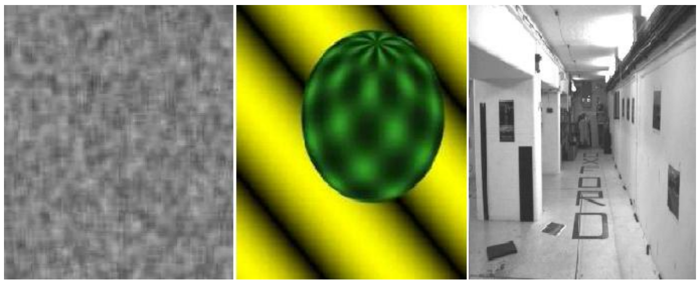


Figure 1: Input images

CSE252a – Computer Vision – Assignment 4

Instructor: Ben Ochoa
Due Date: Dec, 18th. 2014
Revision 0

Instructions:

- Submit your assignment electronically by email to iskwak+252a@cs.ucsd.edu with the subject line *CSE252 Assignment 4*. The email should have one file attached. Name this file: CSE_252_hw04_lastname_studentid.zip. The contents of the file should be:
  1. A pdf file with your writeup. This should have all code attached in the appendix. Name this file: CSE_252_hw04_lastname.pdf.
  2. All of your source code in a folder called code.
- No physical hand-in for this assignment.
- In general, code does not have to be efficient. Focus on clarity, correctness and function here, and we can worry about speed in another course.

1 Optical Flow

In this problem 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.

1.1 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

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

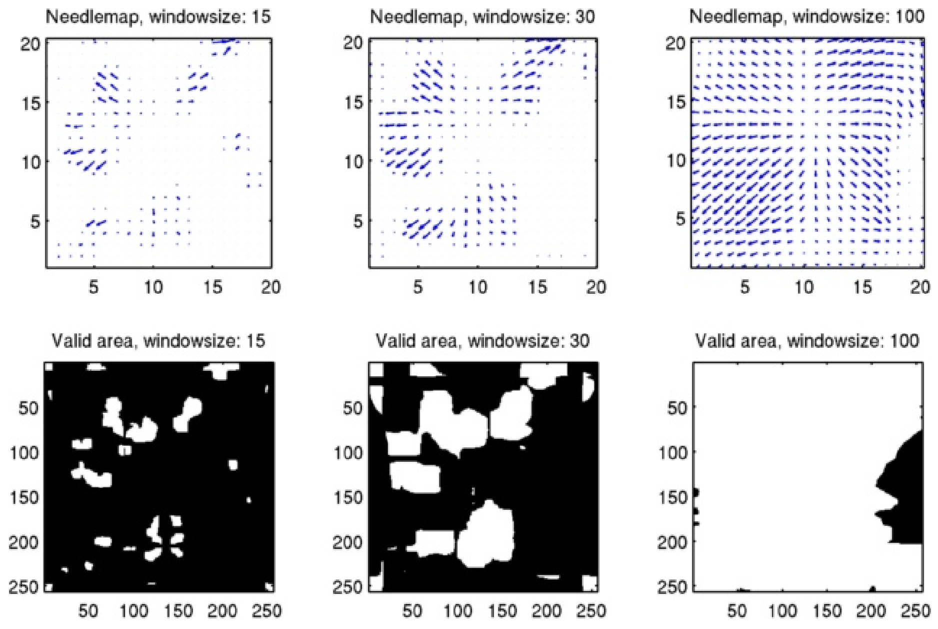


Figure 2: Result for the dense optical flow problem on the corridor image.

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,

```
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^TA 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

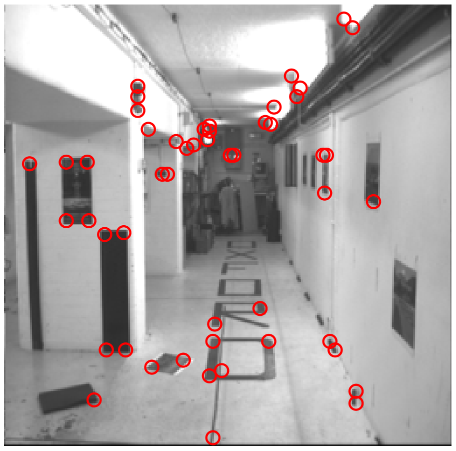
$$A^TA = \begin{pmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_yI_x & \sum I_y^2 \end{pmatrix}$$

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.

1.2 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.

[3]Availible on the course webpage. Password at Piazza


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

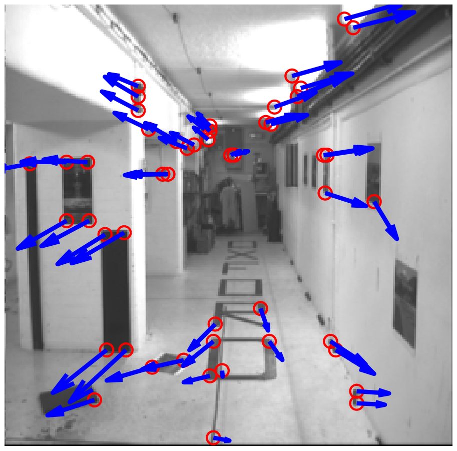

(b) Result of sparse optical flow algorithm on the corridor image.

Figure 3: Corner detection and sparse optical flow

1.3 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]

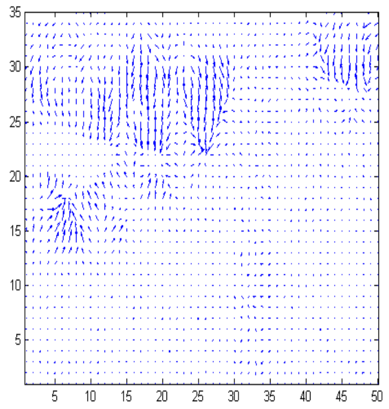
Run the developed code on an image pair that you capture. Does the method work? Any problems? You will probably want to down-sample the images significantly if captured with a modern digital camera.

2 Iterative Coarse to Fine Optical Flow [10pts]

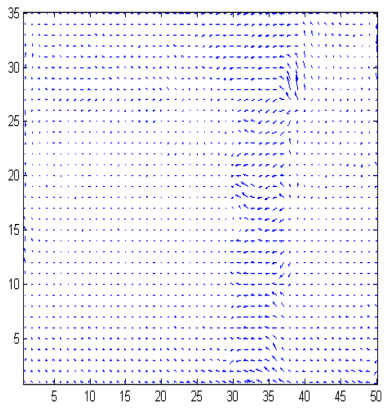
Implement the iterative coarse to fine optical flow algorithm described in the class lecture notes (pg 7 in lecture 16). Show how the coarse to fine algorithm algorithm works better on the first two frames inside of flower.zip than dense optical flow. You can do this by creating a quiver plot using your code from problem 1 and a quiver plot for the coarse to fine algorithm. Try 3 different window sizes: one of your choice, 5 and 15 pixels. Where does the dense optical flow algorithm struggle that this algorithm does better with? Can you explain this in terms of depth or movement distance of pixels? Comment on how window size affects the coarse to fine algorithm? Do you think that the coarse to fine algorithm strictly better than the standard optical flow algorithm? Example output shown in Fig 4. Note: Like in problem 1, convert the image to intensity gray scale images.

3 Bonus: Background Subtraction

In this problem you will remove the dynamic portions of an image from a static background. In this case, the camera is not moving and objects within the scene or moving. For each consecutive pair of frames, background subtraction will calculate $|I(x, y, t) - I(x, y, t-1)| > \tau$, where $I(i, j, k)$

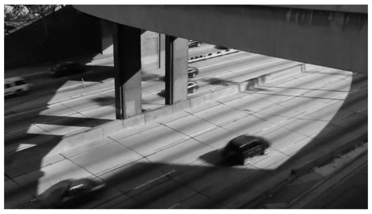


(a) Result of the dense optical flow on the flower sequence.

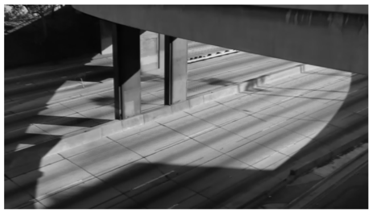


(b) Result of coarse to fine optical flow algorithm on the flower sequence.

Figure 4: Example dense optical flow vs coarse to fine



(a) A frame from the highway sequence..



(b) Result of the background subtraction algorithm.

Figure 5: Example background subtraction output.

is the i, j pixel intensity in the frame at time k, and τ is a threshold. $|I(x, y, t) - I(x, y, t-1)| > \tau$ is the foreground mask for the frame at time t. By masking out the pixels that are greater than τ, you can create an estimate background for the frame. By calculating the mean of the estimated backgrounds you can create a global background for the sequence, see Fig 5. Note: Like in problem 1, convert the image to intensity gray scale images.

Your function prototype should look like:

```
function [background] = backgroundSubtract(framesequence, tau)
```

The variable framesequence can be a string representing a directory of frame images or cell array of frames or any other reasonable input.

Run your code on the highway and truck sequence and include the backgrounds for each sequence as a figure.

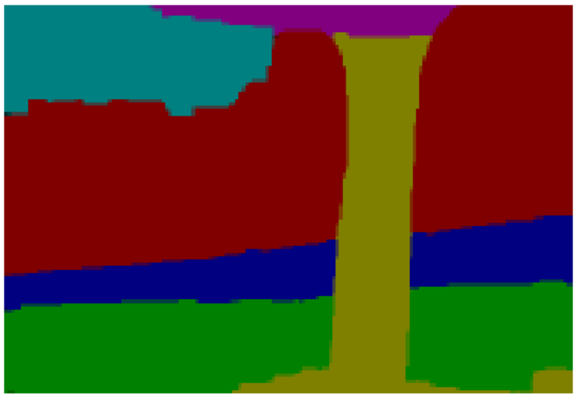


Figure 6: Example motion segmentation from the flower sequence.

4 Bonus: Motion Segmentation [5pts]

The previous algorithm only works on static cameras and a stable background. However, it is also possible to do a similar segmentation using motion cues, however it is much harder. An example motion segmentation is shown in Fig 6.

Using the outputs of your iterative coarse to fine optical flow algorithm, can you segment out the tree from the rest of the first frame of the flower sequence? *Hint: the magnitudes of the motion vectors at different depths can be quite different.*

Provide a figure of the segmented tree (an image with just the tree in it) and explain how you were able to do this. Note: Like in problem 1, convert the image to intensity gray scale images.

Good luck!