CS512 – Computer Vision Assignment 6 report

Problem Statement:

To estimate Optical Flow in a Video.

Proposed Solution:

I am implementing Affine Motion to estimation the Optical Flow in the frames of a Video.

In Affine motion, location motion is described by an affine map.

Affine motion, unlike other block-based approaches does not assume that optical flow or gradient of optical flow are equal in the local neighborhood. Instead, the optical flow vector is estimated at each point.

In Affine flow, we have to estimate 6 parameters.

Implementation Details:

I read frames from a video one at a time and process each of them.

First, I convert the frame to gray scale.

I find the components of Image derivatives i.e, the spatial derivatives, using Sobel filter in opencv using a window size of 5.

I find the frame difference i.e, the temporal derivative, using forward difference. (Taking the difference with the current frame and next frame)

Then, I form the matrices required to find the coefficients are shown below:

Solution is given by:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} \Sigma I_x^2 & \Sigma I_x^2 x & \Sigma I_x^2 y & \Sigma I_x I_y & \Sigma I_x I_y x & \Sigma I_x I_y y \\ \Sigma I_x^2 x & \Sigma I_x^2 x^2 & \Sigma I_x^2 xy & \Sigma I_x I_y x & \Sigma I_x I_y x^2 & \Sigma I_x I_y xy \\ \Sigma I_x^2 y & \Sigma I_x^2 xy & \Sigma I_x^2 y^2 & \Sigma I_x I_y y & \Sigma I_x I_y xy & \Sigma I_x I_y y^2 \\ \Sigma I_x I_y & \Sigma I_x I_y x & \Sigma I_x I_y y & \Sigma I_y^2 & \Sigma I_y^2 x & \Sigma I_y^2 y \\ \Sigma I_x I_y x & \Sigma I_x I_y x^2 & \Sigma I_x I_y xy & \Sigma I_y^2 x & \Sigma I_y^2 x^2 & \Sigma I_y^2 xy \\ \Sigma I_x I_y y & \Sigma I_x I_y xy & \Sigma I_x I_y y^2 & \Sigma I_y^2 y & \Sigma I_y^2 xy & \Sigma I_y^2 y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\Sigma I_x I_t \\ -I_x I_t x \\ -I_x I_t y \\ -I_y I_t \\ -I_y I_t x \\ -I_y I_t y \end{bmatrix}$$

I consider a window of size 5 x 5

I ignore the boundary cases and don't process them

For each pixel as center, I consider its neighbors with the 5 x 5 window and form the matrices as shown below summing up over the pixels in the window.

Before taking the inverse of the 6 x 6 matrix shown above, I find the eigen values of the matrix and find the ratio of second largest and largest eigen value. If the ratio is lesser than 0.7, I ignore the window and don't solve the above equation in the window to find the 6 parameters and then the optical flow itself.

If the ratio is greater than or equal to 0.7, I find the inverse of the 6 x 6 matrix and solve of the parameters, as below:

```
try:
    params = np.matmul(np.linalg.inv(A), B)
except np.linalg.linalg.LinAlgError as lae:
    continue
```

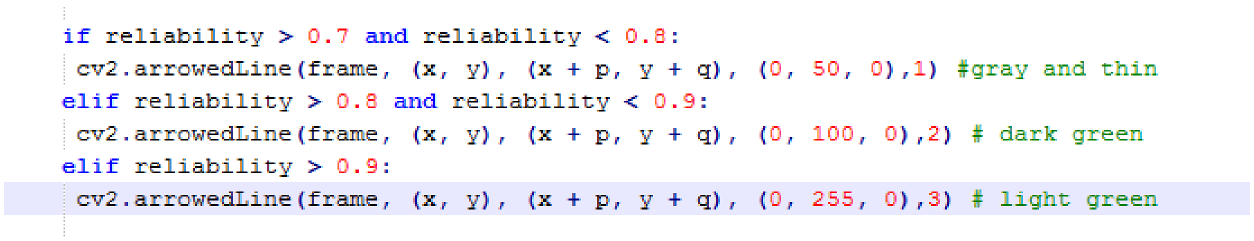
If the matrix is singular and don't have a inverse, LinAlgError is thrown, which I handle and continue without any processing.

After finding the parameters, I find the x and y components of the optical flow vector, as below

v_x = a1 + a2 * x + a3 * y

v_y = a4 + a5 * x + a6 * y

I draw this optical flow vector on the frame, varying the intensity based on the reliability value calculated before, as below:



```
if reliability > 0.7 and reliability < 0.8:
    cv2.arrowedLine(frame, (x, y), (x + p, y + q), (0, 50, 0),1) #gray and thin
elif reliability > 0.8 and reliability < 0.9:
    cv2.arrowedLine(frame, (x, y), (x + p, y + q), (0, 100, 0),2) # dark green
elif reliability > 0.9:
    cv2.arrowedLine(frame, (x, y), (x + p, y + q), (0, 255, 0),3) # light green
```

After processing each frame, I write the results (optical flow vectors for this frame plotted in the frame) to the "data/output/frame_(seq number).png" file for reference.

To evaluate, I have used the code [1], which used Lukas-kanade method. I have run this implementation too on the same data I am using for my implementation.

Instruction to execute my program:

1. Source code is present in "src/AS6.py"
2. My program takes 2 command line arguments, video file path and threshold value for reliable optical flow detection.

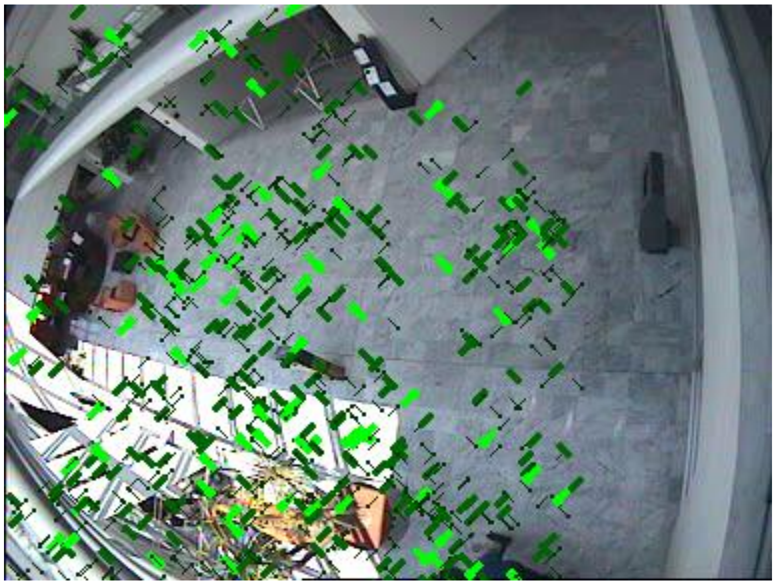
python AS6.py ../data/Walk1.mpg 0.7

3. If these arguments are not specified, the code by default uses "../data/Walk1.mpg" and 0.7.
4. After processing each frame, the frame with optical vectors plotted are written into "../data/output/frame_(seq number).png"

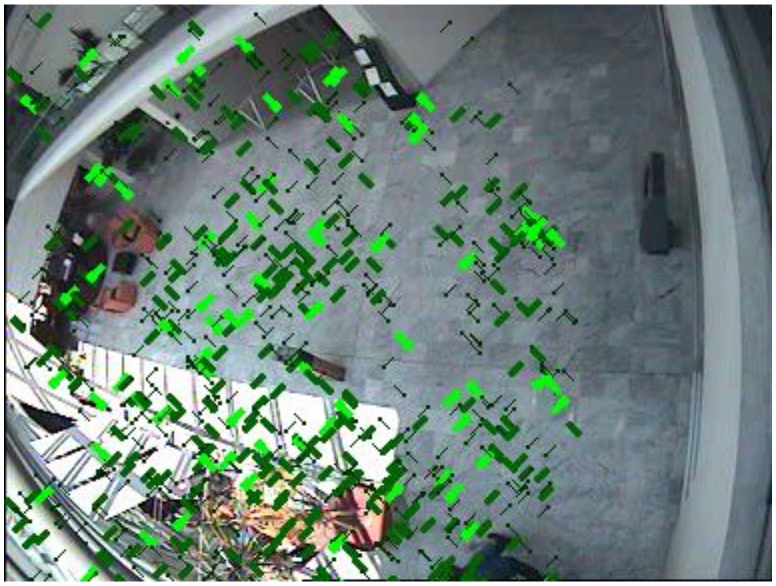
Results and discussion:

In the video sequence I am using, there was not much change in first few frames. So, I dropped processing first 26 frames.

Below is the optical flow plot of the 27th frame:



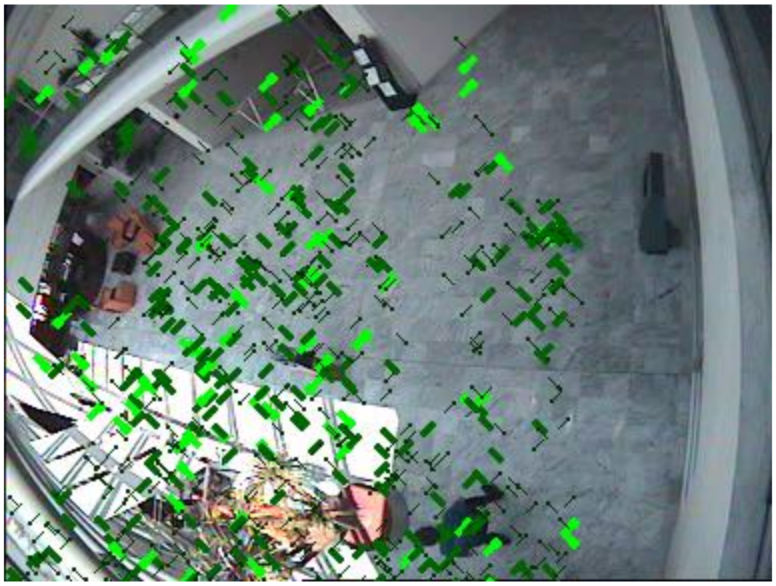
28th frame:



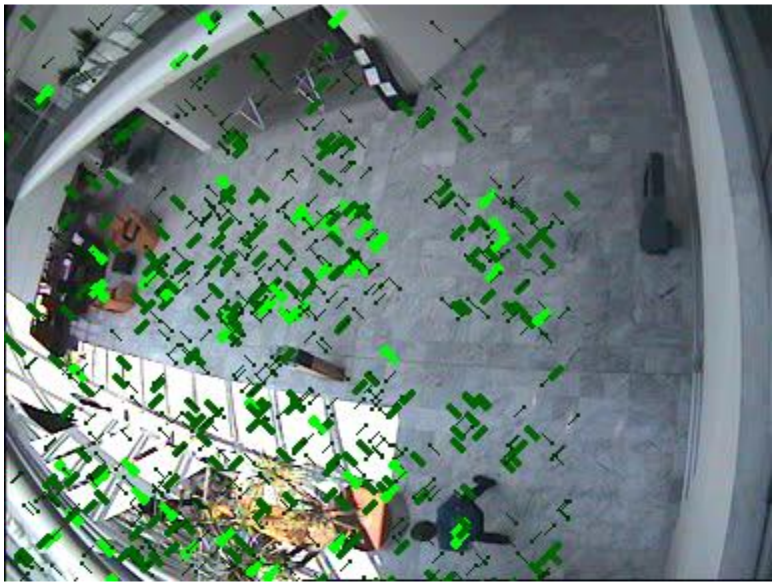
We can clearly see that my algorithm is identifying the corners in the tiles on the floor.

The threshold we choose to detect the reliability make a big difference in the number of optical flow vectors identified in a frame. I used the threshold as 0.7 in my execution.

Frame 36th, a person enter in to view:

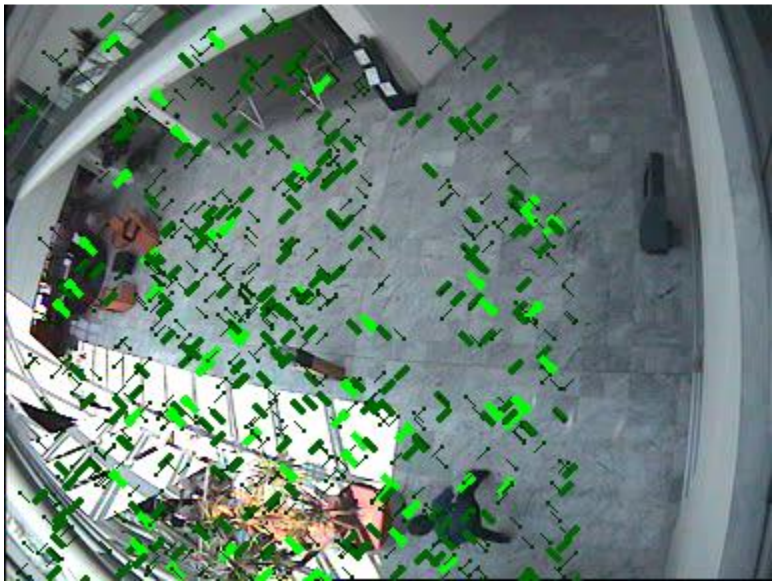


Frame 37th:



We can see that there are a lot of optical flow vector identified around the person as he moves.

38th Frame:



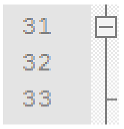
We see a lot of reliable points in the left bottom of the frame, as there were people working in this area and there was a lot of movement in this area.

There are a lot of points detected in the floor, due to corners, but there are not very reliable as seen from the plot.

References:

1. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html#lucas-kanade

Note:



```
while i < 27:
 ret, frame = cap.read()
 i = i + 1
```

If executing my code on some other data and don't want to drop processing first 26 frames, please comment the shown lines from my code.