CS517: Digital Image Processing & Analysis

Lab 4: Canny Edge Detector

Aim:

- To get hands-on experience implementing a very popular segmentation algorithm

Introduction

The purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing. Several algorithms exists, and this lab focuses on a particular one developed by John F. Canny (JFC) in 1986 [2]. Even though it is quite old, it has become one of the standard edge detection methods and it is still used in research [3] [1]. The aim of JFC was to develop an algorithm that is optimal with regards to the following criteria:

- **Detection:** The probability of detecting real edge points should be maximized while the probability of falsely detecting non-edge points should be minimized. This corresponds to maximizing the signal-to-noise ratio.
- **Localization:** The detected edges should be as close as possible to the real edges.
- **Number of responses:** One real edge should not result in more than one detected edge (one can argue that this is implicitly included in the first requirement).

With JFC's mathematical formulation of these criteria, Canny's Edge Detector is optimal for a certain class of edges (known as step edges). A C++ implementation of the algorithm has been written, and this will be further described in Section 3. The images used throughout this worksheet are generated using this implementation.

Test Image

The image in Figure 1 is used throughout this Lab to demonstrate how Canny edge detection works. It depicts a partially assembled pump from Grundfos, and the edge detection is a step in the process of estimating the pose (position and orientation) of the pump.

The image has been preprocessed as follows:

- Determining ROI (Region of Interest) that includes only white background besides the pump, and cropping the image to this region.
- Conversion to gray-scale to limit the computational requirements.
- Histogram-stretching, so that the image uses the entire gray-scale. This step may not be necessary, but it is included to counter-compensate for automatic light adjustment in the used web camera.

Figure 1: *The image used as example of Canny edge detection.*

The Canny Edge Detection Algorithm (`Canny`)

The algorithm runs in 5 separate steps:

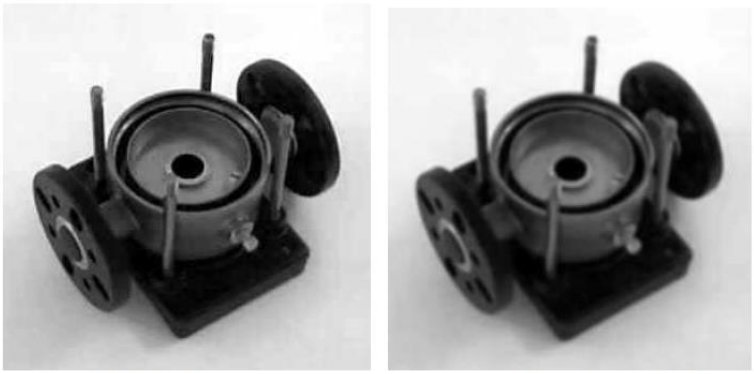
1. Smoothing: Blurring of the image to remove noise.
2. Finding gradients: The edges should be marked where the gradients of the image has large magnitudes.
3. Non-maximum suppression: Only local maxima should be marked as edges.
4. Double thresholding: Potential edges are determined by thresholding.
5. Edge tracking by hysteresis: Final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.

Each step is described in the following subsections.

Smoothing (`CEDSmooth`)

It is inevitable that all images taken from a camera will contain some amount of noise. To prevent that noise is mistaken for edges, noise must be reduced. Therefore the image is first smoothed by applying a Gaussian filter. The kernel of a Gaussian filter with a standard deviation of σ = 1.4 is shown in Equation (1). The effect of smoothing the test image with this filter is shown in Figure 2.

$$B = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{1}$$



(a) Original (b) Smoothed

Figure 2: The original grayscale image is smoothed with a Gaussian filter to suppress noise.

Finding gradients (`CEDGrads`)

The Canny algorithm basically finds edges where the grayscale intensity of the image changes the most. These areas are found by determining gradients of the image. Gradients at each pixel in the smoothed image are determined by applying what is known as the Sobel-operator. First step is to approximate the gradient in the x- and y-direction respectively by applying the kernels shown in Equation (2).

$$K_{\mathrm{GX}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad K_{\mathrm{GY}} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

The gradient magnitudes (also known as the edge strengths) can then be determined as an Euclidean distance measure by applying the law of Pythagoras as shown in Equation (3). It is sometimes simplified by applying Manhattan distance measure as shown in Equation (4) to reduce the computational complexity. The Euclidean distance measure has been applied to the test image. The computed edge strengths are compared to the smoothed image in Figure 3.

$$|G| = \sqrt{G_{\mathrm{x}}^2 + G_{\mathrm{y}}^2} \tag{3}$$

$$|G| = |G_{\mathrm{x}}| + |G_{\mathrm{y}}| \tag{4}$$

Where Gx and Gy are the gradients in the x- and y-directions respectively.

It is obvious from Figure 3, that an image of the gradient magnitudes often indicate the edges quite clearly. However, the edges are typically broad and thus do not indicate exactly where the edges are. To make it possible to determine this, the direction of the edges must be determined and stored as shown in Equation (5).

$$\theta = \arctan\left(\frac{|G_{\mathrm{y}}|}{|G_{\mathrm{x}}|}\right) \tag{5}$$

(a) Smoothed

(b) Gradient magnitudes

Figure 3: *The gradient magnitudes in the smoothed image shown in 3b as well as their directions are determined by applying e.g. the Sobel-operator.*

Non-maximum suppression (`CEDNonMaxSup`)

The purpose of this step is to convert the "blurred" edges in the image of the gradient magnitudes to "sharp" edges. Basically this is done by preserving all local maxima in the gradient image, and deleting everything else. The algorithm is for each pixel in the gradient image:

1. Round the gradient direction θ to nearest 45◦, corresponding to the use of an 8-connected neighborhood.

2. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction. I.e. if the gradient direction is north (theta = 90◦), compare with the pixels to the north and south.

3. If the edge strength of the current pixel is largest; preserve the value of the edge strength.

4. If not, suppress (i.e. remove) the value.

A simple example of non-maximum suppression is shown in Figure 4. Almost all pixels have gradient directions pointing north. They are therefore compared with the pixels above and below. The pixels that turn out to be maximal in this comparison are marked with white borders. All other pixels will be suppressed. Figure 5 shows the effect on the test image.

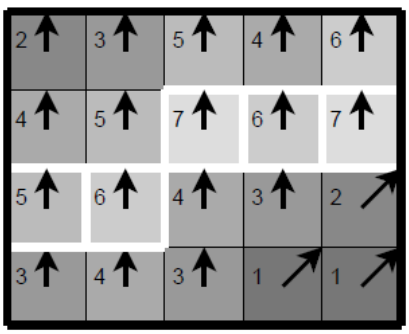


Figure 4: *Illustration of non-maximum suppression. The edge strengths are indicated both as colors and numbers, while the gradient directions are shown as arrows. The resulting edge pixels are marked with white borders*

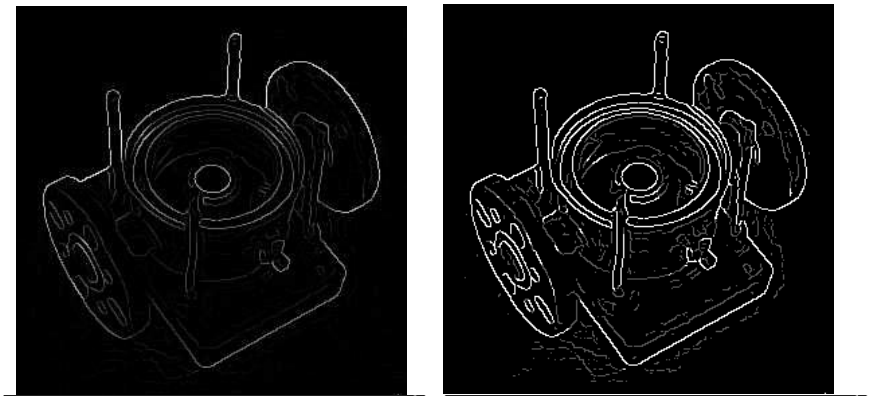
.

(a) Gradient Values (b) Edges after non-maximum suppression

Figure 5: *Non-maximum suppression. Edge-pixels are only preserved where the gradient has local maxima.*

Double thresholding (`CEDHysteris`)

The edge-pixels remaining after the non-maximum suppression step 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 for instance due to rough surfaces. The simplest way to discern between these would be to use a threshold, so that only edges stronger that a certain value would be preserved. The Canny edge detection algorithm uses double thresholding. 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. The effect on the test image with thresholds of 20 and 80 is shown in Figure 6.



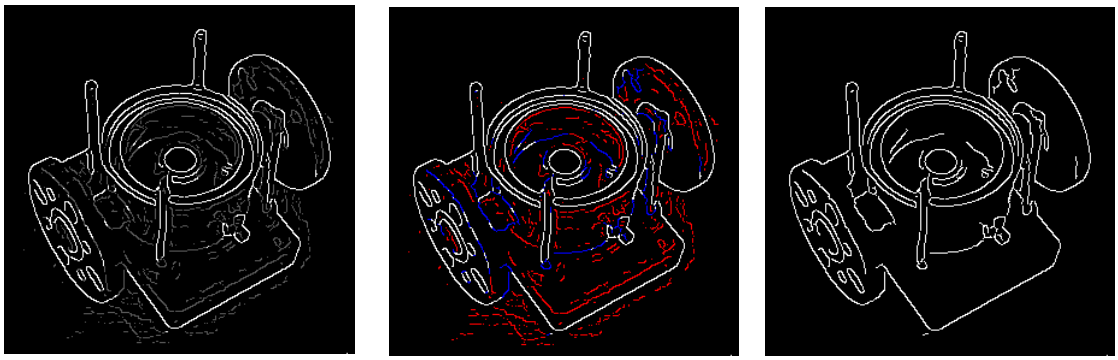
(a) Edges after non-maximum suppression (b) Double thresholding

Figure 6: *Thresholding of edges. In the second image strong edges are white, while weak edges are grey. Edges with a strength below both thresholds are suppressed.*

Edge tracking by hysteresis (`CEDEdgeTracking`)

Strong edges are interpreted as "certain edges", and can immediately be included in the final edge image. Weak edges are included if and only if they are connected to strong edges. The logic 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. The latter type will probably be distributed independently of edges on the entire image, and thus only a small amount will be located adjacent to strong edges. Weak edges due to true edges are much more likely to be connected directly to strong edges.

Edge tracking can be implemented by BLOB-analysis (Binary Large Object). The edge pixels are divided into connected BLOB's using 8-connected neighborhood. BLOB's containing at least one strong edge pixel are then preserved, while other BLOB's are suppressed. The effect of edge tracking on the test image is shown in Figure 7.



(a) Double thresholding (b) Edge tracking by hysteresis (c) Final output

Figure 7: *Edge tracking and final output. The middle image shows strong edges in white, weak edges connected to strong edges in blue, and other weak edges in red.*

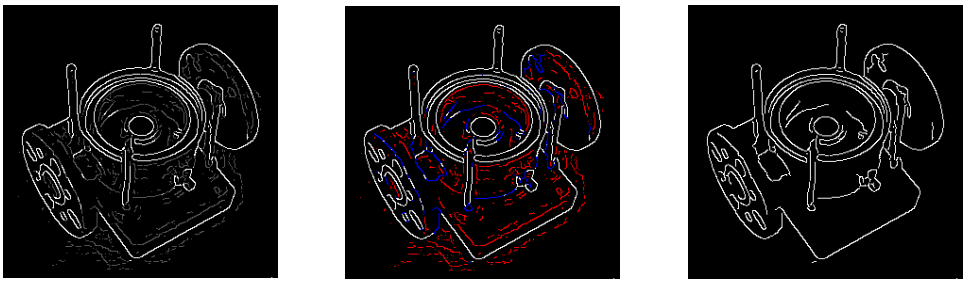
Implementation Details

a. The (source) image and the thresholds can be chosen arbitrarily (input parameters)
b. Only a smoothing filter with a standard deviation of σ = 1.4 is needed (shown in Equation 1).
c. Use the "correct" Euclidean measure for the edge strength
d. The difference filters cannot be applied to edge pixels. This causes the output image to be 8 pixels smaller in each direction.

The last step in the algorithm known as edge tracking can be implemented as an iterative approach. First all weak edges are scanned for neighbor edges and joined into groups. At the same time it is marked which groups are adjacent. Then all of these markings are examined to determine which groups of weak edges are connected to strong edges (directly or indirectly). All weak edges that are connected to strong edges are marked as strong edges themselves. The rest of the weak edges are suppressed.

Figure 8 shows the complete edge detection process on the test image including all intermediate results.

(a) Original (b) Smoothed (c) Gradient magnitudes (d) Edges after non-maximum suppression



(e) Double thresholding (f) Edge tracking by hysteresis (g) Final output

***Figure 8**: All steps of the edge detection.*

Submitting your work:

- Create a test module (`TestCanny`) that tests your algorithm on both sample images
  - Rice & Tire (play around with thresholds to customize for each image)
  - Output of each stage should be displayed as in Figure 8 with appropriate titles

- Upload your exercises with your ID filename **2008CS1001–L4.ipnyb** file on google classroom
  - Your file should have the following modules
    - `Canny` `[10 Points]`
    - `CEDSmooth` `[5 Points]`
    - `CEDGrads` `[5 Points]`
    - `CEDNonMaxSup` `[10 Points]`
    - `CEDHysteris` `[10 Points]`
    - `CEDEdgeTracking` `[10 Points]`
    - `TestCanny` `[20 Points]`
  - ***Negative marks for any problems/errors in running your programs***
  - Submit/Upload it to Google Classroom