Assignment 3: Harris Corner Detector & Optical Flow

Davide Belli
11887532
davide.belli@student.uva.nl

Gabriele Cesa
11887524
gabriele.cesa@student.uva.nl

Introduction

In this assignment, we are going to discuss and experiment with Harris Corner Detector, an algorithm aiming to identify important features in images such as corners in the objects. Then, we will be analyzing Optical Flow in images (defined as the apparent flow in pixels from one frame to the other) using Lucas-Kanade Algorithm. Finally, we will combine the two and use them to study the movement of objects by focusing on the changes around relevant feature points. As a final result, we will be able to produce image animations showing the Optical Flow of different parts of the scene as time passes.

1 Harris Corner Detector

1.1

In Fig. 1 we show the output of Harris Corner Detector on sample images. The number of corners in the images were 28 for the *person toy* and 18 for *pingpong*. For both images, we used a Gaussian derivative filter with size 5×5 and $\sigma = 1$, while the size of neighborhood matrix was also 5×5. The threshold value for the H matrix for pixels to be corners was set to 0.2 for *person_toy/00000001.jpg* and to 0.003 for *pingpong/0000.jpeg*.

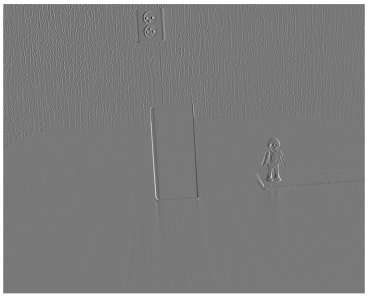
If we rotate a sample image, as shown in Fig. 2, Harris Corner Detector may find a different set of corners. This is only because, when considering the neighborhood matrix of dimension $n \times n$, pixels in the corner areas are different (unless we rotate by a multiple of 90 degrees). In our example, we used an algorithm which fills the background left empty by the rotation using pixel colors from the edges of the image. In particular, the number of corners detected in this case was 34, while in the original image it was 28. Anyway, with an ideal kernel that looks at the same pixels for both the original and the rotated image, the algorithm should find the same corners. An intuition behind the invariance of the Harris Detector to rotation is that, by combining orthogonal derivatives (on x and y axes) in the autocorrelation matrix, both the values found in the original and rotated images describe the same information. In particular, if we think at the images as continuos functions over the 2D space, the derivatives computed on an image along an angulated axis can be also described with some linear combination of the derivatives on x and y axes.

1.2

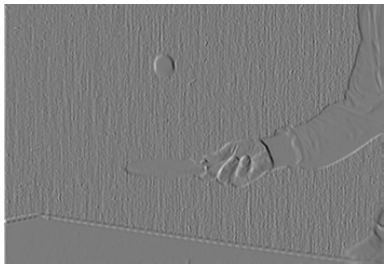
The difference between Shi-Tommasi definition and Harris' one is that the former use as *cornerness* function:

$$H = \min(\lambda_1, \lambda_2) \tag{1}$$

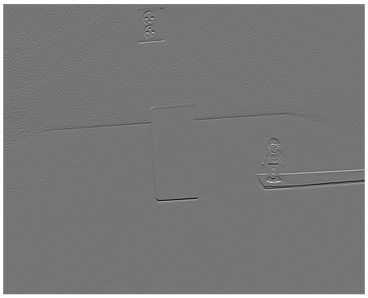
Unfortunately, this requires to compute the 2 eigenvalues explicitly for each point to evaluate. We can not compute the eigenvalues of the whole image because having only one value (a pair) for all the pixels would not be useful to discriminate pixels themselves. An useful improvement to save



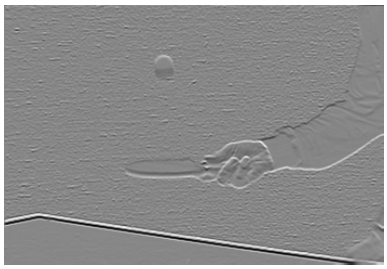
(a) Image derivative I_x



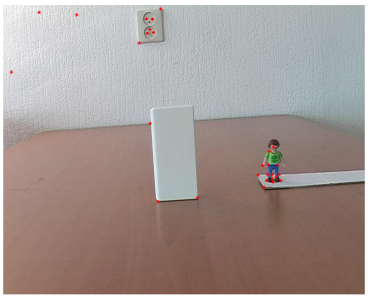
(b) Image derivative I_x



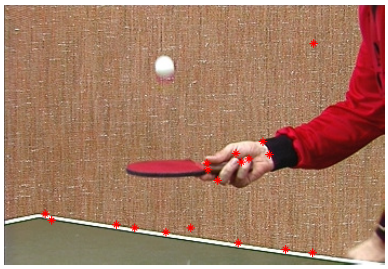
(c) Image derivative I_y



(d) Image derivative I_y



(e) Corners detected in *person_toy/00000001.jpg*



(f) Corners detected in *pingpong/0000.jpeg*

Figure 1: Harris Corner Detector outputs and Image derivatives on sample images

computations is to calculate the Harris cornerness for each point and, if above a threshold, then compute the eigenvalues and the Shi-Tommasi one.

Using Shi-Tommasi, when both the eigenvalues are close to 0, the cornerness will be very small indicating a flat region (the derivatives are very small in all the directions). When only one of them is close to zero whereas the other one is large, the cornerness will still be very small (since the minimum is taken). This case could happen in edge regions (in one direction the derivative is large but in the other it is not). Conversely, when both the eigenvalues are large, the cornerness will be large too, indicating a corner point (in that case the derivative is large in all the directions, which means this is a local maximum). For instance, see Fig 3 for a comparison between the two algorithms.

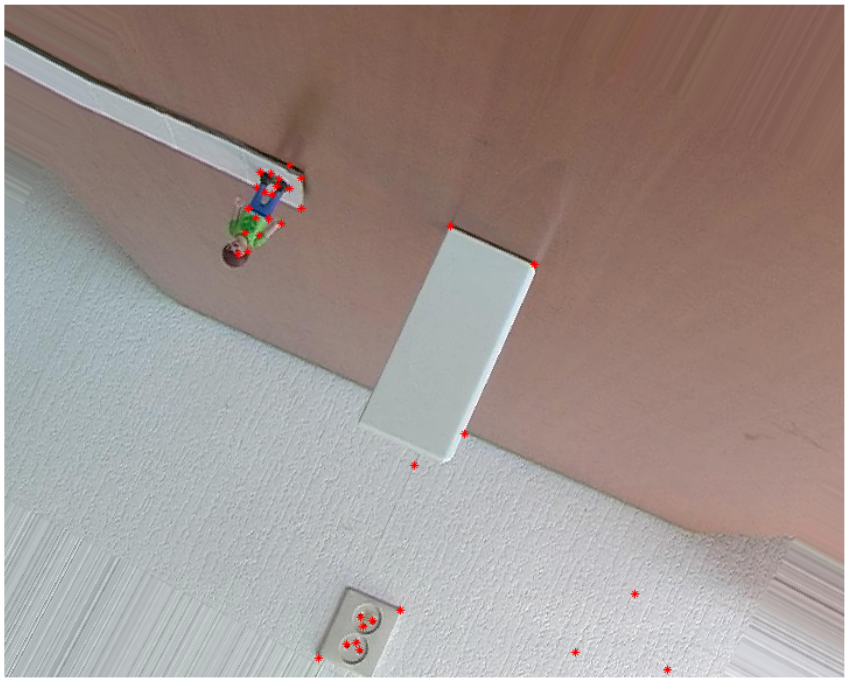


Figure 2: Corners Detected on the rotated image may differ from the original one

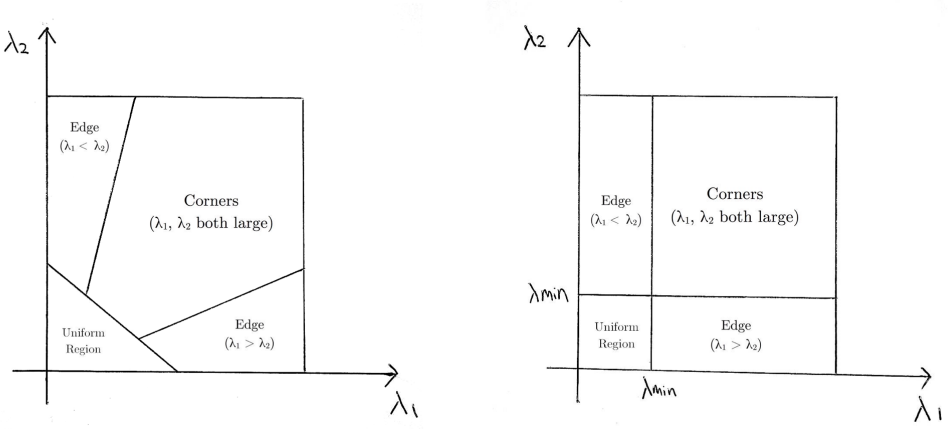
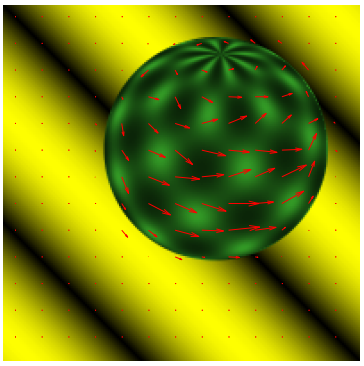


Figure 3: Different classification of points using Harris (left) and Shi-Tommasi (right) algorithms

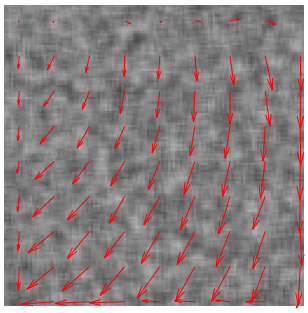
2 Optical Flow with Lucas-Kanade Algorithm

2.1

In Fig. 4 we show the outputs of applying Lukas-Kanade algorithm for movement tracking onto two pairs of images.



(a) Movement tracked on *sphere.ppm*



(b) Movement tracked on *synth.png*

Figure 4: Lukas-Kanade algorithm tracking movement between pairs of sample images

2.2

An alternative to Lukas-Kanade method is Horn-Schunk algorithm. This algorithm acts at global scale trying to minimize an energy function defined as:

$$\int\int |I_1(x+\mathbf{u}(x,y), y+\mathbf{v}(x,y)) - I_2|^2 + \lambda(|\nabla\mathbf{u}(x,y)|^2 + |\nabla\mathbf{v}(x,y)|^2)dxdy$$

where I_1 and I_2 are the two frames and $\mathbf{u}(x,y)$ and $\mathbf{v}(x,y)$ are the 2 components of the flow. In other words, Horn-Schunk algorithm searches the best flow that changes one frame to the other, preferring those flows which change smoothly throughout the images (thanks to the regularization term in the integral that make up the laplacians of the components of the flow).

Conversely, Lukas-Kanade algorithm acts at local scale considering non-overlapping windows and computing an unique flow vector for each of them.

Lukas-Kanade method might have problems in case the window considered includes only a flat region. In that case, under small translations the content of the window changes only slightly (or does not change at all) and, so, this method is likely to interpret it as no movement.

Instead, Horn-Schunk algorithm can better deal with this problem. If there is at least some region moving on the image, it will be able to compute a good flow in that region and, as it minimizes the changes in the flow, the resulting flow of the rest of the image (or at least the closer area) will be similar to the one in the moving region.

3 Feature Tracking

3.1

In Fig. 5 and 6 you can see some frames from the animation of movement tracking implemented in this task. At the first frame, feature points (corners) are computed. Then, their movements from one frame to the next one are found with Lukas-Kanade and plotted as vectors over every image. In the table tennis scene, it is noticeable that in the last frames, when the camera zooms out, feature points at the edge of the table are classified as moving towards the center (or getting further from the point of view). In fact, zooming in or out with the camera violates one of the assumptions that must be made to have meaningful results with this algorithm, namely that there is no transformation in the point of view (zooming, rotation, translation) during the sequence of frames. Concerning the implementation of this task, we tried two different approaches. At first, we split the image in regions of 15×15 pixels, computing the flow between frames just as in task 2. Then, we defined the flow in feature points as the flow of the region they belong to (see files: *person_toy_fixed_patches.avi* and *pingpong_fixed_patches.avi*).

The second implementation, used to generate the outputs shown in the report, is based on in considering the neighbors centered in each feature point (15×15 pixels patches centered on the features) and computing the optical flow for that patches (see files: *person_toy_moving_patches.avi* and *pingpong_moving_patches.avi*). Similar results were obtained with both algorithms, with the first one being sometimes slightly more accurate

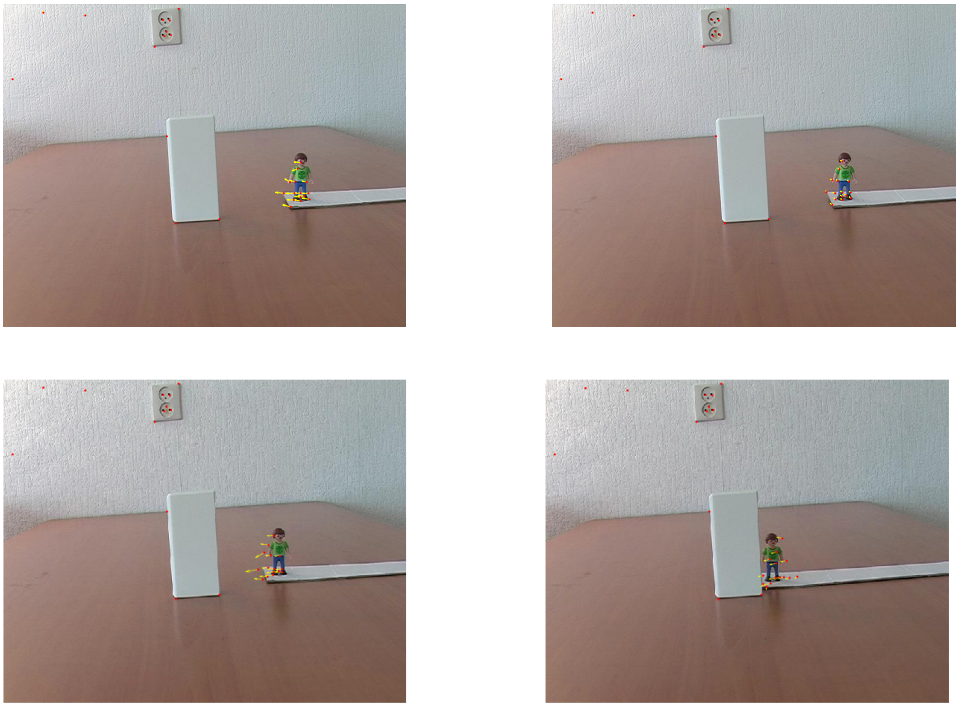


Figure 5: Frames from motion tracking for the *toy_person* scene

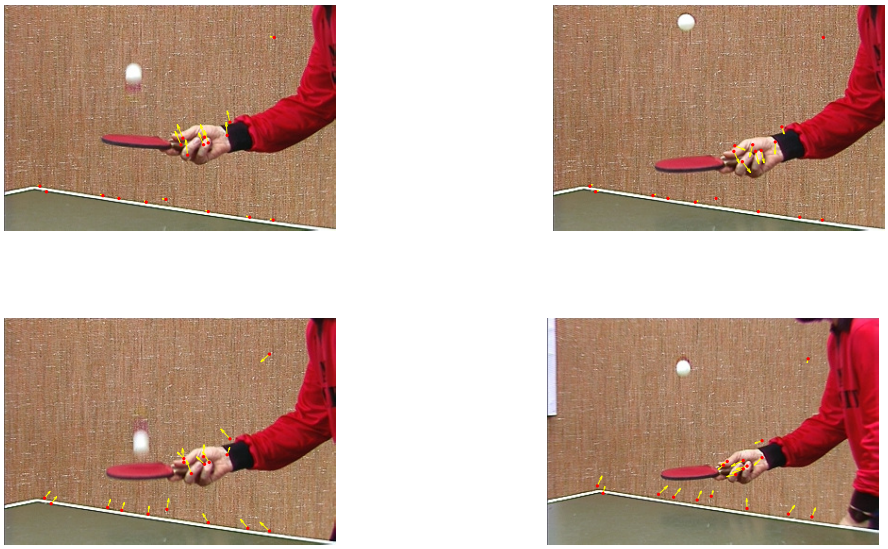


Figure 6: Frames from motion tracking for the *pingpong* scene

3.2

Feature tracking can be more useful than detecting features at each frame because it enables us to track single features: given the initial set of features, its is possible to follow their movement through the frames and trace the features of a frame back to the ones of any previous frame. Conversely, computing new features at each step does not allow us to track which feature in the new frame was which in the previous one. Moreover, it is not even necessary that the features found in two different frames will be the same, making things harder. This approach is particularly useful when tracking movements in scene with similar objects. This is because the matching of future points between the same objects appearing in consecutive frames may result inaccurate due to the presence of similar features in the other objects. An example of this can be tracking the movement of some people in a crowd with similar clothing.

Conclusion

In this assignment, we have seen how Harris Corner Detector is able to correctly distinguish corner points from normal edges. However, an accurate tuning of threshold, sigma and neighborhood size is needed to filter out duplicated or irrelevant corner from the set of detected ones. In particular, we discussed as one of the strength points of Harris Corner Detector is its invariance to image rotations. Finally, we compared it with the definition of cornerness proposed by Shi and Tommasi. When experimenting with Lucas-Kanade, we were able to identify with enough accuracy the apparent movement detected in different regions of the image, noticing how the vectors appear connected each one with its neighbors, resulting in a sort of flow, or vector field. Additionally, we compared this algorithm with another one cited in literature, Horn-Schunck. A flaw of this approach is the impossibility to detect movements in flat, homogeneus region of the image. For the feature tracking task, we initially computed the feature points using Harris Corner Detector, tracking them in the next frames using Lucas-Kanade algorithm. As a result, we were able to produce animations containing the optical flow of feature points at each time step.