LUCAS KANADE TRACKER

IMPLEMENTATION:

1.OWN IMPLEMETATION USING GAUSS NEWTON

First, I initialize the parameters for Shi Tomasi corner detection. Then I took first frame and found corners in it. Then I created mask data structure to create a mask image for drawing purposes. Then I took next frame and found gradient using CV2.sobel() function for future purposes. Then I made a function get_New_Coordinate() to get new position of the pixel in next frame.
Within the function: First I created data structure T which contains original frame data. Then I implemented the algorithm which was described in Simon Baker Paper.

Algorithm:

Pre-compute:

```
Evaluate the gradient $\nabla T$ and the second derivatives $\frac{\partial^2 T}{\partial \mathbf{x}^2}$
Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and the Hessian $\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}$ at $(\mathbf{x}; \mathbf{0})$
Compute $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$, $[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\frac{\partial^2 T}{\partial \mathbf{x}^2}][\frac{\partial \mathbf{W}}{\partial \mathbf{p}}] + \nabla T[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}]$, and $[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$
```

Iterate:

```
    Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
    Compute the error image $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$
    Compute the Hessian matrix $\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}$ using Equation (82)
    Compute $[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}]^{\mathrm{T}} = \sum_{\mathbf{x}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
    Compute $\Delta \mathbf{p}$ using Equation (85)
    Update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta \mathbf{p})^{-1}$
```

Until $\|\Delta \mathbf{p}\| \leq \epsilon$.
Here Equation (82) is:

$$\frac{\partial^2 G}{\partial \mathbf{p}^2} = \left(\left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\frac{\partial^2 T}{\partial \mathbf{x}^2}\right] \left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] + \nabla T \left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right] \right) [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] + \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]$$

And Equation (85) is:

$$\Delta \mathbf{p} = -\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}$$

I have made Warp_Inverse() function to get inverse of W(x;p). After converging the above iteration, I get my new coordinate of original points in next frame by using W(x;p). Thus, after getting the points fuction returns it and now I have coordinate of features in current frame and in next frame, so I joined them using a line to track the object path.
Now after that, new features is again extracted using Shi Tomasi Corner detection and it again goes back to loop and repeats the process.

2.USING OPENCV LIBRARY

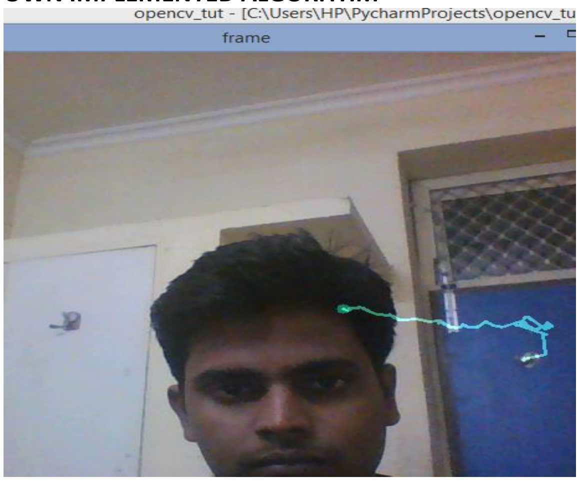
This is done using the inbuilt library function for Lucas Kanade tracking. First I used function for feature detection that is: cv2.goodFeaturesToTrack(). Then I used the inbuilt function for getting the new coordinate that is: cv2.calcOpticalFlowpyrLK().
Now after getting the coordinate in both current and next frame. I join them using the line and hence the objects.

EXPERIMENTAL ANALYSIS:

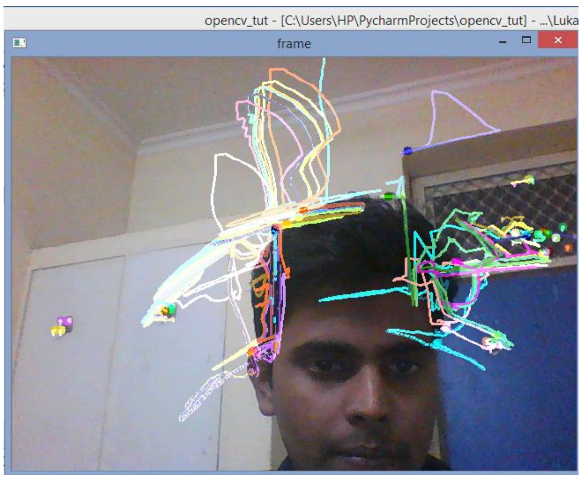
I have experimented on both the implementation and got following results:

Video1:

OWN IMPLEMENTED ALGORITHM

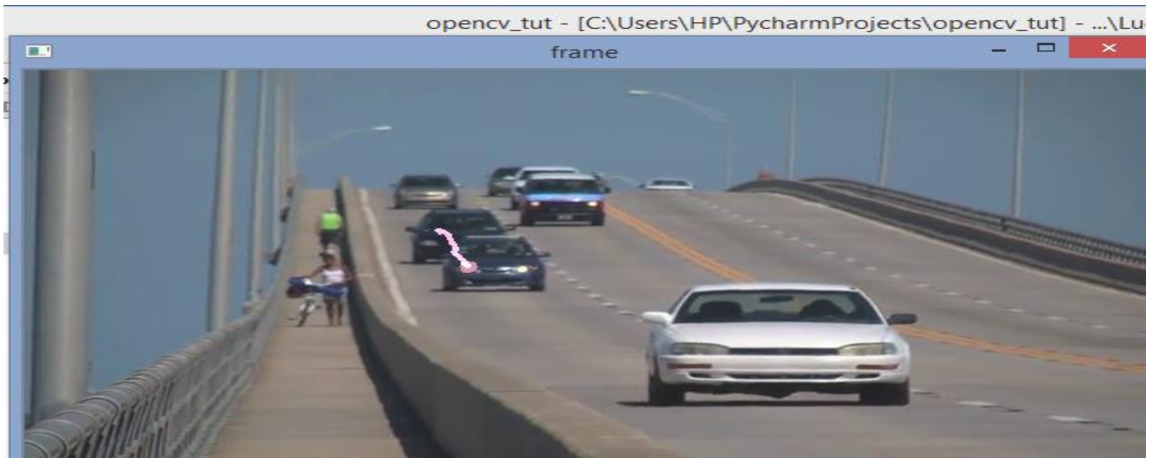


USING OPENCV

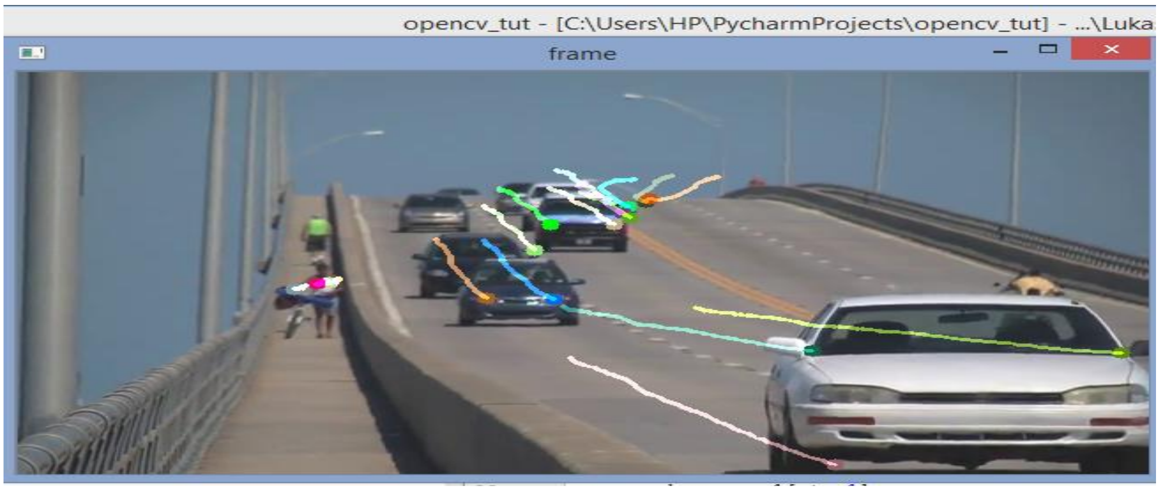

opencv_tut - [C:\Users\HP\PycharmProjects\opencv_tut] - ...\Luka
frame

Video2:

OWN IMPEMENTED ALGORITHM


opencv_tut - [C:\Users\HP\PycharmProjects\opencv_tut] - ...\Lu
frame

USING OPENCV


opencv_tut - [C:\Users\HP\PycharmProjects\opencv_tut] - ...\Luka
frame

OBSERVATION:

I have observed lots of important things and differences in two implementation one is using own implementation by gauss newton and other is using OpenCV. Some of the important points are follow as:

1.My implementation is extracting many features but tracking one of the dominant features, but in inbuilt it is extracting many features and lots of features tracking is working simultaneously. This is not a great deal, this can be adjusted. The main important thing is rate of convergence issue. In inbuilt function, it is converging very fast but in my case, it is converging but at moderate speed.

2.The difference in speed occurs due to many reasons. Some of the important reasons are: inbuilt function uses pyramid, which helps in performance, also better hyper parameter tuning. Also, in case of inbuilt function the convergence algorithm is implemented in optimal way.

3.My own implementation is also giving the approximate result and converging at moderate speed. Also, changing the value of €(epsilon) effects the rate of convergence. For very small and very large values of epsilon, algorithm is not running efficiently. But it is running at moderate speed when values of epsilon lie between 0.00001 – 0.001.

4.This method is basically based on optical flow concept. I mark the track within the video in direction of optical flow between current and previous frame of each pixel values.

5.It has lots of other useful advantages applications like:
   a) Video stabilization
   b) Image Mosaicing
   c) To track anything within the videos etc.

I will describe the two important use of Lucas kanade tracker system:

1. Video stabilization
2. Image Mosaicing

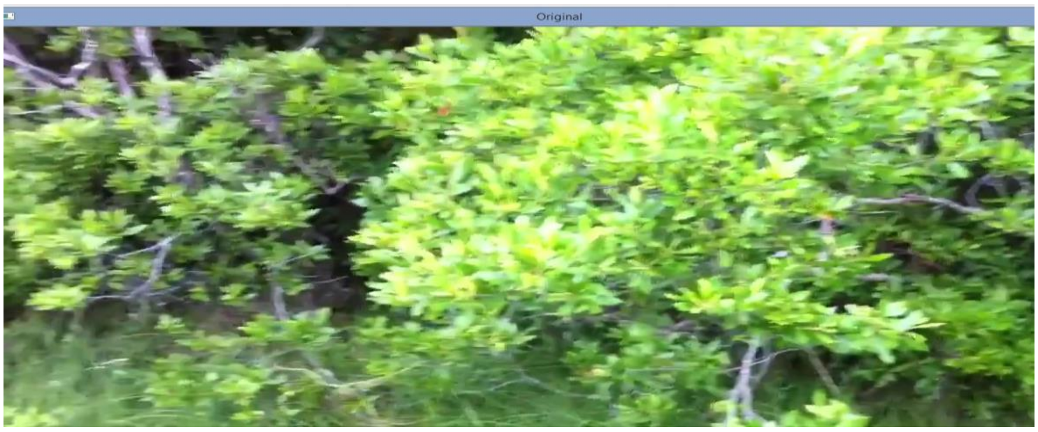
VIDEO STABILIZATION

Algorithm:

1. Find the transformation from previous to current frame using optical flow for all frames. The transformation only consists of three parameters: dx, dy, da (angle). Basically, a rigid Euclidean transform, no scaling, no sharing.
2. Accumulate the transformations to get the “trajectory” for x, y, angle, at each frame.
3. Smooth out the trajectory using a sliding average window. The user defines the window radius, where the radius is the number of frames used for smoothing.
4. Create a new transformation such that new_transformation = transformation + (smoothed_trajectory – trajectory).
5. Apply the new transformation to the video.

EXPERIMENTAL ANALYSIS:

opencv_tut - [C:\Users\HP\PycharmProjects\opencv_tut] - ...\videostab.py - PyCharm Community Edition 2017.1.5
Original

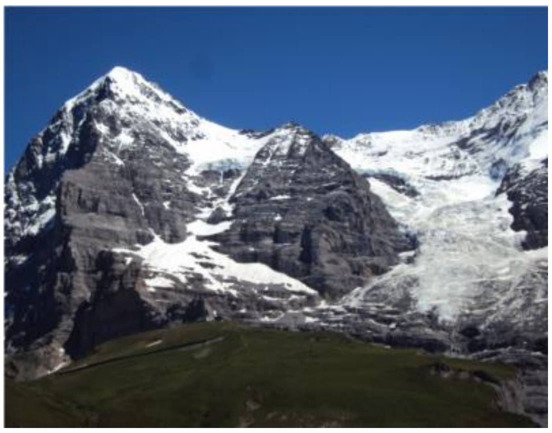
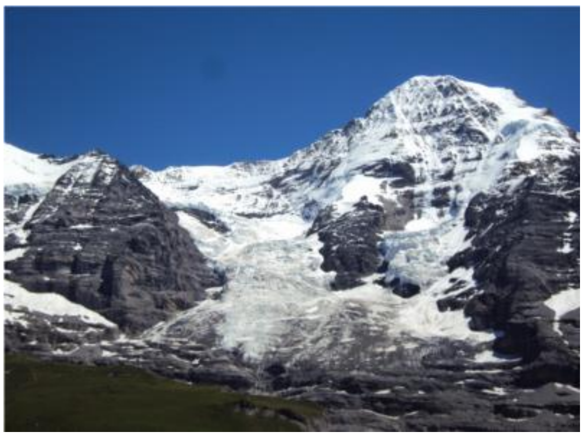


OBSERVATION:
I am getting video stabilization approximately good enough. As the time video stabilization capacity of stabilizing the frame increases. It is now not much affected by camera motion.

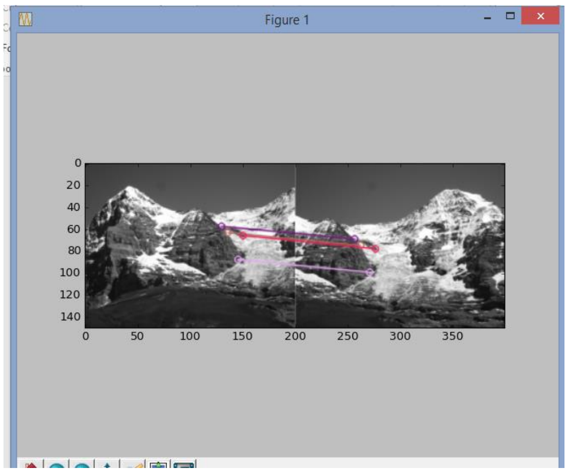
IMAGE MOSAICING

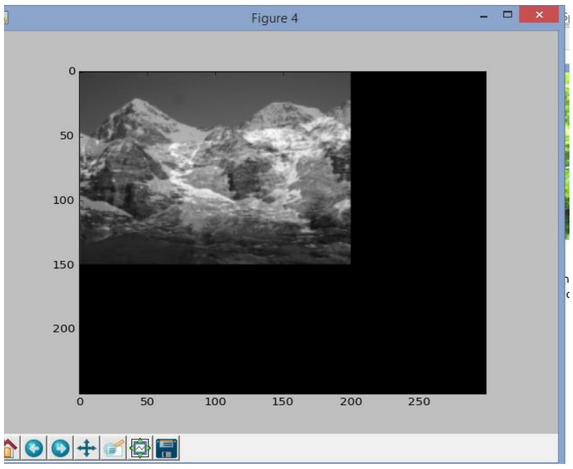
EXPERIMENTAL ANALYSIS:

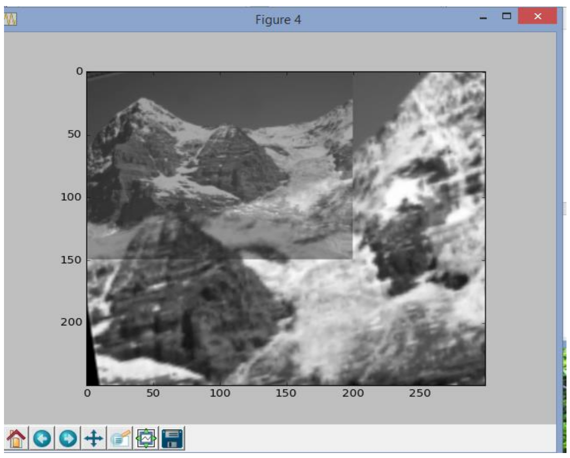
INPUT IMAGES:

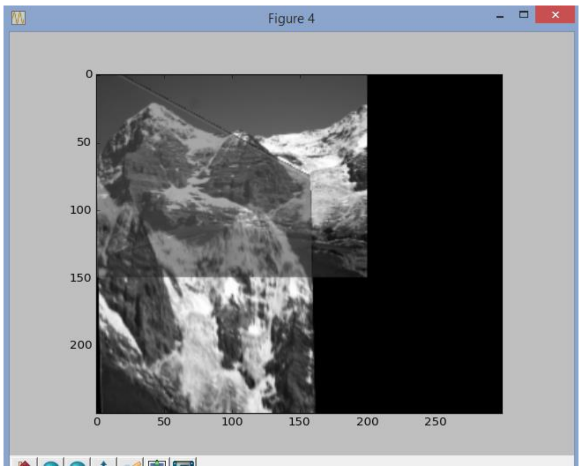



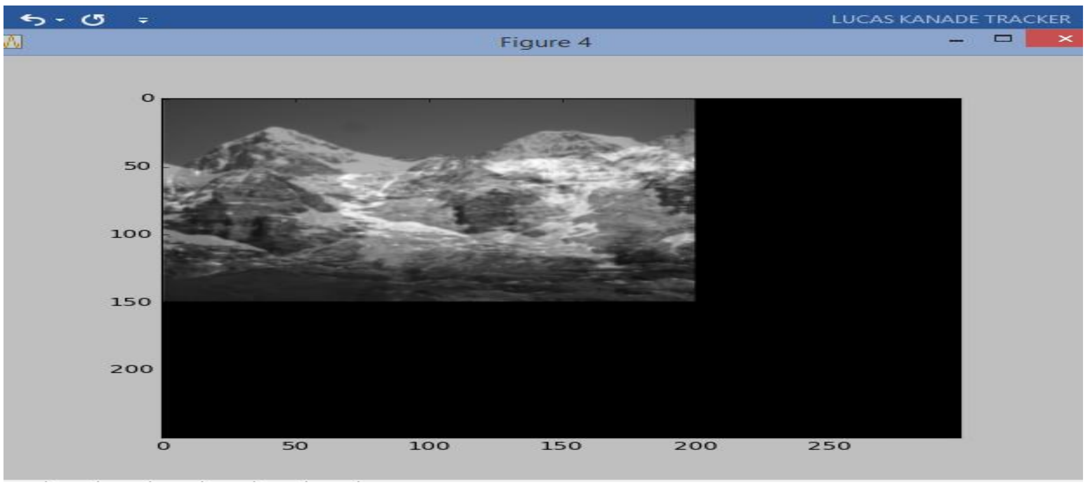
OUTPUT IMAGES:











OBSERVATION:

I have observed that performance of image mosiacing become better as algorithm proceeds and this phenomena is based on optical flow. Finally, in last figure I got better image mosiacing of two input images. Also, In first images, it is showing that first I extract the common features in both and then used optical flow to mosaic the pairs of images.