# Computer Vision 1: Assignment 3 Report

**Petru Neague**
(11844523)
petru.neague@student.uva.nl

**Tomas Fabry**
(11868414)
tomas.fabry@student.uva.nl

## 1 Introduction

In this project report we describe and implement the following tasks:

- The Harris Corner-Detector Algorithm to detect corner features in images;
- The Lucas-Kanade Algorithm to detect the direction of movement of patches in an image, given two instances of time of the given image
- Feature Tracking using the Harris Corner-Detector and Lucas Kanade Algorithms to find corner-features and track them across a sequence of images.

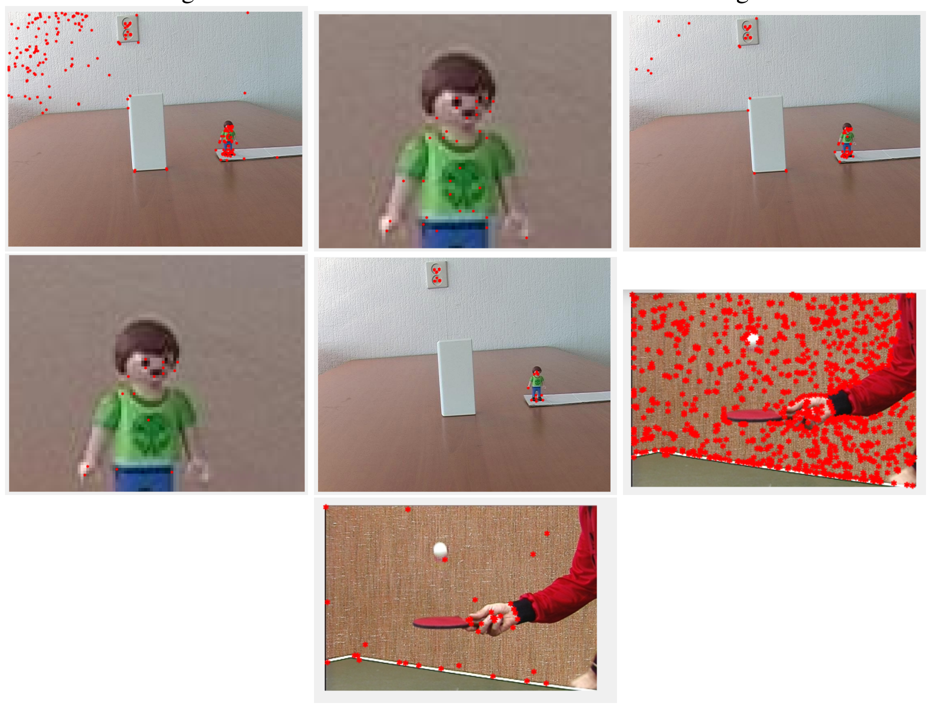
We adhere to the structure given by the assignment instructions and provide questions with the respective answers right below accordingly. Code provided in the zip file serves as the answer to implementation tasks.

## 2 Harris Corner-Detector

Question 1.

- Q: Create a function to implement Harris Corner Detector. Your function should return the H matrix, the rows of the detected corner points r, and the columns of those points c, where the first corner is given by (r(1), c(1)). Name your script as harris corner detector.m;
- Q: Your function should also plot three figures, the computed image derivatives Ix and Iy, and the original image with the corner points plotted on it. Show your results on example images person toy/00000001.jpg and pingpong/0000.jpeg in your report separately. Remember to experiment with different threshold values;
  A:

Figure 1: Harris Corner Detector for the Provided Images

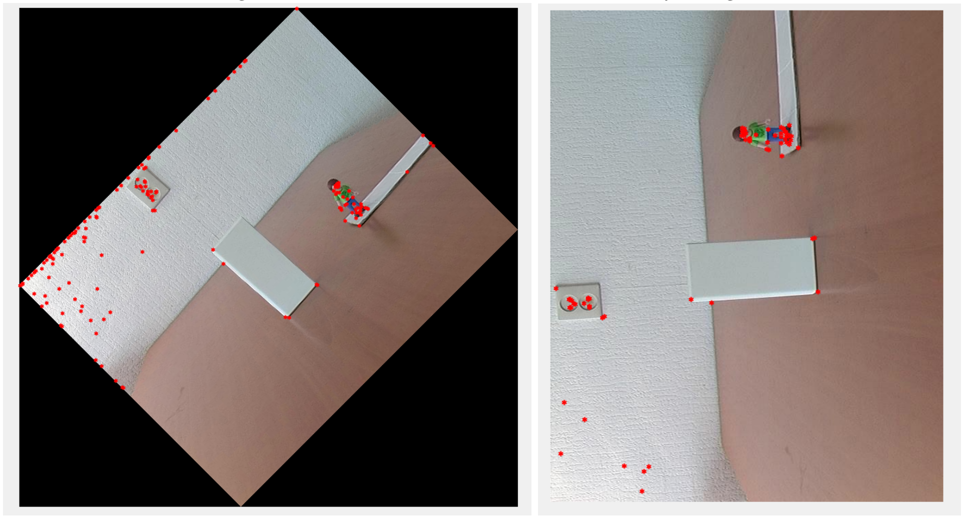


From left to right : toy/00000001.jpg for threshold = $0.01$, a close-up of the toy-figure for this threshold; the same image for threshold = $0.025$; another close-up of the toy-figure for the new threshold; and the same image for threshold = $0.1$; the pingpong/0000.jpeg image for threshold = $0.03$; the same image with threshold = $0.15$; and a close-up of the hand for this threshold.

It is evident that the threshold has massive influence on the quality of corner detection; It can also be seen that different thresholds are suitable for different images (for example the best threshold is around $0.025$ for the toy image and around $0.15$ - $0.2$ for the pingpong image, 7-9 times larger) meaning that the hyper parameters cannot be generally determined.

- Q: Randomly rotate person toy/00000001.jpg image and run the Harris Corner Detector algorithm on the rotated image. Is the algorithm rotation-invariant? Explain your answer and support it with your observations.
  A:

Figure 2: Corner Detection for Rotated Toy Image



The Threshold here was set to $0.025$ as before.

It can be seen that if the rotation is 90 degrees the algorithm reveals exactly the same results; however at less than other rotations differences can be seen. This probably comes from the fact that the matrix to which we are calculating the eigenvalues are initiated in a different way if the differing angle is different from 0, 90, 180 and 270, leading to deformed results for H. Furthermore, if the rotation is not a multiple of 90, we no longer make use of the replicate padding, which eliminates corner detection on the very edge of the image. 'Replicate' padding pads with pixels of the same color as their neighbours.

Question 2.

- Q:How do they define cornerness? Write down their definition using the notations of Equation 10.
  A:Shi and Tomasi define a corner as a patch whose smallest eigenvalue is larger than a threshold.

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

- Do we need to calculate the eigendecomposition of the image or the patches? Explain your answer.
  We have to calculate the eigendecomposition of the patches as they are the ones which are telling us whether the point around the patch is constructed can be considered a corner.
- In the following scenarios, what could be the relative cornerness values assigned by Shi and Tommasi? Explain your reasoning: a) Both eigenvalues are near 0; b) One eigenvalue is big and the other is near zero; c) Both eigenvalues are big.
  a) The image patch is almost uniform, no edges and no corners;
  b) One eigenvalue is large means there is an edge and there is the change is only around a line (as the smallest of the eigenvalues is small => no corner detected);
  c) Both eigenvalues are big means there is a corner (smallest of the eigenvalues is big => corner detected).

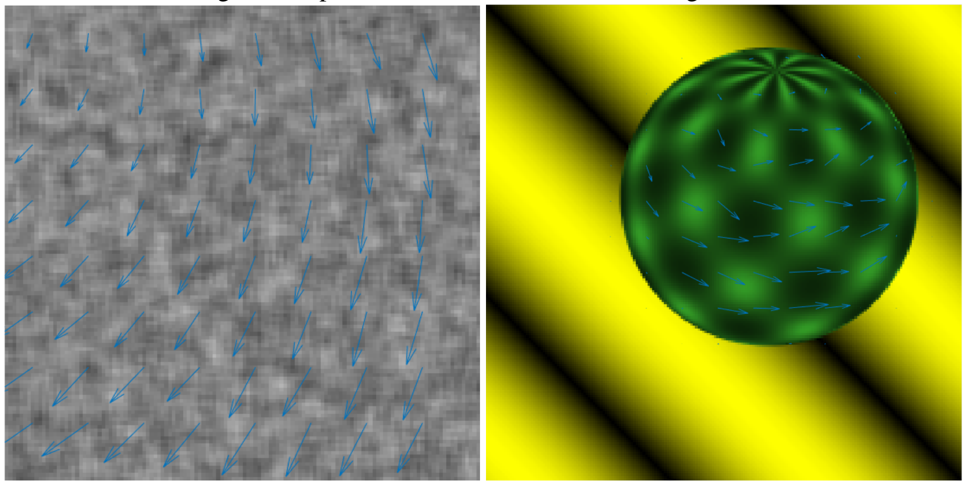
## 3 Optical Flow with Lucas-Kanade Algorithm

Question 1:

- Q: Divide input images on non-overlapping regions, each region being $15x15$;
- Q: For each region compute A, AT and b. Then, estimate optical flow as given in Equation 20;

- Q: When you have estimation for optical flow (Vx,Vy) of each region, you should display the results. There is a MATLAB function quiver which plots a set of two-dimensional vectors as arrows on the screen. Try to figure out how to use this to plot your optical flow results.
  A:

Figure 3: Optical Flow with Lucas-Kanade Algorithm



Question 2:

- Q: At what scale those algorithms operate; i.e local or global? Explain your answer.
  A:Lucas-Kanade works at local level as the matrix A is formed of derivatives of pixels in the current window. The Horn-Schunck algorithm assumes smoothness in the flow of color over the whole image. Thus, it tries to minimize distortions in flow and prefers solutions which show more smoothness. The flow in this case is formulated as a global energy functional which is then sought to be minimized. Thus, the global equation is determined to be:

$$\int\int (I_x * V_x + I_y * V_y + I_t)^2 + \alpha^2(||\nabla V_x||^2 + ||\nabla V_y||^2)dxdy \tag{2}$$

- Q: How do they behave on flat regions?
  A: By "flat" here we understand "homogeneous". For the Lucas-Kanade, a homogeneous area is not possible to be determined because there would be no changes in color from one pixel to the next. The Horn–Schunck algorithm yields a high density of flow vectors, i.e. the flow information missing in inner parts of homogeneous objects is filled in from the motion boundaries, thus performing better than the Lucas-Kanade algorithm.

## 4 Feature Tracking

We can see on the video that the features are being tracked in real time, frame-by-frame, while the detection of corners only happens in the very first frame. After corner detection, we keep track of their positions by adding the optical flow vectors. We scale the optical flow up by 10 because the magnitude of the vectors was not big enough to actually move fast enough to keep track of the corners, however. At the end of the video we can see the corners on the toy's head get left behind a bit because of the comparatively greater speed-up of the toy's direction. Once the tracking algorithm thinks the feature is not on the head, but on the background, it will never move again because the color of the background doesn't change much with movement. A dynamic scaling factor might help resolve this issue.

### 4.1 Question 2:

- Q: Why do we need feature tracking while we can detect features for each and every frame?
  A: Feature detection is generally an expensive operation in comparison to simply tracking

the features. While feature detection often makes use of convolutional filters sliding over the whole image or even neural networks comprising essentially of many filters and operations, feature tracking only requires simpler calculations around the area of the features(and not the entire image), once detected. This way, we can only detect features once and track their movement, or detect features every $n$ steps to correct for cummulative errors, instead of expensively detecting features every frame.

## 5 Conclusion

We have implemented Harris Corner Detection and Lucas-Kanade algorithms and combined them to make a feature tracking algorithm. We have also experimented with different parameter values to better understand the way they influence the accuracy and performance of the algorithms. While there are many more features one could detect on an image than corners, even an algorithm such as the Harris Corner Detector is quite costly in comparison to detecting the optical flow and by tracking the once detected features we are able to achieve better performance while still being able to keep decent accuracy. Of course, various parameters can be optimized to achieve better results on different images.