Rico Mossinkoff - 12805157
Ewoud Vermeij - 11348860
Hannah Lim - 10588973
Yke Rusticus - 11306386

Introduction

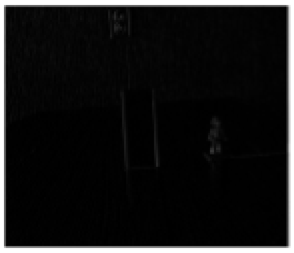
In this assignment, we will focus on several algorithms for image processing. We will first look at the Harris Corner Detector, which is an algorithm to recognize corners in images. This is important to distinguish objects and it plays an important role in object tracking. We will answer several questions to get a greater understanding of the algorithm.
Second, we will look at the Lucas-Kanade algorithm. This is an algorithm to classify optical flow. This means that it can recognize motion in different frames from a recording. We will plot this motion for several different images and see how the algorithm performs.
Finally, we will combine the previous algorithms to make a tracking algorithm. This algorithm uses both corner detection and optical flow detection to track features in an image.
We will demonstrate this with a short video for several image sets. This video will show the motion in a sequence of images.

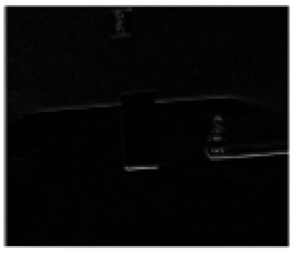
1. Harris Corner Detector (45pts)

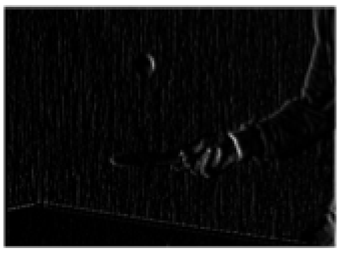
Q1.1:
See attached file "harris_corner_detector.m".

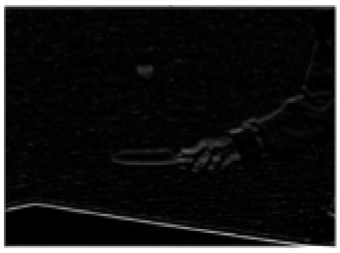
Q1.2:
The function plots both I_x and I_y in a subplot, as well as an additional figure which shows the original image in grayscale and next to it, the image where corner points are indicated by red crosses. The images below show the results for "person_toy/00000001.jpg" and "pingpong/0000.jpeg".

The following images are normalized, such that the largest value in each image is 255.

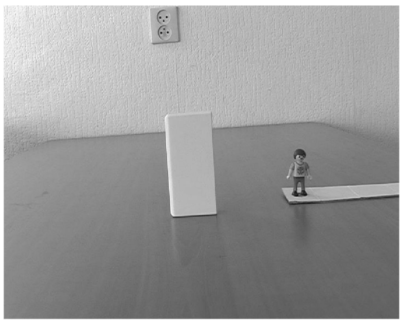

person_toy: I_x

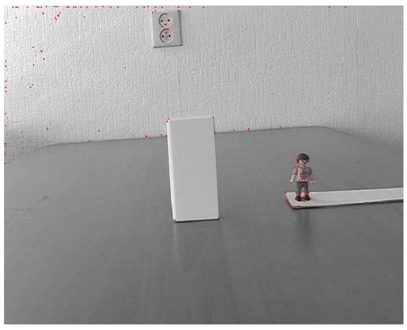

person_toy: I_y

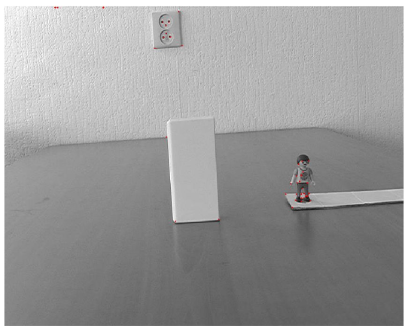

pingpong: I_x

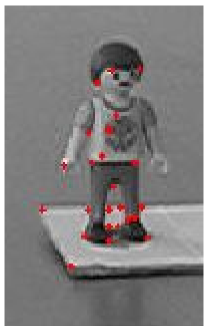

pingpong: I_y

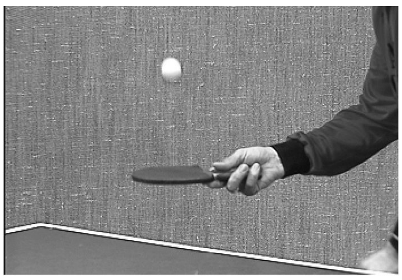
Rico Mossinkoff - 12805157
Ewoud Vermeij - 11348860
Hannah Lim - 10588973
Yke Rusticus - 11306386

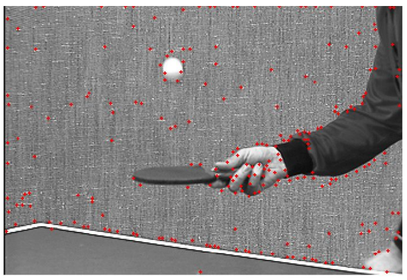

Original (grayscale)

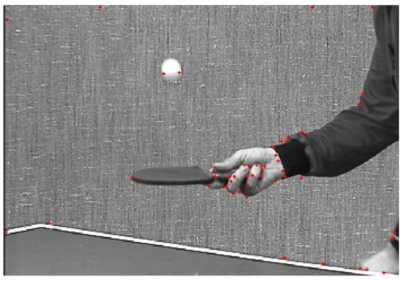

Threshold = 1000
Corners = 127

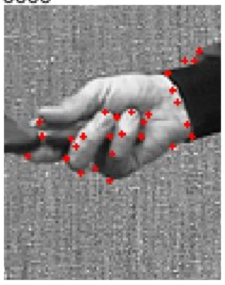

Threshold = 10000
Corners = 34


zoomed
threshold =
10000

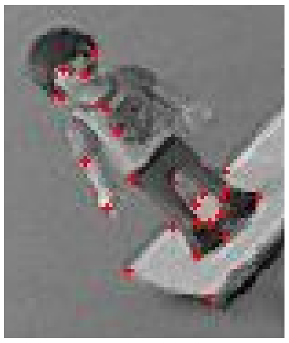

Original (grayscale)

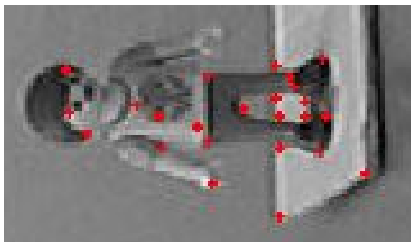

Threshold = 1000
Corners = 233

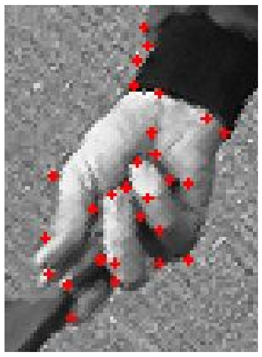

Threshold = 10000
Corners = 49

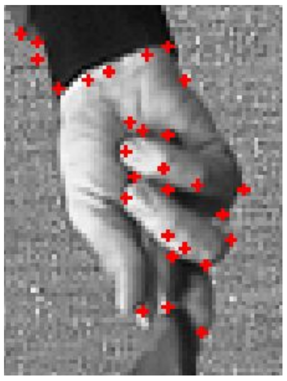

zoomed
threshold
= 10000

Q1.3:
The following images show the corner detections for the rotated images for a threshold of 10000 and these are compared with the zoomed-in images above (previous question). The indicated number of corners applies to the image as a whole (so not just the zoomed image).


Rotation = 45 deg
Corners = 44


Rotation = 90 deg
Corners = 39


Rotation = 45 deg
Corners = 71


Rotation = 90 deg
Corners = 49

We can see that the implemented algorithm is more robust against rotation over 90 degrees than over 45 degrees just by looking at the total number of corners. For a rotation of 45 degrees, the algorithm seems to find the most corners in both images. This could be due to the fact that in our implementation, we do not look for corners closer than n pixels away from the edge of the image. Here n is the size of the kernel in which we check for local minima of H. Since a rotation of 45 degrees gives you an image with zero-padded corners (not to be confused with the corners we try to detect), the edges of the original image suddenly become available for corner detection, resulting in more detected corners than in the cases of 0 or 90 degrees of rotation. The zoomed images shown above show, however, a great similarity with the original image presented in Q1.3. Though most differences can be found again in the case of 45 degrees of rotation. In this case, some corners are not detected, while in the unrotated image they are detected. So, although we may not say the algorithm is rotation-invariant based on the presented data, we may conclude that the algorithm is highly robust against rotation.

(**Q1:** Include a demo function to run your code.)
See attached file "demo_q1_1.m".

Q2.1:
The Shi and Tomasi algorithm is almost similar compared to the Harris algorithm, except for their definition of cornerness, which is defined by Shi and Tomasi as:

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

With λ_1 and λ_2 being the two eigenvalues of Q(x,y).

Q2.2:
We do not need to calculate the eigen decomposition at all since we only need to estimate the two eigenvalues for each point Q(x,y) and take the minimum between the two eigenvalues. We do expect the Shi and Tomasi definition of cornerness to be more computationally expensive compared to Harris' cornerness definition. For Harris, we first calculate I_x and I_y. Then, by using these we calculate Q, after which we can directly form the matrix H. However, for Shi and Tomasi, we add an extra step before we get to matrix H, which is finding the minimum eigenvalue of Q.

Q2.3:

a)
When both eigenvalues are near 0, the minimum of both will be near zero. Given some threshold and the high probability of not having the highest value in the area of neighboring pixels, this will not be indicated as a corner

b)
Since the minimum of both will be evaluated, it will have the same result when having both eigenvalues near 0. It will not be indicated as a corner.

c)
Here, the minimum of both eigenvalues will also be big. Now, the probability of being higher than some threshold and highest in the area of neighboring pixels is high. Therefore, the point is likely to be indicated as a corner.

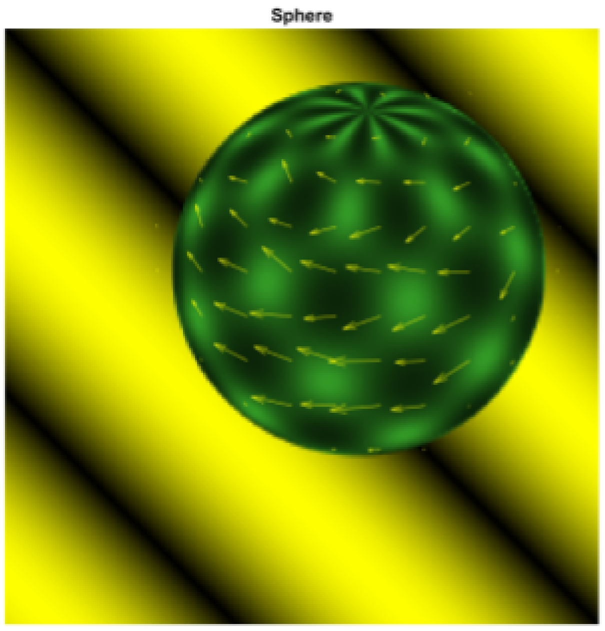
Rico Mossinkoff - 12805157
Ewoud Vermeij - 11348860
Hannah Lim - 10588973
Yke Rusticus - 11306386

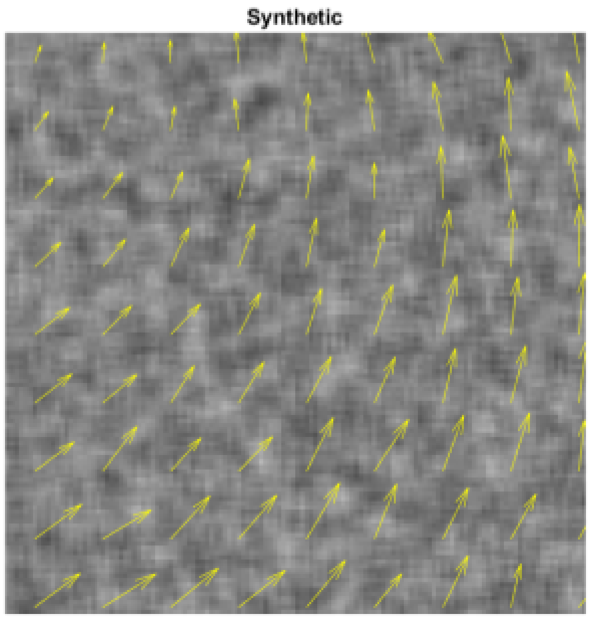
2. Optical Flow with Lucas-Kanade Algorithm (35pts)

Q1.1:
See attached file "lucas_kanade.m".

Q1.2:
See attached file "lucas_kanade.m".

Q1.3:





As we can see in the images above, the yellow lines indicate a combination of the horizontal and vertical motion of each 15x15 patch in the image, when motion is detected. The sphere is turning slightly to the left. In the synthetic image, motion is detected in all patches.

(**Q2:** Include a demo function to run your code.)
See attached file "demo_q2_1.m".

Now you have seen one of the optical flow estimation methods developed by Lucas and Kanade. There are several more methods in the literature. Horn-Schunck method is one of them. Check their method, compare it with Lucas-Kanades and answer the following questions:
Q2.1: At what scale do those algorithms operate; i.e local or global? Explain your answer.

Horn-Schunck operates at a global scale, estimating the flow everywhere. The Horn-Schunk algorithm assumes that the flow is smooth, and therefore does not handle noise well. On the other hand Lucas-Kanade operates on a local scale, estimating the velocity in a small, NxN patch of an image, assuming the flow is constant in the local neighbourhood of a pixel and therefore is less sensitive to noise.

Q2.2:
Lukas-Kanades will have more difficulties with flat regions because it operates on a local scale. Therefore it will often find very small derivatives since it looks at the local neighborhood. If the derivatives are too small, the algorithm will not notice any changes and the optical flow estimation will be inaccurate. Horn-Schunk looks globally and therefore will probably perform better (because the derivatives are bigger when measuring globally).

3. Feature Tracking (20pts)

Q1.:
See attached file "tracking.m".

(**Q1:** Include a demo function to run your code.)
See attached file "demo_q3_1.m".
For the person_toy image sequence we would expect the corners from the person toy to have a velocity only in the horizontal direction to the left of the window and the corners of the box in the middle of the image and the powerpoint in the wall not to have any velocity in whichever direction. In the results for person_toy, we can see that both the box and the power outlet in some cases give a velocity, which is incorrect. Also in some cases, the person toy has a velocity in the wrong direction (both horizontally and vertically wrong). In the video it can be observed, that these incorrectly computed velocities from the corner points for the box and the power outlet are correlated with the incorrect velocities of the person toy. This can be explained by the fact that the constant brightness assumption is violated with some images, at some corners. This will have some influence on the calculated velocity of some non-moving corners as well as the moving corners.

For the ping pong image sequence we would expect a velocity for the hand and the ping pong ball vertically. In this case, we have set a lower threshold, because the corners of the round-shaped ping pong ball, are slightly harder to detect. In the results, we can see that the velocity computed for the hand is in line with expectations, but that the velocity for the ball has some issues. This seems to be caused by the speed the ping pong ball is moving in the sequence of pictures. Also, we can see that the corners of the ping pong table have an incorrect velocity, which is perhaps caused by the fact that the image is zoomed out. Although zooming out does not seem to have an effect on the velocity of the corners of the hand.

To deal with the high motion of objects between images, we could use the coarse-to-fine method. This method simply builds image pyramids in such a way that each pixel on the top layer of the pyramid represents multiple pixels of the original image. Therefore, the high motion gets also reduced to small motion and gives the algorithm a chance of performing better at the velocity detection.

Q2:
With feature tracking, we can follow the same feature in different frames. If you detect features for different frames without tracking, an object might be recognized as another object than the previous frame. By tracking it, you know for sure that you are looking at the right thing. Also, with tracking, you can measure the motion of an object, and the magnitude of this motion. Just measuring features does not give you any information about the movement.

Conclusion

The goal of this assignment was to learn more about feature recognition and tracking. We learned that the Harris Corner Detector is a great tool to recognize corners in images. There is no specific threshold that will always result in the best result when running the algorithm. In some images with big contrast, the threshold could be small to avoid false corner detection. In other lower contrast images, however, this might not result in a good performance. The Harris Corner Detector algorithm seems to be quite robust against image rotation. Our results showed some differences, but these can be explained by the zero paddings that are inserted by Matlab when rotating an image. Another method very similar to Harris corner detection is the method from Shi and Tomasi, which measures cornerness with the minimum value of the two eigenvalues of Q(x,y). Although we didn't test this algorithm, other sources indicate that the method of Shi and Tomasi performs better than Harris corner detection.
We also learned that the Lucas-Kanade algorithm proves to be very useful when estimating optical flow. It might not be very suitable for estimating motion for flat surfaces, because it makes use of the gradient in the x and y direction, which will be 0 or close to 0 for flat surfaces.
Finally, we had some hands-on experience with feature tracking. Here we combined what we learned before and experienced the use of the algorithms with a realistic example. This helped us in gaining deeper understanding of feature tracking.