Computer Vision Lab 3

Maximilian Schlögel UvA
Anca-Diana Vicol VU
Benjamin Kolb UvA
Philipp Lintl UvA

1 Introduction

In this paper we will discuss the Harris Corner Detector (HCD), Optical Flow and how to combine it into a feature tracking algorithm.

Firstly we describe our implementation of the Harris Corner Detection algorithm, that is based on the given description and hints. We apply the algorithm to two images and demonstrate its functionality, as well as the influence of parameters. In the following, we show that our implementation does not lead to the same detected corners for rotated images. Finally, an alternative, that slightly extends the HCD algorithm and overcomes its shortfalls is elaborated on theoretically.

In the second part we will explain the Lucas-Kanade Algorithm and how we can use it to estimate optical flow of consecutive images. Additionally we will discuss another optical flow method which does not only rely on local assumptions for better estimation, like the Lucas-Kanade method does, but rather on global properties of the optical flow.

Lastly, we will combine our knowledge from section 1 and section 2 to develop a feature tracker. Specifically we will first apply the Harris Corner Detector to detect corners in an image and then use consecutive images to compute the optical flow of the area around the corners. This allows us to track their movements.

2 Harris Corner Detector

2.1 Question 1

The first part of this assignment deals with the Harris corner detection (HCD) algorithm. In general, the task of corner detection deals with the extraction of features within an image. A feature is considered good, if it is discrete (not a continuum like textures), meaningful, reliable and local. Good features are furtherly considered as such, if they hold as much information as possible. This is the case, if both x and y direction gradients are large. [1] The Harris algorithm detects such good corners by taking advantage of the gradients as well as the neighborhood of a point. We do not focus on the theoretic derivations, that can be seen in the assignment sheet, but rather focus which version of the HCD algorithm we implemented. If we consider equation (10) and (11)/(12) of the assignment, we notice that two possibilities arise. We decided for (12), which does not entail a rather costly eigen decomposition. To end up with detected corners in an image, we followed the given hints closely.

This algorithm was then at first applied to the `PingPong` image. After trying several specifications for both the *window size* of the neighborhood and the parameters of the Gaussian filter, the result in Figure 1 was obtained. The following parameters were set:

- window size of Gaussian filter = 9, respective sigma = 1.5
- threshold followed the formula $threshold = 100 * mean(mean(corner\_points))$
- neighborhood window size = 6

The effects of the parameters can be described as such: Increasing the window size and the sigma of the Gaussian filter led to a blurrier image and thus tendentially fewer corners, as derivatives are

Preprint. Work in progress.

smoothed. The threshold according to which a corner point is compared to other points, was set according to the heuristic of 100 times the mean of all the detected corner points. Lastly, an increasing neighborhood window size also yields less corners, as the point is compared to more values and thus less likely is a local maximum. We chose a setting that substantially reduces noise by a rather strong Gaussian filter and a large neighborhood window. These effects can be observed by adjusting these values within our harris$_{c}orner_{d}etection.mfunction$.

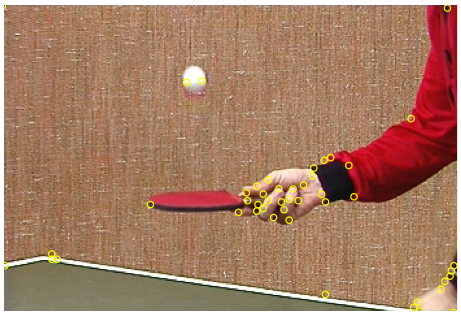

Figure 1: Corners for `PingPong` image.

Figure 2: Derivative along x.

Figure 3: Derivative along y.

We used the same parameter setting for the `Toy` image (Figure 4), as it detected corners quite well. We regard them as satisfying, as they are indeed mostly corners, few in number and except for the lego figure, represent unique local regions. Furthermore, as required in the beginning, they represent interesting parts of the image. Meaning, that textures like the table or wall are not considered, but corners as in the plug or the object on the table are detected.

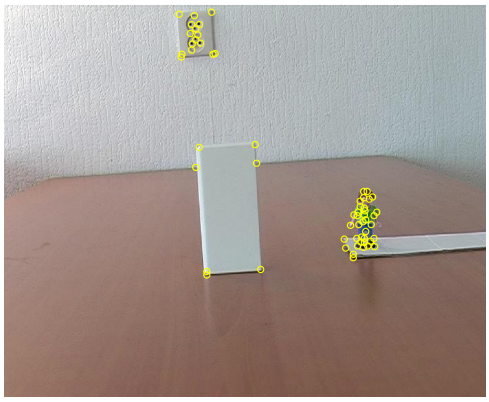

Figure 4: Corners for `Toy` image.

Figure 5: Derivative along x.

Figure 6: Derivative along y.

A general advantage of corner detection is its robustness. This also regards rotation of an image. However, our implementation does not cover this feature, as seen in the 45 degree rotated image depicted in Figure 7. It clearly shows slightly more features, as the non rotated image in Figure4. Notably, the 90 degree rotation (Figure 10) is almost identical to the non rotated image. We think that our implementation does not guarantee rotation invariance, as we do not draw on the original version, that relies on an eigen decomposition. The latter is in fact rotation invariant, as rotation only changes a matrix eigenvectors but not its eigenvalues. So rotating Q would not influence λ_1, λ_2. As demonstrated in our demo function, the dimensions of our rotated image do not match the ones of the original image. Thus, more pixels are available, which opens up more corners to be detected. We observe this effect in the 45 degree rotation.

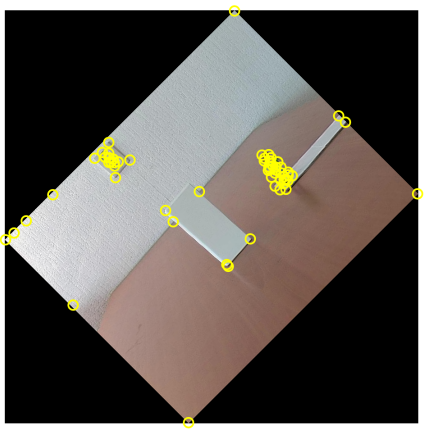

Figure 7: Corners for Toy image.

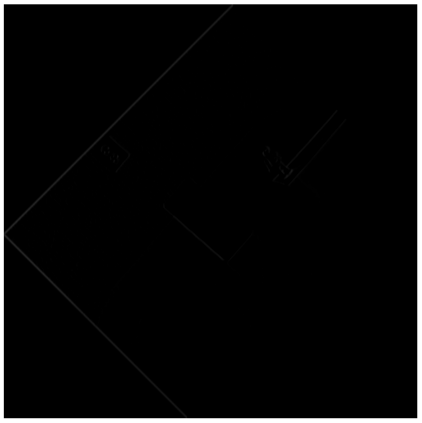

Figure 8: Derivative along x.

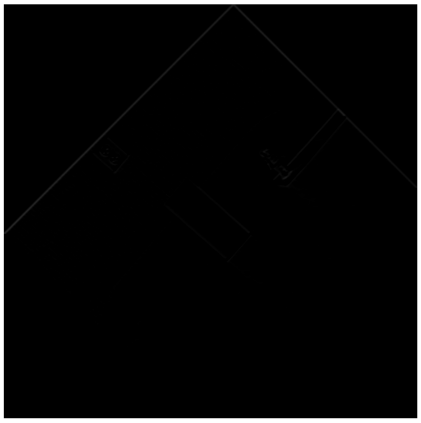

Figure 9: Derivative along y.

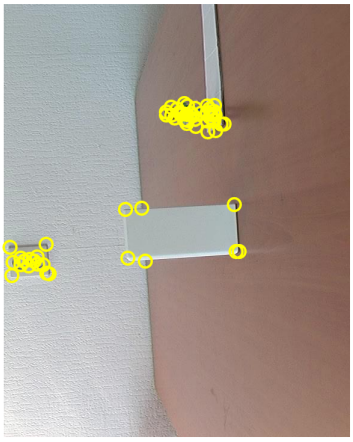

Figure 10: Corners for Toy image.

Figure 11: Derivative along x.

Figure 12: Derivative along y.

2.2 Question 2

Shi and Tomasi [3] based their *cornerness* definition almost entirely on the definition of Harris:

$$H = \lambda_1\lambda_2 - 0.04 * (\lambda_1 + \lambda_2)^2, \tag{1}$$

with λ_1, λ_2 being the eigenvalues of $Q(x, y)$. However, they introduced a small alteration to (1):

$$H = min\{\lambda_1, \lambda_2\}. \tag{2}$$

As shown in the assignment sheet,

$$Q(x, y) = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \tag{3}$$

$$\Rightarrow \quad det(Q) = \lambda_1\lambda_2 = AC - B^2 \tag{4}$$

$$trace(Q) = \lambda_1 + \lambda_2 = A + C \tag{5}$$

hold. If we now use the following property of the the $min(x, y)$ function:

$$min\{x, y\} = \frac{1}{2}(x + y - |x - y|) \tag{6}$$

$$= \frac{1}{2}\left(x + y - \sqrt{(x - y)^2}\right) \tag{7}$$

If we apply this to the eigenvalues λ_1, λ_2, the min function arises as:

$$\begin{aligned}
min\{\lambda_1, \lambda_2\} &= \frac{1}{2}(\lambda_1 + \lambda_2 - \sqrt{(\lambda_1 - \lambda_2)^2} \\
\text{(multiply square)} \quad &= \frac{1}{2}\left(\lambda_1 + \lambda_2 - \sqrt{\lambda_1^2 - 2\lambda_1\lambda_2 + \lambda_2^2}\right) \\
\text{(rewriting quadratic form)} \quad &= \frac{1}{2}\left(\lambda_1 + \lambda_2 - \sqrt{(\lambda_1 + \lambda_2)^2 - 4\lambda_1\lambda_2}\right) \\
\text{(plugging in det() and trace() from above)} \quad &= \frac{1}{2}\left(A + C - \sqrt{(A + C)^2 - 4(AC - B^2)}\right) \\
\text{(simplify by multiplication)} \quad &= \frac{1}{2}\left(A + C - \sqrt{A^2 + C^2 - 2AC + 4B^2}\right) \\
\text{(apply quadratic form again)} \quad &= \frac{1}{2}\left(A + C - \sqrt{(A - C)^2 + 4B^2}\right)
\end{aligned}$$

These transformations show, that eigen decomposition is not nececessary. Only, the underlying formulas of A, B and C have to be applied in order to end up with the H matrix.

Again, both eigenvalues λ_1 and $\lambda 2$ are assessed according to a threshold or minimum value λ. [3] So, a corner point is given, when

$$min\{\lambda_1, \lambda_2\} > \lambda \tag{8}$$

holds. In case of two eigenvalues close to zero, *relative cornerness* will be as small as the minimum out of both. Depending on the given threshold, it will possibly lay below the given λ and thus be not considered a corner point. If one eigenvalue is big and the other near zero, the situation stays the same, as only the minimum value is considered. Finally, a point is considered a corner point, if both eigenvalues are larger than the given threshold and thus high in cornerness. [3]

3 Lucas-Kanade Algorithm

In this section we will shortly discuss the Lucas-Kanade Algorithm and then portray the results of applying it

amples. The Lucas-Kanade algorithm uses the assumption of brightness constancy to estimate optical flow. This assumption means, that the brightness and color of parts of an object remains constant over time.

There are a few issuos with this assumption. First, the brightness of the same object in an image might vary, due to shadows, reflections, etc. and secondly, for very homogeneous regions it is hard to decipher the direction of flow. The second issue is partially due to the discrete origin of images represented by pixel, and also photos, taken in discrete time steps, represent discrete instances of a normally real valued dimension 'time'. These issues make it hard to directly apply the assumption of brightness constancy.

One way to deal with the problem of homogeneous regions, is the Lucas-Kanade Algorithm. By further assuming that the optical flow is constant in a local neighborhood of a pixel, the Lucas-Kanade Algorithm tries to deal with the ambiguity of the optical flow.

This idea of constant optical flow in a neighborhood is implemented by dividing the image into non-overlapping similar sized areas. In our case we have two image-pairs, one with pixel size 200x200 and one with pixel size 128x128. We use non-overlapping regions of size 15x15. This leaves an overhang of 5 pixels in the first case, and 8 in the second case. We decided to not do padding, but to ignore those slices of the image. It would be interesting to see the different results, tho, and potentially change the size of the non-overlapping region size to a smaller size or a size that is a dividor of the pixel size of the image.

The results of the application of the Lucas-Kanade Algorithm can be seen in the two following image sets in figures 13-15 and figures 16-18.

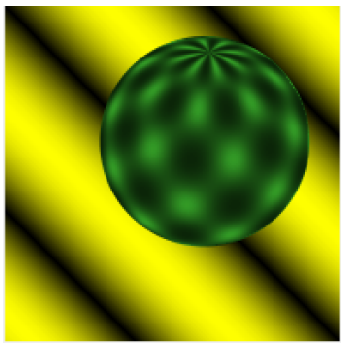

Figure 13: Image 1 of the sphere

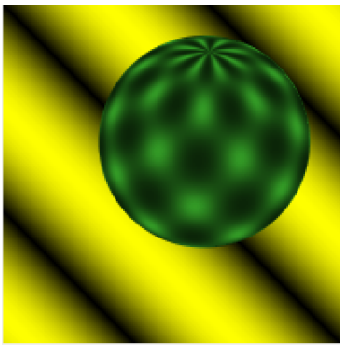

Figure 14: Image 2 of the slightly rotated sphere

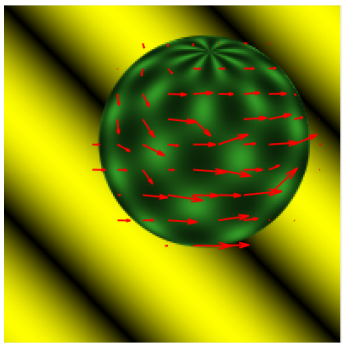

Figure 15: Image of the optical flow on top of the original sphere image

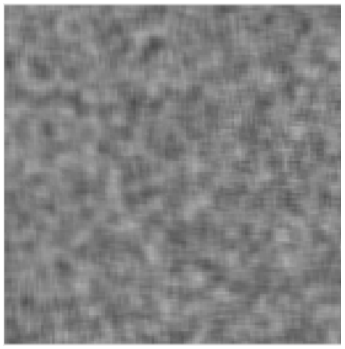

Figure 16: Synthetic image 1

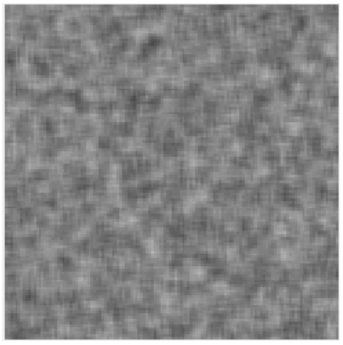

Figure 17: Synthetic image 2 slightly moved

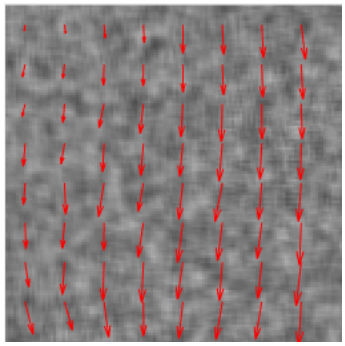

Figure 18: Image of the optical flow on top of the original sphere image

We can see on image 15 that the Lucas-Kanade algorithm seems to work well in images with a clear underlying structure and little noise. If we have no clear structure and a lot of noise, as can be seen in figures 16-18, it is really hard for the algorithm to properly estimate the optical flow. To be fair, though, it is even hard for human eyes to determine the optical flow for those noisy images.

3.1 Question 2-2-1

We will now discuss the Horn-Schunk method and compare it to the above discussed Lucas-Kanade algorithm.

As discussed above, the Lucas-Kanade algorithm operate locally. It assumes that in a local neighborhood of a pixel, the optical flow is the same. The area of influence on the estimation of the optical flow at a certain point is depending on the size of this region, which is set as a hyperparameter. But this area is just acting locally. In the method of Horn-Schunk, we do not have a local area of influence. This method operates globally. Similar to the Lucas-Kanade case, it assumes that optical flow is similar in a local neighborhood, but this time the focus lies on the smoothness of the optical flow on a global level [2]. Note: Smoothness in this case, means that there are no sudden changes in optical flow. The estimation of the optical flow for small area is also dependent on the global behavior, since the method tries to find a optical flow with high global smoothness.

3.2 Question 2-2-2

How do they behave on flat regions? The issue with a flat region is, that we do not have structure that can help us etimate the optical flow. This gets problematic for the Lucas-Kanade method, when there are larger regions, that do not have structure and therefore make it hard to find the derivatives (no structure means often the same values over time and locally constant values, both make it hard to find the derivative). Larger areas of constant values mean that we get optical flow vectors of 0 length, which is problematic. This behavior can be observed when making the region size smaller for our example above. As we can see in figure 19 there are no optical flow vectors visible in the homogeneous areas. The Horn-Schunk method, on the other hand, delivers a higher density in optical

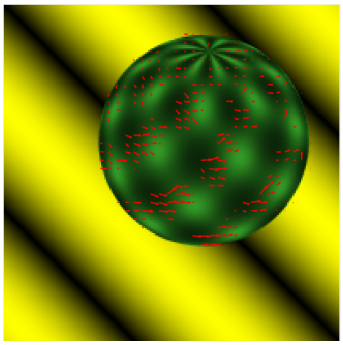

Figure 19: Smaller local regions with optical non-existent flow vectors

flow vectors, that still allow a high smoothness. While in the Lucas-Kanade method the optical flow in a region is the same, and therefore does not offer smoothness at the boundaries of those regions, the Horn-Schunk method propagates the optical flow vectors from boundery parts in the image, to the homogeneous parts of the area. This leads to a consistent vector field of optical flow vectors.

4 Feature Tracking

4.1 Question-1

We use the gif format for the videos.

4.2 Question-2

One reason to track features may be that it helps if we want to obtain an understanding of the ongoing physical processes. E.g. moving body parts could potentially be tracked over time. Another reason is that feature detection may be computationally expensive compared to optical flow estimation, in which case feature tracking helps reducing the computational cost.

5 Conclusion

The Harris Corner Detection section revealed that the desired characteristic of rotation invariance depends on the chosen implementation. In fact, our version, that is not conducting an eigen decomposition was shown to deviate from its non rotated counterpart. Furthermore, the detection of corners heavily depends on the chosen parameters of the Gaussian filter, the Gaussian derivatives, the threshold, as well as the size of the neighborhood window. More extensive hyper parameter tuning might be of interest to motivate the choice of parameters more in depth than just trying a few configurations. Nevertheless, as seen in the application, the HDP is a powerful tool, which can easily be improved with the alternative provided by Shi and Tomasi.

After implementing the Lucas-Kanade algorithm and applied it to different images, we saw that it performs quite well on images with a clear structure and no noise. But we also saw some drawbacks

of it, namely locally homogeneous surfaces and very random or noisy images. To solve the issue of local homogeneous surfaces we introduce the Horn-Schunck algorithm, which acts on a global level. It would be interesting so see actual differences in performance between those two algorithms, as we only discussed their differences theoretically.

Workload-division: Harris Corner Detector: Anca-Diana Vicol and Philipp Lintl Optical Flow: Maximilian Schlögel Feature Tracking: Benjamin Kolb

References

[1] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[2] B. K. Horn and B. G. Schunk. Determining optical flow. Technical report, Massachusetts Institute of Technology Cambridge, 1980.

[3] Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1994.