Computer Vision 1: Harris Corner Detector and Optical Flow

Bogdan Teleaga, 11413565
Mircea Mironenco, 11164182

March 8, 2017

1 Harris Corner Detector

We have implemented the Harris Corner Detection algorithm in harris.m. The result of applying the algorithm on the two provided images, can be seen in Figure 2 and 3. To run the algorithm with the tuned hyper-parameters, see harris_demo.m. The hyperparameters can be used to increase the granularity for the corner detection, for example by creating a less smooth Gaussian and using a smaller window size we can detect every corner within the wall socket for example. In Figure 2 and 3 we chose to detect coarser corners.

(a) Instead of $H = \lambda_1\lambda_2 - k \cdot (\lambda_1 + lambda_2)^2$ where $k = 0.04$ we use:

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

(b) For both methods we need to compute eigenvalues for patches instead of the whole image. The intuition is that we are interested in the correlation between changes in intensity in both directions which for a corner will only happen in a region close to the corner. However, for the Harris detector we can just compute the determinant and trace of the Q matrix.

(c) In short, if H is greater than a predefined value, it can be considered a corner. To answer the mentioned cases and illustrate, see Figure 1, taken from the openCV documentation [1].

In the orange areas we can see that either λ_1 or λ_2 are lower than a certain threshold value, and thus do not correspond to corners. These areas correspond to *edges*.

The gray area corresponds to neither corners or edges, and here both the eigenvalues are less than a required minimum.

The green area illustrates that both eigenvalues are big, or rather, larger than a certain value. This corresponds to *corners*.

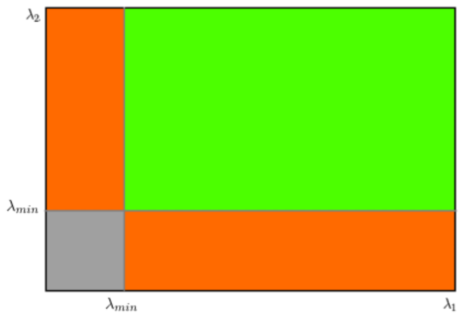

Figure 1: Shi-Tomasi space.

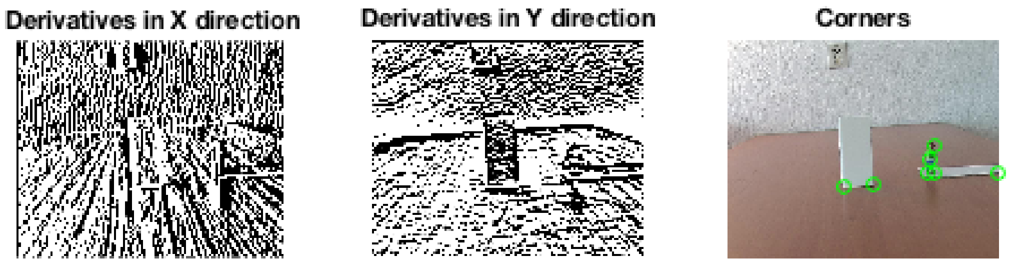

Figure 2: Harris corner detection for toy image.

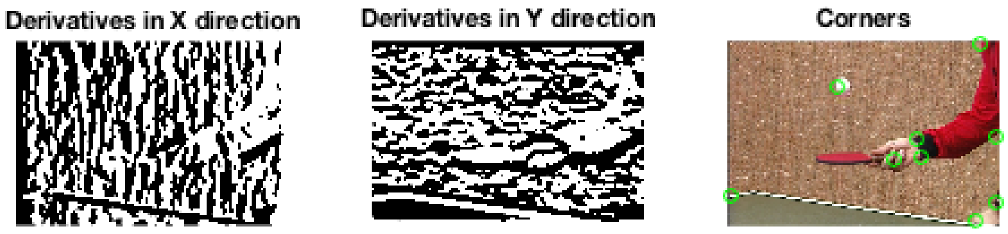

Figure 3: Harris corner detection for ping-pong image.

2 Optical Flow using Lucas-Kanade Algorithm

Optical Flow using Lucas-Kanade has been implemented in 2 separate files (lucas_kanade.m and lucas_kanade_points.m), using 2 different implementations, depending on whether or not we are applying it to the whole image or just a set of predefined points. The results can be see in Figures 4 and 5.

(a) HornSchunck assumes smoothness in the flow over the whole image, i.e. at the global level, whereas Lucas-Kanade operates at a local level, inside a predefined neighborhood.

(b) Lucas-Kanade will fail on flat regions, as the derivatives are zero and as such the inverse will not exist. Horn-Schunck treat flat regions as ambiguous, by applying a smoothing constraint to the optimization problem. This allows information regarding to the velocity to propagate towards the flat regions, solving the problem given by the gradient being close to zero.

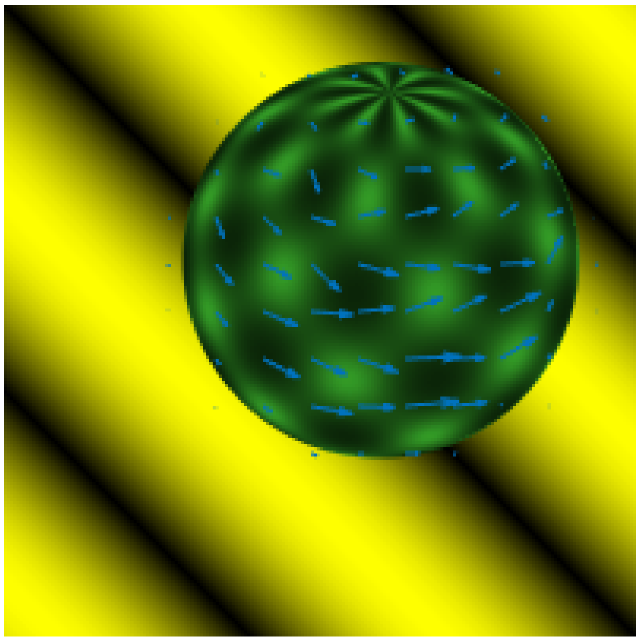

Figure 4: Optimal flow in sphere images using Lucas-Kanade

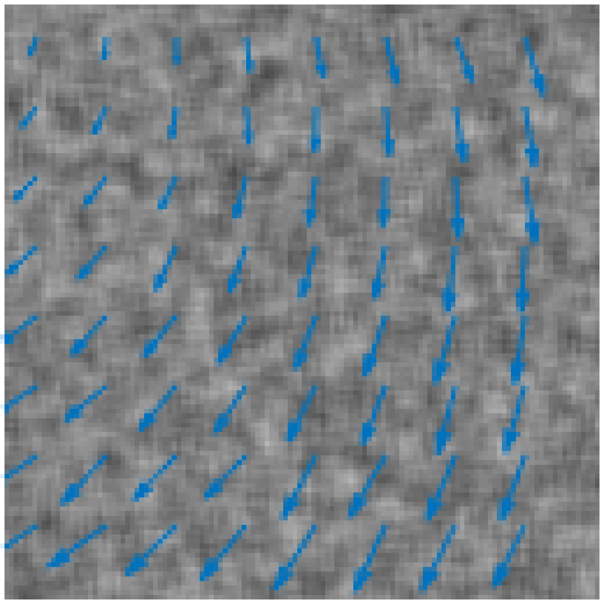

Figure 5: Optimal flow in synth images using Lucas-Kanade

3 Tracking

We have implemented a tracking algorithm that utilizes both Harris Corner Detection and the Lucas-Kanade algorithm to track optical flow. The corners of the image are recomputed on every fifth image. The tracking videos can be seen by accessing *pingpong.avi* and *person_toy.avi*.

References

[1] http://docs.opencv.org/3.0-beta/doc/py$_{t}utorials/py_{f}eature2d/py_{s}hi_{t}omasi/py_{s}hi_{t}omasi.html$.