Computer Vision 1
Harris Corner Detector & Optical Flow

Gongze Cao, Kai Liang, Xiaoxiao Wen and Zhenyu Gao
Faculty of Science, University of Amsterdam

1 Introduction

In this assignment, we cover some topics about the Harris Corner Detector and optical flow. We begin with implementing the Harris Corner Detector and discussing the Harris algorithm with different parameters and cornerness defined by [1] in section 2. Then, we implement the Lucas-Kanade algorithm for optical flow estimation and discuss other methods in section 3. At the end, we combine these two algorithms to extract visual features and track them in videos in section 4.

Throughout the assignment, we distribute our work fairly in terms of coding and writing the report.

2 Harris Corner Detector

2.1 Question-1

1. The detailed implementation can be found in `harris_corner_detector.m`. The function takes 3 main arguments as input, which are namely for reading the original image, setting the threshold and the size of the window (i.e. $N \times N$: the size of local neighbors). Besides, we set $\sigma = 1$ and $kernel\_size = 3 \times 3$ for the Gaussian and its first order derivative filter because it is natural and works well.

2. The image derivatives I_x and I_y for **person_toy/00000001.jpg** and **pingpong/0000.jpeg** are shown in Figure 1 and Figure 3, and the original images with the corner points plotted using different thresholds and window sizes for **person_toy/00000001.jpg** and **pingpong/0000.jpeg** are shown in Figure 2 and Figure 4.

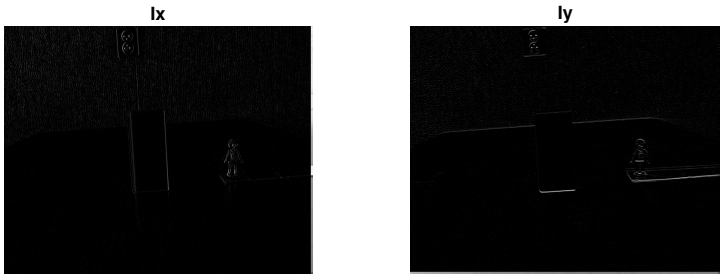

Figure 1: *Image derivatives I_x and I_y for **person_toy.jpg**.*

After experimenting with different thresholds and window sizes, we find that for **person_toy.jpg**, when we set $threshold = 500000$ and $window\_size = 5 \times 5$, the detection effect is the best. As for **pingpong.jpeg**, it has the best outcome when we set $threshold = 400$ and $window\_size = 5 \times 5$.

3. The algorithm is rotation-invariant. Harris algorithm uses eigenvalues of the local patch to detect the corners and the eigenvalues will remain the same regardless of rotations.

   However, in implementation, there are some small deviations. The size of the window is defined by a square of N-by-N, so there may be a small difference between the patches of the same corner before and after the rotation, which we think will lead to the difference because of different eigenvalues and the local maximum of $\mathbf{H}$. Specially, when the rotation angle is 90 degrees, there is no difference because the patches are the same after rotation. For example, we can infer from Figure 5 that the detected corners are the same in the original image and the image after rotating 90 degrees, but the corners detected in the image rotated by 45 degrees is slightly different from the original one.

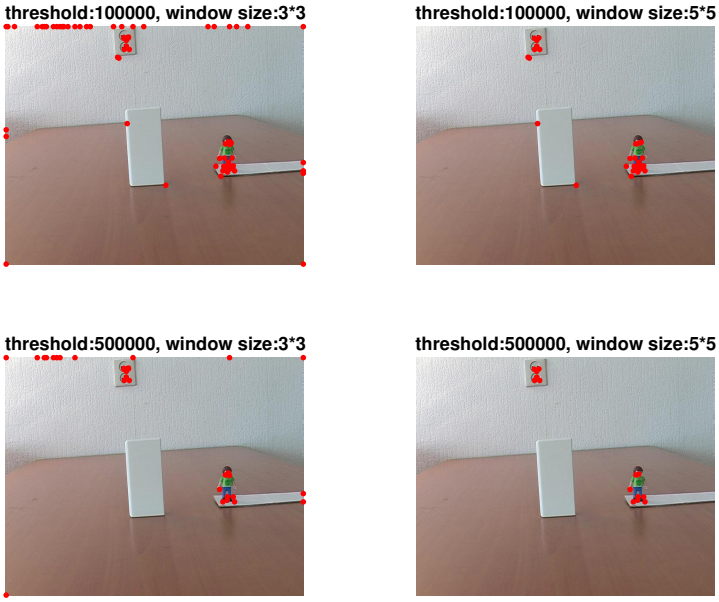

Figure 2: ***person_toy.jpg*** *with corners under different thresholds and window sizes.*

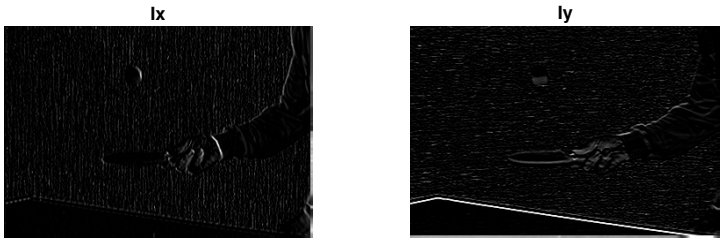

Figure 3: *Image derivatives I_x and I_y for* ***pingpong.jpeg****.*

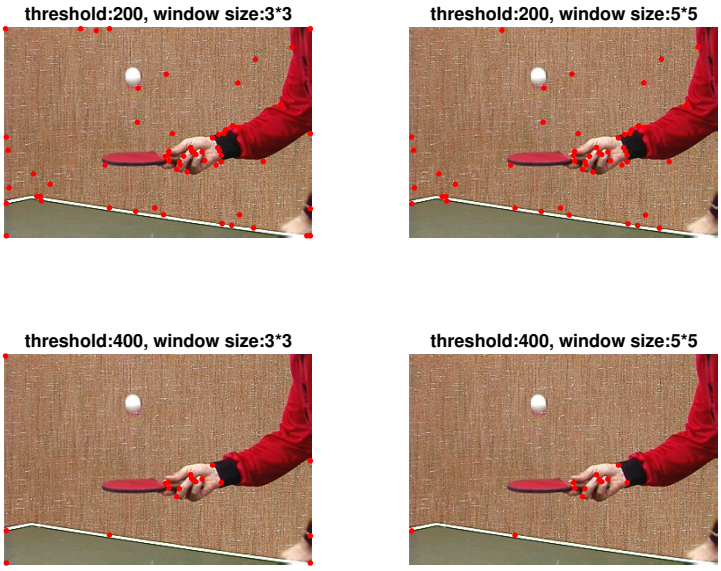

Figure 4: ***pingpong.jpeg*** *with corners under different thresholds and window sizes.*

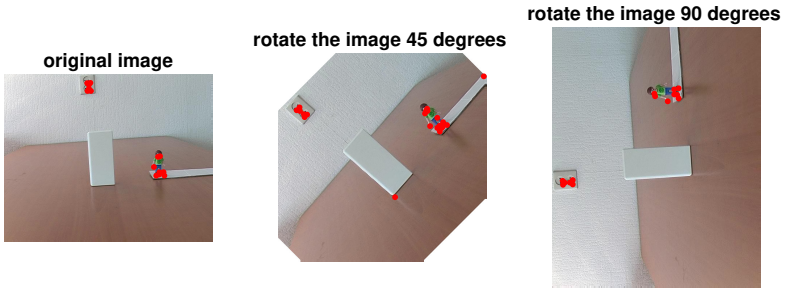

Figure 5: ***person_toy.jpg*** *with corners in different rotations.*

2.2 Question-2

1. According to [1], the cornerness $H(x, y)$ for the Shi and Tomasi corner detector is defined as:

$$H(x, y) = \min(\lambda_1, \lambda_2)$$

   where the constant $k = 0.04$ originally from Harris Corner Detector is spared.

2. Although the eigenvalues of $Q(x, u)$ need to be computed, the eigen decomposition is expensive in terms of complexity and is evitable. Given $Q(x, y) = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$, $\lambda_1\lambda_2 = AC - B^2$ and $\lambda_1 + \lambda_2 = A + C$, we can easily find the eigenvalues λ_1 and λ_2.

3. (a) When both eigenvalues are near 0, taking the minimum of the two, the cornerness H would be near 0 as well, indicating a flat surface in the image.
   (b) When one eigenvalue is big and the other is near 0, H would be near 0, indicating an edge in either direction in the image.
   (c) When both eigenvalues are big, H would be big, indicating a corner point.

3 Optical Flow with Lucas-Kanade algorithm

3.1 Question-1

The detailed implementation can be found in `lucas_kanade.m`. The function takes two major input arguments, which are namely the two frames of the same object. For computing the partial derivatives of pixels in the x- and y-directions, the Sobel kernel is used. In `lucas_demo.m`, we test the function on the two given pairs of images (i.e. *sphere* and *synth*), and the estimations are shown in Figure 6a and Figure 6b.

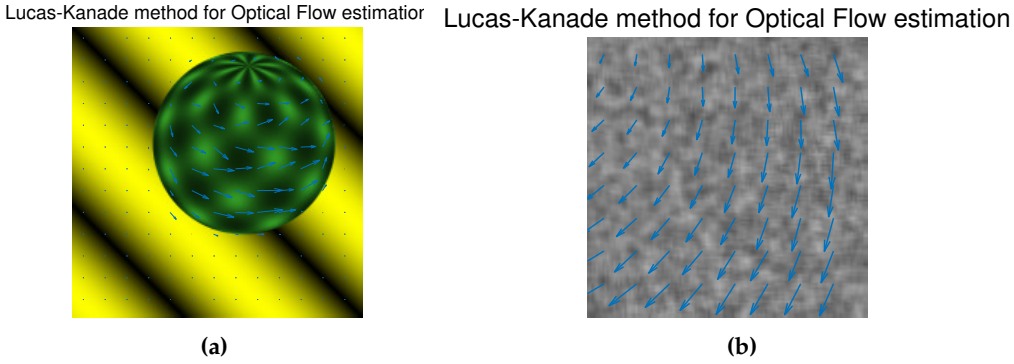

Figure 6: *Lucas-Kanade method for optical flow estimation for sphere (a) and synth (b).*

As can be inferred from the two figures, the function estimates the optical flow for the two pairs of images properly as expected.

3.2 Question-2

1. The Lucas-Kanade method operates on a local scale because it assumes that the optical flow is constant in a local neighborhood of the pixel under consideration, which is an additional assumption to resolve the aperture problem. On the other hand, the Horn-Schunck method operates on the global scale as it assumes that neighboring points on the objects have similar velocities and assumes a smooth variance in the optical flow (velocity field) over the whole image [2].

2. On flat regions, the Lucas-Kanade method fails because the gradients of the local neighboring points in both x- and y-directions become (close to) 0, which makes the linear system no longer solvable. In terms of the Horn-Schunck method, as it operates on a global scale, as long as there is tractable motion in the image, the optical flow on flat regions would be affected by the optical flow computed on the motion boundaries via the iterative procedure.

4 Feature Tracking

4.1 Question-1

The detailed implementation can be found in `tracking.m`. We test our implementation on the *ping-pong* and *person_toy* datasets. Specifically, we set our thresholds in the detector to be 400 and 500000 respectively and set the window size to be 5×5. In the optical flow part, we set the region size to be 15×15 same as in section 3. Besides, we empirically set the time interval between two frames, namely the coefficient we use to scale the optical flow before adding to the tracking point, to be 10. Some key frames from the two generated videos are shown in Figure 7. As can be inferred from the graphs (and

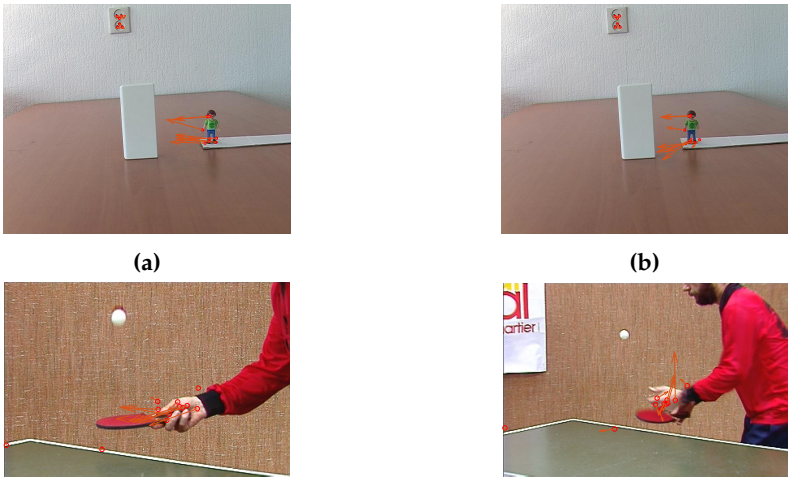

Figure 7: *Tracking visualization from two key frames of person_toy.avi and pingpong.avi.*

videos), our implementation functions properly as expected on tracking the feature points of moving objects in the whole process.

4.2 Question-2

Firstly, using optical flow to track feature points is cheaper and more natural than detecting features for each frame. Secondly, by fixing the tracking points in the first place, we can avoid sudden immersion or disappearance of some points. Besides, the position of edge points estimated per frame might show unstable behaviors such as bouncing around in a region, while using optical flow is much more stable because the norm of optical flow is controllable.

5 Conclusion

In the previous sections, we first study and implement the Harris Corner Detector for extracting corner points and the Lucas-Kanade algorithm for estimating the optical flow. We then apply the two methods for tracking the interesting corner points. Finally, we test the feature tracking method on the *pingpong* and *person_toy* examples and obtain expected results.

For the Harris Corner Detector, it is shown that the effects of the parameters are significant, hence, the parameters are in need of tuning for different inputs. Besides, it is researched that although there are some small deviations, the algorithm is rotation-invariant. Finally, the algorithm is compared with its variant: the Shi and Tomasi detector. It is found that the variation spares one tunable parameter k that is originally used in the Harris Corner Detector for computing the corner H, and produces better results [1] in similar computational complexity.

For the Lucas-Kanade algorithm, the implementation is described with the results illustrated. Besides, it is compared against the Horn-Schunck method, where the difference between local and global methods for optical flow estimation is discussed. Furthermore, their different behaviors and robustness against flat regions are discussed and qualitatively reasoned, which shows that the Horn-Schunck method, operating on a global scale, is more robust w.r.t. flat regions.

Finally, the implementation of feature tracking is described with results shown. Also, the importance of first detecting and fixing the features and then tracking their motions is briefly discussed.

References

[1] Jianbo Shi and Tomasi, "Good features to track," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, pp. 593–600, IEEE Comput. Soc. Press, 2002.

[2] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185 – 203, 1981.