Assignment 3: Harris Corner Detector & Optical Flow

Vasiliki Vasileiou, Victor Kyriacou, Irene Papadopoulou

October 2021

Introduction

In this assignment we practised two basic Computer Vision algorithms, Harris Corner Detector and Lucas-Kanade. The former is used for detecting interest points and particularly corners. The latter is used for optical flow calculation. In the first two parts of this report, we implement these algorithms and test our results in different images. In the third part, we combine these two algorithms, in order to implement feature tracking.

1 Harris Corner Detector

Question 1.1.1: In this part of the lab we implemented the Harris - Stephens [1] method for corner detection. As a corner we define the intersection point of two edges. This method is based on the approach of image intensity change using Taylor's polynomials. The cornerness $H(x, y)$ is defined by the two eigenvalues of $Q(x, y)$, and is given by $H = \lambda_1\lambda_2 - 0.04\,(\lambda_1 + \lambda_2)^2$. The eigenvectors correspond to the directions of maximum and minimum change of the image, and the eigenvalues λ_1, λ_2 are a measure of this change. There are three different cases when both eigenvalues are small we have a flat region, when the one is much more smaller than the other one we have edges. Finally when both of them are large we have corners. Based on the above observations we expect that the cornerness will be positive and quite large in the corners. Then we reject points that are not maximum in a neighborhood using dilation. We also reject points that do not have a cornerness value above a certain threshold.

Question 1.1.2: We plot our results for three different values of threshold. We observe that the first value is too small and our algorithm detects only a small number of corners.

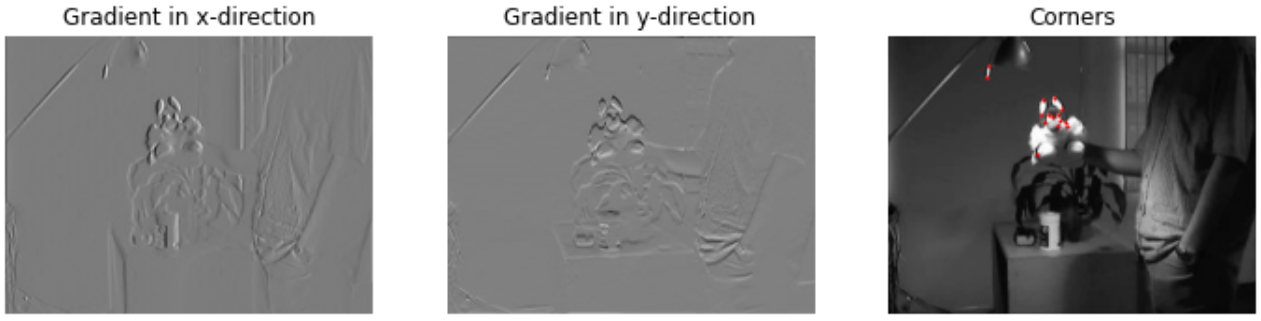


Figure 1: Toy Threshold 0.01

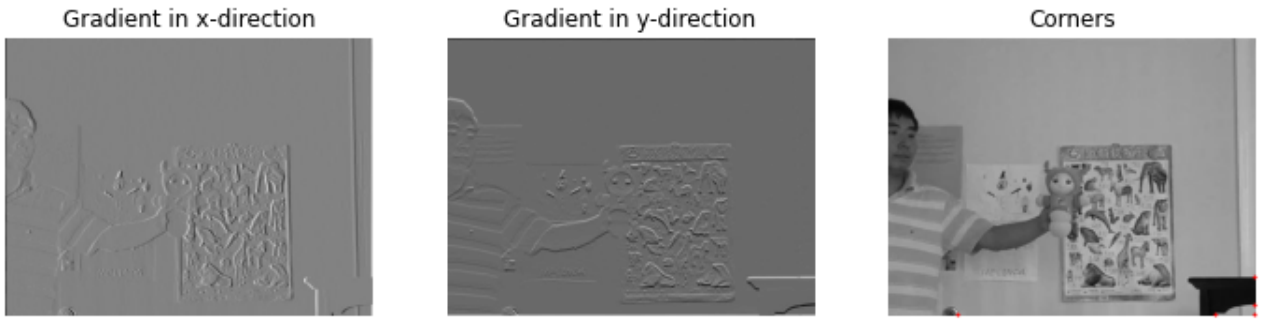


Figure 2: Doll Threshold 0.01

In the second case the number of detected corners is sufficient enough in both of the two images.

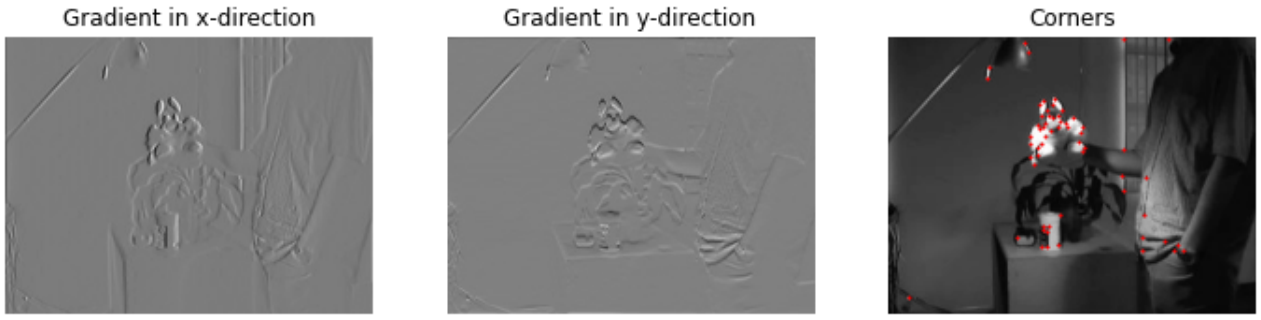


Figure 3: Toy Threshold 0.001

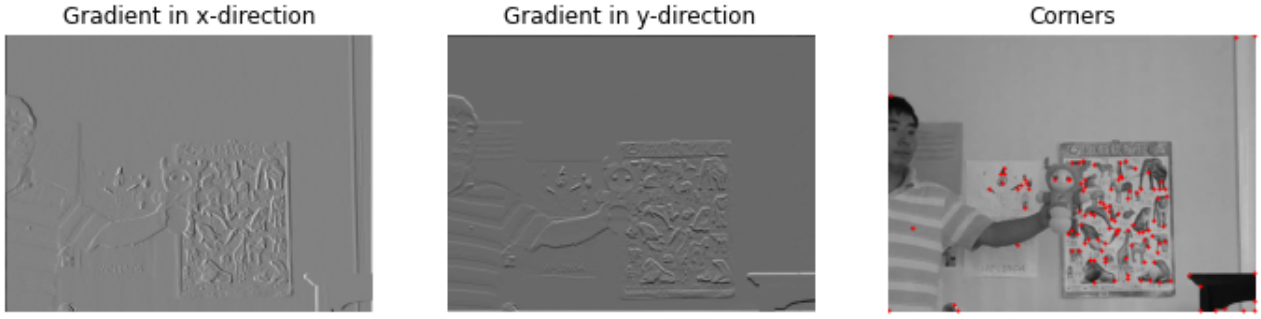


Figure 4: Doll Threshold 0.001

Finally in the last case we have a vary small threshold, this decrease makes the criterion stop working properly and it falsely recognises more pixels as corners.

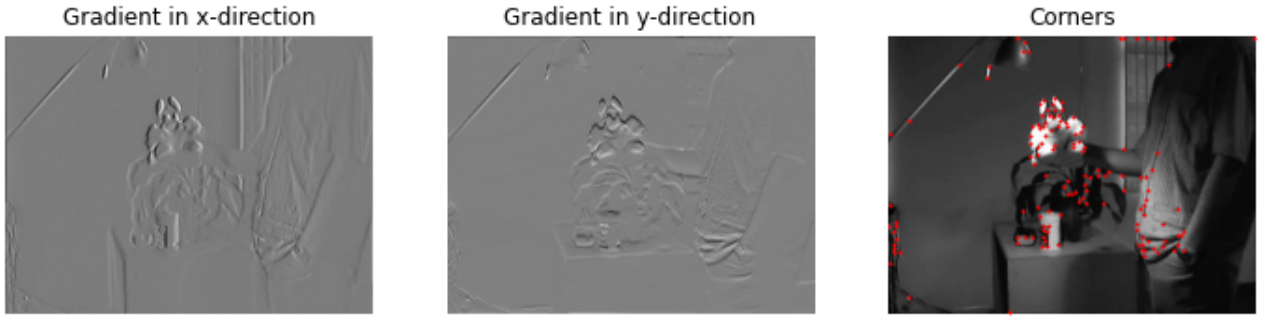


Figure 5: Toy Threshold 0.0001

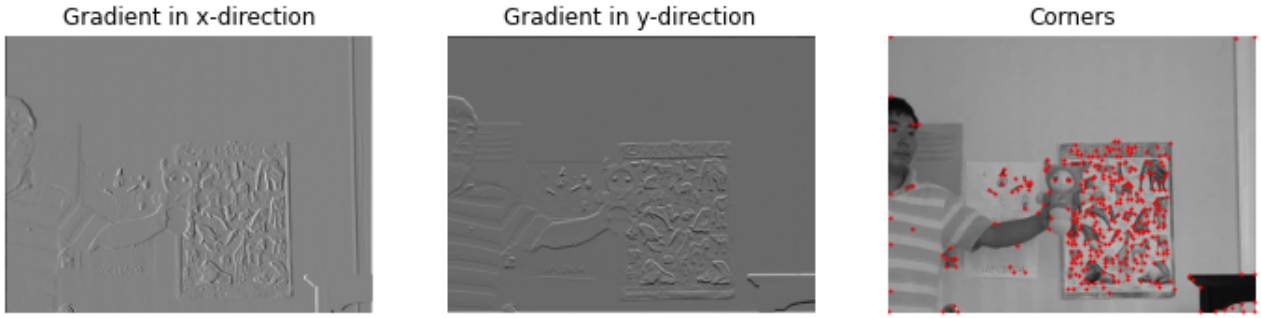


Figure 6: Doll Threshold 0.0001

Question 1.1.3: Harris corner response measure depends only on the eigenvalues of the matrix M, so it is rotation-invariant. The previous sentence is also proved by the 90^o rotation. But in the case of 45^o rotation we observe some points that appear or disappear at random corners. These are caused by imperfect interpolation and unavoidable floating-point rounding errors. Overall, most corners are detected correctly at all roatating angles.

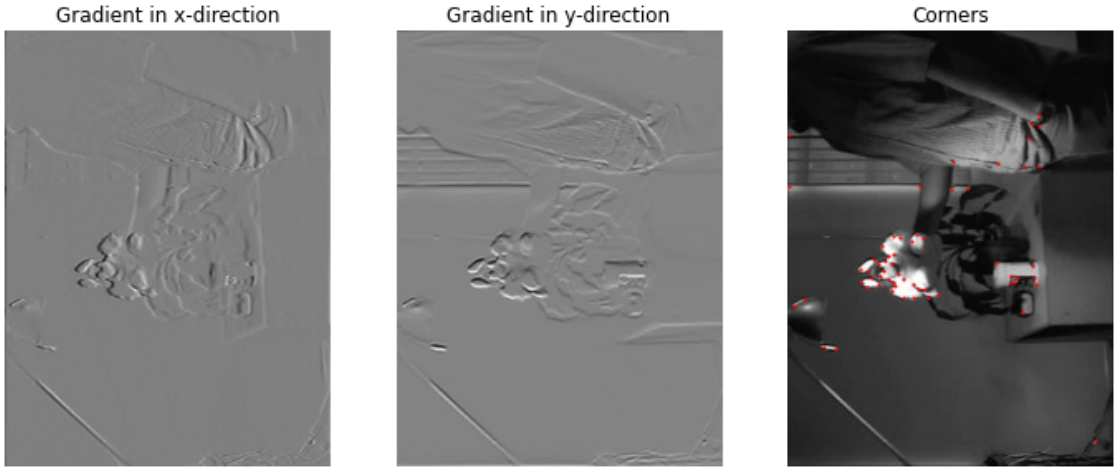


Figure 7: Toy Rotated 90^o

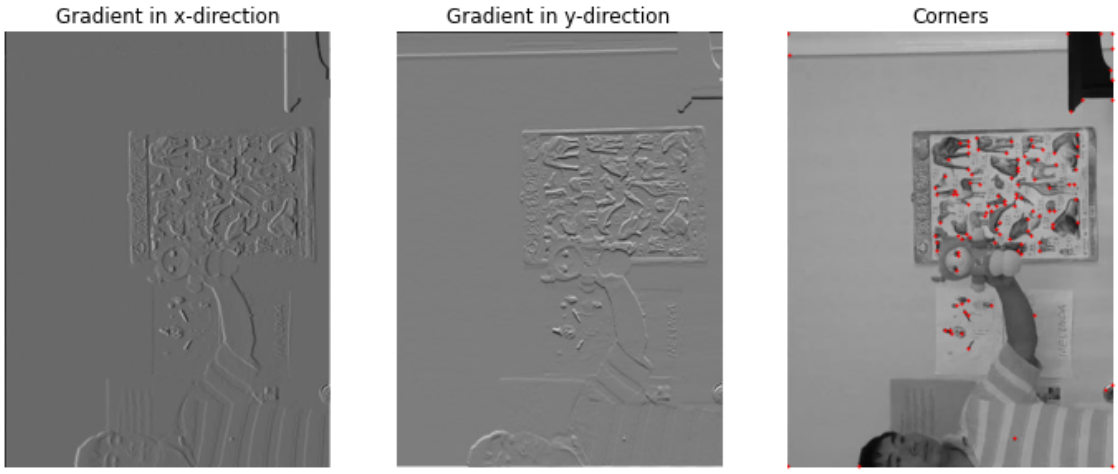


Figure 8: Doll Rotated 90^o

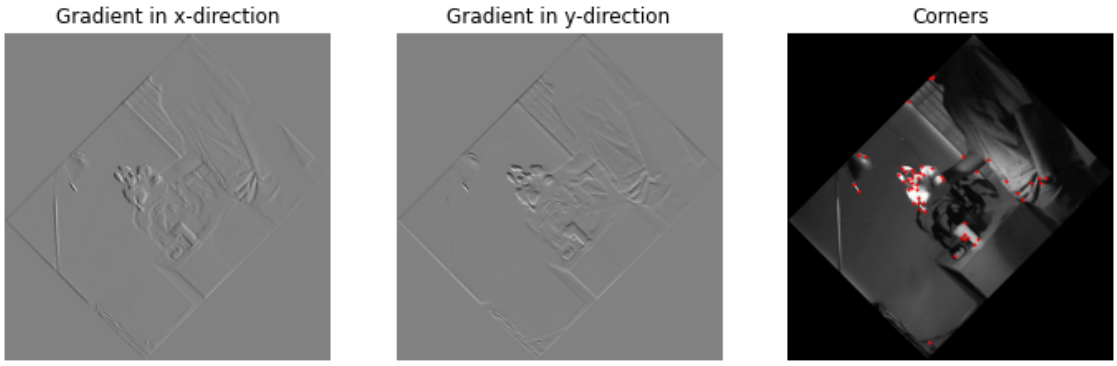


Figure 9: Toy Rotated 45^o

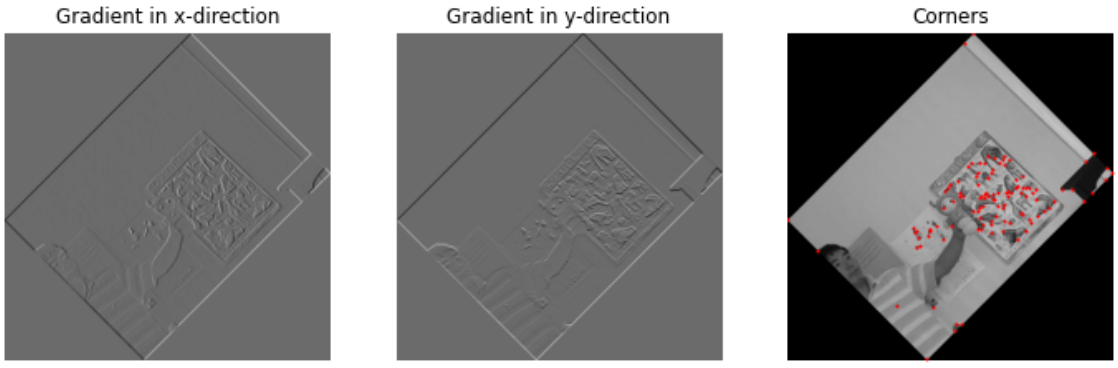
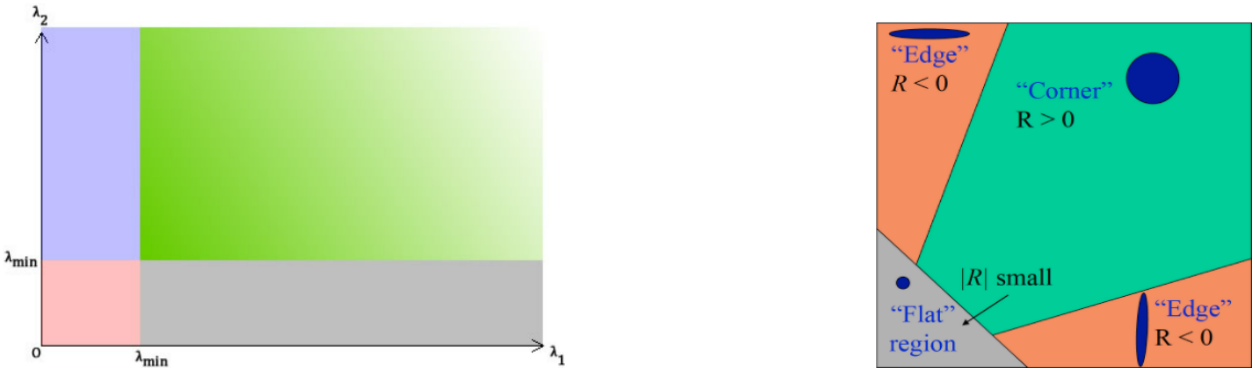


Figure 10: Doll Rotated 45^o

Question 1.2.1: Shi and Tomasi introduced a new algorithm for corner detection. This approach is based in the Harris Corner Detection with the only difference between them being the way they define the corner. To be more precise, for Harris-Stephens method the cornerness is given by the equation $H = \lambda_1\lambda_2 - 0.04\,(\lambda_1 + \lambda_2)^2$ while for Shi-Tomasi is given by $H = \min\{\lambda_1, \lambda_2\}$. Both of two approaches uses the eigenvalues of $Q(x, y)$ with the former passes them into a function while the latter uses them as they are to

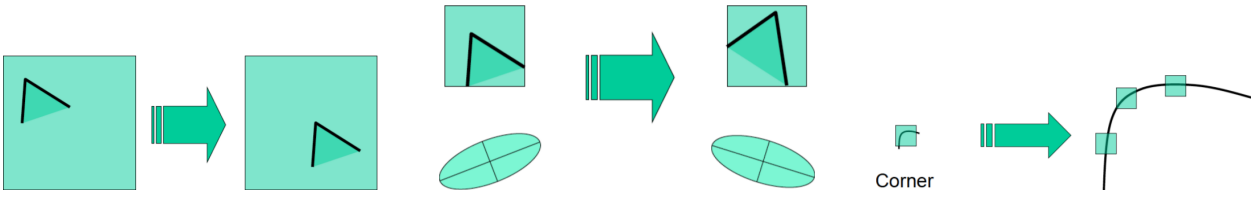
identify the corner. Also for both of methods the cornerness H is compared with a threshold value and if it is higher of this value the algorithms identify this point as corner point. In their paper [4], Shi and Tomasi demonstrated experimentally that this score criteria was much better.



(a) Shi-Tomasi (b) Harris-Stephens

Figure 11: Comparisons of Corner Detection Algorithms

Question 1.2.2: The Shi-Tomasi algorithm uses the eigenvalues for defining the cornerness of an image. So it depends in these values if the algorithm is translation, rotation and scale invariant. It is translation invariant, because the derivatives and window function are translation invariant. It works in the same manner as the Harris corner detector, so this detector is also rotation invariant. But it is not scale invariant because the re-scale changes the values of the eigenvalues. In case of enlargement a corner could be interpreted as an edge and in case of shrinkage an edge could be interpreted falsely as corner.



(a) Translation (b) Rotation (c) Scale

Figure 12: Translation and Rotation Invariant, Scale Variant

Question 1.2.3: The answer to this question could be more clear, if we observe the Figure 11a. So if both eigenvalues are near to zero (a), they belong to the red area and the point will be interpreted as "flat" region. In the second case where one of them is big and the other is near to zero (b)we have the same interpretation as the cornerness is defined by the minimum value of the eigenvalues. But in case that the small value is higher the threshold we are transferred in one of the blue or grey areas which represent edges. Finally, the case of both eigenvalues being big (c) is represented by the green area of the Figure 11a which is a corner.

2 Optical Flow - Lucas-Kanade Algorithm

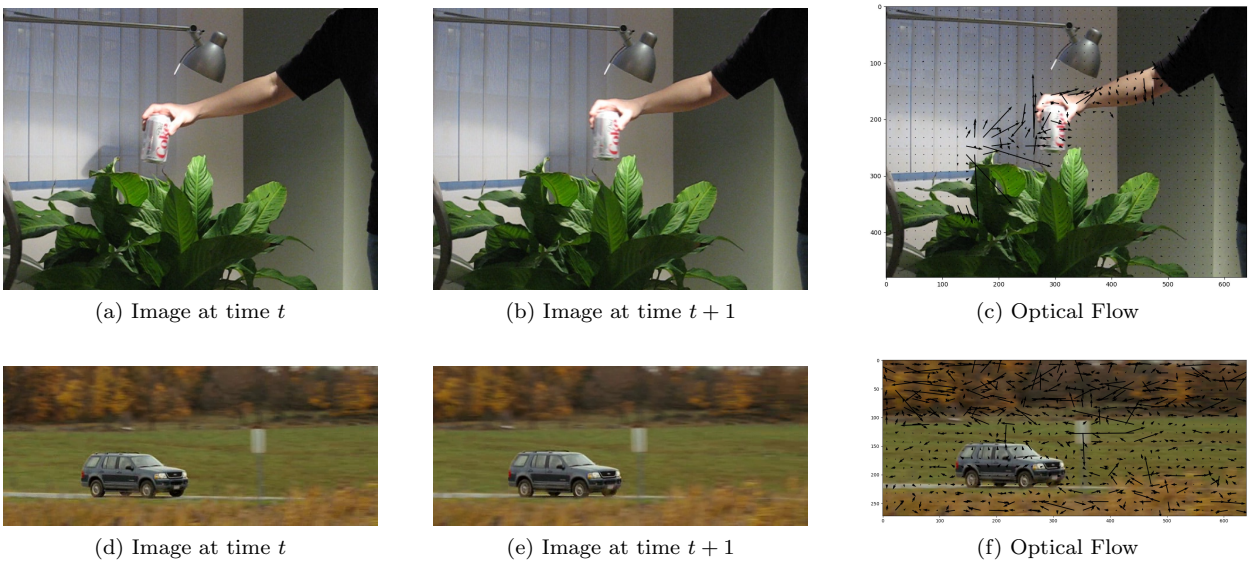
Lucas-Kanade algorithm [3] is a differential method used for the estimation of the optical flow. The main assumption is that the optical flow for each pixel p of interest, is constant for its whole neighbourhood. Thus, it utilises the gradients I_x, I_y, I_t for all the pixels in that region, to obtain the velocities V_x and V_y in x and y direction, respectively, by the least squares solution. These equation can be written as the following system:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ \vdots & \vdots \\ I_x(q_{N^2}) & I_y(q_{N^2}) \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad and \quad b = \begin{bmatrix} I_t(q_1) \\ \vdots \\ I_t(q_{N^2}) \end{bmatrix}, \tag{1}$$

where N is the number of an $N \times N$ square region with pixel p in its center. The Lucas-Kanade algorithm obtains a compromised solution by the weighted least squares principle by solving the following equation:

$$v = \left(A^T A\right)^{-1} A^T b. \tag{2}$$

To obtain the optical flow between two images namely, I_t and I_{t+1}, we convert them to gray-scale and calculate the gradients I_x, I_y, I_t. Then we separate the image into smaller square regions of $N \times N$ size. For each sub-region we obtain a pair of $[V_X, V_y]$ using Equation (2), which then, we map on the corresponding pixel p. The resulting optical flow for the two pairs of images can be seen in Figure 13.



(a) Image at time t (b) Image at time $t+1$ (c) Optical Flow

(d) Image at time t (e) Image at time $t+1$ (f) Optical Flow

Figure 13: Optical flow calculated for both pictures using Lucas-Kanade algorithm

2.1 Lucas-Kanade and Horn-Schunk Algortihm

As we described, Lucas-Kanade separates the image into smaller sub-regions and calculates the optical flow of each sub-region. Thus, Lucas-Kanade, operates locally on the image. On a "flat" sub-region that although its pixels move their values stay constant and so the estimated flow is zero. This is the aperture problem. The algorithm suffers in estimating the optical flow in such cases, because it utilises only local information. On the other hand, Horn-Schunk [2] is an algorithm that operates globally in order to deal with the aperture problem. To do that, it introduces a global constraint of smoothness. More specifically it assumes, that the flow all over image is smooth, in order to try minimizing the distortions in flow. Horn-Schunk tries to minimize equation (3) which is the optical flow expressed as a global energy function:

$$E = \iint \left[(I_x u + I_y v + I_t)^2 + \alpha^2 \left(\|\nabla u\|^2 + \|\nabla v\|^2 \right) \right] \mathrm{d}x \ \mathrm{d}y \tag{3}$$

The additional part $\mathbf{V} = [u(x,y), v(x,y)]^T$ denotes the optical flow vector, and the parameter α is a regularization constant, which controls the smoothness of the flow. Because the algorithm provides more dense flow vectors, in flat regions where there is no information because the gradients are zero, the missing flows are filled in from the motion boundaries.

3 Feature Tracking

Question 3.1: We implemented a feature tracking algorithm by combining the two previous algorithms, Harris Corner Detector and Lucas-Kanade for optical flow. Using the first algorithm, we locate the feature points we want to track and using the second one, we calculate where these points are going to move in the next frame.

Specifically, we start by obtaining the corners of the first frame, using the Harris Corner Detector algorithm in the same manner as before. Now that we have our feature points, we need to estimate their optical flow, based on their location in the next frame.

In order to calculate the optical flow only for our points of interest, we need to update our Lucas-Kanade algorithm; so that it calculates the flow given specific points and not uniformly in the whole image. For this, we created a new function called *calculate_optical_flow_with_LK_for_corners* which takes as input the indices of the rows and columns of the points we want to track. Next, we compute the sub-regions of these points, by considering each point as the centre of each region_size x region_size sub-region (in our case 15x15). We also cut the points that are close to the image's boundaries and can not fit a 15x15 sub-region around them. Finally, following the Lucas-Kanade algorithm, we estimate each point's optical flow, by calculating the velocities Vx and Vy in the x and y direction.

The last step is to update each corner's location, based on their movement between the two frames. In order to do that, we add on each corner (x, y) the corresponding velocity Vx and Vy, multiplied by a scaling factor z, and round up to the closest integer, to get where this corner moved in the next frame. We use the scaling factor in order to make corner updating more noticeable. After some trials we set the scaling factor $z = 3.25$.

We repeat this process for each frame, and we get our final video showing the estimated optical flow between each pair of frames. The result is tracking the corners of the first frame throughout the frames that follow. We tested our results and created two sample videos, named *doll.avi* and *toy.avi*. We notice that the tracking works pretty well, except at some corner points where the optical flow is pointing at different directions.

Question 3.2: Even though we can detect features for each and every frame, we need feature tracking because we do not know whether the points we are detecting correspond to the points of the feature we want to track. Generally, we need feature tracking, so that we remember our feature points and update only their location in each frame. Instead of trying to match each point we detect with the corresponding point in the next frame, we can do it directly, which is much more efficient.

Conclusion

In conclusion, in this assignment we implemented the Harris Corner Detector algorithm, which is used to find the corners in an image, Additionally, we implemented Lucas-Kanade algorithm, which is a local optical flow estimation method. Finally, we combined these methods in order to achieve tracking an object in a video sequence.

References

[1] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[2] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.

[3] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. Vancouver, British Columbia, 1981.

[4] J. Shi and C. Tomasi. Good features to track. *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.