CS4495 Fall 2013 — Computer Vision
Problem Set 5: Optic Flow

DUE: Sunday November 10 - 11:55pm

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

1. Images (either as JPG or PNG or some other easy to recognize format) clearly labeled using the convention PS¡number¿–¡question number¿–¡question sub¿–counter.jpg
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 PS1-1-code. For some parts – especially if using Matlab – the entire code might be just a few lines.
3. Finally a PDF file that shows all the results you need for the problem set. This will include the images appropriately labeled so it is clear which section they are for and the small number of written responses necessary to answer some of the questions. Also, for each main section, if it is not obvious how to run your code please provide brief but clear instructions. **If there is no Zip file you will lose at least 50%!**

This project uses files stored in the directory `http://www.cc.gatech.edu/~afb/classes/CS4495-Fall2013/ProblemSets/PS5` There are a few sequences there. First, there is a test sequence called `TestSeq` that just translates a textured rectangle over a textured background. In that directory you'll find the images `Shift0, ShiftR2, ShiftR10, ShiftR20, ShiftR40` and `ShiftR5U5`. You'll use these images to test your code and to see the effect of the hierarchy. `DataSeq1` has 3 frames of the so-called "Yosemite" sequence and `DataSeq2` has larger displacements. There are also two extra credit sequences: `taxi` and `Juggle`.

1 Gaussian and Laplacian Pyramids

In class we described the Gaussian pyramid constructed using the `REDUCE` operator. On the class website's Problem Sets main page there is a link to the original Burt and Adelson Laplacian Pyramid paper that defines the `REDUCE` and `EXPAND` operators. And it's in the main directory given above.

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 images (3 Laplacian images and 1 Gaussian image) of the first image 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 filtering 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 $v(x, y)]$. These are *displacement* images. You can also use the MATLAB `quiver` function which draws little arrows - though you may have to scale the values to see the arrows. `TestSeq` has images of a smoothed image texture with the different texture center rectangle displaced by a set number of pixels. `Shift0` is the "base" image; images listed as `ShiftR2` have the center portion shifted to the right by 2 pixels; `ShiftR5U5` have the center portion shifted to the right by 2 pixels and up 5, etc.

When you try your code on the images `TestSeq` you should get a simple translating rectangle in the middle and everything else zero. Try your LK on the base image and the `ShiftR2` and between the base image and `ShiftR5U5`. Remember LK only works for small displacements with respect to the gradients so you might have to smooth your images a little to get it to work; try to find a blur amount that works for both cases but keep it as little as possible.

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 computing motion between (1) the base and `ShiftR2` and (2) the base and `ShiftR5U5`. If you blur (smooth) the images say how much you did.

2.2 Now try the code comparing the base image `Shift0` with the remaining images of `ShiftR10, ShiftR20` and `ShiftR40`. **Use the same amount of blurring as you did in the previous section.** Does it still work? Does it fall apart on any of the pairs?

Output: The code and the images showing the x and y displacements either as images or as arrows when applied to `ShiftR10, ShiftR20` and `ShiftR40`. Describe your results.

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.

2.3 Apply your single-level LK code to the`DataSeq1` sequence (from 1 to 2 and 2 to 3). 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. If you flicker (rapidly show them back and forth) the warped image 2 and the original image 1, you should see almost no motion: the second image pixels have been "moved back" to the location they came from in the first image. Same is true for image 2 to 3. Next try this for `DataSeq2` where the displacements are greater. You will likely need to use a coarser level in the pyramid (more blurring) to work for this one.

Note: for this question you are only comparing between images at some chosen level of the pyramid! In the next section you'll do the hierarchy.

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 should try it on `TestSeq` between the base and the larger shifts, or 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 amount and make sure your hierarchical 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?). First apply to the `TestSeq` for the displacements of 10, 20 and 40 pixels. Then 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 for each of the cases.

4 The Juggle Sequence

The sequence `Juggle` has significant displacement between frames — the juggled balls move significantly. Try your hierarchical LK on that sequence and see if you can warp frame 2 back to frame 1.

4.1 Apply your hierarchical LK to the `Juggle` sequence.

Output: Displacement image (as image or quiver diagram) and the difference image between the warped I_2 and the original I_1.

5 The Taxi Sequence

The sequence `taxi` has distinct moving objects. Use the output of hierarchical LK to create a segmentation of the scene. A segmentation would be a set of image regions where the motion all belongs to the same parametric flow, like translation (all pixels with same velocity) or similarity (pixels movement is determined by 4 coefficients and the $< x, y >$ value of the pixel.

5.1 Apply your hierarchical LK to the `taxi` sequence and then generate a segmentation map based upon consistent motion in the regions. State your model (translation, similarity, affine, or what?)