CSE252a – Computer Vision – Assignment 4
Instructor: Prof. David Kriegman.
Revision 0

Instructions:

- This assignment should be solved, *and written up* in groups of 2. Work alone *only* if you can not find a partner. No groups of 3 are allowed.
- Submit your assignment electronically by email to obeijbom@cs.ucsd.edu with the subject line *CSE252 Assignment 4*. The email should have two files attached.

  1. A pdf file with your writeup. This should have all code attached in the appendix. Name this file: CSE_252_hw4_writeup_lastname1_lastname2.pdf.
  2. A compressed archive with all your matlab code files. Name this file: CSE_252_hw4_code_lastname1_lastname2.zip.

  The code is thus attached *both* as text in the writeup appendix and as m-files in the compressed archive.
- Please make this a proper report, with methods, thoughts, comments and discussions. All code should be tucked away in an appendix.
- No physical hand-in for this assignment.
- You may do problems on pen an paper, just scan and include in the writeup pdf file.
- In general, MATLAB code does not have to be efficient. Focus on clarity, correctness and function here, and we can worry about speed in another course.

Overview

In this assignment you will implement the Lucas-Kanade algorithm for computing a dense optical flow field at every pixel. You will then implement a corner detector and combine the two algorithms to compute a flow field only at reliable corner points. 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]. I use the synthetic dataset in the instructions below. Please include results on *all three datasets* in your presentation. For reference, your optical flow algorithm should run in seconds if you vectorize properly (for example, the eigenvalues of a 2x2 matrix can be computed directly). Again, no points will be taken off for slow code, but it will make the experiments more pleasant to run.

Dense Optical Flow [5pts]

Implement the single-scale Lucas-Kanade optical flow algorithm. This involves finding the motion (u,v) that minimizes the sum-squared error of the brightness constancy equations for each pixel in a window. As a reference, 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,

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

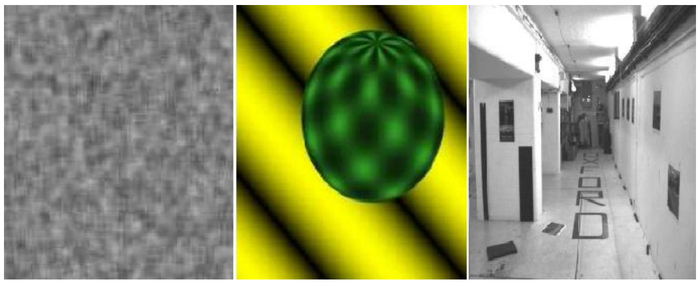


Figure 1: Input images

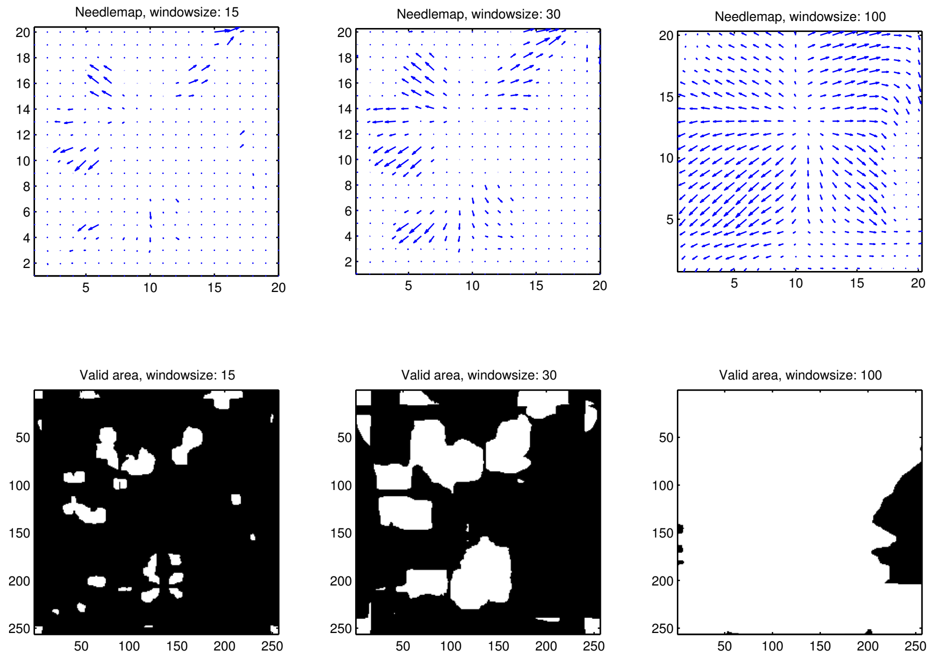


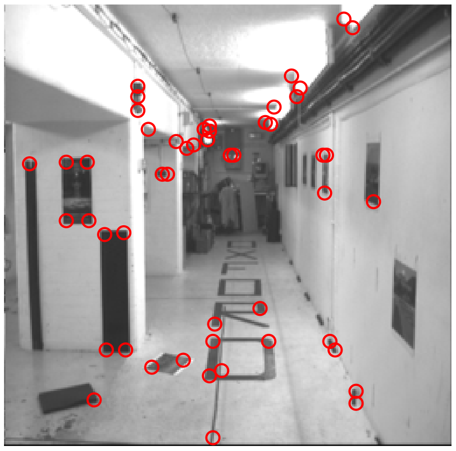
Figure 2: Result for the dense optical flow problem on the corridor image.

```
function [u, v, hitMap] = opticalFlow(I1,I2,windowSize, tau)
```

Here, u and v are the x and y components of the optical flow, hitMap a binary image indicating where the corners are valid (see below), I1 and I2 are two images taken at times $t = 1$ and $t = 2$ respectively, windowSize is the width of the window used during flow computation, and τ is the threshold such that if the smallest eigenvalue of $A^T A$ is smaller than τ, then the optical flow at that position should not be computed. Recall that the optical flow is only valid in regions where

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

has rank 2 (why?), which is what the threshold is checking. A typical value for τ is 0.01. Using this value of τ, run your algorithm on all three image sets (the first two images of each set), for three different windowsizes of your choise, to produce an image similar to Fig. 2. Also provide some comments on performance, impact of windowsize etc.


(a) Result of the corner detection problem on the corridor image.

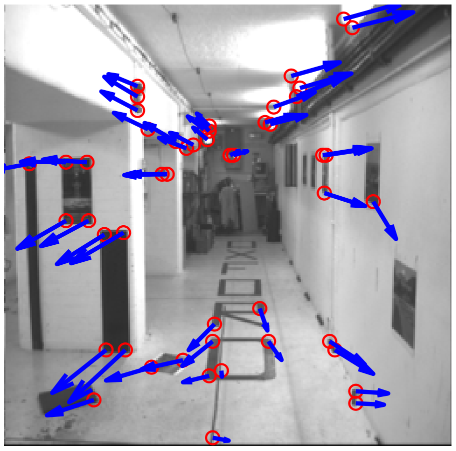

(b) Result of sparse optical flow algorithm on the corridor image.

Figure 3: Corner detection and sparse optical flow

Corner Detection [2pts]

Use your corner detector from Assignment 3 to detect 50 corners in the provided images. Use a smoothing kernel with standard deviation 1, and windowsize of 7 by 7 pixels for your corner detection throughout this assignment. Include a image similar to Fig. 3a in your report. If you were unable to create a corner detection algorihtm in the previous assignment, please email the TA for code.

Sparse Optical Flow [3pts]

Combine Parts A and B to output an optical flow field at the 50 detected corner points. Include result plots as in Fig. 3b. Select appropriate values for windowsize and τ that gives you the best results. Provide a discussion about the focus of expansion (FOE) and mark manually in your images where it is located. Is it possible to mark the FOE in all image pairs? Why not / why?

Your own images [3pts]

Run the developed code on an image pair that you capture. Does the method work? Any problems? You will probably want to down-sample the images significantly if captured with a modern digital camera.

Good luck!