CS 6780: Advanced Machine Learning **(Due: 15/05/2020)**

CCE5205: Computer Vision Assignment

Instructor: Dr. Ing. Reuben Farrugia *Name:* Luca Zammit

1 Introduction

The aim of this project is to create an algorithm that successfully tracks a specified object within a video stream. Harris Corner detection is used to detect features of a specified object within the first frame of the video stream. Then the Horn and Schunck Optical Flow algorithm is used to compute the flow between each two consecutive frames, and a simple tracking algorithm is implement to track the features, within the video sequence.

2 Questions and Implementations

Question 1: Load first frame and display bounding box (5 points)

Using the OpenCV VideoCapture() function, the video is first opened and the first frame is read using the .read() function. The top right and bottom left corners for the bounding box are declared as (291, 181) and (399, 261) respectively. The height and width of the bounding box are calculated from the corner points, and the first frame is displayed with the bounding box around the car.

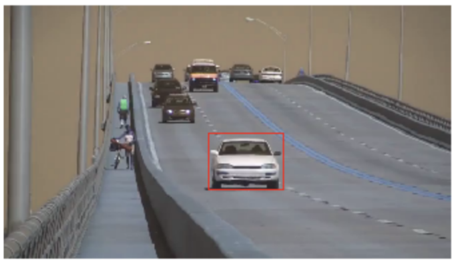


Figure 1: Result for Question 1

Question 2: Develop Harris Corner detection algorithm from scratch (30 points)

The Harris Corner detection algorithm is implemented from scratch using numpy, OpenCV and scipy. OpenCV's Sobel function is used to compute the image gradients I_x and I_y. I_{xx}, I_{yy}, and I_{xy} are obtained by using the image gradients as follows: $I_{xx} = I_x \circ I_x$, $I_{yy} = I_y \circ I_y$, and $I_{xy} = I_x \circ I_y$, where $\circ$ means elementwise multiplication. The summation over a patch for I_{xx}, I_{yy}, and I_{xy} is calculated to obtain Sum_{xx}, Sum_{yy}, and Sum_{xy} respectively. The sums over a patch are arranged in the Structural Tensor $\begin{bmatrix} Sum_{xx} & Sum_{xy} \\ Sum_{xy} & Sum_{yy} \end{bmatrix}$ and the np.linalg.eigvals() function is used to calculate the Eigen values for the patch using the Structural Tensor. The value of R at coordinate (x, y), the centre of the patch, is then calculated by using the equation $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda)^2$, where λ_1 is the Eigen value in the x direction and λ_2 is the Eigen value in the y direction. Then using a sliding window approach the algorithm is computed for the whole image. A threshold of 0.17% of the maximum R value is used to reduced the amount of feature points so that only the points that have an R value higher than the threshold are kept as valid features. The points above this threshold are considered as having a value of '255'. Using the OpenCV findContours() function the contours of the areas having value '255' are found and then by using the moments() function the centre of these contours is obtained. The result is as can be seen in Figure 2. Furthermore comparing the points marked with a value of 255 through the first principle implementation, to the ready made function from OpenCV very similar results are obtained as can be seen in Figure 3.

Question 3: Develop the Horn and Schunck Optical Flow algorithm from scratch (30 points)

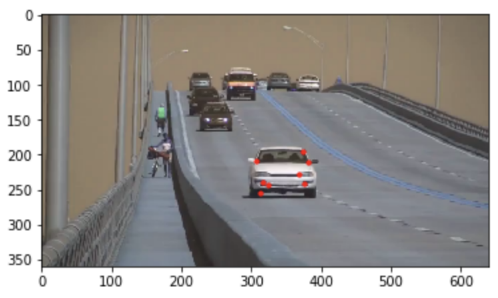
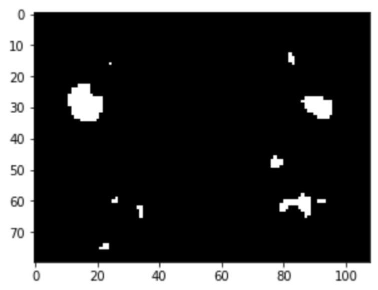
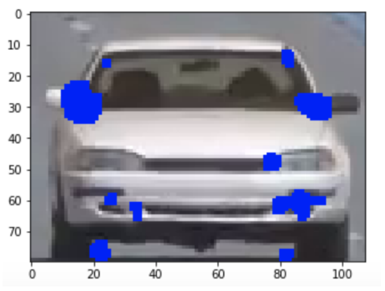


Figure 2: Result for Question 2



(a) Points marked by Harris Corner Detection first principle



(b) Points marked by OpenCV Harris Corner Detection

Figure 3: Comparison between first principle and OpenCV Harris Corner detection

The Horn and Schunck algorithm is developed from scratch using numpy and scipy. The Mean Kernel and the Kernels for computing the gradients f_x, f_y, and f_t are declared. Then the gradients are computed as follows $f_x = \left[\begin{smallmatrix}-1 & 1\\-1 & 1\end{smallmatrix}\right] * Img_1 + \left[\begin{smallmatrix}-1 & 1\\-1 & 1\end{smallmatrix}\right] * Img_2$, $f_y = \left[\begin{smallmatrix}-1 & -1\\1 & 1\end{smallmatrix}\right] * Img_1 + \left[\begin{smallmatrix}-1 & -1\\1 & 1\end{smallmatrix}\right] * Img_2$, and $f_t = \left[\begin{smallmatrix}-1 & -1\\-1 & -1\end{smallmatrix}\right] * Img_1 + \left[\begin{smallmatrix}1 & 1\\1 & 1\end{smallmatrix}\right] * Img_2$. The flow vectors u and v are initialized as matrices of 0s with the image dimensions. To estimate the flow vectors first their means are calculated by convolving the flow vectors with the mean kernel $\bar{u} = u * \begin{bmatrix}1/9 & 1/9 & 1/9\\1/9 & 1/9 & 1/9\\1/9 & 1/9 & 1/9\end{bmatrix}$ and similarly for $\bar{v}$. P is calculated by using $P = f_x\bar{u} + f_y\bar{v} + f_t$ and D is calculated by using $D = \lambda + f_x^2 + f_y^2$ where λ is the regularization constant. Then the flow vectors are estimated using $u = \bar{u} + f_x\frac{P}{D}$ and similarly for v. This is performed iteratively, improving the estimate with each iteration. The result of the optical flow algorithm is as can be seen in Figure 4, obtained using Quiver plot from Matplotlib.

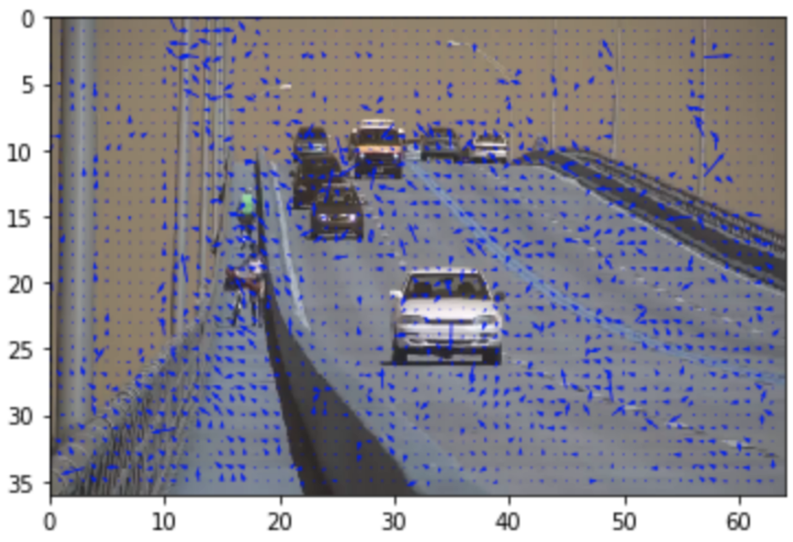
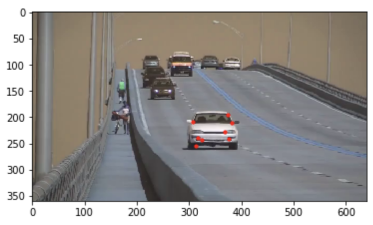
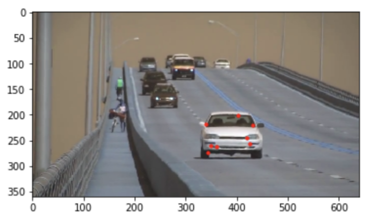


Figure 4: Result for Question 3

Question 4: Track the keypoints using Optical Flow over the first 100 frames (20 points)

Using the Optical flow algorithm developed for Question 3, the keypoints identified in Question 2 are tracked with a simple algorithm over the first 100 frames of the video sequence. The tracking function takes as inputs image 1 and image 2 which should be two consecutive frames in a video sequence. It then calls the optical flow function to compute the Optical flow between the two frames. For every feature coordinate, the flow vectors u and v at that location are taken. To calculate the new location for the features, the value for u and v at the location of the features are added to the coordinates of the features, $x_2 = x_1 + u(x_1)$ where x_1 is the old x value for the keypoint coordinate and x_2 is the new x value. The same is done for the y values with flow vector v. In Figure 5 the result of the tracking algorithm can be seen by comparing frames 1, 50, and 90. In Figure 6 the performance of tracking using OpenCV's calcOpticalFlowPyrLK() tracking implementation is used, where we can see the the performance of the algorithm is a bit better, especially since the keypoint on the top right corner of the windscreen is not failing to be tracked correctly unlike the first principle implementation. However other than this keypoint all other keypoints are being tracked correctly with the first principle implementation.


(a) Frame 1


(b) Frame 50

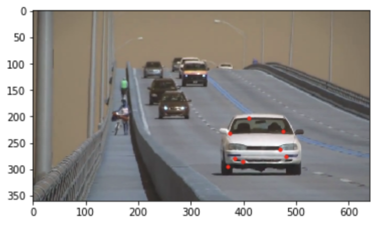
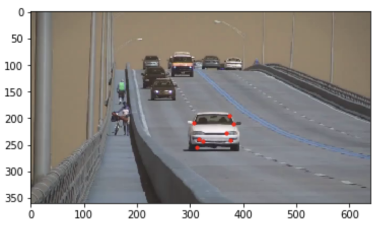
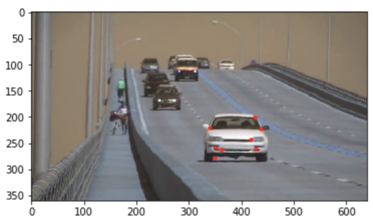

(c) Frame 90

Figure 5: Tracking Result


(a) Frame 1


(b) Frame 50

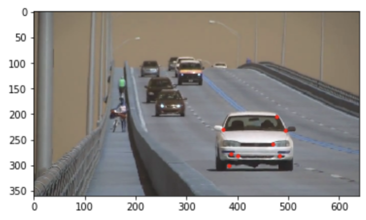

(c) Frame 90

Figure 6: Tracking Result using OpenCV

Question 5: Plot the trajectory of the white vehicle in the first 100 frames (5 points)

To compute the trajectory of the white vehicle in the first 100 frames, the bottom left and the top right corners of the bounding box which incorporates all features are calculated for the first and the 100th frame and the centres for these bounding boxes are calculated. Using the Quiver function from Matplotlib the trajectory is displayed on the image of the first frame, showing the path of the vehicle throughout the first 100 frames. The result can be seen in Figure 7.

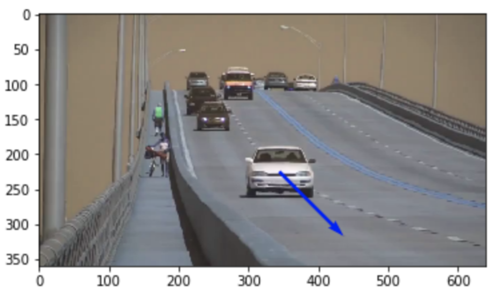


Figure 7: Result for Question 5

3 Conclusion

Within this assignment, a simple tracking algorithm is developed from scratch using Horn and Schunck Optical flow to track features obtained by Harris Corner detection. The tracking algorithm implemented from scratch is then compared with the tracking algorithm implemented from OpenCV. The result of the OpenCV tracking algorithm perfoms better, especially since one of the points is not tracked correctly in the tracking algorithm implemented from scratch. Moreover the Harris Corner detection algorithm is compared to the one implemented by OpenCV, where it can be seen that the performance for this is very close to the one implemented from scratch.