HW2: Optical Flow and Video Stabilization

Due date: 28/4/2020 @ 23:50

Part 1 – General Questions

In this part you will need to answer a few questions. Keep your answers short. Sometimes there is more than one correct answer, so please explain your answer.

Let's remember the original optical flow formula:

$$0 = I_t + I_x u + I_y v$$

Note that for each pixel we have two unknowns (u and v), and only one equation.

1. What is the constant brightness constraint? In you answer relate to:
   a. How does it help us to solve the optical flow between two images?
   b. Is this assumption correct in real world scenarios? (what happens when we reflective objects in the image?).
2. What is the aperture problem and what can we do about it?
3. How did Lucas-Kanade solve the optical flow problem? (what did they assume about the movement of each pixel?). Hint: it's related to the movement of the neighbourhood of each pixel.
4. Is Lucas-Kanade assumption true around the object boundaries? Why?
5. Propose a general idea how to correctly find the optical flow on the object's boundaries, given you can get any input you desire (except from the true movement of each pixel). For example, you can get a depth map / label image (you know for each pixel to which object it belongs to or what is its depth). Write which inputs you assume to have and the general idea of your solution.

Part 2 – Lucas-Kanade Optical Flow

In this part you are required to compute the velocity fields (u,v) between images I1 and I2. Using those values, you are required to back-project from I2 to I1 (the result should be very similar to I1, in the sense that there should be little movement between the two frames).

The Optical Flow (velocity field) will be computed using the Lucas-Kanade Algorithm (as you learned in class). For this assignment you will use a 2D-translation warp.

***Note:** in order to read I1, I2 use the library scipy.io.

Reminder:

LK Overall Algorithm:

Input: I1,I2

Output: P- warp parameters

```
1. Build pyramid*
2. For each level of the pyramid
   Iterate:
   $\Delta P = -(B^T B)^{-1} B^T I_t$
   $P = P + \Delta P$
```

Notice: For 2D-translation-

$$\frac{\partial w}{\partial P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (Jacobian)$$

$$\rightarrow \nabla I_2 \frac{\partial w}{\partial P} = [I_x \; I_y]$$

$$\rightarrow B = [I_x \; I_y] \in R^{nX2}$$

$$\rightarrow \Delta P = -(B^T B)^{-1} B^T I_t = -\begin{bmatrix} I_x^T I_x & I_x^T I_y \\ I_x^T I_y & I_y^T I_y \end{bmatrix}^{-1} \begin{bmatrix} I_x^T I_t \\ I_y^T I_t \end{bmatrix}$$

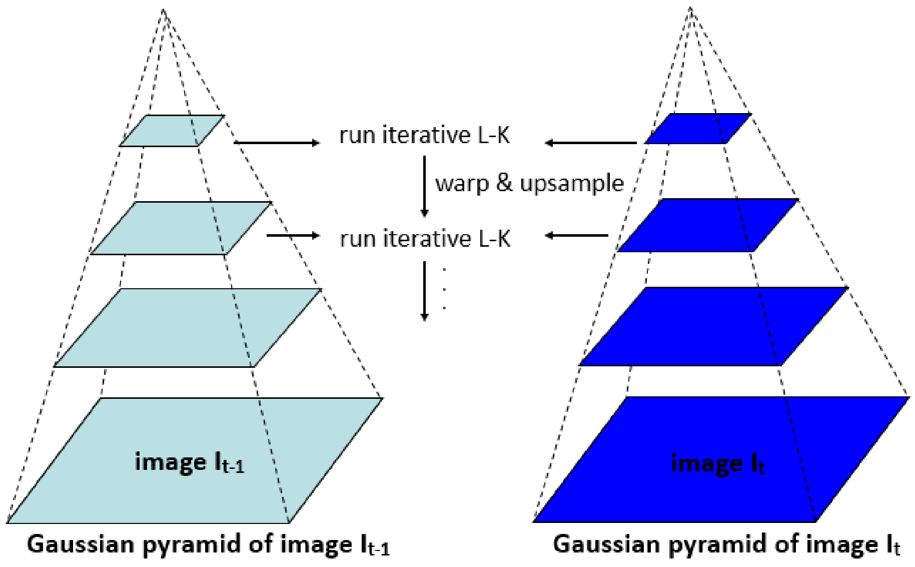
Build Pyramid Algorithm*:

Input: I-image, K-number of levels

Output: Pyramid P

```
1. $P_1 = I * g$        (g is a Gaussian Filter)
2. For i=2:K
   $P_i$ = subsample $P_{i-1}$
   $P_i = P_i * g$
```

Graphical representation of the whole Algorithm:



For this part you will need to create the following functions:

1. LucasKanadeOpticalFlow– this is your main function, you will call the following functions from it.
2. LucasKanadeStep
3. WarpImage

- **<u>LucasKanadeStep</u>**

  This function will compute optical flow using a **<u>single iteration</u>** of Lucas-Kanade Algorithm.

  Use the following API:

  (du,dv)= LucasKanadeStep(I1,I2,WindowSize)

  <u>Input:</u>

  I1,I2- images (we want to back-project from I2 to I1)

  WindowSize- size of local neighbourhood around the pixel (in which we assume that the optical flow is constant)

  <u>Output:</u>

  (du,dv) – Optical flow warp parameters (velocity fields)

- **<u>WarpImage</u>**

  This function back projects image I2 using u and v. the purpose of this is to see how we go back from I2 to I1 by knowing the motion between both images. The result should be very similar to I1.

  Use the following API:

  I_warp = WarpImage(I,u,v)

  <u>Input:</u>

  I - image

  (u,v) – Optical flow warp parameters (velocity fields)

  <u>Output:</u>

  I_warp – warped image

  ***Hint:** use the function meshgrid as well as be-linear interpolation.

  ***Note:** you might get 'holes' in the resulting image. replace them with the value they had in the original image (I2).

- **<u>LucasKanadeOpticalFlow</u>**

This function will compute optical flow using Iterative Pyramid Lucas-Kanade Algorithm

Use the following API:

(u,v) =LucasKanadeOpticalFlow(I1,I2,WindowSize,MaxIter,NumLevels)

<u>Input:</u>

I1,I2- images (we want to back-project from I2 to I1)

WindowSize- size of local neighbourhood around the pixel (in which we assume that the optical flow is constant)

MaxIter- maximum iterations that we allow (for each level of the pyramid)

NumLevels- number of levels in the image pyramid

<u>Output:</u>

(u,v) – Optical flow warp parameters (velocity fields)

***Hint:** for building image pyramids use OpenCV functions pyrUp and pyrDown.
***Note:** please notice that when moving up between levels of the pyramid you should upsample u and v (they need to be the same size as the image) and multiply their value by 2.

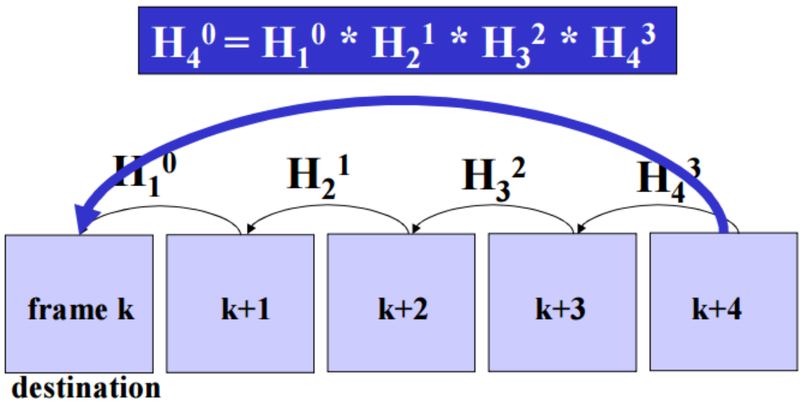
Part 3 – Video Stabilization

In this part you are required to stabilize a video file using Lucas-Kanade Optical Flow. For that you will use the function you created in Part 2.

Given a video sequence, for each frame compute the LK Optical Flow with respect to the previous warped frame and warp the entire image.

For example, suppose you've warped frame k to frame 1. Now, compute the warp from frame k+1 to frame k, add the warp you computed from frame k to frame 1 and use the combined warp, to warp frame k+1 to the coordinate system of frame 1.

Graphical representation:



Use the following API:

StabilizedVid = LucasKanadeVideoStabilization(InputVid,WindowSize,MaxIter,NumLevels)

Input:

InputVid- video we wish to stabilize

WindowSize (for LK)- size of local neighbourhood around the pixel (in which we assume that the optical flow is constant)

MaxIter (for LK)- maximum iterations that we allow (for each level of the pyramid)

NumLevels (for LK)- number of levels in the image pyramid

Output:

StabilizedVid- stabilized version of InputVid (**save it as: ‘StabilizedVid_ID1_ID2.avi’**).

__Notes:__

- Print a message to the screen specifying which frame is being processed.
- You should run the video stabilization on the entire frames in the video (don’t assume you know the number of frames in the video, extract the number of frames from the video object). **This note goes to all other properties of the input video (frame rate, image size).**

General Notes

- In both parts, you may add functions as you please, which will be called from the functions you were given.
- In the main file (HW2.py) you will call all your functions. Don't change this file. Only thing that you need to do is determine the values of the parameters: **WindowSize,MaxIter,NumLevels.** Choose values that gives the best result. I will test your functions from your main file.
- Files that you need to submit:
  - PDF file (**ex2_ID1_ID2.pdf**) with answers to part 1.
  - StabilizedVid_ID1_ID2.avi
  - HW2.py (fill there your IDs)
  - ex2_funcsions.py, which would contain the implementation of the following functions:
    - LucasKanadeOpticalFlow.
    - LucasKanadeStep.
    - WarpImage.
    - LucasKanadeVideoStabilization.
    - Any other function that you written for this assignment

Any compilation errors of any sort equal zero points for that respective question. Please double check your solutions before submitting them!

ALL FILES WILL BE PLACED IN ONE ZIP FILE CALLED: **vp2020_ex2_ID1_ID2.zip** , where ID1&ID2 should be your ID's.

Only one of each student pair should submit the HW.

Enjoy and Good Luck! ☺