**Image Understanding – Assignment #1**

**Edge Detection**

### Introduction

#### Organization of the code

The main function is EdgeDetectionCanny.m. The information of this function is shown in Table 1.

Table 1. Description of EdgeDetectionCanny

|  |  |  |  |
| --- | --- | --- | --- |
| Name | EdgeDetectionCanny | | |
| Description | Detect the edge of the given grayscale image by using Canny`s edge detection algorithm. | | |
|  | Name | Type | Description |
| Input | path | string | the path and name of the given image |
| sigma | double | the standard deviation for the gaussian filter |
| threshold | vector | The input magnitude threshold vector. If only one threshold is input, a single-threshold method will be called. If the input parameter is a vector, the first two elements will be regarded as a high limit and a low limit and a hysteresis method is called. |
| Output | J | matrix | the image after edge detection |

Also, there are several helper functions which called by the main function. Below is the description of all those functions.

Table 2. Description of GaussianSmooth

|  |  |  |  |
| --- | --- | --- | --- |
| Name | GaussianSmooth | | |
| Description | Generate one horizontal and one vertical gaussian filter by the given standard deviation. Implement the 2D gaussian filter as a sequence of a 1D horizontal and vertical gaussian convolution. | | |
| Input | Name | Type | Description |
| I | matrix | the image needed to be smoothed |
| sigma | double | the standard deviation for the gaussian filter |
| Output | G | matrix | the image after smoothing |

Table 3. Description of Convolution

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Convolution | | |
| Description | Calculate the convolution matrix of two input matrices | | |
| Input | Name | Type | Description |
|  | I | matrix | first input matrix |
|  | J | matrix | second input matrix |
| Output | H | matrix | convolution of matrix I and matrix J |

Table 4. Description of GradientCalculation

|  |  |  |  |
| --- | --- | --- | --- |
| Name | GradientCalculation | | |
| Description | Calculate the Gradient of each element of the input matrix, including the magnitude and direction | | |
|  | Name | Type | Description |
| Input | I | matrix | the matrix needed to calculate the gradient |
| Output | M\_pdx | matrix | the matrix saving the x component of gradient direction |
|  | M\_pdy | matrix | the matrix saving the x component of gradient direction |
|  | M\_magnitude | matrix | the matrix saving the magnitude of gradient |

Table 5. Description of RidgeDetection

|  |  |  |  |
| --- | --- | --- | --- |
| Name | RidgeDetection | | |
| Description | Calculate local maximum of the gradient magnitude matrix | | |
|  | Name | Type | Description |
| Input | M\_pdx | matrix | the matrix saving the x component of gradient direction |
|  | M\_pdy | matrix | the matrix saving the x component of gradient direction |
|  | M\_magnitude | matrix | the matrix saving the magnitude of gradient |
| Output | M | matrix | the matrix saving the “ridge” of the gradient |

Table 6. Description of EdegDetect\_1threshold

|  |  |  |  |
| --- | --- | --- | --- |
| Name | EdegDetect\_1threshold | | |
| Description | Detect the edge of input image by thresholding the matrix of magnitude | | |
|  | Name | Type | Description |
| Input | I | matrix | the image needed to detect edges |
|  | M\_magnitude | matrix | the matrix saving the “ridge” of the gradient |
|  | threshold | double | given threshold to locate edges |
| Output | J | matrix | the image after detecting the edges |

Table 7. Description of EdegDetect\_2thresholds

|  |  |  |  |
| --- | --- | --- | --- |
| Name | EdegDetect\_2thresholds | | |
| Description | Detect the edge of input image by thresholding the matrix of magnitude, a DFS method is used. The high threshold and low threshold will be determined automatically. | | |
|  | Name | Type | Description |
| Input | I | matrix | the image needed to detect edges |
|  | M\_magnitude | matrix | the matrix saving the “ridge” of the gradient |
|  | threshold | double | first threshold |
|  | threshold2 | double | double threshold |
| Output | J | matrix | the image after detecting the edges |

Also, in order to implement a DFS method, stack structure is used. However, there isn`t an integrated stack in Matlab, so a self-defined class of stack structure is created. This class has the same member functions with the structure stack in C++, such as push() and pop().

#### Convolution method

In order to implement the 2D convolution and handle the boundary conditions, several steps are used.

Step 1: expand the matrix by padding zeros.

First, calculate the size of the first matrix and save in variable “*rowNum\_I*” and “*colNum\_I*”, and then calculate the size of the second input matrix and saved in variable “*rowNum\_J*” and “*colNum\_J*”. The expanded matrix size will be [*rowNum\_I*+2\*(*rowNum\_J*-1)] X [*colNum\_I* + 2\*(*colNum\_J*-1)].

Second, add (*rowNum\_J*-1) rows of zeros above and below the first matrix.

Third, add (*colNum\_J*-1) columns of zeros on the left and right side of the first matrix.

Step 2: Rotate the second input matrix by 180 deg.

According to the definition of convolution, the subscripts of the second matrix are both negative. In order to simplify the calculation, rotate the second matrix by 180 deg. Then the two subscripts are both positive.

Step 3: Calculate convolution

First, align the center of the rotated second input matrix to each element of the first matrix in the expanded matrix.

Second, calculate the sum of the product of each corresponding element in first matrix and second matrix.

Third, loop all the elements in the first matrix and save the result in a new matrix.

#### Hysteresis implements

In order to implement the hysteresis method, a depth first search algorithm is used. A stack is defined to store the strong edges, which means the pixels above the high threshold. Then each time a pixel coordinate is popped from the stack, all its eight neighbors will be checked if the intensity is between the high threshold and low threshold. If a pixel is found to be a part of “weak” edge (intensity between the high threshold and low threshold) which has a neighbor of strong edges. The pixel coordinate will be immediately pushed into the stack and the intensity will be modified to a value bigger than high threshold to avoid repeatedly check and push of a single pixel.

After all the strong edge pixels are checked. The corresponding pixel of the image will be defined as edge and the intensity will be changed to 255, and the intensity of other pixels will be changed to 0.

#### Discussion of the results

### 2.Result of Edge Detection

#### Result of the gaussian smooth

Test code:

I = zeros(101,101);

I(51,51) = 255;

imshow(I);

G = GaussianSmooth(I,5);

imshow(G);

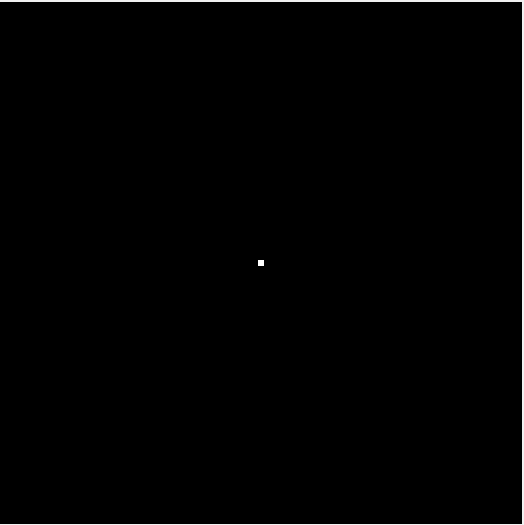
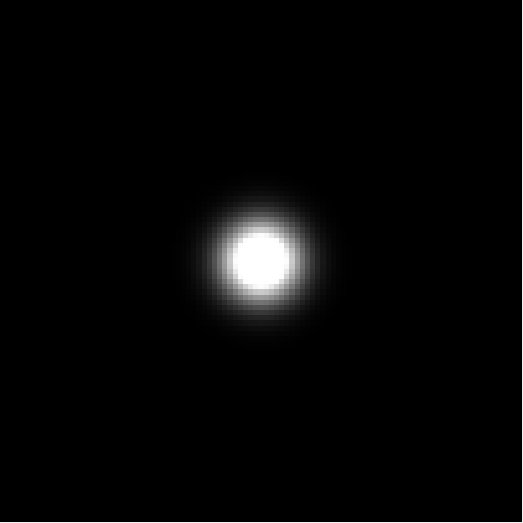


Fig 1.1 A black image(101X101) with one while pixel in the center Fig 1.2 The image smoothed by gaussian filter(σ=5)

#### Result of the one threshold Canny`s method

1. Elephants.pgm

Test code:

EdgeDetectionCanny('elephants.pgm',0.5,20);

EdgeDetectionCanny('elephants.pgm',1,20);

EdgeDetectionCanny('elephants.pgm',2,20);

EdgeDetectionCanny('elephants.pgm',1,30);

EdgeDetectionCanny('elephants.pgm',1,40);

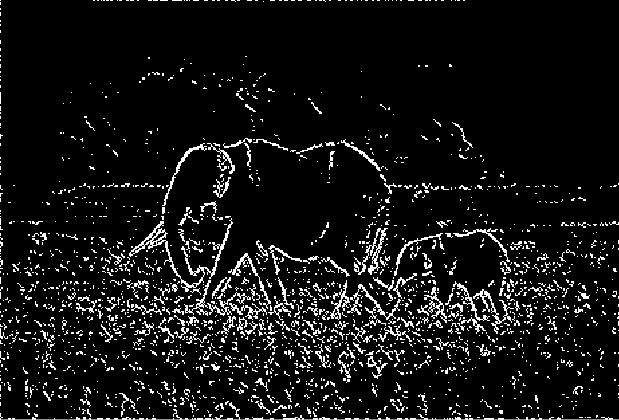
Fig 1.3 Origin image of “elephants.pgm” Fig.1.4 Edge detection result（σ=0.5, threshold =20）

Fig 1.5 Edge detection result of（σ=1, threshold =20） Fig 1.6 Edge detection result（σ=2, threshold =20）

Fig 1.7 Edge detection result of（σ=1, threshold =30） Fig 1.8 Edge detection result of（σ=1, threshold =40）

1. bear.pgm

Test code:

EdgeDetectionCanny('bear.pgm',0.5,15);

EdgeDetectionCanny('bear.pgm',1,15);

EdgeDetectionCanny('bear.pgm',2,15);

EdgeDetectionCanny('bear.pgm',1,20);

EdgeDetectionCanny('bear.pgm',1,25);

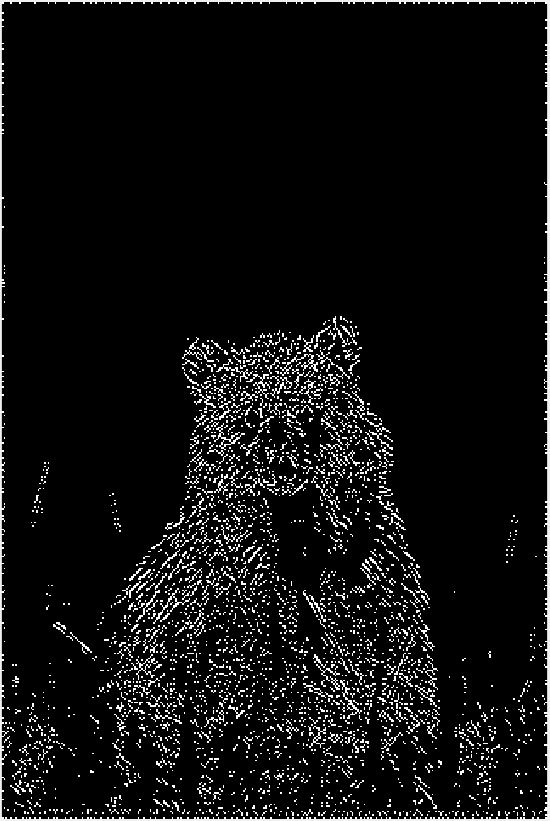
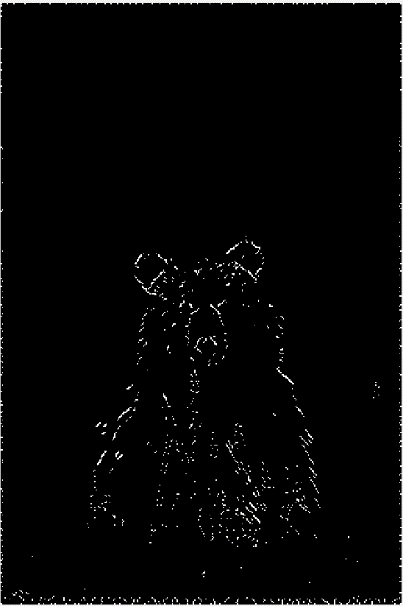
Fig 1.9 Origin image of “bear.pgm” Fig.1.10 Result(σ=0.5, threshold =15) Fig.1.11 Result(σ=1, threshold =15)

Fig 1.12 Result(σ=2, threshold =15) Fig.1.13 Result(σ=1, threshold =20) Fig.1.14 Result(σ=1, threshold =25)

1. boat.pgm

Test code:

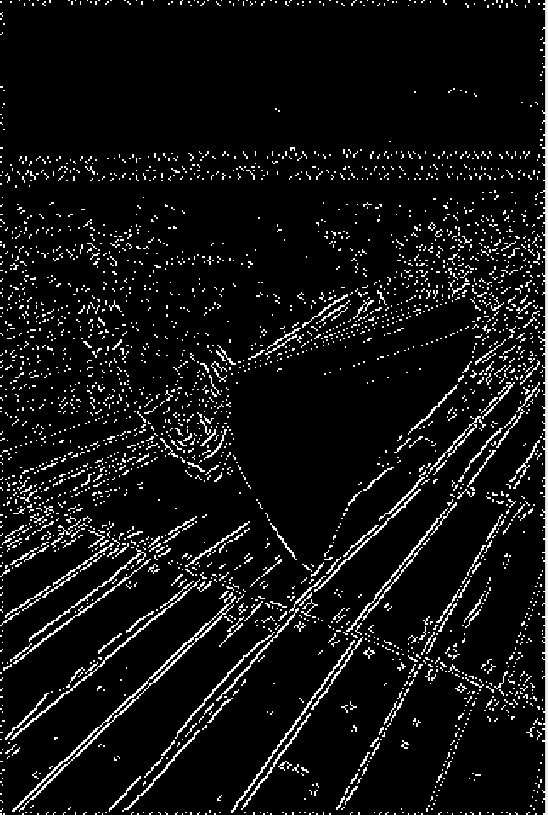
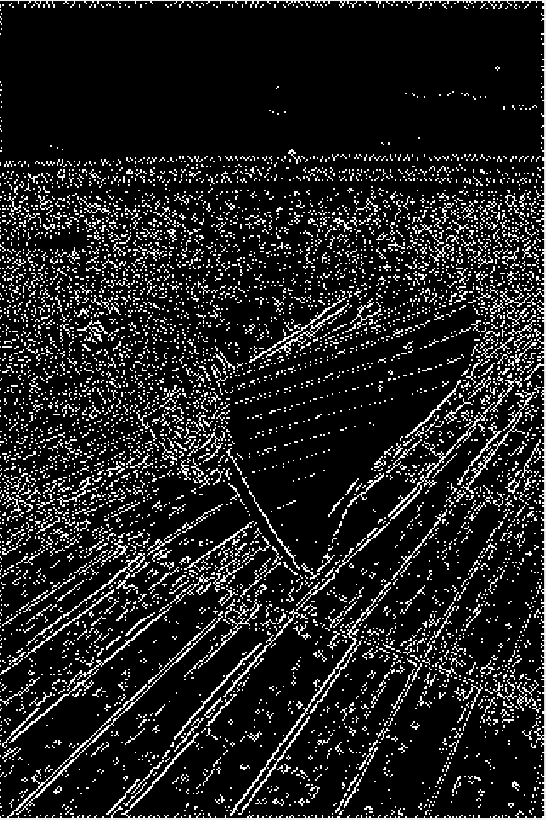
EdgeDetectionCanny(‘boat.pgm',0.5,15);

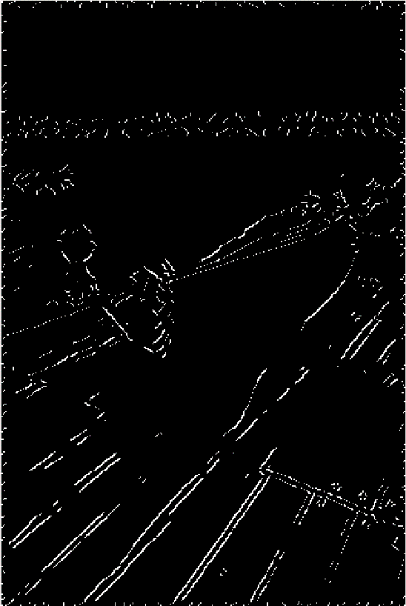
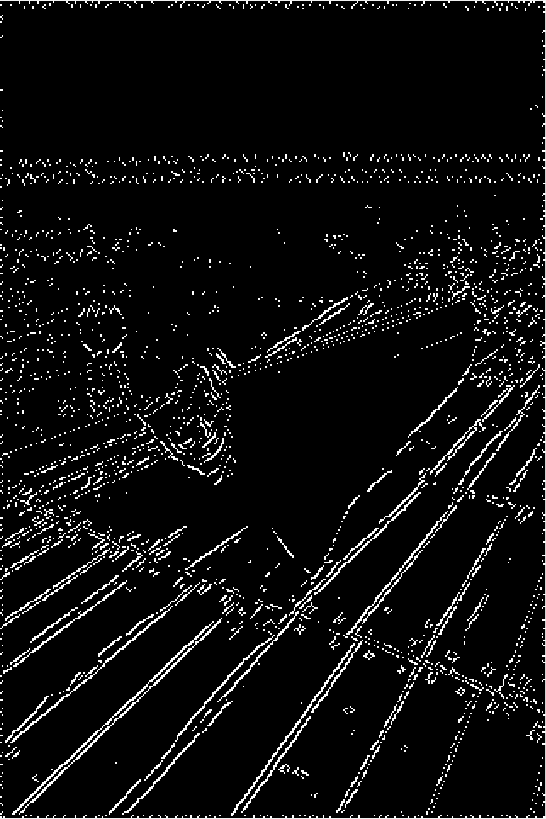
EdgeDetectionCanny(‘boat.pgm',1,15);

EdgeDetectionCanny(‘boat.pgm',2,15);

EdgeDetectionCanny(‘boat.pgm',1,20);

EdgeDetectionCanny(‘boat.pgm',1,25);

Fig 1.15 Origin image of “boat.pgm” Fig.1.16 Result(σ=0.5, threshold =15) Fig.1.17 Result(σ=1, threshold =15)

Fig 1.18 Result(σ=2, threshold =15) Fig.1.19 Result(σ=1, threshold =20) Fig.1.20 Result(σ=1, threshold =25)

1. tsukuba.pgm

Test code:

EdgeDetectionCanny(‘tsukuba.pgm’,0.5,15);

EdgeDetectionCanny(‘tsukuba.pgm',1,15);

EdgeDetectionCanny(‘tsukuba.pgm',2,15);

EdgeDetectionCanny(‘tsukuba.pgm',1,20);

EdgeDetectionCanny(‘tsukuba.pgm',1,25);

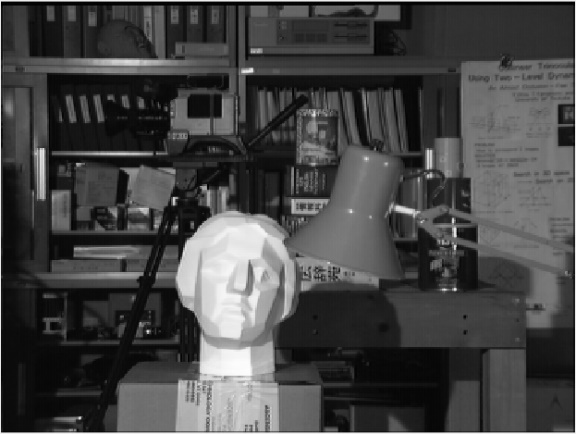


Fig 1.21 Origin image of “tsukuba.pgm” Fig 1.22 Result(σ=0.5, threshold =15)

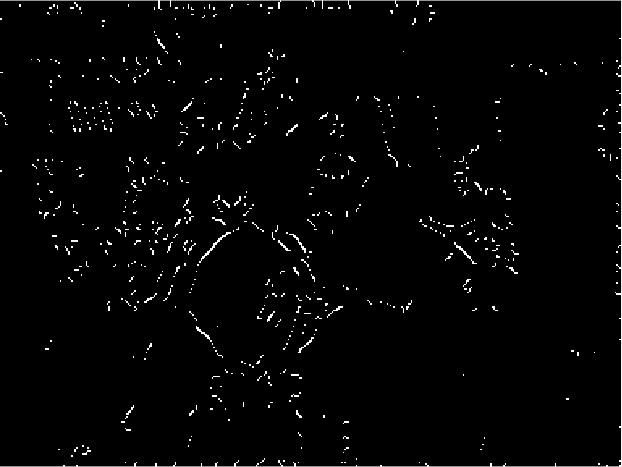


Fig.1.23 Result(σ=1, threshold =15) Fig 1.24 Result(σ=2, threshold =15)



Fig.1.25 Result(σ=1, threshold =20) Fig 1.24 Result(σ=1, threshold =25)

#### Result of the hysteresis method

EdgeDetectionCanny('elephants.pgm',1,[10,30])



EdgeDetectionCanny('elephants.pgm',0.5,[10,40])



EdgeDetectionCanny('elephants.pgm',1,[10,40])



EdgeDetectionCanny('elephants.pgm',1,[10,20])

EdgeDetectionCanny('tsukuba.pgm',1,[10 25])

