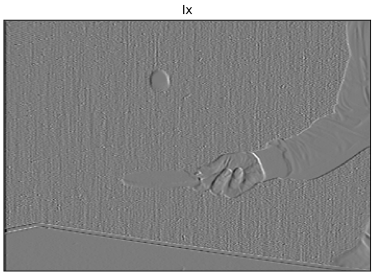
1 Introduction

In this assignment we focus on corner detection and optical flow tracking. Firstly, we implement Harris corner detection algorithm. We tune parameters for two images and test rotation invariance for our algorithm. Secondly, we focus on Lucas Kanade Algorithm to track two images' optical flow. Finally, we combine Harris corner detection and Lucas Kanade Algorithm to track the corners in consecutive images.

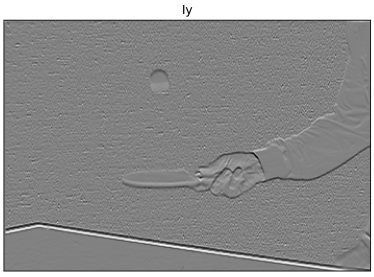
2 Harris Corner Detector

Question 1:In this section, we implement the Harris corner detector algorithm. There are several parameters in this algorithm that need to be tuned, which are the Gaussian filter scale σ and kernel size k, the window size n that is used to identify local maxima and the threshold of recognizing corners. Firstly, we tune the Gaussian filter parameters. After several experiments we found $\sigma = 1.5$ and $k = 5$ to yield decent results. Window size n is how many surroundings pixels are used to determine the corners. We try $n = 3, 5, 7, 9$ on images and we find out $n = 9$ have the best result.

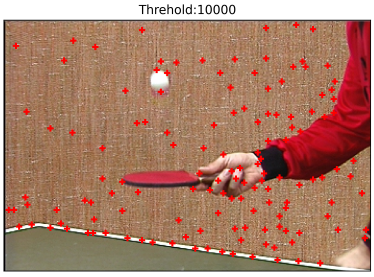
Figure 1 and Figure 2 shows two images gradients I_x and I_y, and the original image with detected corners. From the gradient images we can clearly see that only vertical or horizontal textures are left in I_x and I_y, respectively. Additionally, it is clear to see that the threshold 10000 is not ideal for both images. There are many corners on the wall which are all false positive. Thus, we need to tune the threshold parameter for these two images.



(a) I_x of pingpong image

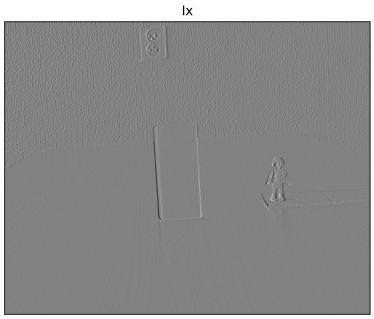


(b) I_y of pingpong image

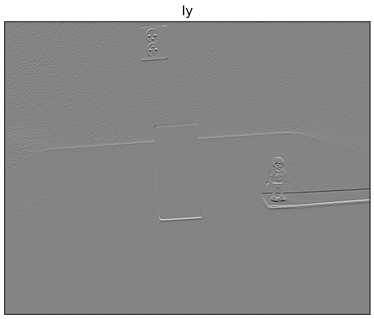


(c) Corners of pingpong image and threshold = 10000

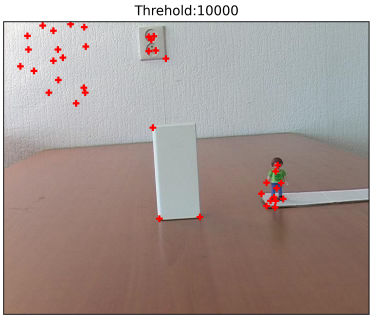
Figure 1: (a-c) are ping pong image derivatives I_x and I_y, and the original image with the corner points plotted on it. $\sigma = 1.5, k = 5, n = 9$. Threshold =10000.



(a) I_x of toy image



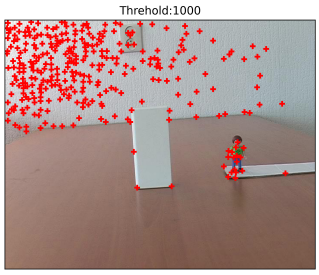
(b) I_y of toy image



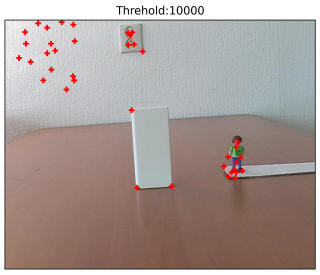
(c) Corners of toy image and threshold = 10000

Figure 2: (a-c) are toy image derivatives Ix and Iy, and the original image with the corner points plotted on it. $\sigma = 1.5, k = 5, n = 9$. Threshold =10000.

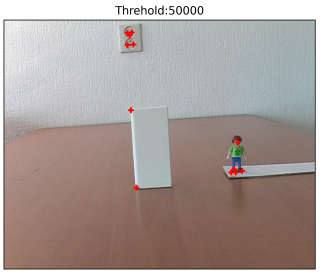
Figure 3 shows four different results for varying thresholds of the toy image. It is clear to see that increasing threshold parameters improve the performance of corners detection. The detected corners on the wall are eliminated when threshold is bigger than 50000. And when comparing Figure 3(c) and Figure 3(d) we can see Figure 3(d) loses many true corners in the image. Thus, for toy images 50000 threshold is the best.



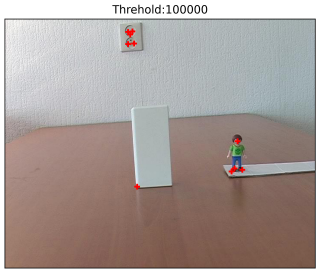
(a) Threshold = 1000



(b) Threshold = 10000



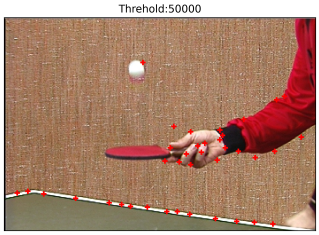
(c) Threshold = 50000



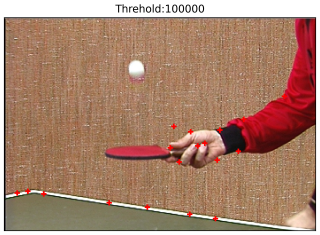
(d) Threshold = 100000

Figure 3: Threshold parameter tuning for toy image. And threshold is range from 1000 to 100000.$\sigma = 1.5, k = 5, n = 9$.

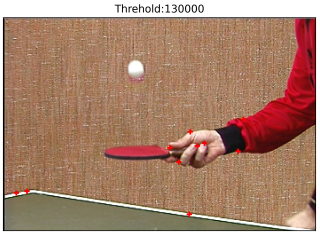
Furthermore, we explore the ping pong image. In Figure 4(a) we can see that the corners on the brown wall are eliminated when the chosen threshold is bigger than 50000, but there are still some wrong results along the edge of table. Figure 4(c) shows that the result for a threshold of 130000 is decent. There are only two small errors along the table. If we further increase the threshold we will lose true corners on the hand (see Figure 4(d)). Thus, a threshold of 130000 is the best for the toy images.



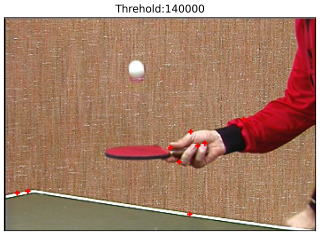
(a) Threshold = 50000



(b) Threshold = 100000



(c) Threshold = 130000

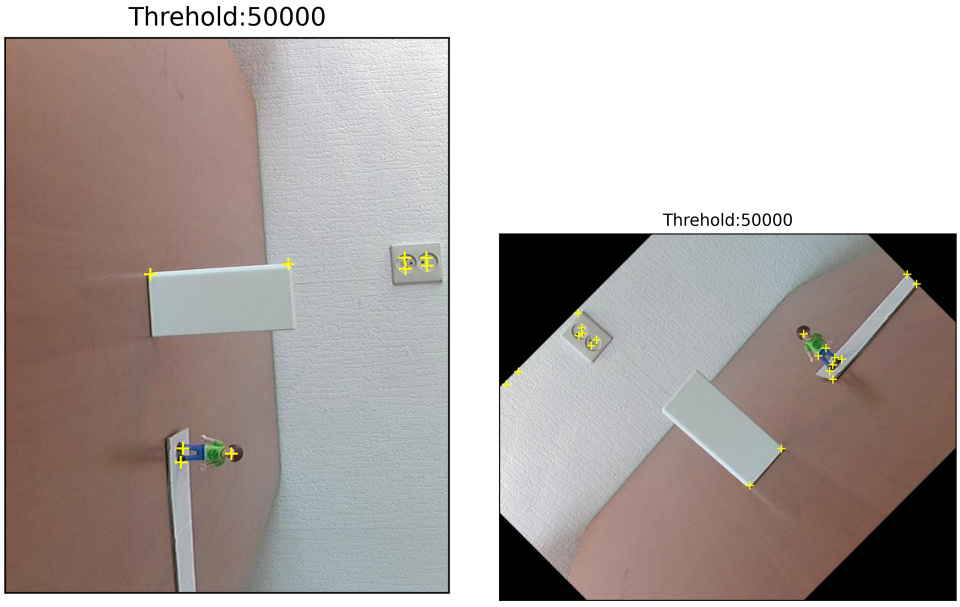


(d) Threshold = 140000

Figure 4: Threshold parameter tuning for ping pong image. And threshold is range from 50000 to 140000.$\sigma = 1.5, k = 5, n = 9$.

The Harris corners detection is a rotation-invariant algorithm. That's because when the image is rotated the shape still remains the same, which means the eigenvalues are still the same. The only change in rotation is the orientation of the gradient. Thus, Harris corners detection is rotation-invariant.

From Figure 5(a) we can see that when images are rotated 90 degree the result is still decent and the same as without rotation(see Figure 3(c)), which further shows that Harris corners detection is rotation-invariant algorithm. When images are rotated 45 degree though, the results are not identical as Figure 3(c) shows. We can see that there are some new false positives along the edge of images because of 0 padding. The possible explanation is that when we rotate the image with 90 degree the nine neighbors we chosen are still the same. Thus, the final result is consistent with Figure 3(c). But when we rotate the image with 45 degree, we can see that the new image has many blank areas around the edge of the true image. The nine neighbors are changed because of rotation, which significantly influences the pixels around the edge of true image because we compare the value of the centered pixel with its eight neighbors. When the centered pixel is located in the edge of true image, most of neighbors pixels are zero (pure black area). Thus, the center pixels can be easily chosen as corners.



(a) 90 rotated toy image

(b) 45 rotated toy image

Figure 5: Results of corners with rotated toy images. (a) is rotated 90 degree image and (b) is rotated 45 degree.

Question 2: The Shi and Tomasi algorithm is quite similar to Harris algorithm. The only difference is the way a corner is identified. Shi and Tomasi calculate the corner by the following equation:

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

where λ_1 and λ_2 are eigenvalues of the auto-correlation matrix for each pixel. These two eigenvalues still need to be bigger than certain a threshold. Then, the smaller eigenvalue is chosen. We need to calculate eigenvalues for each pixel in Shi and Tomasi algorithm, while for Harris we don't need to this.

Shi and Tomasi algorithm still satisfies translation invariance, rotation invariance but not scale invariance. As we pointed out before, Shi and Tomasi algorithm is similar to Harris algorithm. Translation and rotation don't change the shape of an image, which means the eigenvalues are still the same as before. The only change happens in orientation of the derivatives. But as Figure 6 shows, when the scale changes, the edges are detected as one corner with the same window size, which means the eigenvalues are different from the original scale.

As for the three scenarios:

- If both eigenvalues are near 0, the cornerness response will be low because the eigenvalues need to be bigger than threshold. Thus, it will not be classified as a corner.
- If only one eigenvalue is big, the cornerness response will still be low. Because the algorithm picks the smaller eigenvalue which needs to be bigger than the threshold. Thus, it will still not be classified as a corner.
- If two eigenvalues are big, the cornerness response will be high.

Figure 7 also verifies our statement. We can see only when both eigenvalues are bigger than the threshold (green area), this pixel will be considered as corner.

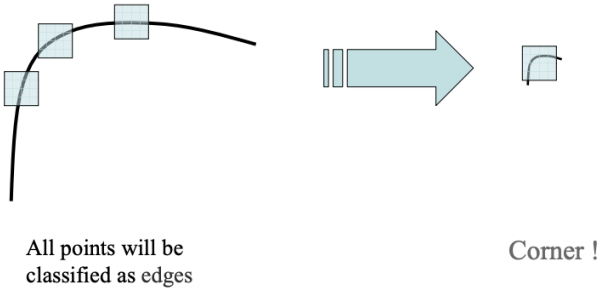


Figure 6: Scale invariance of Shi and Tomasi algorithm (taken from [2]).

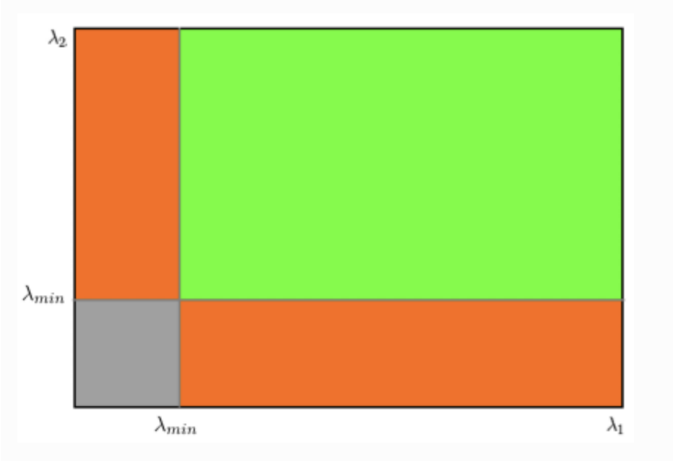


Figure 7: $\lambda_1 - \lambda_2$ plot: Only if both eigenvalues are bigger than the threshold (green area), the pixel will be considered as a corner (taken from [3]).

3 Lucas Kanade Algorithm

In this part of the assignment we are dealing with the Lucas Kanade algorithm that is used to compute the motion of pixels in a sequence of images. We implemented and applied this algorithm on the given sphere and synth images. The results are shown in Figure 8 and 9. For the sphere image, the background is static which we can see from the vectors being zero.

The optical flow of the sphere indicates a clockwise rotation. Since the velocity of pixels at the equator of the sphere have the highest velocity, the optical flow vector is the largest at this point, while the contrary is the case for pixels located at the poles.

From the optical flow vector field for the synth sequence, we can derive that the motion is a planar rotation, i.e. we don't have any static patches in this sequence.

While the Lucas-Kanade algorithm assumes a constant optical flow within a local neighbourhood of pixels, the Horn-Schunck method assumes the optical flow to be smooth over the whole image, i.e. Lucas-Kanade operates on a local scale and Horn-Schunck on a global scale. Equation 2 shows the equation of Horn-Schunck. It is clear to see that the first item is identical to Lucas-Kanade method. Then this method incorporate the second item: smoothness constraint, which will find the smoothest flow in the whole image.[1]

$$E = \int\int (I_x V_x + I_y V_y + It)^2 + \lambda(\|\bigtriangledown V_x\|^2 + \|\bigtriangledown V_y\|^2) dxdy \tag{2}$$

Furthermore, Lucas-Kanade can't deal with flat regions since the matrix A does only contain zeros for them. In this case one can't calculate $(AA^T)^{-1}$, i.e. the optical flow isn't defined here. With the Horn-Schunck method on the other hand, it is possible to compute the optical flow even for flat regions since all pixels in the image contribute to the optical flow at one point, i.e. even if the gradient is zero in this point one can still derive an optical flow.

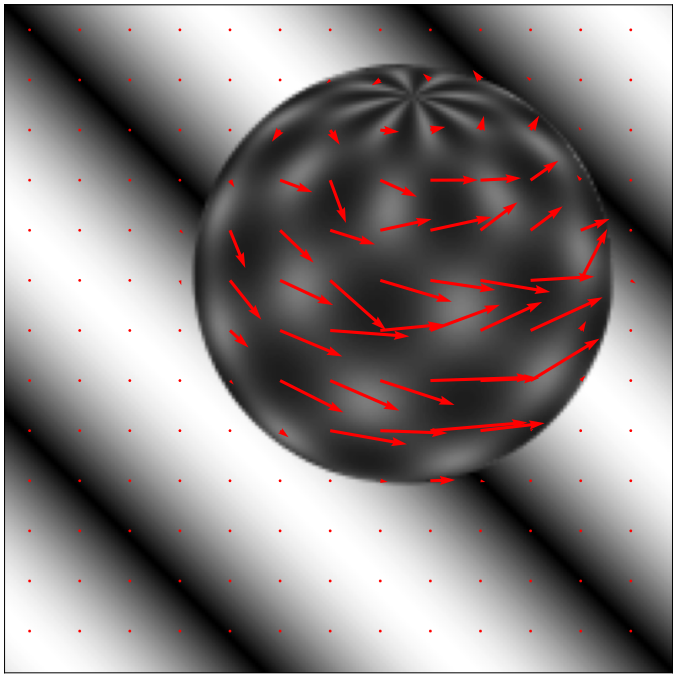


Figure 8: Visualization of the optical flow of the sphere sequence using the Lucas Kanade algorithm.

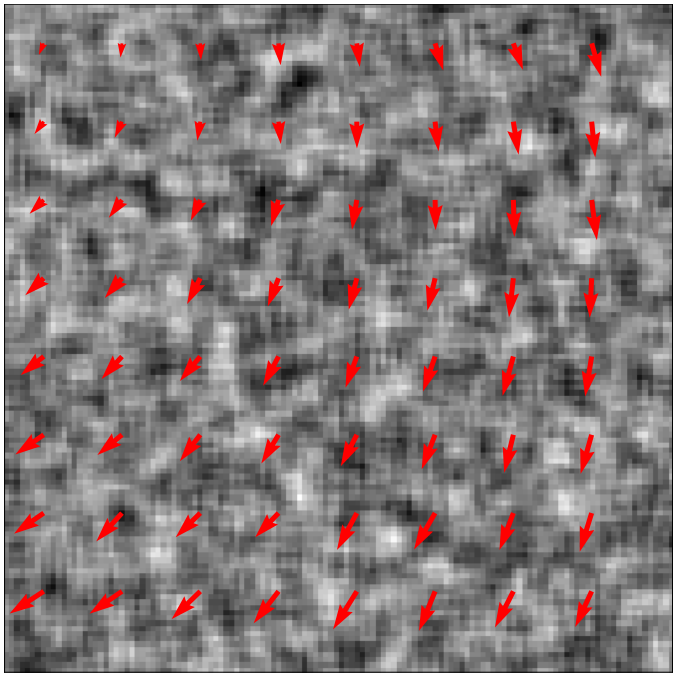
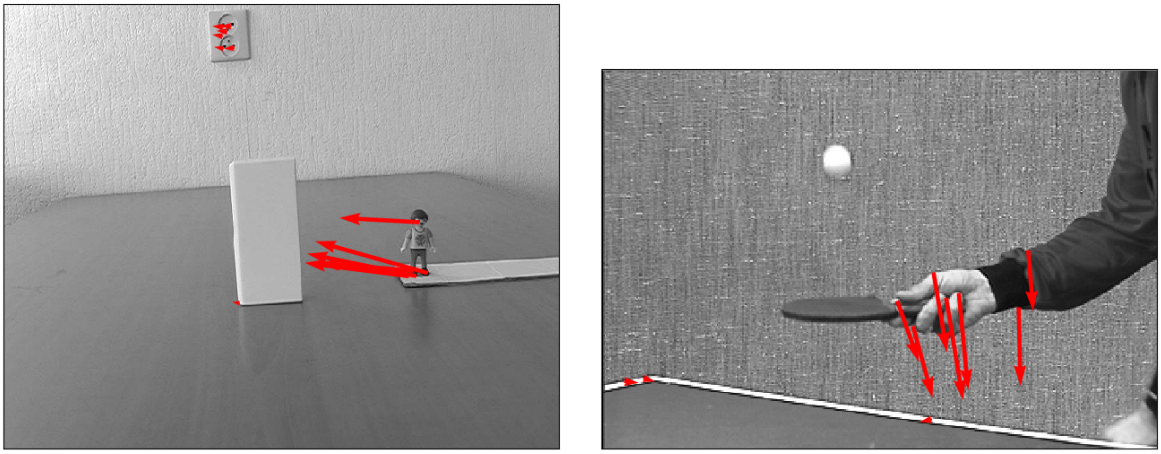


Figure 9: Visualization of the optical flow of the synth sequence using the Lucas Kanade algorithm.

4 Tracking

In this part we combine Harris corners detection and Lucas-Kanade algorithm to do feature tracking. Figure 10 shows the first frame of tracking. Firstly, we use Harris corners algorithm to locate the corners. Then, we use Lucas-Kanade algorithm to do the optical flow tracking. For toy images we find out the speed of moving pixels is quite different of each frame. And steady points in image also have small vectors to move because of computational error. Thus, we round these small errors to zero which makes the steady points stay still. And for actual moved points we use V_x and V_y multiply by 2 which will amplify the movement and compensate the different speed of each frame. The DIF of toy image is in Zip file. For ping pong images it is tricky to make the steady points stay still because of computational error of least square. We use V_x and V_y multiply by 3 which will amplify the movement and compensate the different speed of each frame. But the result shows the steady points still move between the frame. The GIF of ping pong image is also in Zip file.

For the tracking code, we first use tracking function to make all the points and save them in corresponding file. The we read all these plots and make a GIF. To run the demo, first you need to run the tracking function and then run the create_gif function.



(a) Tracking of toy image

(b) Tracking of ping pong image

Figure 10: Results of corners with rotated toy images. (a) is rotated 90 degree image and (b) is rotated 45 degree.

The first reason is that the process of Harris corner detection is quite computationally expensive. If the number of image is big, the whole tracking process will take long time to finish. Another reason is the new detection of new images may yield new corners, which will not match the old corners. In this way, we may lose or add new corners in process.

5 Conclusion

This assignment gives us insight to deal with corner detection and optical flow tracking. We implemented Harris corner detection but our results of rotation invariance are not ideal. For Lucas Kanade Algorithm our results shows decent optical flow. And when we combine these two method for video tracking the result are also decent.

References

[1] Horn. https://fzheng.me/2015/03/25/optical-flow/.

[2] Shi. https://courses.cs.washington.edu/courses/cse576/06sp/notes/HarrisDetector.pdf.

[3] Shicorner. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html.