Problem Set 5: Optic Flow

Due Monday, March 30, 2015 at 7:00 am EST

Description

In lectures 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 at 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.

Some sequences are provided in the input directory. First, there is a test sequence called `TestSeq` that just translates a textured rectangle over a textured background. In that directory you will find the images `Shift0, ShiftR2, ShiftR10, ShiftR20, ShiftR40` and `ShiftR5U5`. You need to 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. Also, there is an extra credit sequence called `Juggle`.

Reading: Burt, P. J., and Adelson, E. H. (1983). The Laplacian Pyramid as a Compact Image Code (question 2)

What to submit

Download and unzip the `ps5` folder (also under: https://www.udacity.com/wiki/ud810): ps5.zip
Rename it to `ps5_xxxx` (i.e. `ps5_matlab`, `ps5_octave`, or `ps5_python`) and add in your solutions:

`ps5_xxxx/`

- input/ - input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other files your code generates
- ps5.m or ps5.py - code for completing each part, esp. function calls; all functions themselves must be defined in individual function files with filename same as function name, as indicated
- *.m or *.py - Matlab/Octave function files (one function per file), Python modules, any utility code
- ps5_report.pdf - a PDF file with all output images and text responses

Zip it as `ps5_xxxx.zip`, and submit on T-Square.

Guidelines

1. Include all the required images in the report to avoid penalty.

2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and GTID on the report.
5. Even if the code is not working, submit the code as the instructors can read through the algorithms to give partial credit. Comment your code appropriately.
6. Late submissions should be emailed to the TAs to be graded for partial credit.

Questions

1. Lucas Kanade Optic Flow

In this part, you need to implement the basic LK step. You need to write a code to create gradient images and 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:

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

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 function like a 5x5 or bigger (or smaller!) box filter or smoothing filter (e.g. Gaussian) instead of actually looping. Convolution is just a normalized sum.

a. 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. They can be displayed using `imagesc` along with the false-color colormap created by using `colormap jet` and show it by `colorbar`.

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 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:
- the `base Shift0` and `ShiftR2 as ps5-1-a-1.png`
- the `base Shift0` and `ShiftR5U5 as ps5-1-a-2.png`

Output (textual response):
- If you blur (smooth) the images say how much you did

b. 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 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:
- the `base Shift0` and `ShiftR10` as `ps5-1-b-1.png`
- the `base Shift0` and `ShiftR20` as `ps5-1-b-2.png`
- the `base Shift0` and `ShiftR40` as `ps5-1-b-3.png`

Output (`textual response`):
- Describe your results.

2. Gaussian and Laplacian Pyramids

In lectures we described the Gaussian pyramid constructed using the REDUCE operator. Here is the original paper that defines the REDUCE and EXPAND operators:
Burt, P. J., and Adelson, E. H. (1983). The Laplacian Pyramid as a Compact Image Code

a. 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 4 images that make up the Gaussian Pyramid (in a subplot) `as ps5-2-a-1.png`

b. 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.
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 Laplacian pyramid images *(3 Laplacian images and 1 Gaussian image)* of the first image of `DataSeq1 as ps5-2-b-1.png`

3. Warping by flow

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);
        %use Matlab interpolation routine
        warpI3=interp2(x,y,i2,x+vx,y+vy,'*nearest');
        %use Matlab interpolation routine
        warpI2=interp2(x,y,i2,x+vx,y+vy,'*linear');
        I=find(isnan(warpI2)); warpI2(I)=warpI3(I);
```

In OpenCV there is the function remap which behaves similarly.

a. 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 images showing the x and y displacements for DataSeq1 *(either as images or as arrows)* as ps5-3-a-1.png

- show the difference image between the warped image 2 and the original image 1 for DataSeq1 as ps5-3-a-2.png
- the images showing the x and y displacements for DataSeq2 *(either as images or as arrows)* as ps5-3-a-3.png
- show the difference image between the warped image 2 and the original image 1 for DataSeq2 as ps5-3-a-4.png

4. 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 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 D_x and D_y.
6. Add these to original flow: U = U + D_x and V = V + D_y.
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) $\log_2(\delta)$.

a. Write the function to compute the hierarchical LK. Apply to the TestSeq for the displacements of 10, 20 and 40 pixels (ShiftR10.png, ShiftR20.png and ShiftR40.png).

Output:

- the displacement images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as ps5-4-a-1.png

- the difference images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as ps5-4-a-2.png

b. Apply your function to DataSeq1 for the images yos_img_01.jpg, yos_img_02.jpg and yos_img_03.jpg .

Output:

- the displacement images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as ps5-4-b-1.png

- the difference images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as ps5-4-b-2.png

c. Apply your function to DataSeq2 for the images 0.png, 1.png and 2.png.

Output:

- the displacement images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as `ps5-4-c-1.png`

- the difference images between the warped I_2 and the original I_1 for each of the cases

create a subplot showing these results in one image as `ps5-4-c-2.png`

EXTRA CREDIT QUESTION (question 5)

5. 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.

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

   Output:

   - the displacement images between the warped I_2 and the original I_1 for each of the cases

   create a subplot showing these results in one image as `ps5-5-a-1.png`

   - the difference images between the warped I_2 and the original I_1 for each of the cases

   create a subplot showing these results in one image as `ps5-5-a-2.png`