

Problem Set 2
CMSC 426
Due February 21, 2017

Written Exercises (10 points each)

1. Consider the image below:

3	4	5	6
4	5	6	7
5	6	7	8
6	7	8	9

- a. What is the gradient at the pixel in bold face (with a value of 6)?
 - b. What is the magnitude of the gradient?
 - c. What is the direction of the gradient?
2. Suppose the intensities in an image can be described by the equation: $I(x,y) = (x-50)^2 + (y-50)^2$. Answer the following questions analytically, by taking derivatives of this equation.
- a. What is the gradient at the position (40,45)?
 - b. What is the direction of the gradient there?
 - c. What is the magnitude of the gradient there?
3. Suppose there is an image, I , such that the gradient at (x,y) is always $(x,0)$.
- a. Give an example of a 3x3 image that would fit this description.
 - b. Suppose $I(3,2) = 7$. What is the intensity at $I(5,4)$?

Programming Exercises

The goal of this assignment is to write a simple edge detector based on the Canny edge detector. Your program will identify pixels in which the magnitude of the gradient is a local maxima when compared to two neighboring points in the direction of the gradient. To compute this you must smooth the image prior to finding the gradient. Then you must compute the gradient magnitude and direction. Using the gradient direction, for each pixel, (x,y) , you will compute two points, one in the direction in which the image intensity is increasing most rapidly, the other in the opposite direction, where the intensity is decreasing most rapidly. You will compare the gradient magnitude at (x,y) to the magnitude at these two points. (x,y) is an edge only if its gradient is bigger than at these two points.

Keep in mind that whenever you do image filtering, you should use the option 'replicate' for handling boundary pixels. Note that to use the test functions we describe below, you will have to name your functions the same as ours, or modify our test functions. For example, to use the test function *test_smooth_image* you will have to name your function *smooth_image* in the first problem.

Hint: It will be a good idea to turn your image into a matrix of floating point numbers using the *double* function. If not, you will get into trouble, because images are constrained to be non-negative integers below 255. If, for example, the gradient is sometimes negative, it cannot be stored in an image.

1. **Image Smoothing (15 points)** Write a function that will smooth an image with a Gaussian filter. You've already done this for Problem Set 1, but write a new version using built-in Matlab functions, to get familiar with them. Your function should take as input an image and a value for sigma that will be used to smooth the image. You should use the function *fspecial* to create the Gaussian filter, and the function *imfilter* to perform correlation with it. You should be sure that the width of your filter is at least six sigma, to fully capture the Gaussian. For example, if sigma is 1, you might use a filter that has a length of 7 (the first odd number after $6*1$). For sigma = 2, you might use a filter of length 13.

You can test your function using the test function that we provide, *test_smooth_image*. When we execute this test function, we get the following result:

```
>> test_smooth_image
```

```
ans =
```

```
    0.0050    0.0661    0.3397    0.7858    0.9983    0.7858    0.3397    0.0661
0.0050
    0.0661    0.8708    4.4776   10.3582   13.1602   10.3582    4.4776    0.8708
0.0661
    0.3397    4.4776   23.0227   53.2587   67.6658   53.2587   23.0227    4.4776
0.3397
    0.7858   10.3582   53.2587  123.2040  156.5322  123.2040   53.2587   10.3582
0.7858
    0.9983   13.1602   67.6658  156.5322  198.8760  156.5322   67.6658   13.1602
0.9983
    0.7858   10.3582   53.2587  123.2040  156.5322  123.2040   53.2587   10.3582
0.7858
    0.3397    4.4776   23.0227   53.2587   67.6658   53.2587   23.0227    4.4776
0.3397
    0.0661    0.8708    4.4776   10.3582   13.1602   10.3582    4.4776    0.8708
0.0661
```

```
0.0050 0.0661 0.3397 0.7858 0.9983 0.7858 0.3397 0.0661
0.0050
```

Your results might be slightly different, if you produce a filter of a slightly different length than we do. But they should be quite similar. Hand in your code and the result of this test.

2. **(20 points)** Now write a function that will compute the gradient of an image. Your function should take an image as input and return two matrices. Each matrix will be the size of the original image. One will contain the x component of the image gradient for each pixel, the other will contain the y component of the gradient.

You can test your function using the test function that we provide, `test_image_gradient`. When we execute this test function we get the following result:

```
>> [Dx Dy] = test_image_gradient
Dx =
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
-3.5000 -6.0000 -4.0000 -2.0000 0 2.0000 4.0000 6.0000
3.5000
Dy =
-3.5000 -3.5000 -3.5000 -3.5000 -3.5000 -3.5000 -3.5000 -3.5000 -
3.5000
-6.0000 -6.0000 -6.0000 -6.0000 -6.0000 -6.0000 -6.0000 -6.0000 -
6.0000
-4.0000 -4.0000 -4.0000 -4.0000 -4.0000 -4.0000 -4.0000 -4.0000 -
4.0000
-2.0000 -2.0000 -2.0000 -2.0000 -2.0000 -2.0000 -2.0000 -2.0000 -
2.0000
```

```

      0      0      0      0      0      0      0      0      0
    2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000
2.0000
    4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000
    6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000
6.0000
    3.5000  3.5000  3.5000  3.5000  3.5000  3.5000  3.5000  3.5000
3.5000

```

Hand in your code and the result of this test.

3. **(15 points)** Now write a function to compute the magnitude and direction of the gradient. This function should take as input the two matrices produced by the previous function, which contain the x and y components of the gradient. It should return three matrices, which contain the magnitude of the gradient at each pixel, and the x and y components of the direction of the gradient. That is, the second and third results will contain the cosine of the direction of the gradient and the sine of the direction of the gradient.

You can test your function using the test function that we provide, `test_gradient_magnitude_direction`. When we execute this test function we get the following result:

```
[R, X, Y] = test_gradient_magnitude_direction
```

```
R =
```

```

    1.4142    2.2361    3.1623    4.1231
    2.2361    2.8284    3.6056    4.4721
    3.1623    3.6056    4.2426    5.0000
    4.1231    4.4721    5.0000    5.6569

```

```
X =
```

```

    0.7071    0.8944    0.9487    0.9701
    0.4472    0.7071    0.8321    0.8944
    0.3162    0.5547    0.7071    0.8000
    0.2425    0.4472    0.6000    0.7071

```

```
Y =
```

```

    0.7071    0.4472    0.3162    0.2425
    0.8944    0.7071    0.5547    0.4472
    0.9487    0.8321    0.7071    0.6000
    0.9701    0.8944    0.8000    0.7071

```

Notice that with these values, we can find in Matlab:

```
X.^2+Y.^2
```

```
ans =
```

```

    1.0000    1.0000    1.0000    1.0000

```

```
1.0000 1.0000 1.0000 1.0000
1.0000 1.0000 1.0000 1.0000
1.0000 1.0000 1.0000 1.0000
```

This is because X and Y contain the cosine and sine of the direction of the gradient, and cosine squared plus sine squared equals one. Hand in your code and the result of this test.

4. **(20 points)** Now combine these functions together to create a function that will find positions in the image in which the gradient is a local maxima, and reasonably large. This function will take three inputs: an image, sigma, and t . sigma is the parameter used in *smooth_image* to determine how much to smooth the image. t is a threshold. We will only detect edges at pixels in which the magnitude of the gradient is greater than this threshold. The output of this function will be a binary matrix. For every pixel, there will be a 1 if there is an edge at that pixel, and a zero if there is no edge.

A pixel will be an edge if it satisfies two criteria. First, it must be a local maxima in gradient magnitude. That means that its gradient magnitude must be bigger than that at two neighboring locations. These must be locations that are a distance of one pixel away, and that are in the directions at which the gradient is changing most rapidly. For example, suppose pixel (10,10) has a gradient direction that is 45 degrees from the x axis. We would describe this gradient direction as (.7071, .7071). Then we can only have an edge at (10,10) if the magnitude of the gradient there is bigger than the magnitude of the gradient at (10.7071,10.7071), and also greater than the gradient magnitude at (9.2929, 9.2929). Note that these locations do not have integer coordinates, so we must interpolate the gradient magnitude to estimate its value at these locations. Since we haven't discussed how to do this, we will provide you with a Matlab function that performs this interpolation. This function is called: *interpolate_gradients*. Look at the comments in this function to see how to use it.

Test your function using the image of the swan in swanbw.jpg. We have provided a second image, swanedges.jpg, which shows the output of our code when we run on this image, using a value of 2 for sigma, and a value of 15 for t . Turn in your code, and an image that shows the edges that your code produces when you run with these same values. You may find it useful to use the Matlab function *imwrite*.

5. **Challenge Problem (20 points):** Now add hysteresis to your edge detector. It should take two thresholds as input, along with the image and sigma. A pixel is an edge if the gradient magnitude is greater than the first threshold, but also if the gradient magnitude is bigger than the second threshold and it has a neighbor that is an edge (note that this is a recursive definition). The image swanedges_h.jpg shows the results of running this function with the swan image, with sigma = 2, and with thresholds of 15 and 2. The results are not too different from those in

swanedges.jpg, but a little better in the beak and head of the swan and the bottom of its body.