, and the derivative in the t direction. Because we assume that image values remain constant over time we know that the derivatives of the pixel intensities in the x and y directions times the optical flow in the x and y directions must equal the negative of the change in pixel intensities over time t. We can estimate than estimate the optical flow using regression, by solving the following system of equations:

$$A^T A\vec{v} = A^T\vec{b} \tag{5}$$

Where A is simply the matrix of the derivatives of the pixel intensities in the x and y direction, $\vec{v}$ is the vector of optical flows and $\vec{b}$ is the negative of the derivative of the pixel intensity w.r.t. t.

We used a window size of 15, a sigma of 3 and a kernel length of 9.

2.2 Results

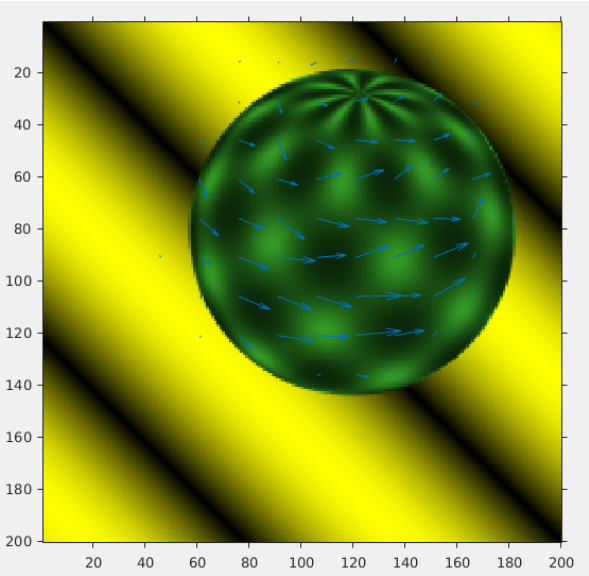

Figure 3: Optical flow computation for sphere1.ppm and sphere2.ppm files. Arrows go to the right direction.

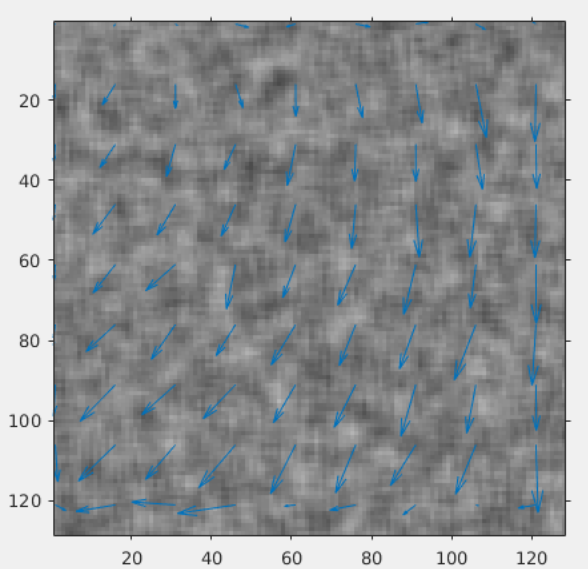

Figure 4: Optical flow computation for synth1.pgm and synth2.pgm files. Arrows go down.

3 Tracking

3.1 Implementation

We use the implemented Harris corner detector to determine key points to track them in next frames. Optical flow algorithm is executed for each pair of consecutive frames. Coordinates of key points are updating regarding to the offset determined by the optical flow vectors.

3.2 Results

Results are reported in the separate video files out.avi and out2.avi. It can be noticed that with each next frame key points are going a little further away from the original anchor [Fig 5, Fig 6] that is caused by:

- error propagation of the offset value computed by the Optical Flow algorithm [Fig 5];
- the fast moving object that cannot be captured by the local changes of the frame [Fig 6].

An increase of error can be fixed by evaluating tracked points by results returned by the Harris corner detector each N frames. The corner detector is not supposed to be executed each time during the performance issue. Finally, kernel and windows size can be increased to reduce influence of noise by incorporating more neighborhood pixels.

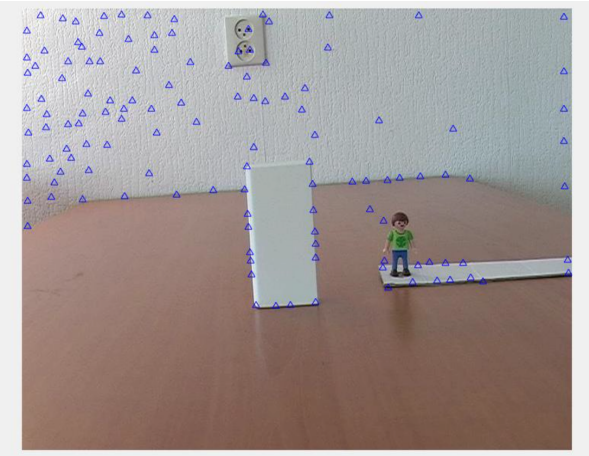

Figure 5: Tracking the *toy-person*

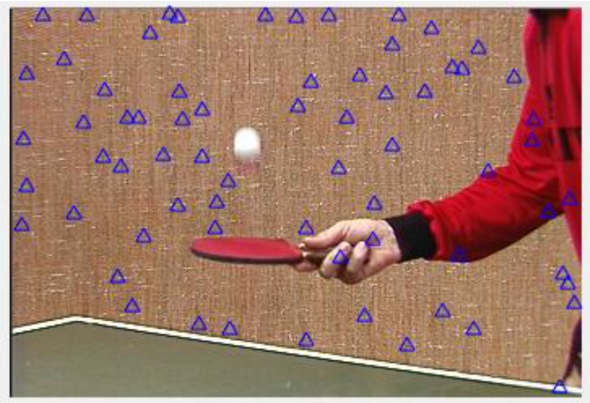

Figure 6: Tracking the *pingpong*