Summer Design Project 2016
Week 3

Kartik Sayani : 201451030
Meghna Ayyar : 201451027
Mentor : Prof. Pratik Shah

20 May, 2016

---

1 Week 3

1.1 Topics

Iterative Lucas Kanade Optical Flow :

So far, we used Lucas Kanade optical flow method for one iteration on two images. This method refines the original algorithm by iteratively running the original Lucas Kanade algorithm. Given an initial estimate of the Optical Flow in an image, we run by first shifting the original image to the points determined by the initial estimate, and then running the algorithm again. This gives us a more resolved and finer optical flow. The code for this implementation is being tested at the time of writing this report and will shortly be submitted.

Image Resolution Pyramid :

Image resolution pyramids are constructed by down-sampling the original image and then further down-sampling it at each level such that the lowest level is the one with highest resolution and the resolution grow coarser as one goes up the levels. This comes in handy when the motion of objects in the image is large over a single frame difference, which ca be missed if the window chosen for calculating optical flow is not adequately large. By down-sampling we bring down this large motion to fit in the window. Down-samplig can be easily done by a method described in *Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the algorithm* by *Jean-Yves Bouguet* OpenCV 3.0 provides a built-in utility *pyrDown(image)* under the cv2 package which implements the same method as described above.

Lucas Kanade Algorithm over the Image Resolution Pyramid :

The Iterative Lucas Kanade Method is applied over the image pyramid to capture large motions. This is done by first considering the initial guess of optical flow to be zero and calculating the optical flow on the top most image of the pyramid, i.e. on the coarsest one. Once the optical flow of this image is found to a satisfying degree of accuracy using LK iterations, we feed this optical flow as the initial guess of optical flow for LK iterations of a level below this level i.e. for a finer version of the image. We do this for all levels until we reach the original image. This captures the optical flow of motion of large objects in the image sequence. The larger is the motion, the higher(coarser) one has to go in the pyramid levels to capture it. Information is lost on down-sampling which means that smaller motions might be lost completely and therefore it is important to chose the levels appropriately.

OpenCV 3.0 has a built-in utility *cv2.calcOpticalFlowPyrLK(Previmg , nextimg , prevpts , nextpts, Winsize , Maxlevel , criteria , minEigThreshold)* to do this. The function calculates optical flow between the two images *Previmg* and *nextimg*. It takes an array of *prevpts* (the points over which we are interested in calculating the optical flow) and returns an array of corresponding coordinates of the points in the *nextimg*. It also returns an array of *status* which has value 1 for each point if that feature was tracked successfully. An array containing the error in the measurement of the corresponding features is also returned.

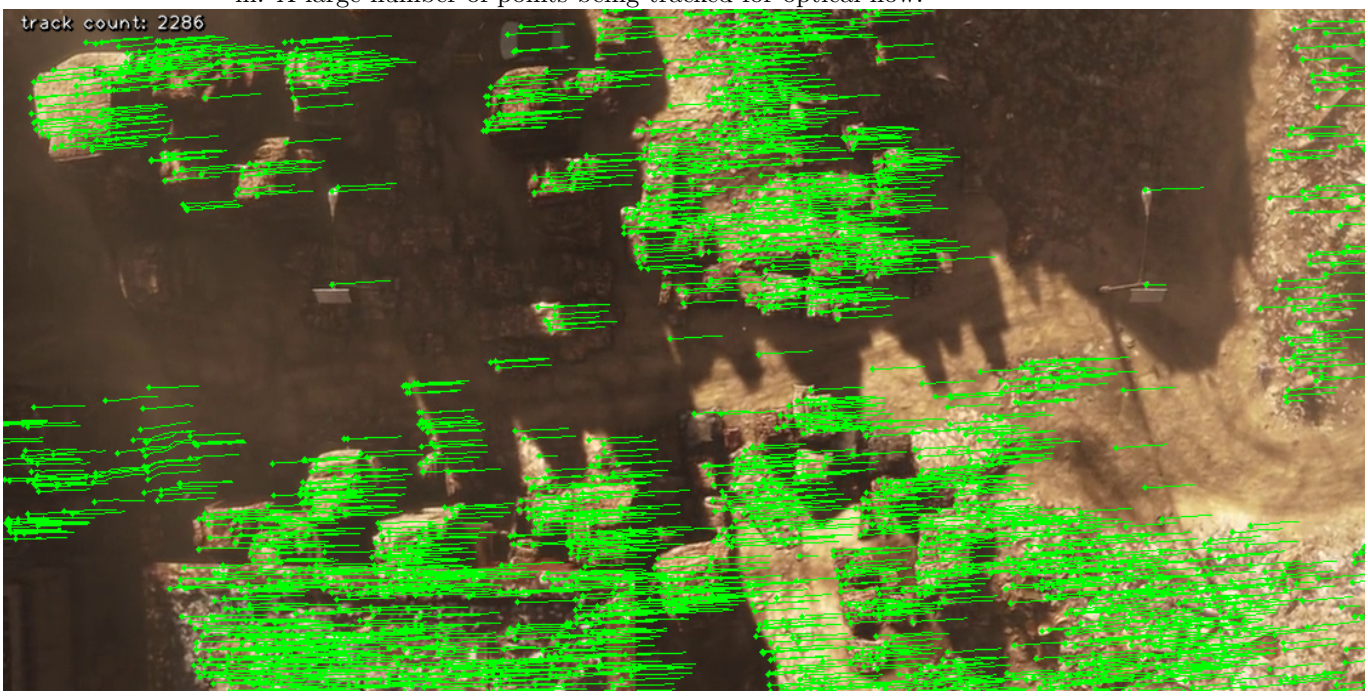
It is worth to notice these parameters that are passed on to this function:
winsize : it is the search window size in the pyramid. It is set in accordance with the levels so that large motion is captured in smallest possible number of levels but at the same time keeping unnecessary processing by setting it too large.
maxlevel : the number of levels of pyramid that will be used to calculate the optical flow.
criteria : the termination criteria for the LK iterations to stop, i.e. the satisfying amount of accuracy in the optical flow.
mineigthreshold : The function also calculates the eigen values of the corresponding optical flow equations. These eigen values are useful in determining the significant corner and edge points in the image. It is similar to doing *goodFeaturesToTrack()* instead we point out which of the good feature points passed to the function are not good enough anymore. If these values fall below the threshold , then that feature is filtered out as a bad feature and its flow is not processed.

We have implemented this function from a sample provided in the OpenCV documentation over a video of good 30fps and it has run smoothly with good results. We shall upload the implementation over the Drive folder. Here are the screen captures of the optical flow shown in the video.

i. With less number of points being tracked for optical flow.

ii. With the number of points being tracked increased for optical flow.

iii. A large number of points being tracked for optical flow.



Implementation of Lucas Kanade Iterative approach on Image Pyramids :

We are currently working on implementing Lucas Kanade's Iterative approach on Image Pyramids and are halfway through it. We shall hopefully be done with it in 2 days. We will post the results in the shared Drive folder.

Libraries and Dependencies :

The following libraries and dependencies are required to be able to run the python implementation scripts:
The *array* module from *numpy* module in Python3. Required for storing and manipulating images as arrays.
The *video* module of the *cv2* module of OpenCV. It is required for functions like *cv2.VideoCapture(VideoSource)* which renders video and provides various utilities to work with videos and *cv2.VideoCapture.read()* which reads the video frame by frame after decoding based on its format.The function *cv2.goodFeaturesToTrack (image , maxcorners , qualitylevel , miindistance , corners , mask , blocksize , use HarrisDetector)* is also provided by the same module. This function returns a list of coordinates of strong corners in the image. we can use these points as our features for the initial optical flow calculation.
The module *clock* from *time* module of OpenCV is needed to correctly render the videos according to their frame rates.
Further *draw_2* module of the *common* module from OpenCV is needed to draw the optical flow indicated by green lines on the images.

References

Most of the references were taken from online OpenCV documentation, wikipedia and a paper *Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the algorithm* by *Jean-Yves Bouguet*