ENGG5104 Image Processing and Computer Vision

Assignment 3 – Optical Flow Estimation

Due Date: 11:59pm on Sunday, March 20th, 2016

1. Background

Motion estimation among a sequence of images is a widely used task in computer vision, such as image segmentation, object classification, and driver assistance.

In this community, researchers use 'optical flow' to describe the motion field. 'Optical flow' is a dense vector field, where a 2D displacement vector is calculated for each pixel. The displacement vector points to its corresponding pixel from another image. An example is shown in Fig. 1.

FIGURE 1 OPTICAL FLOW EXAMPLE

Since much of the structural information of a 3D scene gets lost in the imaging process, so does the motion information. The estimation of the accurate 2D motion in an image sequence is therefore ill-posed and has to be aided by additional priors, such as the smoothness property of motion.

2. Horn-Schunck Method

2. 1 Objective function

Horn-Schunck method is a classical optical flow estimation algorithm. It assumes that motion field is piece-wised smooth in most regions. Thus, the methods tries to minimize distortions in flow and prefers solutions which show more smoothness.

In this assignment, we will implement the modern version of Horn-Schunck method. Different from the basic method mentioned in lecture slides which solves the flow field in one pass and in single resolution, we use

pyramid-based coarse-to-fine scheme to achieve better performance, similar to that of Lucas-Kanade method. Many current optical flow algorithms are built upon its framework.

We model this problem into an optimization problem. We first write down the objective function, then by minimizing this objective function, we will get the optical flow field.

We denote a pixel as $\boldsymbol{p} = (x, y)$ and the corresponding flow vector is $\boldsymbol{w}(\boldsymbol{p}) = (u(\boldsymbol{p}), v(\boldsymbol{p}))$, where u(**p**) and v(**p**) are the horizontal and vertical components of the flow field, respectively. Under the brightness constancy assumption, the corresponding pixels should have consistent colors. In addition, flow field should be piecewise smooth. This results in an objective function as follows (continuous spatial domain)

$$E_1(u, v) = \int |I_2(\boldsymbol{p} + \boldsymbol{w}) - I_1(\boldsymbol{p})|^2 + \lambda(|\nabla u|^2 + |\nabla v|^2) d\boldsymbol{p} \tag{1}$$

where $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$ is the gradient operator, λ is regularization weight, I_1 and I_2 are two images. $|x|^2$ means l2-norm for x (consider x as a vector).

Hints: Intuitively, $I_1(\boldsymbol{p})$ (pixel $\boldsymbol{p}$ in image I_1) corresponds to $I_2(\boldsymbol{p} + \boldsymbol{w})$ (pixel $(\boldsymbol{p} + \boldsymbol{w})$ in image I_2), which should be similar color. So $|I_2(\boldsymbol{p} + \boldsymbol{w}) - I_1(\boldsymbol{p})|^2$ should be small. And flow field $\boldsymbol{w}(\boldsymbol{p}) = (u(\boldsymbol{p}), v(\boldsymbol{p}))$ is assumed to be smooth, so $|\nabla u|^2$ and $|\nabla v|^2$ should be small.

2.2 Variational Framework

We use an iterative solver to minimize Eq. (1). Our solver first assumes that a current estimate of the flow field is $\boldsymbol{w}$, and now we need to estimate the best increment $\Delta_{\boldsymbol{w}} = (\Delta_u, \Delta_v)$, to update $\mathbf{w}$. The objective function in Eq. (1) is then changed to

$$E_2(\Delta_u, \Delta_v) = \int |I_2(\boldsymbol{p} + \boldsymbol{w} + \Delta_{\boldsymbol{w}}) - I_1(\boldsymbol{p})|^2 + \lambda(|\nabla(u + \Delta_u)|^2 + |\nabla(v + \Delta_v)|^2) d\boldsymbol{p} \tag{2}$$

Let

$$I_z(\boldsymbol{p}) = I_2(\boldsymbol{p} + \boldsymbol{w}) - I_1(\boldsymbol{p}) \tag{3}$$

$$I_x(\boldsymbol{p}) = \frac{\partial}{\partial x} I_2(\boldsymbol{p} + \boldsymbol{w})$$

$$I_y(\boldsymbol{p}) = \frac{\partial}{\partial y} I_2(\boldsymbol{p} + \boldsymbol{w})$$

Thus $I_2(\boldsymbol{p} + \boldsymbol{w} + \Delta_{\boldsymbol{w}}) - I_1(\boldsymbol{p})$ can also be linearized by its first-order Taylor expansion

$$I_2(\boldsymbol{p} + \boldsymbol{w} + \Delta_{\boldsymbol{w}}) - I_1(\boldsymbol{p}) \approx I_z(\boldsymbol{p}) + I_x(\boldsymbol{p}) \cdot \Delta_u(\boldsymbol{p}) + I_y(\boldsymbol{p}) \cdot \Delta_v(\boldsymbol{p})$$

For ease of computation, we vectorize u, v, Δ_u, Δ_v into U, V, Δ_U, Δ_V (column vectors). Let $\boldsymbol{I}_x = diag(I_x)$ and $\boldsymbol{I}_y = diag(I_y)$ be diagonal matrices where the diagonal values are vectorized images I_x and I_y. We use $\boldsymbol{D}_x$ and $\boldsymbol{D}_y$ to denote the matrix corresponding to x- and y-derivative filters, i.e. $\boldsymbol{D}_x U$ results in the vectorized form of image convolution result: $u * [1 \; -1 \; 0]$.

We introduce column vector δ_p that has only one nonzero value 1.0 at location $\boldsymbol{p}$, e.g., $\delta_p^T \boldsymbol{I}_z \delta_p = I_z(\boldsymbol{p})$ and $\delta_p^T \boldsymbol{I}_x \delta_p = I_x(\boldsymbol{p})$. The continuous function in Eq. (2) can be discretized as

$$E_2(\Delta_U, \Delta_V) = \sum_p \left(\delta_p^T\left(I_z + \boldsymbol{I}_x \Delta_U + \boldsymbol{I}_y \Delta_V\right)\right)^2 + \lambda(\left(\delta_p^T \boldsymbol{D}_x (U + \Delta_U)\right)^2 + \left(\delta_p^T \boldsymbol{D}_y (U + \Delta_U)\right)^2$$

$$+ \left(\delta_p^T \boldsymbol{D}_x (V + \Delta_V)\right)^2 + \left(\delta_p^T \boldsymbol{D_y} (V + \Delta_V)\right)^2) \quad (4)$$

2.3 Optimization

The main idea to solve the above equation is to find Δ_U, Δ_V so that the gradient $\left[\frac{\partial E_2}{\partial \Delta_U}; \frac{\partial E_2}{\partial \Delta_V}\right] = 0$.

We can derive

$$\frac{\partial E_2}{\partial \Delta_U} = 2\sum_p \boldsymbol{I}_x \delta_p \delta_p^T \boldsymbol{I}_x \Delta_U + \boldsymbol{I}_x \delta_p \delta_p^T \left(\boldsymbol{I}_z + \boldsymbol{I}_y \Delta_V\right) + \lambda\left(\boldsymbol{D}_x^T \delta_p \delta_p^T \boldsymbol{D}_x + \boldsymbol{D}_y^T \delta_p \delta_p^T \boldsymbol{D}_y\right)(U + \Delta_U) \quad (5)$$

$$= 2((\boldsymbol{I}_x^2 + \lambda \boldsymbol{L})\Delta_U + \boldsymbol{I}_x \boldsymbol{I}_y \Delta_V + \boldsymbol{I}_x I_z + \lambda \boldsymbol{L} U) \quad (6)$$

From Eq. (4) to Eq. (5) we use $\frac{d}{dx}(x^T \boldsymbol{A} x) = 2\boldsymbol{A}x$, $\frac{d}{dx}(x^T b) = b$, where x and b are column vectors and $\boldsymbol{A}$ is a matrix. From Eq. (5) to Eq. (6) we use the fact that $\sum_p \delta_p \delta_p^T$ is the identity matrix, and $\boldsymbol{I}_x$, $\boldsymbol{I}_y$ are diagonal matrices. $\boldsymbol{L}$ is a Laplacian filter defined as

$$\boldsymbol{L} = \boldsymbol{D}_x^T \boldsymbol{D}_x + \boldsymbol{D}_y^T \boldsymbol{D}_y \quad (7)$$

Similarly one can show

$$\frac{\partial E}{\partial \Delta_V} = 2((\boldsymbol{I}_y^2 + \lambda \boldsymbol{L})\Delta_V + \boldsymbol{I}_x \boldsymbol{I}_y \Delta_U + \boldsymbol{I}_y I_z + \lambda \boldsymbol{L} V)$$

Therefore, solving $\left[\frac{\partial E}{\partial \Delta_U}; \frac{\partial E}{\partial \Delta_V}\right] = 0$ can be performed in the following linear system

$$\begin{bmatrix} \boldsymbol{I}_x^2 + \lambda \boldsymbol{L} & \boldsymbol{I}_x \boldsymbol{I}_y \\ \boldsymbol{I}_x \boldsymbol{I}_y & \boldsymbol{I}_y^2 + \lambda \boldsymbol{L} \end{bmatrix} \begin{bmatrix} \Delta_U \\ \Delta_V \end{bmatrix} = -\begin{bmatrix} \boldsymbol{I}_x I_z + \lambda \boldsymbol{L} U \\ \boldsymbol{I}_y I_z + \lambda \boldsymbol{L} V \end{bmatrix} \quad (8)$$

2.4 A coarse-to-fine Schema

In this framework, we also build a coarse-to-fine refining scheme on Gaussian pyramids constructed on input images. Each two successive levels are with downsampling rate 0.8 (can be adjusted), i.e., each low-level pixels corresponds to 0.8 up-level pixels. It is achieved by bicubic or bilinear interpolation.

2.5 Evaluation

Once you build the pyramids and solve Eq. (8) in each level iteratively, the result in the finest level is obtained. We evaluate optical flow results by two measures: *Average Angle Error* (AAE) and *End Point Error* (EPE). They are defined by comparing your flow result with the ground truth flow. The lower the AAE and EPE are, the better your optical flow performance is.

Also you can visually estimate your result from visualization of the flow map (using `flowToColor` function).

2.6 Algorithm Flowchart

```
Input: frame1,frame2,λ
Build image pyramid; %TODO#1
Initialize flow = 0;
For I = numPyramidLevel downto 1
   Initialize flow from previous level;
   Build gradient matrix and Laplacian matrix; %TODO#2
   For j = 1:maxWarpingNum
      Warp image using flow vector; %TODO$3
      Compute image gradient Ix, Iy, and Iz; %TODO#4
      Build linear system to solve HS flow; %TODO#5
      Solve linear system to compute the flow;
      Use median filter to smooth the flow map; %TODO#6
   EndFor
EndFor
Output: flow
```

3. Existing Files

We provide you the following files in the skeleton code:

`runflow.py`: The overall code to run the system.

`estimateHSflow.py`: Estimate the HS flow

`flowTools.py`: Useful tools to read / write flow file, visualize flow field and calculate AAE, EPE

In file `flowTools.py`, there is auxiliary functions you need to use to build the optical flow framework, including the IO for flow file (`readFlowFile()`, `writeFlowFile()`), plot the flow in a color format (`flowToColor()`), compute the error between your flow and ground truth flow (`flowAngErr.m`).

We provided 8 test cases in folder "data/":

The input two-frame data are put into subfolder "other-data/"

The ground-truth flow is put into subfolder "other-gt-flow/"

You need to test your implementations on the 8 cases and show your result (colorized flow and AAE / EPE) in the report.

4. Implementation Details

Given the skeleton code for Horn-Schunck optical flow, you need to write your own code in it. The requirement is as follows.

TODO #1: build Gaussian pyramid for coarse-to-fine optical flow estimation (func:*estimateHSflow()*)

For each input image, the first layer of the pyramid is the original image.

For each layer, first adopt a Gaussian filter to remove details. Then resize it to 0.8 of its current size to obtain next layer.

Hints: The 0.8 ratio can be achieved by calling `cv2.resize()` in Python.

TODO#2: build gradient matrix ($\boldsymbol{D_x}, \boldsymbol{D_y}$) in Eq.(7) and Laplacian matrix $\boldsymbol{L}$ in Eq.(8) (func: *estimateHSflowlayer()*)

$\boldsymbol{D}_x U$ is the x-direction gradient of image produced by convolution $u * [1 \; -1 \; 0]$. $\boldsymbol{D}_y U$ is the y-direction gradient of image $u * [1 \; -1 \; 0]^T$.. They are used in Eq. (7) for calculation.

Hints: 1. $\boldsymbol{D}_x, \boldsymbol{D}_y$ are constructed as follows. The input image is with size $[height, weight]$. Define $n = height \times weight$. The matrix $\boldsymbol{D}_x, \boldsymbol{D}_y$ are thus with size $n \times n$.

First, build an all 1 vector as `e = numpy.ones([n,1])`.

For $\boldsymbol{D}_y$, use `scipy.sparse.spdiags()` in Python to set the main diagonal values as `-e`, and set its first diagonal (the one above the main diagonal) as `e`.

For $\boldsymbol{D}_x$, use `scipy.sparse.spdiags()` in Python to set the main diagonal values as `-e`, and set its $height$-th diagonal (the $height$-th diagonal above the main diagonal) as `e`.

The different diagonals are illustrated in Fig. 2. The main diagonal is labeled as 0, and the first diagonal is labelled as 1 respectively.

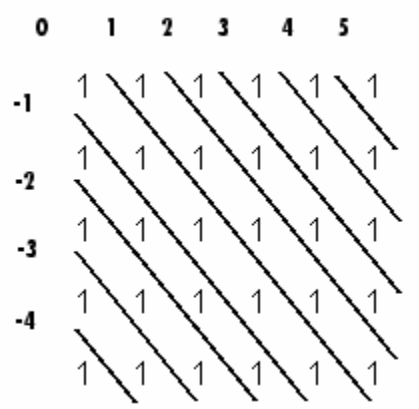

FIGURE 2 DIAGONAL ILLUSTRATION

2. Laplacian matrix $\boldsymbol{L}$ is defined in Eq. (7) from the gradient matrix.

TODO#3: warp image using the flow vector (func: *estimateHSflowlayer()*)
In each level, once you compute flow in one pass, warp the left image to the right based on this flow. Warping frame2 into frame1 is via `cv2.remap()` function.
Hints: An example is given in *runflow.py*

TODO#4: compute image gradient I_x, I_y, and I_z (func: *estimateHSflowlayer()*)
Compute image gradient I_x and I_y using the given gradient kernel.
Compute I_z using Eq. (3).
Hints: 1. Both I_x and I_y are computed on the warped image from frame2. You can use the gradient kernel $\boldsymbol{D}_x$ and $\boldsymbol{D}_y$ computation. A better choice is to convolve the images with kernel `h = [1 -8 0 8 -1]/12` for better result generation.

2. I_z is computed as the difference between warped image and frame1 for every corresponding pixel pair..

TODO#5: build linear system to solve HS flow (func: *estimateHSflowlayer()*)
In this step, you need to build matrix A and b for the linear system $Ax = b$ in Eq. (8), using previous computed image gradient I_x, I_y, I_z, and the Laplacian matrix.
Hints: 1. In Eq. (8), $\boldsymbol{I}_x = diag(I_x)$ and $\boldsymbol{I}_y = diag(I_y)$. All U, V, I_z are the vectors.

2. Use sparse matrix to save memory. Search Python and Scipy documents online to understand how to use sparse matrices. (use latest version of Scipy lib, if you cannot find certain functions)

3. Solve sparse linear system using functions from module `scipy.sparse.linalg`

TODO#6: use median filter to smooth the flow result in each level in each iteration (func: *estimateHSflowlayer()*)
Hints: Use Python function `cv2.medianBlur()` and set the default window size as [7, 7] (can be adjusted).

4. Marking

Basic Part (100%)

(80%) Implementation of Horn-Schunk optical flow. Tune parameters for different cases to achieve the best performance for them, respectively.

(20%) Report (including flow color map, AAE and EPE results for the 8 test cases, brief introduction of the algorithms, execution time, the extra credit parts etc.)

Extra Credit

You can consider the following bonus:

Bonus 1 (10%): Use robust function instead of L2 norm in Eq. (1) as

$$E(u,v) = \int \psi(|I_2(\mathbf{p}+\mathbf{w}) - I_1(\mathbf{p})|^2) + \lambda\phi(|\nabla u|^2 + |\nabla v|^2)d\mathbf{p}$$

The robust function ψ and ϕ are defined as L1 norm, which result in a piecewise smooth flow field, i.e., $\psi(x^2) = \sqrt{x^2+\epsilon^2}, \phi(x^2) = \sqrt{x^2+\epsilon^2}$ $(\epsilon = 0.001)$. The above equation can be solved using *iterative reweighted least square* (IRLS) method. We list the following changes you need to make in order to change your code into robust version.

First, change the Laplacian filter in Eq. (7) as the generalized Laplacian filter as

$$\boldsymbol{L} = \boldsymbol{D}_x^{\mathrm{T}}\boldsymbol{\Phi}'\boldsymbol{D}_x + \boldsymbol{D}_y^{\mathrm{T}}\boldsymbol{\Phi}'\boldsymbol{D}_y$$

And the linear equation in Eq. (8) is changed as,

$$\begin{bmatrix} \boldsymbol{\Psi}'\boldsymbol{I}_x^2 + \lambda\boldsymbol{L} & \boldsymbol{\Psi}'\boldsymbol{I}_x\boldsymbol{I}_y \\ \boldsymbol{\Psi}'\boldsymbol{I}_x\boldsymbol{I}_y & \boldsymbol{\Psi}'\boldsymbol{I}_y^2 + \lambda\boldsymbol{L} \end{bmatrix} \begin{bmatrix} \Delta_U \\ \Delta_V \end{bmatrix} = -\begin{bmatrix} \boldsymbol{\Psi}'\boldsymbol{I}_x\boldsymbol{I}_z + \lambda\boldsymbol{L}U \\ \boldsymbol{\Psi}'\boldsymbol{I}_y\boldsymbol{I}_z + \lambda\boldsymbol{L}U \end{bmatrix}$$

The weight $\boldsymbol{\Phi}'$ and $\boldsymbol{\Psi}'$ are defined as $\boldsymbol{\Phi}' = \mathrm{diag}(\phi')$ and $\boldsymbol{\Psi}' = \mathrm{diag}(\psi')$, where $\phi' = [\phi'_p]$ is a vector with each value corresponds to the robust function for each pixel p. For $\phi(x^2) = \sqrt{x^2+\epsilon^2}$, its first order derivative is

$$\phi'(x^2) = \frac{1}{\sqrt{x^2+\epsilon^2}}$$

It is similar for the definition of ψ'.

In IRLS, instead solving the linear system in Eq. (8) for one time to obtain the final result in our original version, we perform several following fixed-point iterations ($dU \to \Delta_U, dV \to \Delta_V$ in the following figure):

(a) Initialize $dU = 0,\ dV = 0$.

(b) Compute the "weight" $\boldsymbol{\Psi}'$ and $\boldsymbol{\Phi}'$ based on the current estimate dU and dV.

(c) Solve the following linear equation

$$\begin{bmatrix} \boldsymbol{\Psi}'\mathbf{I}_x^2 + \alpha\mathbf{L} & \boldsymbol{\Psi}'\mathbf{I}_x\mathbf{I}_y \\ \boldsymbol{\Psi}'\mathbf{I}_x\mathbf{I}_y & \boldsymbol{\Psi}'\mathbf{I}_y^2 + \alpha\mathbf{L} \end{bmatrix} \begin{bmatrix} dU \\ dV \end{bmatrix} = -\begin{bmatrix} \boldsymbol{\Psi}'\mathbf{I}_x I_z + \alpha\mathbf{L}U \\ \boldsymbol{\Psi}'\mathbf{I}_y I_z + \alpha\mathbf{L}V \end{bmatrix} \quad \text{(A.12)}$$

(d) If dU and dV converge, stop; otherwise, goto (b).

Refer to the Appendix A of Ce Liu's thesis for more detailed derivations.

Bonus 2 (10%): Change the median filter in *TODO#6* to weighted median filter.

The median filter with window size 7×7 finds the median value in the local region of the flow map as show in Fig. 3(a). The weight for each pixel value can be regarded as 1.

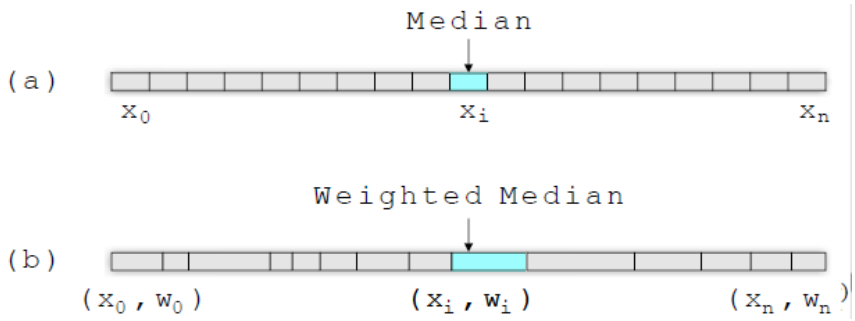

FIGURE 3. MEDIAN FILTER AND WEIGHTED MEDIAN FILTER

The weighted median filter has different weights for each pixels. For each local region 7×7 centered at location i in our case, the weight for each neighboring pixel j (in the local region) can be determined as color difference on frame1 as

$$w_j = \exp(-\frac{|I_i - I_j|^2}{2\sigma^2})$$

where I_i and I_j are the pixel values for pixels i and j in original frame1. σ is the range variance and is usually set as 0.1. Again sort all local values in each 7×7 region via the flow map, and the value located at 50% of total weight ($= \frac{1}{2} \cdot \sum_j w_j$) is the 'weighted median value', as illustrated in Fig. 3(b).

Note that you should change the window size to find the best one for the best performance for input images.

Bonus 3 (10%): 5 students producing the best results regarding the EPE measures (average of 8 cases) will automatically win this bonus.

5. Submission

Your submission should contain two things: **code files (in Python)** and **a report (in HTML)**

Python Code

The completed code files, including your implementation and the skeleton code we provide.

Report Requirements

1. The report must be written in **HTML**.
2. In the report, you should describe your algorithm and any decisions you made to write your algorithm a particular way.
3. You should also show and discuss the results of your algorithm (including results, parameter settings, and analysis etc.).

4. Discussion on algorithms' efficiency (based on time elapsed).
5. Hightlight any extra credit you did.
6. Ideas and algorithms from others **MUST** be clearly claimed. List papers or links in the **Reference** section if needed.
7. Feel free to add any other information you feel is relevant.

Submission Format

The folder you hand in must contain the following:

1. **README.txt** - containing anything about the project that you want to tell the TAs, maybe brief introduction of the usage of the codes
2. **code/** - directory containing all code for this assignment.
3. **html/** - directory containing all your html report for this assignment (including images). Your web page should only display compressed images (e.g. jpg or png). (case sensitive)
4. **html/index.html** - home page for your results. (case sensitive)

Please compressed the folder into <*your student ID*>-asgn3**.zip** or <*your student ID*>-asgn3**.rar**, and upload it to e-Learning system.

6. Remarks

1. Five late days total, to be spent wisely.
2. 20% off per day for late submission.
3. Your mark will be deducted if you do not follow the instructions for the submission format.
4. The assignment is to be completed **INDIVIDUALLY** on your own.
5. You are encouraged to use methods in related academic papers.
6. Absolutely **NO sharing or copying** of code! **NO sharing or copying** of reports! Offender will be given a failure grade and the case will be reported to the faculty.