CS 512 Assignment 6

Report

Li Xu A20300818

1. Problem Statement

In this assignment, I implemented Horn-Schunk algorithms to compute optical flow. I started from the optical flow example in OpenCV tutorials [1] and modified my own method rather that directly calling cv2.calcOpticalFlowFarneback function. Due to large calculation, the program is a little bit slow, please be patient.

2. Proposed Solutions

The objective equation is:

$$E\big(V(x,y.t)\big) = \iint_{image} E_{OF}^2\left(V(x,y,t)\right) + \alpha^2 E_S^2\big(V(x,y,t)\big)$$

$$where\ E_{OF}^2 = \left(I_x u + I_y v + I_t\right)^2$$

$$E_{SOF}^2 = u_x^2 + u_y^2 + v_x^2 + v_x^2$$

The optical flow can be compute as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} - \begin{bmatrix} (I_x\bar{u} + I_y\bar{v} + I_t)I_x \\ (I_x\bar{u} + I_y\bar{v} + I_t)I_y \end{bmatrix} \frac{1}{I_x^2 + I_y^2 + \alpha^2}$$

Then we need to have an initial set of u and v to compute the average of u and v.

And iterate to compute the optical flow (u,v)

3. Implementation Details

1. Load the test video from terminal: python as6.py test.mpg
2. Instead of have an initial guess of u and v, I just set the initial u,v to zero
3. From the initial u and v to iteratively compute u,v using the equation above. And we can also use a Laplacian approximate to estimate optical flow. [3]

$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$
$\frac{1}{6}$	-1	$\frac{1}{6}$
$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$

Fig 1. Laplacian filter to estimate optical flow

4. Plot the optical flow onto the video
5. You can press p to pause the video. (I implemented this pause function by setting the cv2.waitKey delay value to a negative number. When the it's negative, the program will wait for a keyboard input for an infinite time until another key is pressed.) [2]

4. Results and Discussion

The result of the optical flow is shown below:

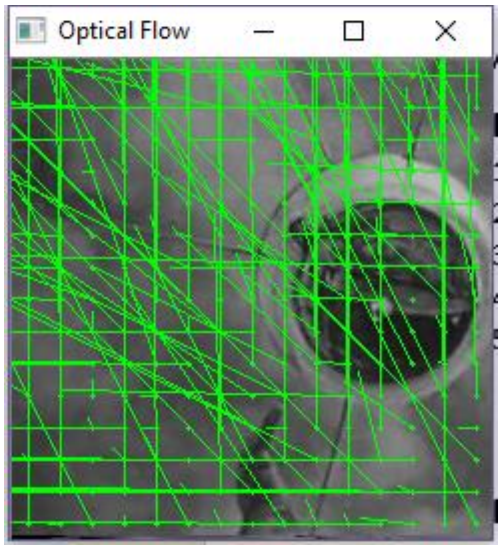

Fig 2. Optical flow shown in my function

The drawing function and calculation need to be improved for a better result.
I also checked the OpenCV official tutorials to have a good-looking result (calling cv2.calcOpticalFlowFarneback function) shown below

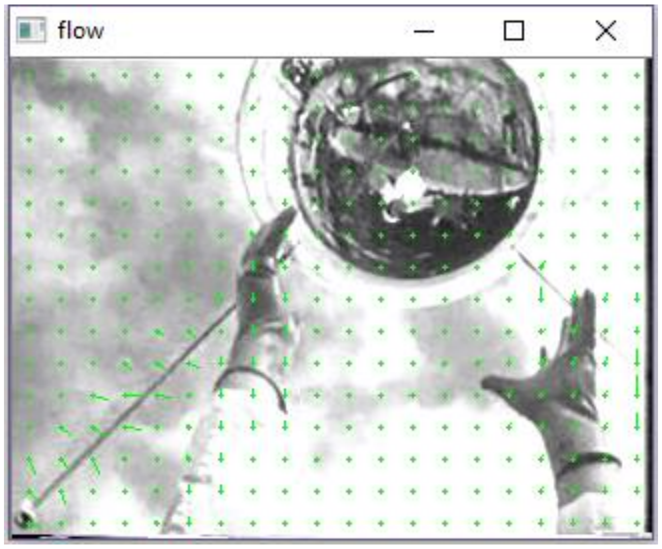

Fig 3. Optical Flow shown in OpenCV tutorials function

Some future works can be done to improve my function:

1. Instead of starting compute u and v from 0, we can have an initial guess of them. We can use Lucas-Kanade or affine-flow method to compute corresponding point from two frames, then calculate the initial u and v base on that. In this way, it's more efficient to get the final u,v.
2. In my Implementation, at first, I want to set a threshold to determine when to stop compute u and v. I found it's very hard to set an universal threshold for every input video. Therefore, I choose to use a fixed iteration loops to compute the final u and v.

5. Reference

1. https://github.com/opencv/opencv/blob/master/samples/python/opt_flow.py
2. http://blog.csdn.net/xinyu3307/article/details/7382228
3. http://people.csail.mit.edu/bkph/papers/Optical_Flow_OPT.pdf