CS4495 Fall 2011 — Computer Vision
Project 5: Optic Flow

DUE: ****TUESDAY*** November 29 - 4:30pm

In class we discussed *optic flow* as the problem of computing a *dense flow field* where a flow field is a vector field $< u(x,y), v(x,y) >$. We discussed a standard method — Hierarchical Lucas and Kanade — for computing this field. The basic idea is to compute a single translational vector over a window centered about a point that best predicts the change in intensities over that window by looking a the image gradients.

For this problem set you will implement the necessary elements to compute the LK optic flow field. This will include the necessary functions to create a Gaussian pyramid.

As always, a reminder as to what you hand in: A Zip file that has

1. Output: Images (either as JPG or PNG or some other easy to recognize format) clearly labeled using he convention Proj<number>-<question number>-<question sub>-counter.
2. Code you used for each question. It should be clear how to run your code to generate the results. Code should be in different folders for each main part with names like Proj2-1-code. For some parts — especially if using MATLAB — the entire code might be just a few lines.
3. Sometimes the question asks for output other than an image, such as a matrix or a written response — put all non-image output in a readme.txt file in the code folder.
4. Zip file should be name <gt_username>.zip, where <gt_username> is replaced by your Georgia Tech username as it appears in T-Square.

This project uses files stored in the directory `http://www.cc.gatech.edu/~afb/classes/CS4495-Fall2011/projectfiles/project5` There are a two mini sequences of 3 frames each called `DataSeq1` and `DataSeq2` that you will use for the assignment. There is also a test sequence called `TestSeq` that just translates a square in the middle. Use this to test your code.

1 Gaussian and Laplacian Pyramids

In class we described the Gaussian pyramid constructed using the `REDUCE` operator. On the class website's calendar there is a link to the original Burt and Adelson Laplacian Pyramid paper that defines the `REDUCE` and `EXPAND` operators.

1.1 Write a function to implement `REDUCE`. Use this to produce a Gaussian Pyramid of 4 levels (0-3). Demonstrate using the first frame of the `DataSeq1` sequence.

Output: The code, and the 4 images that make up the Gaussian Pyramid

Although the Lucas and Kanade method does not use the Laplacian Pyramid, you do need to expand the warped coarser levels (more on this in a minute). Therefore you will need to implement the `EXPAND` operator. Once you have that the Laplacian Pyramid is just some subtractions.

1.2 Write the function `EXPAND`. Using it and the above `REDUCE`, create the 4 level Laplacian pyramid of `DataSeq1` (which has 1 Gaussian image and the 3 Laplacian images).

Output: The code and the Laplacian pyramid of `DataSeq1`.

2 Lucas Kanade optic flow

Next you need to implement the basic LK step. Given the two operators above and code to create gradient images (you did those last problem set) you can implement the Lucas and Kanade optic flow algorithm. Recall that we compute the gradients I_x and I_y and then over a window centered around each pixel we solve the following:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

Remember a weighted sum could be computed by just smoothing the gradient image (or the gradient squared or product of the two gradients) by a funciton like a 5x5 box filter or a 5x5 smoothing filter (wide Gaussian) instead of actually looping. Convolution is just a normalized sum.

2.1 Write the code to do the LK optic flow estimation. For each pixel you solve the equation above. You can display the recovered values a variety of ways. The easiest is to make two new images U and V that are the x and y displacements $[u(x, y)$ and $v(x, y)]$. You can also use the MATLAB `quiver` function which draws little arrows - though you may have to scale the values. Because LK only works for small displacements with respect to the gradients, `TestSeq` is a smoothed image with the center square displace 2 pixels to the right and 3 pixels up. So when you try this on `TestSeq` you should get a simple translating square in the middle and everything else zero.

Output: The code and the images showing the x and y displacements either as images (make sure you scale them so you can see the values) or as arrows when applied to `TestSeq`.

Next you'll try it on the two Data sequences. You'll need to determine a level of the pyramid where it seems to work. To test your results you should use the recovered motion field to warp the second image back to the first (or the first to the second).

The challenge in this is to create a `Warp` function and then use it correctly. This is going to be somewhat tricky. I suggest you try and use the test sequence or some simple motion sequence you create where it's clear that a block is moving in a specific direction. The first question is are you recovering the amount you need to move to bring the second image to the first or the first to the second. If you look carefully at the slides you'll see we're solving for the amount that is the change from I_1 to I_2 Consider a case the image moves 2 pixels to the right. This means that

$I_2(5,7) = I_1(3,7)$ where I am indexing by x, y and not by row and column. So to warp I_2 back to I_1 to create a new image W, would set $W(x,y)$ to the value of $I_2(x+2,y)$. The W would align with I_1.

MATLAB has a function to do this interpolation: INTERP2. To use this, you'll need to understand the function MESHGRID - which tends to think about matrices as X and Y not rows and columns. So the call:

```
[M,N]=size(i2);
[x,y]=meshgrid(1:N,1:M);
```

creates a matrix called x which is just repeated columns of the x value - the column index, and y is the row index. In this case, if M - the number of rows - is 4 and N is 3, then x and y are 4x3. This can be confusing. Another way to think about it is that these are the x and y values(column and row) of the (i,j) locations. So if you want to get a value from somewhere near by in the image, you would add the displacement to this value. This can be seen in the following very good warp function (courtesy Yair Weiss when still a student):

```
function [warpI2]=warp(i2,vx,vy)
   % warp i2 according to flow field in vx vy
   % this is a "backwards" warp: if vx,vy are correct then warpI2==i1
   [M,N]=size(i2);
   [x,y]=meshgrid(1:N,1:M);
   warpI3=interp2(x,y,i2,x+vx,y+vy,'*nearest'); % use Matlab interpolation routine
   warpI2=interp2(x,y,i2,x+vx,y+vy,'*linear'); % use Matlab interpolation routine
   I=find(isnan(warpI2));
   warpI2(I)=warpI3(I);
```

In OpenCV there is the function `remap` which behaves similarly.

2.2 Apply your LK code to the Data1 and Data2 sequences. Because LK only works for small displacements, find a Gaussian pyramid level that works for these. To show that it works, you will show the output flow fields as above and you'll show a warped version of image 2 to the coordinate system of image 1. That is, you'll warp Image 2 back into alignment with image 1.

Output: The code and the images showing the x and y displacements for `DataSeq1` and `DataSeq2` either as images or as arrows. Then, for each sequence, show the difference image between the warped image 2 and the original image 1.

3 Hierarchical LK optic flow

Recall that the basic steps of the hierarchy:

1. Given input images L and R. Initialize $k = n$ where n is the max level.

2. REDUCE both input images to level k. Call these images L_k and R_k.

3. If $k = n$ initialize U and V to be zero images the size of L_k; otherwise expand the flow field and double (why?) to get to the next level: U = 2*EXPAND(U), V =2* EXPAND(V).

4. Warp L_k using U and V to form W_k.

5. Perform LK on W_k and R_k to yield two incremental flow fields Dx and Dy.

6. Add these to original flow: $U = U + Dx$ and $V = V + Dy$.

7. If $k > 0$ let $k = k - 1$ and goto (2).

8. Return U and V.

When developing this code you might try and create some test sequences of your own. Take a textured image and displace a center square by a fixed translation amount. Vary the amout and make sure your hieracrhical method does the right thing. In principle, for a displacement of δ pixels, you'll need to set n to (at least) $\log_2(\delta)$.

3.1 Write the function to compute the hierarchical LK (surprised, huh?). Apply to both `DataSeq1` and `DataSeq2`.

Output: The code, the displacement images and the difference image between the warped I_2 and the original I_1.