Computer Vision 1-Assignment 3

Michael Mo
University of Amsterdam
michael.mo@uva.student.nl

Changxin Miao
University of Amsterdam
changxin.miao@uva.student.nl

1 Introduction

In this assignment, we are going to explore different techniques to extract important features from an image. We start with experimenting with harris corner algorithms to detect corners in am image. Subsequently, we implement Lucas-Kanade algorithm to estimate Optical Flow of an image. At last, we combine both algorithms into a feature-tracking algorithm, with which we could detect important features such as corners in an image and track its movements.

2 Harris Corner Detector

2.1 Question 1

1
Following the instruction, we build up the algorithm according to harris corner detector to locate corners in the image. At first, edges of image are also classified as corners. This could be due to the zero padding option in the $conv2$ function. As a result, the drastic change on the edge for "Ix" or "Iy" might contribute to high gradient. As long as there is also certain change on the other axis, the pixel might be identified as a corner.
2
We implement the Harris corner detector on the $'persontoy/00000001.jpg'$ and $'pingpong/0000.jpeg'$. During the experiment, we found that as the sigma value of the Gaussian filter increases, the threshold we set for detecting the corner becomes lower. This is due to the fact that as the image become more blurred, the eigen values also decrease. The threshold that we set up for these two images are different. For the person toy image, we set the threshold as the 10 times mean value of the Harris matrix. Results of these images will be displayed in the figure 1.

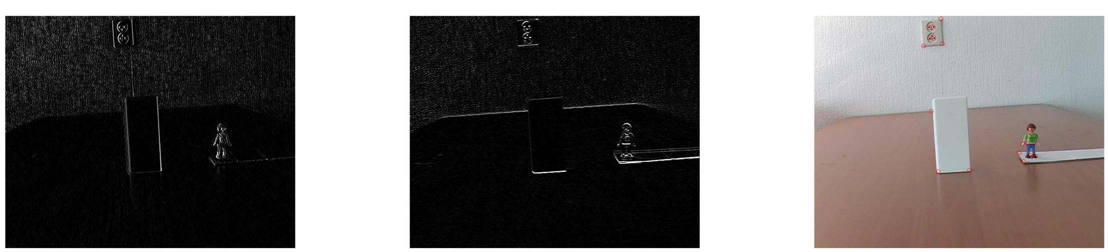

Figure 1: The left, middle and right image correspond to the Ix, Iy and original image with corners

The ping pong image requires higher threshold than the toy image, it will be 30 times the mean value of the matrix. It could be visualized as the images in the figure 2.

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

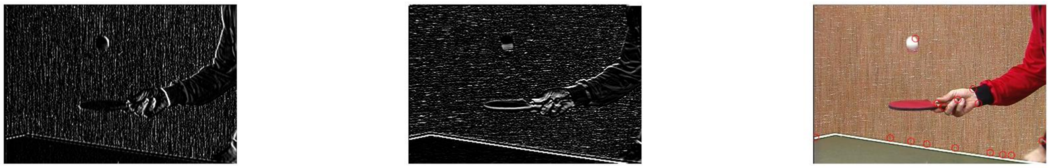

Figure 2: The left, middle and right image correspond to the Ix, Iy and original image with corners

3

The image is rotated with the "imrotate" function in matlab with the option "bilinear". As we could observe from the below images, harris corner detector is rotation-invariant. Harris corner detector works on images with different angles are displayed by figure 3. The edge still causes some problem on the rotated image, we assume that is due to the black background added to the original image, which cause the instant change on the edges of images.

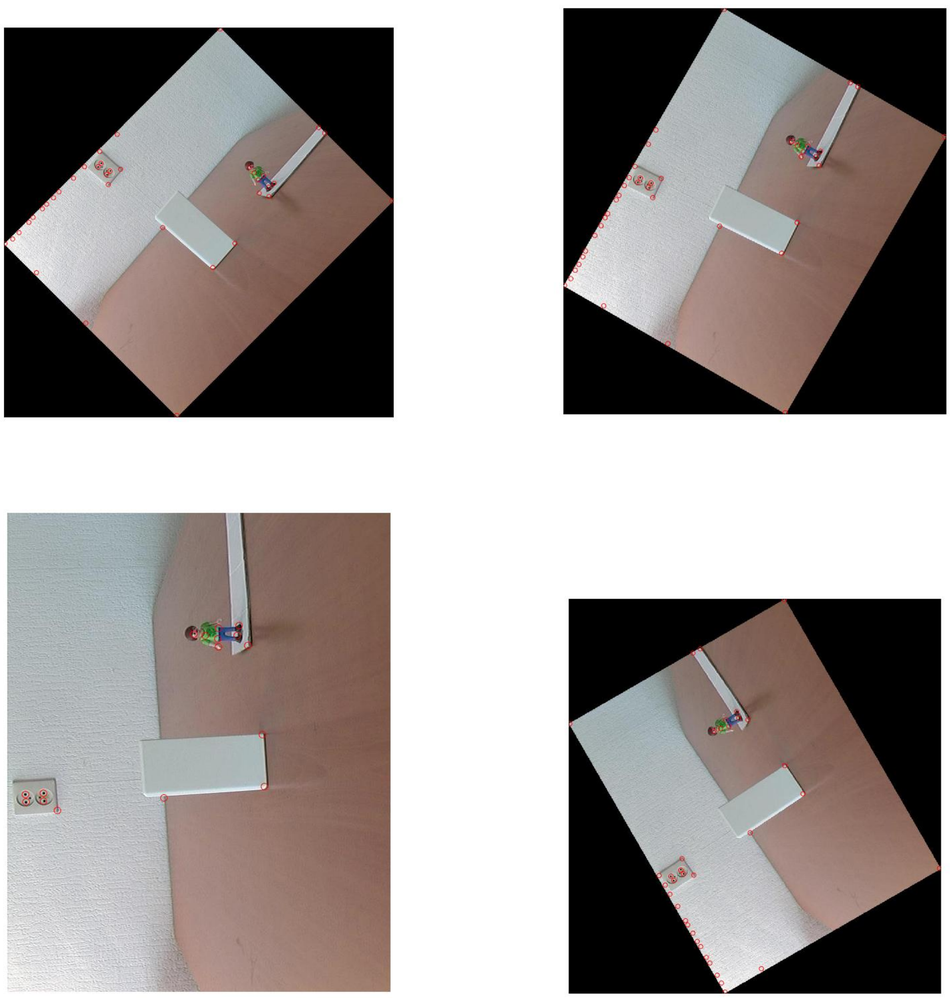

Figure 3: Up: left-45 degree right-60 degree Down: left-90 degree right-120 degree

The rotation-invariant property of harris corner algorithm could be explained by the eigen value used for calculating the Harris matrix. $H = \lambda 1 \lambda 2 - 0.04(\lambda 1 + \lambda 2)^2$ Although "Ix" and "Iy" of the corner changes(ellipse rotate), the eigen value remains the same. Therefore, H will also stay the same.

2.2 Question 2

1

In the original harris corner algorithm, the H matrix is calculated by the mathematical expression $H = \lambda 1 \lambda 2 - 0.04(\lambda 1 + \lambda 2)^2$. However, for Shi-Tomasi, it is calculated as $H = min(\lambda 1, \lambda 2)$. The corner window will only by identified when H is larger than a threshold valueλ. They performed several experiments and found out that when the smaller eigen value is larger than threshold value, the image matrix is well conditioned. As the larger eigenvalue could not be arbitrarily larger due to similar intensity **?**.

2

If we implement Harris corner method, we do not necessarily need to calculate the two eigen values. It could be expressed as $H = det(Q) - 0.04(trace(Q))^2$. Nevertheless, if we use Shi-Tomosi approach, it is essential for us to calculate both eigen values to determine the smaller one.

3

- (a) If both eigen values are near 0, the region has no interesting features.
- (b) If one eigen value is positive while the other is near 0. Then it could be the edge.
- (c) If both eigen values are big, then it could be defined as a corner.

3 Lucas-Kanade

3.1 Question 1

Given a sequence of images made in consecutive time, it can be possible to determine the motion of objects within those images. There are multiple algorithms which try to estimate that "optical flow", and the Lucas-Kanade algoritm is considered here.

1

We start with two pairs of images of a scene which were assumed to be made consecutive in some small time (see Figures 4 and 5).

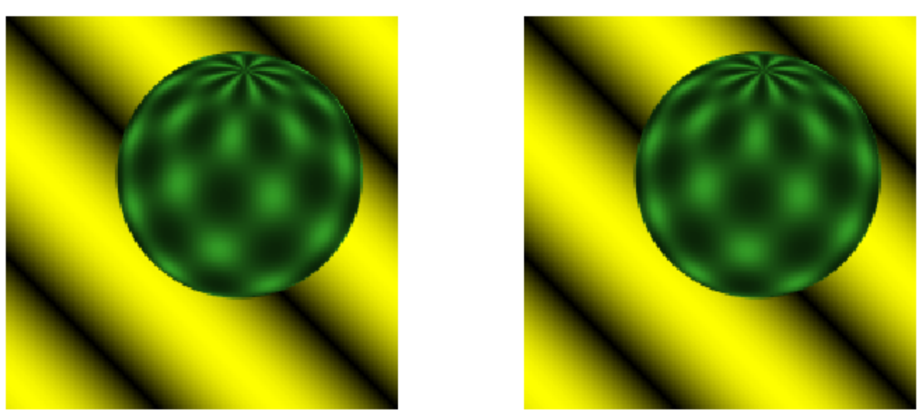

Figure 4: Pair of images: sphere1.ppm (left), sphere2.ppm (right).

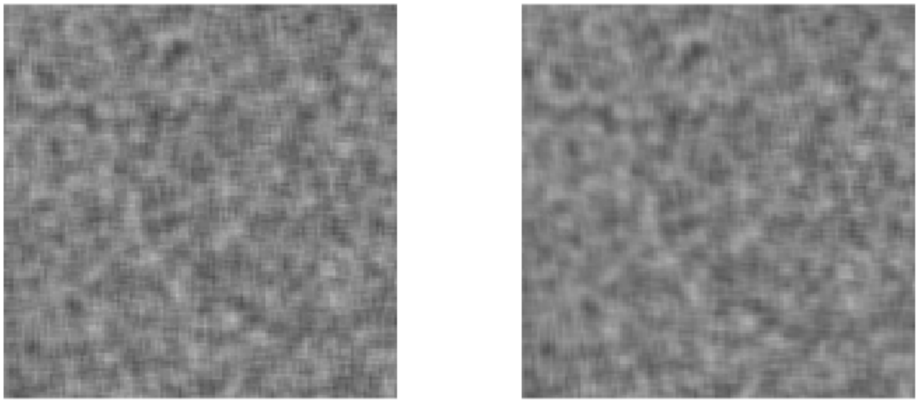

Figure 5: Pair of images: synth1.pgm (left), synth2.pgm (right).

We first divide the image in consecutive non-overlapping regions of a 15x15 size where we start at the top-left corner of the image. After this we will use the Lucas-Kanade algorithm to estimate the optical flow vector at the centre of each region. This means that when we estimate the optical flow at a point, the local neighbourhood of which we make use of is a square with the point at its centre.

2

We implemented a function which can estimate the optical flow at a point p with its local neighbourhood given. In the following experiments we always let a 15x15 sized square with p as its centre be the local neighbourhood of that point. And if p is for example located near the top border of the image and the square does not fit inside the image, we move the square a bit downwards so that it justs fits. After determining the local neighbourhood, we then use the Lucas-Kanade algorithm to solve the optical flow equations using least squares (where each pixel of the local neighbourhood gives rise to one linear equation). Note that when we calculate the image derivatives we convolve the image with the derivative of a gaussian. The sigma for this experiment was set to 2, and different values of sigma give different results. A bigger sigma blurs the image more, and ignores smaller details. So the value of sigma which is best should depend on what scale you want to calculate the movement, but note that this in turn also depends on the size of the local region.

3

Now choosing the set of points p as mentioned in Q1.1, we get the following results for the two pairs of images 'sphere1.ppm','sphere2.ppm' and 'synth1.pgm','synth2.pgm' (Figure 6).

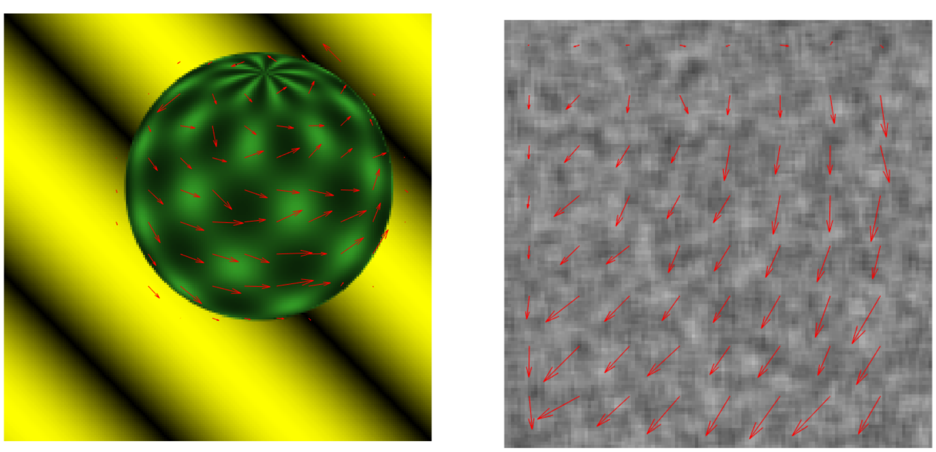

Figure 6: Estimates of optical flow vector at set of points in 'sphere1.ppm' (left) and 'synth1.pgm' (right).

We see that all the estimated optical flow vectors point towards the direction we expect the object really moved. In regions where there is no change between the two images, the optical flow vector is also indeed zero.

3.2 Question 2

1
There are more algorithms available than just the Lucas-Kanade algorithm. One of them is known as the Horn-Schunck method. There is a big difference between those two methods, which has to do on whether they operate on a local or global level.

The Lucas-Kanade algorithm operates on a local level. This is because when we want to estimate the optical flow for a point p, the Lucas-Kanade only uses information from a local neighbourhood around p. This means that the estimate for the optical flow at point p can not make use of any information outside that neighbourhood.

In contrast, the Horn-Schunck method is a method that works on a global level. This means that to estimate the optical flow at a point p, it also uses information outside a local neighbourhood around p. The idea of this method is based on a smoothness constraint on the optical flow vectors. The method also works in an iterative manner. The smoothness constraint says that the optical flow vectors will not abruptly change but in a smooth manner. So when this method estimates an optical flow vector at a point p, it does not only take the local neighbourhood into account, but also takes into account that the estimation for a point p has to be similar to the optical flow vectors around it (which were estimated in the previous iteration).

To see how this can have a big difference is answered in the next question.

2
There could be a big difference in the two methods when we look at how they work at flat regions of the images which are uniform valued. To see why, consider we need to estimate the optical flow at a point p which in both images lies in a uniform valued region. (i.e. We have images of a rectangle bar which moved to the left. In the first image the point p is at the centre of the bar, in the second (because the bar moved to the left) the point p lies more to the right of the bar.)

Lucas-Kanade will only makes use of local neighbourhood around the point p. It will see that in both images the neigbourhood around p is identical, therefore the algorithm regards it as a zero movement. So in the example of the rectangle bar: Even when we can see that the bar moved to the left, this method will not notice that.

Horn-Schunck on the other hand, operates on a global level with a smoothness constraint and in an iteration based way. So when this method is used to estimate the optical flow at a point p, it does not only look at the neighbourhood around p, but also takes the optical flow of its surrounding (which were estimated in previous iteration) into account. The smoothness constraint, says that the optical flow can not abruptly change, meaning the optical flow at point p should not be much different than the optical flow vectors estimated around p. So in the example with the rectangle it means this method will estimate the optical flow at p to be towards the left. The reason is that it will estimate at the bar edges to be movement towards the left, and (after some iterations) therefore because of the smoothness constraint say that the optical flow at p should also be towards the left.

4 Feature tracking

4.1 Question 1

1
We now combine the Harris corner detector and the Lucas-Kanade for a feature-tracking algorithm. Assuming we are given for example a sequence of video frames of some object moving, we thus like the algorithm to be able to detect some features of the object and also show in which direction it moves.

The method for N frames then can be described as follows:

```
// Choose tracked points as the points of interest of first image.
```

```
points = detect_cornerpoints(frame(1))

For k = 1 to (N-1)
    // Only for the tracked points calculate the optical flow.
    opt_flow_vectors = calc_opt_flow(frame(k),frame(k+1),points)
    // Save a plot for the video
    save_plot(frame(k),points,opt_flow_vectors)
    // Update the location of where the point will be in next frame
    points = update_new_positions(points,opt_flow_vectors)
```

2

To see how the feature tracking works, we have performed the algorithm on two different sequences of videoframes. We have created two demos 'pingpong' and 'person_toy'. For the "pingpong" demo, we see that the tracking works reasonable. For the first frame we see some distinctive corner points, and almost all the optical flow vectors seem to be in the correct direction. The positions of the tracked points move towards the optical flow vector and seem reasonable close to the real position of where it should be. We note that for big movement the estimation is much less accurate, than when there is smaller movement. This happens because one of the assumptions of the Lucas-Kanade algorithm is that the movement between a pair of images is small. So if there is a sudden big movement, the estimation will work worse. For the "person_toy" demo, the direction of the tracked points are also mostly in the correct direction. Since this video only contains small movements, we see that at the end of the video all tracked points are still quite close to the position where they should be.

4.2 Question 2

If we want to do a motion tracking given a sequence of images consecutive over time, we need to know for how much the object has moved per image. Even if for each image of the sequence we can detect features, that does not automatically tell us how the object has moved. This is because having only the features themselves, does not necessarily give information to know where each feature went to in the next image. (i.e. taking the closest feature does not necessarily work.) However, knowing the optical flow might help the match of features. Since for each point of where the feature lies, the estimation of the optical flow at that point will give you an estimation of where that point will move to. In the next image with the new position, you can then draw the conclusion that you expect there to be the feature point.

5 Conclusion

For corner detection, we implement the Harris corner algorithm. We found that as the sigma of Guassian filter increases, we have to decrease the threshold in order to make the algorithm performs well. Additionally, for different images, we might need different thresholds, a good metric could be multiples of the mean of Harris matrix. Harris corner detector works sufficiently to detect corners in the image. Nonetheless, it gets confused with corners in the shadow. The Shi-Tomasi algorithm might improve this point by comparing the minimum eigen value with the threshold. This method might have its own drawback that it requires the eigen value calculation every time. While for Harris corner, we could also implement the determinant and trace to compute the Harris matrix.

To estimate the optical flow at points we can use the Lucas-Kanade algorithm. It is based on the assumptions that the movement of a local neighbourhood around the point is constant and that between consecutive images is small.

The Harris corner detector can also be combined with a optical flow estimator to work as a feature tracker. From experiments, we conclude that the biggest problem for this is that the more motion there is, the less accurate the feature tracker was. Using other (global) methods (i.e. Horn-Schunck method) might give more accurate performance.

References

[1] Shi, J. & Tomasi, C. (1994) Good features to track, Technical Report. Cornell University, Ithaca, NY, USA.