INFORMATION TECHNOLOGY UNIVERSITY

COMPUTER VISION ASSIGNMENT 2

Implementation of Canny's Edge Detector

Edge detection is very important to find the sudden changes in an image. In this assignment we are going to learn about the implementation of a very important edge detector called Canny's edge detector.

Guidelines

- Submit all of your code and results in a single zip file with name FirstName_RollNumber_02.zip
- Submit single zip file containing
  a) code b) results c) readme d) report
  There should be Readme.txt explaining how to run your code
  There should be Report.pdf or Report.doc detailing your experience and highlighting any interesting result. Also past you code at the end of your report as the originality of the reports and code will be checked.
- Submit the hard copy of mathematical part.
- **Please note that no late submissions will be accepted. Even a second late.**
- Email instructor or TA if there are any questions. You cannot look at others code or use others code, however you can discuss with each other.
- This document has two parts, first **Implementation** and second is **mathematical** part. Implementation part has two portions, first **Steps for implementation** and second **Problems.**
- Follow the given link for implementation. This has all the information needed to solve the assignment. Steps for implementation in the assignment are also from the given link. [1]
  http://suraj.lums.edu.pk/~cs436a02/CannyImplementation.htm
- Before solving the problem section, first go through the steps which will help you implement Canny's edge detector step by step in Python.

- **Deadline for programming part is Sunday 6th October 2019 before 11:59pm.**
- **Mathematical part should be submitted in Thursday's class on 3rd October 2019. Please submit hard copy of it.**

Implementation

There are some basics steps for the implementation of canny edge detector. They are all described in detail in the given link. [1]

1. Generation of Masks
2. Applying Masks to Images
3. Non Maxima Suppression
4. Hysteresis Thresholding

Steps for implementation:

1. **Generation of Masks:** This module requires the value of **sigma** as an input and generates x- and y-derivative masks as output.
   - Mask size: To generate the masks, the first step is a reasonable computation of the mask size. Mask size depends on the value of sigma and T. Formula of size of half mask is as follows:

     sHalf = round(sqrt(-log(T) * 2 * sigma^2))

   - Where sigma and T are input parameters.
     i. Sigma can be **0.5, 1, 1.5**… Lower limit of sigma is 0.5.
     ii. Width of the mask is based on parameter T. T can be any value **between 0 and 1**. For example **T=0.3.** The above formula will give us size of half mask.
   - The total mask size would then be computed as follows:

     N= 2*sHalf +1 (total mask size)

   Example: sigma=0.5; T=0.3

   sHalf = round(sqrt(-log(T) * 2 * sigma^2)) => 1

   N= 2*sHalf +1 => 3

   [Y, X]=meshgrid(-sHalf:sHalf) => Y=[-1,0,1;-1,0,1;-1,0,1]

   Note: meshgrid is an interesting function. It can be found in Python's numpy library.

   - In Canny's method, the masks used are **1st derivative of a Gaussian** in x- and y-directions. Gaussian formula is as follows:

$$G(x,y)=ce^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- Now take the first derivative of Gaussian w.r.t 'x' and call it 'fx' and put the valve of X,Y in it. Repeat this procedure w.r.t 'y' as well. These (Gx, Gy) are the two masks that will be convolved with image in next step.
- Multiply the Gx and Gy with 255 to scale up and then round off these values so that convolution is performed with integer values rather than floats.

Note: Save the scale factor, because gradient magnitude will be later scaled down (after convolution) by the same factor.

2. **Applying Masks to Images:** The masks are applied to the images using convolution. Store convolution results in fx and fy.

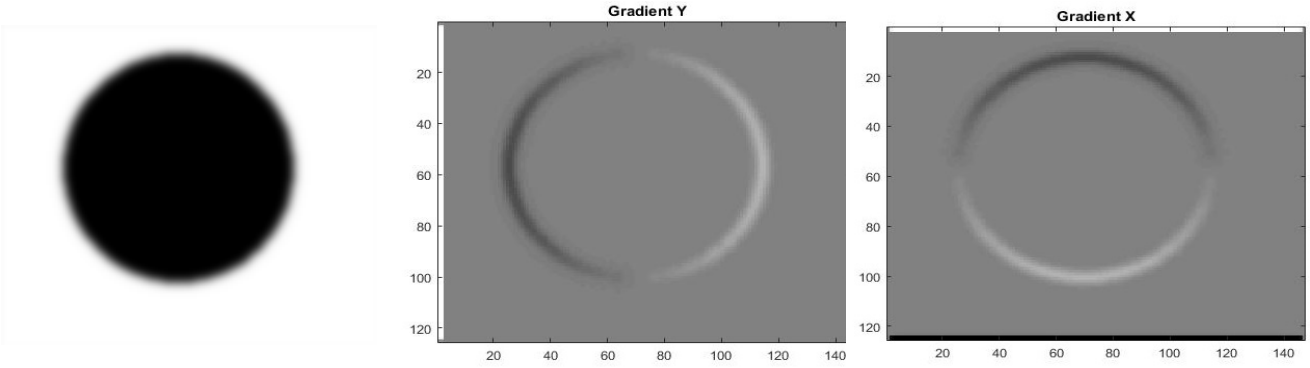

Figure 1 : Image (a) is an input image 'circleBlured.png'. Image (b) is generated after convolving the input image with Gy (Gaussian derivative w.r.t 'y'). Image (c) is generated after convolving the input image with Gx (Gaussian derivative w.r.t 'x').

3. **Compute gradient magnitude**: Compute the gradient magnitude of images (fx,fy) at each pixel after convolving with masks (Gx,Gy). M is computed from fx and fy images, using the magnitude formula:

$$M=\sqrt{f_x^2 + f_y^2}$$

Note: The result is then scaled down by the same factor which was used to scale up the masks. To write output to image files, the min and max values are scaled to 0 and 255 respectively.

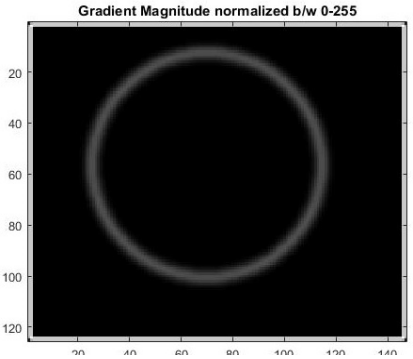

Figure 2: Gradient magnitude of input image normalized b/w 0-255

4. **Compute gradient Direction**: Compute the gradient direction of images (fx,fy) at each pixel after convolving with masks (Gx,Gy). Phi is computed using atan2 function by following formula:

$$\theta = \arctan \frac{f_y}{f_x}$$

Note: Convert the angle returned by atan2 function to degrees and add 180 to get an output range of 0-360 degrees.

5. **Non Maxima Suppression:** Non maxima suppression step makes all edges in M one pixel thick.

- The first step is to quantize gradient direction into just four directions.

Value assigned	Angle
0	0 to 22.5 157.5 to 202.5 337.5 to 360
1	22.5 to 67.5 202.5 to 247.5
2	67.5 to 112.5 247.5 to 292.5
3	112.5 to 157.5 292.5 to 337.5

Table 1 Quantize gradient directions

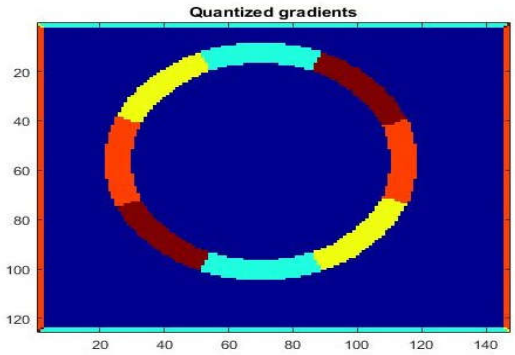

Figure 3 Image after quantizing gradient directions

- The next step is to pick two neighbors of each edge point **along the gradient direction**. This is because gradient direction is perpendicular to the edge, and therefore, this is the direction in which we are searching for edge points.
- Therefore the two neighbors that need to be picked for comparison are the north and south neighbors in the direction of gradient. If the edge point (r,c) is greater than both these neighbors, then it is maintained in M otherwise it is made zero.

6. **Hysteresis Thresholding:** The final step in Canny's edge detection algorithm is to apply two thresholds to follow edges.
   - First made the border pixels zero, so that finding neighbors does not go out of bounds of the image.
   - Next, the image is scanned from left to right, top to bottom. The first pixel in non-maxima suppressed magnitude image which is above a certain threshold, Th, is declared an edge.
   - Then all its neighbors are recursively followed, and those above threshold, Tl, are marked as an edge.
   - Thus there are really two stopping conditions:
     i. if a neighbor is below Tl, we won't recurs on it.
     ii. if a neighbor has already been visited, then we won't recurs on it.

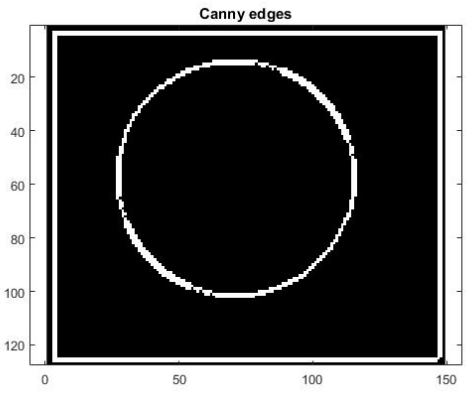

Figure 5 Image after sigma=1 and Th=100,Tl=20

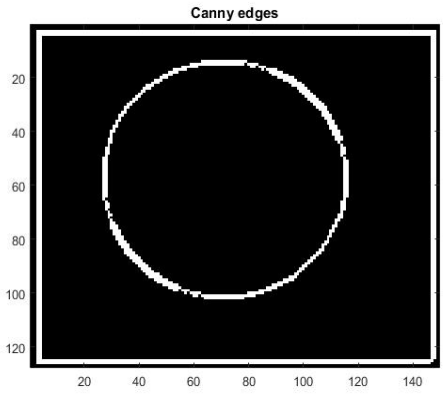

Figure 4 Image after sigma=1 and Th=50,Tl=10

Problem:

For Assignment 2, you have to implement Canny Edge Detector. Test Images are given in the assignment folder. **Create separate functions for all 4 steps.**

For each image you have to submit

a) images showing gradients in X and Y direction **(10)**
b) image showing Magnitude of gradients at each pixel **(10)**
c) Image of quantized orientations **(15)**
d) Final Result for sigma = 1, sigma = 0.5 and sigma = 2 **(20)**
e) For sigma =1, submit 2 different results for 2 different pairs of Th, Tl in Hysteresis thresholds. **(25)**

Mathematical Part

Problem 1: Calculate and plot (using plot3 or mesh) derivative and double derivative of Gaussian filter. **(10)**

Problem 2: Prove that Convolutions are commutative a*b=b*a (* => convolution) **(5)**

Problem 3: Calculate the rank of given matrixes. **(5)**

A= [1 2 3; 3 6 9] **B= [1 2 3; 4 5 6; 7 8 9]**

References:

[1] http://suraj.lums.edu.pk/~cs436a02/CannyImplementation.htm