1. Graded Exercise: Canny Edge Detector

You are allowed 1 hour 45 minutes for this graded exercise. You have to submit your files by 10:00 AM. You may use your notes, books and code from previous exercise sessions, but access to Internet resources is forbidden. Once you are done, make sure that your files are successfully uploaded to the system.

1.1 Introduction

Canny edge detector was developed in 1980s and since then it is one of the most popular edge detection techniques. It satisfies three conditions for a good edge detector, which are:

- Low error rate: meaning a good detection of only existent edges.
- Good localization: the distance between edge pixels detected and real edge pixels has to be minimized.
- Minimal response: only one detector response per edge.

In this exercise session we will implement and test Canny edge detector from scratch, by following five steps described below. These are:

1. Preprocessing - filtering out the noise
2. Computing gradient - magnitude and direction
3. Non-maxima supression
4. Double thresholding
5. Hysterisis thresholding

Write your code either in a file called `main.m` or in a separate function, as requested in the exercise. You can put qualitative answers as comments in `main.m` file. Use the image **pears.png** given inside MATLAB toolbox as input. Convert the image to grayscale and double precision before applying the edge detector.

1.2 Smoothing (10 pts)

Noise present in the image may cause false detections of the edges. In order to prevent this behavior the image has to be smoothed prior to detection.

Exercise 1.1 In `main.m` smooth the image using Gaussian filter of size 7x7. Choose suitable σ. What is the trade-off of using bigger σ? ■

1.3 Computing gradients (20 pts)

Edges are considered to be points where image intensity changes sharply. This change can be quantified by measuring the gradient at each pixel location. Gradient is characterised by magnitude (edge strength) and direction and both of those properties were proved to be useful in Canny edge detector.

Exercise 1.2 In `main.m` compute S_x and S_y, gradient images in horizontal and vertical directions using suitable operators. Next, compute S_{mag} and S_{ang} - magnitude and direction of the gradient. *Hint:*. For gradient direction computation, you can use the function `atan2()`. ■

1.4 Non-maximum suppression (30 pts)

After applying gradient calculation, the edges are typically broad and blurry. In order to "thin" the edge responses and refine the localization, non-maxima suppresion is applied. This is done by preserving all local maxima in the gradient image and deleting everything else. For each pixel in the gradient image, the algorithm is as follows:

- Specify a number of discrete orientations for the gradient vector. For example, in a 3×3 region, we can define four orientations[1] for an edge passing through the center point of the region: horizontal (0°), vertical (90°), 45° and 135°. As Fig. 1.1 shows, if the edge normal is in the range of directions from $-22.5°$ to 22.5° or from $-157.5°$ to 157.5°, we call the edge a horizontal edge and quantize it to 0°. In Fig. 1.1, each shade of gray corresponds to a different quantization of the gradient direction.
- Compare the edge strength of the current pixel with the edge strength of the neighbor pixels in two opposite sides along the quantized direction. E.g., if the quantized gradient direction for the current pixel is 90°, then compare it with the pixels to the north and south.
- If the edge strength of the current pixel is greater than that of the neighboring pixels in the gradient direction, preserve the value of the edge strength. If not, suppress the value, i.e., set the value to zero.

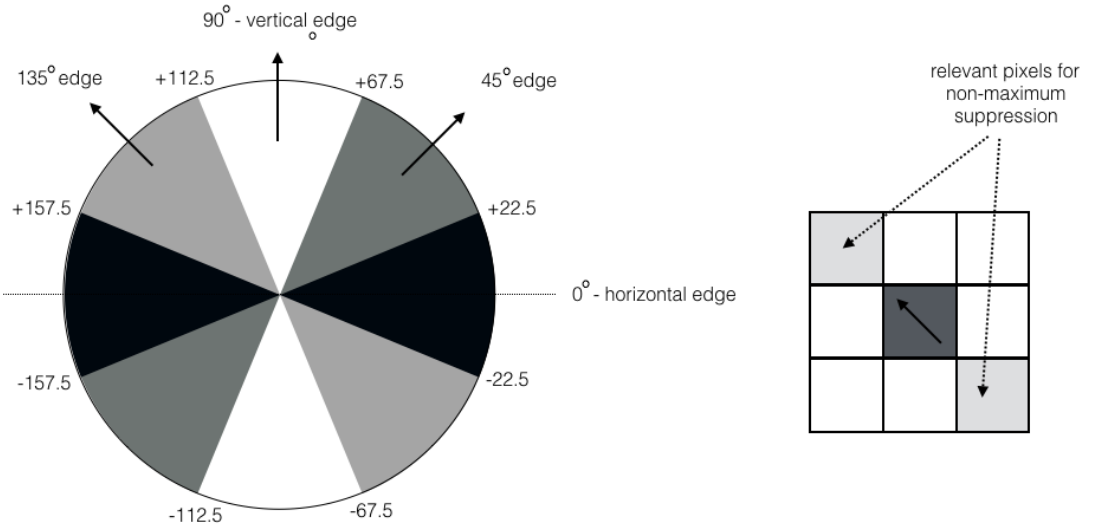

Figure 1.1: (Left) The angle ranges of the edge normals for the four types of edge directions in a 3×3 neighborhood. Each edge direction has two ranges, shown in corresponding shades of gray. (Right) The gradient direction for the current pixel is shown with the bold arrow. Only the neighboring pixels along the gradient direction are relevant, and their edge strengths are compared to that of the current pixel.

Exercise 1.3 Write a function called `nonmaximum_suppression` to implement the above algorithm. The input to the function should be the magnitude (S_{mag}) and direction (S_{ang}) of the gradient, the output should be the thinned image (S_{thin}). ■

[1] Every edge has two possible orientations. For example an edge whose normal is oriented at 0° and an edge oriented at 180° are the same horizontal edge.

1.5 Double threshold (15 pts)

The edge pixels remaining after the non-maximum suppression step above are still marked with their strength pixel-by-pixel. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to discern between these would be to use a threshold, so that only edges stronger than a certain value would be preserved. The Canny edge detection algorithm uses double thresholding for this step. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak.

Exercise 1.4 Implement a function called `double_threshold` to implement the above functionality. The input to the function should be the thinned image weighted with the gradient magnitude and two thresholds. The output of the function should be the set of strong and weak edges. ■

1.6 Hysteresis thresholding (25 pts)

Strong edges can immediately be included in the final edge image. However, weak edges are included if and only if they are connected to strong edges. The idea is of course that noise and other small variations are unlikely to result in a strong edge (with proper adjustment of the threshold levels). Thus strong edges will (almost) only be due to true edges in the original image. The weak edges can either be due to true edges or noise/color variations. Usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection blob analysis is applied. The edge pixels are divided into connected blobs using 8-connected neighborhood. Blobs containing at least one strong edge pixel are preserved, while blobs containing only weak edges are suppressed.

Figure 1.2: Left, middle: A weak edge (gray) with no surrounding strong edges is removed. Right: A weak edge connected to at least one strong edge (white) is kept.

Exercise 1.5 Implement a function called `hysteresis_thresholding` to implement blob analysis on weak and strong edges. The input to the function should be the set of strong and weak edges of the previous step. The output should be the final edge map of the original image.

■