Problem Set 4

CMSC 426

Assigned Tuesday, March 14, Due Tuesday, March 28

1. Suppose we have the two images, H and I, given below:

12	11	10	11	12	13
11	10	9	10	11	12
10	9	8	9	10	11
9	8	7	8	9	10
8	7	6	7	8	9
7	6	5	6	7	8

H

13	12	11	10	11	12
12	11	10	**9c**	10	11
11	10	**9b**	8	**9a**	10
10	9	8	7	8	9
9	8	7	6	7	8
8	7	6	5	6	7

I

   a. **2 points.** Consider the pixel in image I that has an intensity of 9 and is marked with an **a**. What is the gradient at that pixel? Let's assume that the positive y direction is up.

   b. **2 points.** What is the change in intensity at that pixel?

   c. **2 points.** Using the optical flow equation: $0 \approx I_t + \nabla I \cdot [u \ v]$, write an equation that describes the motion of that pixel by the vector [*u,v*].

   d. **8 points.** Repeat these steps for the pixel with intensity 9 that is marked **b**. Derive an equation for motion based on that pixel. Combine these two equations and solve for the motion of the pixels, assuming they are undergoing the same motion.

   e. **6 points.** Suppose you do the same thing for the pixel marked **c**. Does this fit the result you obtained in the last problem? If not, can you explain why not?

2. Suppose we have an image with intensities described by the equation: $H(x,y) = 10 + (x-12)^2 + (y-10)^2$. The camera moves a little, and we get a new image described by the equation: $I(x,y) = 10 + (x-13)^2 + (y-10)^2$.
   a. **4 points.** What is the gradient of image *I* at the location *(15, 13)?*

b. **4 points.** What is the change in intensity between images H and I at the location *(15,13)?*

c. **4 points.** Using the optical flow equation: $0 \approx I_t + \nabla I \cdot [u\ v]$, we want to determine where in image *I* we will find the point that was located at *(15,13)* in image *H.* Give a linear equation that tells us a line where that location might be.

d. **8 points.** Go through the same steps and find a corresponding linear equation at the location (15,10). Using these two equations, solve for the motion.

e. **Challenge Problem 5 points:** Determine what the actual motion of the image is. Show that this is different from what you computed in part (d). Then show that if the motion had been in the same direction, but very small, solving for it in this way would have produced (almost) exactly the right answer.

3. **60 points.** Corner Detection. You should implement the corner detector that we discussed in class. Recall that to do this we perform the following steps. First, for every point in the image, we form a matrix, *C,* which is based on the image gradients in the neighborhood of this point. You should compute image gradients in the same way that you did for edge detection, smoothing the image first. I suggest that you reuse the routines *image_gradient* and *smooth_image* from the edge detection problem set. Use a 5x5 neighborhood around each point to form *C.* Then use singular value decomposition, with the matlab routine *svd*, to find the singular values of *C.* (Or you can use the matlab function *eig* to find the eigenvalues of *C.* In this case it will amount to the same thing). This is summarized in the equations:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \overset{\text{Using SVD}}{=} R_2 \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R_1$$

The measure of cornerness for a point will be the smallest of the two singular values, λ_2. Since you have a measure of cornerness for every pixel in the image, you should output a matrix that is the same size as the input image, and that contains the cornerness for each pixel. So your complete program will have the form: *M = image_cornerness(I,sigma).* Here *I* is the input image, and *sigma* is a value indicating how much to smooth the image before computing the image gradient. I have given you a function *display_best_corners*, which will take as input an image, and the output of your cornerness routine, and will output an image that shows where the corners are. When this routine finds a good corner, it eliminates from consideration nearby points that are slightly less good, so it doesn't exactly find the locations in C that have the highest values.

We can use this, on the swan image for example, with the calls:

```
C = image_cornerness(swanbw, 2);
imshow(display_best_corners(swanbw, C, 20));
```

This will display the 20 best corners in the swan image. A link to a picture of the result is in swancorners.jpg.

Let's step through a simpler example of how this how program will work. Suppose we have an image with a corner:

```
I =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0   10   10   10   10
    0    0    0   10   10   10   10
    0    0    0   10   10   10   10
    0    0    0   10   10   10   10
```

To keep things simpler, I'll do no smoothing (so I call my routine with sigma = 0). Computing the x and y components of the image gradient, I get:

```
Ix =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    5    5    0    0    0
    0    0    5    5    0    0    0
    0    0    5    5    0    0    0
    0    0    5    5    0    0    0

Iy =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    5    5    5    5
    0    0    0    5    5    5    5
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
```

Using these to build a C matrix for the point (4,4), we get the matrix:

```
C =
   150    25
    25   150
```

Why these numbers? Below, I show all the gradients, as pairs (I_x,I_y). In red are the gradients in a 5x5 neighborhood around the pixel at (4,4). If we square all the I_x values we get 150, since there are six such values. Similarly for squaring I_y. Only pixel (4,4) gives a non-zero value, 25, for I_x*I_y though.

0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,5	0,5	0,5	0,5
0,0	0,0	5,0	5,5	0,5	0,5	0,5
0,0	0,0	5,0	5,0	0,0	0,0	0,0
0,0	0,0	5,0	5,0	0,0	0,0	0,0
0,0	0,0	5,0	5,0	0,0	0,0	0,0

We will have to do this for every point. Performing svd on the matrix C, we get:

svd(C)

ans =
175.0000
125.0000

So the cornerness of this point is 125, which is pretty high. Looking at the cornerness of all the points, we get:

0	0	0	0	0	0	0
0	25.0000	39.6447	44.0983	45.9431	21.4986	0
0	39.6447	75.0000	89.6447	94.0983	45.9431	0
0	44.0983	89.6447	125.0000	139.6447	70.1854	0
0	45.9431	94.0983	139.6447	175.0000	94.0983	0
0	21.4986	45.9431	70.1854	94.0983	75.0000	0
0	0	0	0	0	0	0

Notice that the point with the highest cornerness is located at (5,5), even though intuitively this does not seem to be exactly where the corner is. This is because a 5x5 region centered at (5,5) has the greatest diversity of gradients, since it has all the non-zero gradients we got in the neighborhood around (4,4), plus some new ones.

Hand in your results for the swan image, along with your code.

4. **Challenge Problem 10 points:** What happens to the corners if part of an image is in a shadow? To simulate this, take an image and reduce the intensities in one part of the image by a factor of ½. How can this influence which corners in the image are the most salient (ie, have the highest cornerness)? Can you think of a

way to change the corner detector so that its decision about which are the strongest corners will not be affected by shadows? Implement your strategy, and show what it does.