Computer Vision I
CSE 252A, Winter 2007
David Kriegman

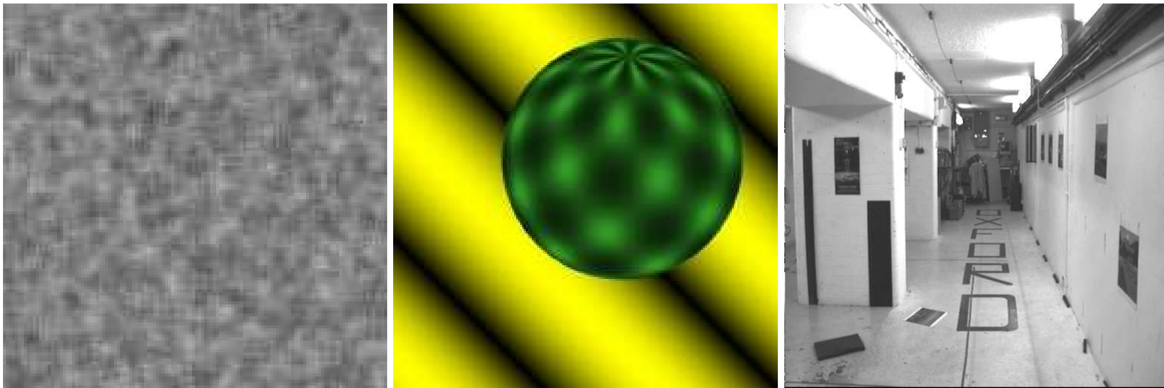
Name :
Student ID :
E-Mail :

Assignment #4 : Optical Flow

(Due date: 3/16/07)

Overview

In this assignment you will implement the Lucas-Kanade optical flow algorithm. Your input will be pairs or sequences of images and your algorithm will output an optical flow field (u, v). Three sets of test images are available from the course website. The first contains a synthetic (random) texture, the second a rotating sphere[1], and the third a corridor at Oxford university[2]. Before running your code on the images, you should first convert your images to grayscale and map intensity values to the range $[0, 1]$.



Part A: Single-Scale Optical Flow (8 points)

Implement the single-scale Lucas-Kanade optical flow algorithm. This involves finding the motion (u, v) that minimizes the the sum-squared error of the brightness constancy equations for each pixel in a window. As a reference, you should read pages 191-198 in "Introductory Techniques for 3-D Computer Vision" by Trucco and Verri[3]. Your algorithm will be implemented as a function with the following inputs,

```
function [u,v] = optical_flow(I1,I2,windowSize).
```

Here, u and v are the x and y components of the optical flow, $I1$ and $I2$ are two images taken at times $t = 1$ and $t = 2$ respectively, and windowSize is a 1×2 vector storing the width and height of the window used during flow computation. In addition to these inputs, you should add a theshold τ such that if τ is larger than the smallest eigenvalue of $A^T A$, then the the optical flow at that position should not be computed. Recall that the optical flow is only valid in regions where

$$A^T A = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_y I_x & \sum I_y^2 \end{pmatrix} \quad (1)$$

has rank 2 (why?), which is what the threshold is checking. A typical value for τ is 0.01; if you use something different please specify this in your report.

[1] Courtesy of http://www.cs.otago.ac.nz/research/vision/Research/OpticalFlow/opticalflow.html
[2] Courtesy of the Oxford visual geometry group
[3] Available from the course webpage.

What to turn in for this section

- Your matlab code.
- Quiver plots of (u, v) for the following image pairs / window sizes (6 quiver plots total),
  - SYNTH : I1 = 'synth_000.pgm', I2 = 'synth_001.pgm'
    `windowSize = 9x9`
    `windowSize = 15x15`
  - SPHERE : I0 = 'sphere.0.ppm', I1 = 'sphere.1.ppm'
    `windowSize = 15x15`
    `windowSize = 21x21`
  - CORRIDOR : I0 = 'bt.000.pgm', I1 = 'bt.001.pgm'
    `windowSize = 15x15`
    `windowSize = 21x21`
- Any modifications and/or thresholds you used.
- Does this algorithm work well for these test images? If not, why?
- Try experimenting with different window sizes. What are the tradeoffs associated with using a small vs. a large window size? Can you explain what's happening?

Part B: Multiple-Scale Lucas-Kanade (7 points)

Modify your algorithm to iteratively refine the optical flow estimate from multiple image resolutions. The basic idea is to initially compute the optical flow at a coarse image resolution. The obtained optical flow can then be upsampled and refined at successively higher resolutions, up to the resolution of the original input images. By computing the optical flow in this manner, we can circumvent the aperature problem to some degree – because the window size remains fixed across all resolutions, the algorithm effectively computes the optical flow using successively smaller aperatures. This approach also works well on images with large displacements, something the single-scale algorithm is unable to handle.

In terms of implementation, you should first modify your code from part A so that it accepts and refines parameters `u0` and `v0`, which correspond to initial estimates of the optical flow,

```
function [u,v] = optical_flow_refine(I1,I2,windowSize,u0,v0).
```

This function should refine the optical flow according to,

$$u = u_0 + \triangle u \tag{2}$$
$$v = v_0 + \triangle v \tag{3}$$

where $\triangle u$ and $\triangle v$ represent the offset between the initial estimate and the refined estimate. To compute $\triangle u$ and $\triangle v$ we simply shift the window in I2 by (u_0, v_0) and compute the optical flow between the shifted windows. Once `optical_flow_refine` is working you should write another function,

```
function [u,v] = optical_flow_ms(I1,I2,windowSize,numLevels)
```

which calls the refinement function. `optical_flow_ms` should implement the following pseudo-code,

```
Step 1. Gaussian smooth and scale I1 and I2 by a factor of 2^(1-numLevels)
Step 2. Compute the optical flow at this resolution
Step 3. For each level,
  a. Scale I1 and I2 by a factor of 2^(1-level)
  b. Upscale the previous level's optical flow by a factor of 2
  c. Compute u and v by calling optical_flow_refine with the previous
     level's optical flow
```

What to turn in for this section

- Your code
- Quiver plots of (u, v) for the following image pairs,
  - SPHERE : I0 = 'sphere.0.ppm', I1 = 'sphere.1.ppm'
    `windowSize = 15x15`, numLevels=3
    `windowSize = 21x21`, numLevels=3
  - CORRIDOR : I0 = 'bt.000.pgm', I1 = 'bt.001.pgm'
    `windowSize = 7x7`, numLevels=5
    `windowSize = 15x15`, numLevels=3
- Any modifications and/or thresholds you used.
- Did this improve results? How and why?

What to turn in for the entire report

1. Email Neil the code and report (nalldrin AT cs.ucsd.edu)
2. Hand in a report which contains
   (a) Hardcopy of code
   (b) Report which includes the output described above. Also, include a short description of your conclusions based on your experience.

Implementation Tips

Useful Matlab Commands

I'm providing the following list of Matlab hints to (hopefully) reduce debug time. Note that some of these commands require the image processing toolbox.

- Spatial gradient.
  The matlab command `gradient` will compute the gradient of an image for you. For example,
  `[Ix Iy] = gradient(I);`
- Gaussian smoothing.
  The following code snippet will smooth an image with a $\sigma = 1.5$ Gaussian kernel,
  ```
  H = fspecial('gaussian',[5 5],1.5);
  I = conv2(I,H,'same');
  ```
- Quiver plots.
  The matlab command `quiver` can be used to generate quiver plots. However, because Matlab uses a different coordinate system for images than for other types of data, getting quiver to correctly plot u and v can be tricky. To save you time, here's a code snippet that will generate a nice looking quiver plot to display your optical flow field.

```
function quiver_uv(u,v)

% Resize u and v so we can actually see something in the quiver plot
scalefactor = 50/size(u,2);
u_ = scalefactor*imresize(u,scalefactor,'bilinear');
v_ = scalefactor*imresize(v,scalefactor,'bilinear');

% Run quiver taking into account matlab coordinate system quirkiness
% and scaling the magnitude of (u,v) by 2 so it is more visible.
quiver(u_(end:-1:1,:),-v_(end:-1:1,:),2);
axis('tight');
```

- Image resizing.
  The matlab command `imresize` will do the trick. To avoid aliasing effects, make sure you specify either bilinear or bicubic filtering.

- Padding an image with zeros.
  You may find it useful at times to pad an image or matrix with zeros on all sides (to avoid boundary problems, for example). The `padarray` function greatly simplifies this task.

Some Common Pitfalls

- In part B, when upsampling u and v, the values also need to be scaled (why?).

- Accidentally switching x and y indices.

- Image boundary issues. There are many ways to handle the image boundary. The simplest methods are either to pad the input images with zeros or to only compute the optical flow for regions where the neighborhood window is well defined. A third approach that may yield slightly better results is to only use portions of the neighborhood window that lie inside the original images. Any of these approaches is perfectly fine for this assignment.