Computer Vision 1 - Assignment 3
Harris Corner Detector & Optical Flow

Stephan Grzelkowski UvA-id: 10342931
Sander Brinkhuijsen UvA-id: 11305517
Sebastiaan Barneveld UvA-id: 11051108 **Whitney Mok** UvA-id: 10624767

October 1, 2019

TO DO's

1. Sectie 3 tracking: a) algo beter laten werken - misschien hyper param tuning b) superKorte bescrhijving bij vraag 1 (anders is het antwoord letterlijk alleen code en een filmpje) 2. Intro and Conclusion

Introduction

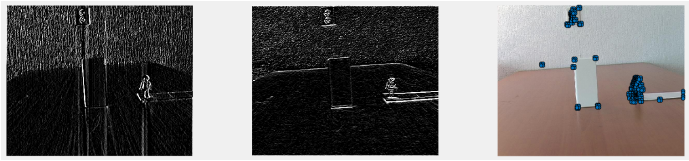
As an extension of our previous work, we explore some more feature extraction algorithms, specifically ones that detect corners or describe motion. Among these, we have implemented the Harris Corner Detector and Lucas-Kanade Optical Flow method, and lastly, we show how optical flow can also be used to track corners or other features.

1 Harris Corner Detector

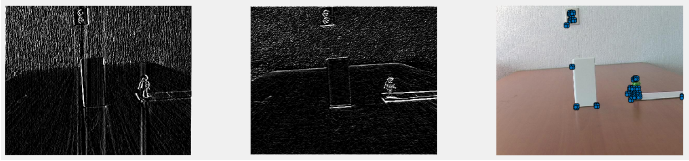
Question-1

1.and 2.

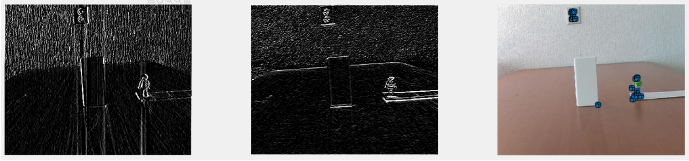
In our previous work, we have shown that the extent of intensity changes can provide clues about where edges are located. Here, we show that this can be further exploited to localize corners- points with large gradients in both directions. Figure 1 and 2 illustrate the results of applying Harris' Corner Detector: increasing the threshold is one way to increase the selectivity of the algorithm, as fewer points will have a cornerness value above the threshold.



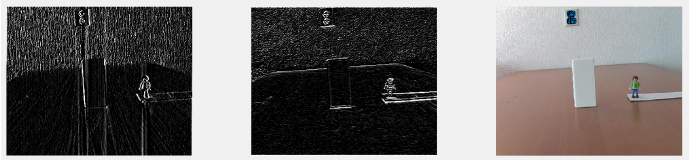
(a) Threshold = 1



(b) Threshold = 10

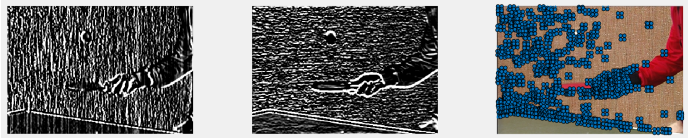


(c) Threshold = 20

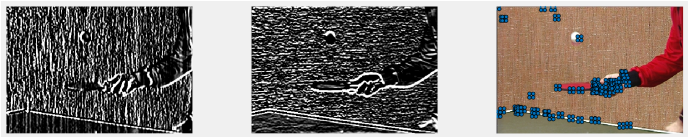


(d) Threshold = 100

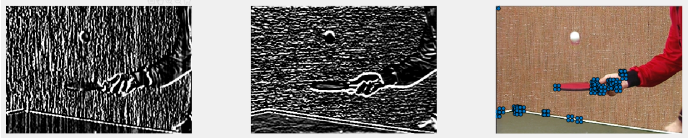
Figure 1: Harris corner detection on the toy person image with different thresholds. The image has been smoothed with a Gaussian kernel with kernel size = 5 and $\sigma = 1$.



(a) Threshold = 1



(b) Threshold = 10



(c) Threshold = 20

(d) Threshold = 100

Figure 2: Harris corner detection on the pingpong image with different thresholds. The image has been smoothed with a Gaussian kernel with kernel size = 5 and $\sigma = 1$.

3.

The Harris corner detection is not completely rotation invariant. For a 90 degrees rotation, the corner responses will remain the same (Figure 3b). The derivatives of the image produce the same output, the only difference is that the x- and y-axis are swapped. This causes matrix Q to swap its main diagonals (A and C). Since H is calculated by Equation 1, the swap does not change the result of H.

For a 45 degrees rotation, however, the derivatives in the x and y direction do change, altering the components within the Q matrix (A, B, and C) and subsquently the corner response H (Figure 3a.

$$H = (AC - B^2) - 0.04(A + C)^2 \tag{1}$$

Question-2

1.

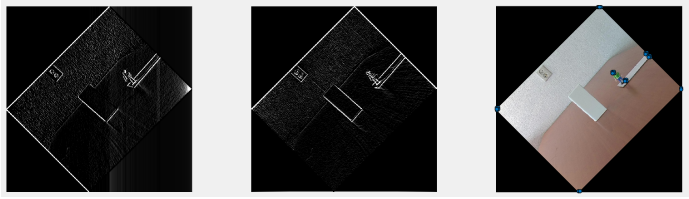
While Harris Corner detector can give reasonable estimations of corners as demonstrated above, Shi and Tommasi have introduced a slight modification to the algorithm that is often reported to yield even better results [2]. Their definition of a cornerness is as follows:

$$H(x, y) = min(\lambda_1, \lambda_2) \tag{2}$$

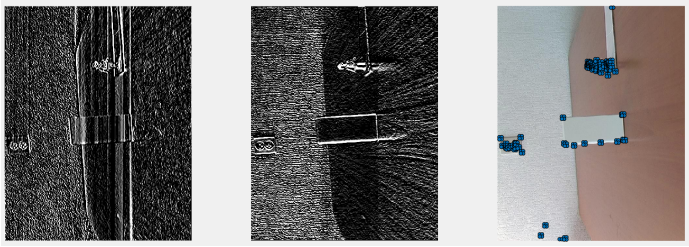
A point (x, y) is considered a corner when its cornerness value $H(x, y)$ is larger than a predefined threshold.

2.

As suggested above, Shi-Tommasi's approach is very similar to the Harris Corner detector: eventually each patch's properties are described by the Q matrix. From its eigenvalues a corner response can be determined. Therefore, both methods require eigen decomposition at patch level.



(a) Image rotated by 45 degrees. Threshold = 10



(b) Image rotated by 90 degrees. Threshold = 10

Figure 3: Harris corner detection algorithm on rotated images. The image has been smoothed with a Gaussian kernel with kernel size = 5 and $\sigma = 1$.

3.

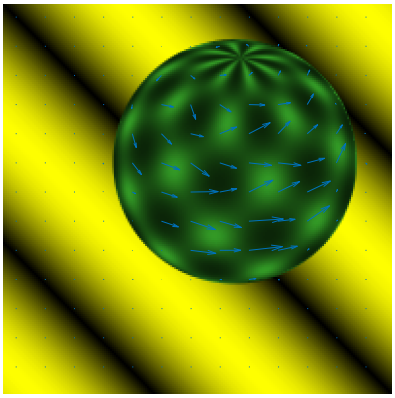
Shi and Tommasi's simpler definition of cornerness stemmed from the consistent observation that along edges, one of the eigenvalues is remarkably small. This led to the idea that, in scenario c, where even the smallest among the two eigenvalues exceeds the threshold T, i.e. both eigenvalues are large, the point must be a corner. When $min(\lambda_1, \lambda_2) < T$, the point is assigned an insufficient cornerness value, whether $max(\lambda_1, \lambda_2)$ is big (scenario b, an edge) or also near 0 (scenario a, flat region).

2 Optical Flow

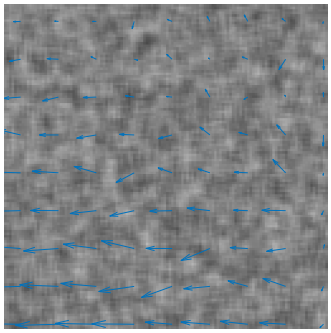
Question-1

1., 2, and 3

As will be shown towards the end of this work, corners we have detected in the previous section or other features could be tracked using optical flow estimates. Here we have implemented the Lucas-Kanade algorithm, which assumes constant optical flow within a region. The division of frames into non-overlapping, rectangular regions following this assumption, is an oversimplified approach, but easy to implement and sometimes gives reasonable results. This is illustrated particularly well by Figure 4a, where all optical flow vectors together show coherent direction, indicative of a rightward rotation from the viewer's perspective.



(a) First frame of a sphere and its optical flow estimates



(b) First frame of something comparable with white noise and its optical flow estimates

Figure 4: Visualization of optical flow vectors estimated for two pairs of frames using the Lucas-Kanade algorithm.

Question-2

1.

Unlike the Lucas-Kanade algorithm, Horn-Schunk operates at global scale. Instead of assuming that optical flow is constant in a neighbourhood, it is assumed that optical flow is smooth across the whole image. Thus, the objective function

$$E = \int\int (I_x u + I_y v + I_t)^2 dxdy + \alpha \int\int \left\{ (\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2 \right\} dxdy \tag{3}$$

not only comprises the image constraint equation, but also incorporates a smoothness constraint (second part of equation 3), where α is a regularisation parameter: larger αs penalize larger flow gradients, resulting in smoother flow. The equation is minimized by solving its corresponding multidimensional Euler-Lagrange equations (see [1]).

2.

As alluded to above, the horizontal and vertical component of the optical flow can be solved for linearly for each pixel. This is done iteratively as follows:

$$\begin{aligned} u_{x,y}^{k+1} &= u_{x,y}^{-k} - \frac{I_x[I_x u_{x,y}^{-k} + I_y v_{x,y}^{-k} + I_t}{\alpha^2 + I_x^2 + I_y^2} \\ v_{x,y}^{k+1} &= v_{x,y}^{-k} - \frac{I_y[I_x u_{x,y}^{-k} + I_y v_{x,y}^{-k} + I_t}{\alpha^2 + I_x^2 + I_y^2} \end{aligned} \tag{4}$$

$\begin{pmatrix} u_{x,y}^{k+1} & v_{x,y}^{k+1} \end{pmatrix}$ denotes the velocity estimate of a pixel at (x, y) in the next iteration $k+1$. $\begin{pmatrix} u_{x,y}^{-k} & v_{x,y}^{-k} \end{pmatrix}$ is the average of the velocity estimates within the neighbourhood, which 'fills in' where a region's image intensity gradient is zero, so that flat regions' optical flow can still be estimated [1].

3 Feature tracking

Question 1

The tracking algorithm uses the Harris Corner Detection algorithm from section 1 to find useful corners within the image. Then it uses the Lucas-Kanade algorithm to find the flow vector in the local neighbourhoods of the image. The locations of the features are updated using the flow-vector. For each image we calculate the flow vectors again and update the locations of the features. This can cause problems if the flow vector is not correctly updating the new location of the features. As this is the only way we update the coordinates of the features the errors can accumulate. We see this in toy-figure video, where some features do diverge from their original image feature.

In the pingpong example we see that the algorithm has great difficulties if the type of motion changes. At first, the update works reasonably, but fails to produce a reasonable update when the motion changes from upwards to downwards and when the camera itself moves, shifting the entire scene. To solve this issue one could redo the harris corner detection after a number of frames.

The code can be found in demoTracking.

Question 2

Instead of feature tracking, which we have visualized in the attached video, it is of course also an option to perform feature detection on every new frame, and it is not expected to take significantly more time in the example we give here. However, for more and larger frames, or when multiple features are to be detected, it is definitely computationally more efficient to track features rather than recomputing them for every frame. The most noteworthy use case of feature tracking, however, is the prediction of the next location of an object. For example, self-driving vehicle systems can predict where an observed car will go based on its optical flow, and adapt own actions accordingly.

Conclusion

In this report, we have discussed our implementation of the Harris Corner Detection algorithm and its behaviour with different threshold values and rotated images. We have also looked into Shi-Tommasi's modification of that method, where, in future work, we could also examine the effects of varying the threshold parameter. Next to that, it would also be worthwhile to practice with the Horn-Schunk method: with its additional smoothness constraint, it has been reported to yield even better results than the Lucas-Kanade

algorithm. Nevertheless, the implementation of Lucas-Kanade allowed us to experience how motion estimates can be applied to feature, and perhaps in future work, to object tracking.

References

[1] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005.

[2] Shi Jianbo and Carlo Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.