Final Project: Multistage Edge Detection with Canny Algorithms

Course: **COSC 6349 - Computer Vision and Image Processing**
Professor: **Dr. Hongyu Guo**
Principal Author: **Matthew Harper**

Overview

Image features in image processing represent essential visual and mathematical characteristics that convey meaningful information within an image. These features include observable attributes such as shape, color, texture, lines, and edges, as well as mathematical descriptors such as pixel intensity, gradients, slope, angle or direction, magnitude, noise, and frequency [1, 2]. Moreover, feature selection and detection is a fundamental concept in image processing and computer vision, enabling the identification of prominent and distinctive patterns while reducing complex pixel-level data into structured and analyzable components [1, 2].

In addition, edge detection is a critical image segmentation and feature detection technique which focuses on identifying object boundaries through abrupt changes in image intensity, commonly modeled using step, ramp, and roof-edge intensity profiles [2, 4]. However, when comparing the available edge detection approaches, the multistage Canny edge detection algorithm remains one of the most widely used due to its robust and optimal performance of its multistage image segmentation algorithm.

The Canny method was specifically designed to satisfy three primary objectives which include achieving a low error rate with minimal noise response, detection of accurate edge localization, and obtaining a prominent true edge [2, 4]. These objectives are approximated mathematically through the use of Gaussian smoothing to reduce noise, gradient magnitude and direction computation to detect candidate edges, non-maximum suppression to refine edge thickness, and thresholding with edge linking to preserve meaningful edges while suppressing false detections [2, 4].

The reliable and well-localized edges produced by the Canny edge detection algorithm provide a strong foundation for subsequent image processing tasks, particularly image segmentation. Image segmentation relies on dividing an image into meaningful regions based on similarity or discontinuity in intensity, and Canny-based edge characteristics are especially effective for segmentation approaches that depend on gradient intensity, magnitude, and direction [2, 4]. As a result, Canny edge detection is an important technique in image segmentation, object recognition, and many computer vision applications [1–4].

The Multistage Canny Edge Detector

Edge detection can be defined as a method utilized for the segmentation of an image(s) based on the identification of rapid and abrupt changes in intensity, referred to as local minima and local maxima in the gradient.

Moreover, edge detection models can also be classified based on their intensity profiles or the way in which the edge transitions in intensity. The ***step-edge*** or ideal edge is an edge characterized as

having two distinct binary intensities occurring over one pixel. The figure below illustrates this step edge phenomenon. Images with intensity profiles described as having slowly increasing slope inversely proportionate to the degree to which the edge is blurred is known as a ***ramp*** profile. ***Roof-edge*** is another commonly observed image edge detection intensity profile in which a model of lines through a localized region are observed cresting to a point [1, 2, 4].

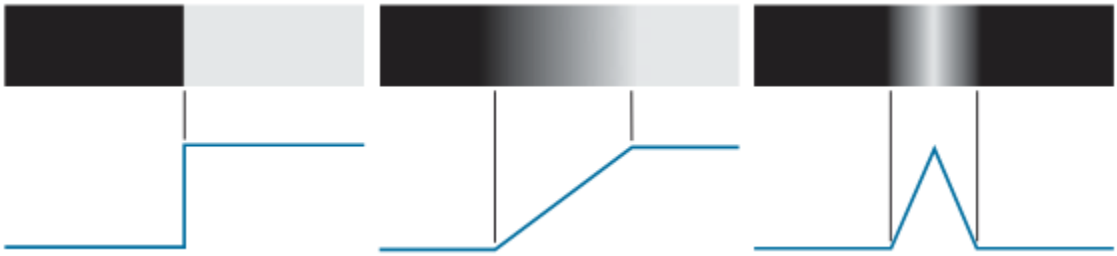


Figure 1 – illustrated the three edge detection models (step, ramp, roof-edge) [2].

As the figures for each edge detection intensity profile suggest, we observe that a first-order derivative is positive during the positive ramp of intensity slope. In addition, the second derivative can be utilized in the detection of zero crossings of constant intensity or constant intensity areas. This is useful in determining the beginning and end of an edge transition and whether a pixel lies in the dark or lighter side of an edge [2, 4].

The figure above displays ideal intensity transitions with little or no observed noise. However, for most image processing this is not the case. As a result, noise filtering (Gaussian noise) is often a necessity for accurate edge detection [2, 4]. Additionally, there are three basic steps required in edge detection which include the following:

1. Image smoothing and noise reduction (Gaussian filter)
2. Edge point detection (first-order derivative)
3. Edge localization (second-order derivative) [2, 4].

Although the ramp and roof edge intensity edge profiles referred to above are common edge models, the ***Canny edge detection algorithm*** (Canny 1986) method is based on the ***step-edge*** model [2]. However, the Canny method was developed as an advanced technique producing more accurate results with the following objectives [2, 4]:

1. **Low error rate** – edge detection with minimum random noise exhibited.
2. **Edge points well localized** – edges located within some t**hreshold** close to the edge point.
3. **Single edge point response** – only one point identified for each **true edge**.

The significance of the Canny edge detection method is the method for expressing the three objectives mathematically [2]. Similar to the three basic steps required in edge detection above, the Canny edge detection algorithm consists of the following steps [2]:

1. Image smoothing with a Gaussian filter for noise reduction.
2. Computing the gradient magnitude identifying local minima and maxima and angle images.
3. Applying nonmaximal suppression to the gradient magnitude image.

4. Utilizing thresholding to identify and detect linked edges.

Image Smoothing Gaussian Filter

Canny identified that an ideal closed form solution did not exist for all three objectives. However, noted that additive white Gaussian noise was a good approximation for real-world applications and that white Gaussian noise had a normal distribution with a first derivative which was also a good approximation for the ideal step-edge detector [2]. The 1-D Gaussian derivative below is illustrated in the formula below [2, 4]:

$$\frac{d}{dx} e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

However, for image edge detection we need two dimensional images where the 2-D Gaussian function G(x,y) becomes [2, 4]:

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

Finally, the edges are smoothed and filtered of noise by using convolution over the image to obtain the smoothed image f_s as follows [2, 4]:

$$f_s(x,y) = G(x,y) \bigstar f(x,y)$$

As a result, the python program below begins with a pre-processed gray image and applies a smoothing Gaussian kernel which acts as a filter consisting of 1x5 element array [0.1, 0.1, .1, 0.1, 0.1]. The algorithm performs 2D convolution in two stages, one for Gx in horizontal edges and another for the vertical edges Gy [2].

```
import numpy as np
import matplotlib.pyplot as plt

#Define a 1D (1x5) Smoothing Gaussian filter
kernel = np.array([0.1, 0.1, .1, 0.1, 0.1], dtype=np.float32) #values can be adjusted based on image noise

#Assign the signal image from the pre-processed gray image
signal=gray_image

# Convolve over the horizontal edge Gx
tmp = np.zeros_like(signal, dtype=np.float32) #create a zeros array same size as image
for y in range(signal.shape[0]): #convlolve over signal array and assign values to tmp array
    tmp[y, :] = np.convolve(signal[y, :], kernel, mode="same")

# Convolve over the vertical edge Gy
out = np.zeros_like(signal, dtype=np.float32)  #create a zeros array same size as image
for x in range(signal.shape[1]):#convlolve over tmp array and assign values to final output array
    out[:, x] = np.convolve(tmp[:, x], kernel, mode="same")

#Assign the output processed image as gray image 0-255 pixel magnitude
out = np.clip(out, 0, 255).astype(np.uint8)
```

This smoothing effect of the Gaussian filter algorithm can be observed as a pronounced blurring effect on the image which translates in a noise reduction for low to medium magnitudes of image signal noise which is utilized multistage processing and image segmentation continued in the next steps.

Gradient Magnitude (Gx, Gy) & Gradient-Based Direction Angle

The next stage of the Canny edge detection after the convolution of the Gaussian smoothing filter is finding or calculating the gradient magnitude of the image pixels. This is accomplished by convolving Sobel derivative operators which are typically 3x3 or 5x5 masks having coefficients which sum to zero such as the 3x3 operators illustrated below [2].

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Sobel Gradient Operator

Mask **m(x,y) = |Gx| + |Gy|** [2].

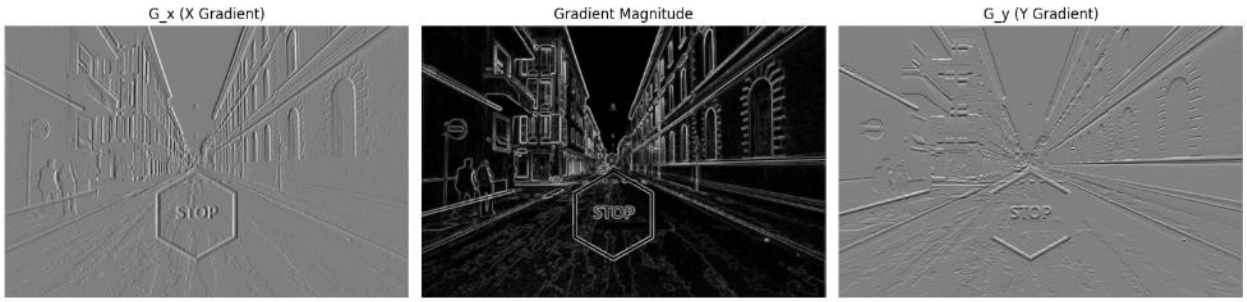
Gradient Magnitude

$$\boldsymbol{M(x,y)} = \sqrt{\boldsymbol{G_x^2} + \boldsymbol{G_y^2}} \qquad [2].$$

In order to perform convolution function over each element in the array (pixel) padding is required to position the convolution window over the edge of the image. As a result, the size and shape of the image is first determined and then used to calculate the kernel height and width and finally the padding. The convolution with padding function is called recursively for each Sobe-based gradient operator and result is returned and plotted for comparison [2, 5].

```python
def convolve2d_manual(image, kernel):
    #determine the height and width of the image for padding
    #padding is needed for convolution window on sides of
    #of image in which convolution window is [2k+1,2k+1]/2
    height, width = image.shape
    kernal_height, kernel_width = kernel.shape
    pad_h = kernal_height // 2
    pad_w = kernel_width // 2

    #Pad image with zeros
    padded = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='reflect')
    out = np.zeros((height, width), dtype=np.float32)

    #Convolve over image
    for y in range(height):
        for x in range(width):
            region = padded[y:y+kernal_height, x:x+kernel_width]
            out[y, x] = np.sum(region * kernel)

    return out

def gradient_magnitude_sobel_operators(image):
    image = image.astype(np.float32)

    # Sobel kernels from Digital Image Processing 3rd.ed.
    #Gonzalez, R. C., & Wintz, P. (2008). Digital image processing (3rd ed.). Addison-Wesley.
    #Page 166 3.6.4 Using First-Order Derivatives for Image Sharpening - The Gradient & Sobel Operators

    kernel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ], dtype=np.float32)

    kernel_y = np.array([
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1]
    ], dtype=np.float32)

    Gx = convolve2d_manual(image, kernel_x)
    Gy = convolve2d_manual(image, kernel_y)

    gradient_magnitude = np.sqrt(np.square(Gx) + np.square(Gy))

    return gradient_magnitude, Gx, Gy

magnitude, Gx, Gy = gradient_magnitude_sobel_operators(signal)
```

The plotted results illustrate how the gradient operators detected the gradient edges consisting of Gx and Gy. As an example, we can observe how the left and right stop sign edges were left out of the Gy gradient gray image, which only identifies vertical edges.



Similar to the three basic steps required in edge detection above, the Canny edge detection algorithm consists of the following steps [2]:

Gradient Direction Angle and Applied Non-Max Suppression

The next step in multistage Canny edge detection is the direction or angle calculation for edge enhancement followed by the suppression of less significant edge transitions. This is required to thin edge lines and ridges around local maxima which tend to create thick lines as the edge transitions from low to high magnitudes [2].

The angle is calculated from the gradient magnitudes Gx and Gx with the equation below as follows [2, 5]:

$$\alpha(x, y) = \arctan\left[\frac{Gy}{Gx}\right]$$

```
theta = np.arctan2(Gy, Gx)

angle = (np.degrees(theta) +180) % 180.0 #convert to angle


def nms_non_max_suppression(magnitude, angle): #magnitude and angle
    height, weight = magnitude.shape
    out = np.zeros((height, weight), dtype=np.float32)

    for y in range(1, height-1):
        for x in range(1, weight-1):
            a = angle[y, x]

            # Quantize to 4 directions
            if (0 <= a < 22.5) or (157.5 <= a < 180):
                m1, m2 = magnitude[y, x-1], magnitude[y, x+1]          # 0°
            elif 22.5 <= a < 67.5:
                m1, m2 = magnitude[y-1, x+1], magnitude[y+1, x-1]      # 45°
            elif 67.5 <= a < 112.5:
                m1, m2 = magnitude[y-1, x], magnitude[y+1, x]          # 90°
            else:
                m1, m2 = magnitude[y-1, x-1], magnitude[y+1, x+1]      # 135°

            out[y, x] = magnitude[y, x] if (magnitude[y, x] >= m1 and magnitude[y, x] >= m2) else 0.0

    return out
```

The resulting image resembles an image with enhanced edging and sharper detail based on the magnitude and direction of the edge angle with thinner, more localized lines.

False Edge Suppression with Canny Edge Thresholding

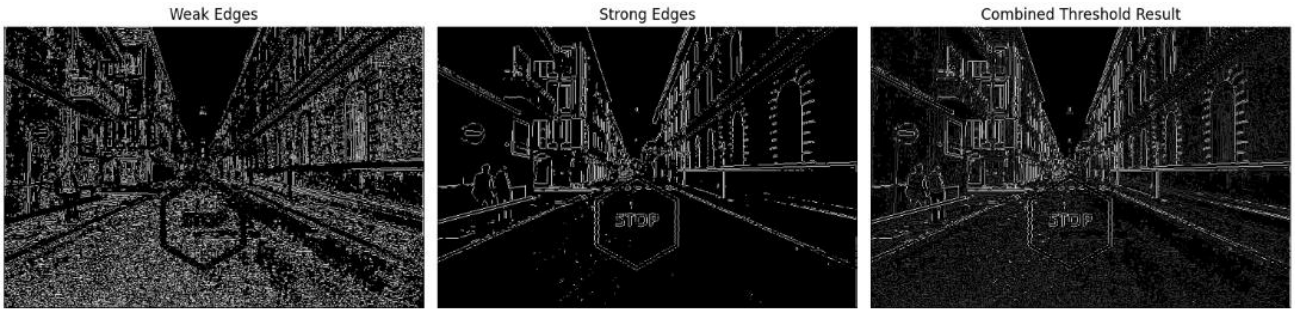
The final step is to filter the image based on edge magnitude intensity by setting threshold values for low and high. When filtering the image for high and low values caution is utilized to prevent the exclusion of weak edges while filtering out false positives and strong edges while filtering out false negatives. The settings are adjustable for both weak and strong and high and low thresholds. This technique is referred to as hysteresis thresholding sometimes called double thresholding [2]. The python function below accomplishes this in two parts, first by creating a zeros matrix based on the shape of the non-maximum suppressed image shape. The image is then processed and filtered for strong and weak edge detection by row and column [2, 5].

```python
def threshold(nonmax_imgage, low_thresh, high_thresh):

    #threshold to an Non-Maximum Suppression image from previous step
    #lowThreshold  : ratio for weak edges
    #highThreshold : ratio for strong edges

    height, width = nonmax_imgage.shape #MxN shape of matrix
    #create zeros matrix for storing weak and strong edges
    img = np.zeros((height, width), dtype=np.uint8)

    # Filter values for strong/weak edge detection
    weak = np.uint8(200)
    strong = np.uint8(250)

    # Identify strong, weak, and non-edge pixels
    strong_i, strong_j = np.where(nonmax_imgage>= high_thresh)
    weak_i, weak_j = np.where((nonmax_imgage >= low_thresh) & (nonmax_imgage < high_thresh))

    img[strong_i, strong_j] = strong
    img[weak_i, weak_j] = weak

    return img, weak, strong

#return threshold image with weak and strong edge array values can be adjusted
#Canny recommended values of 2-3:1 (high-low) - using .10-.30 or 3:1
thresh_img, weak, strong = double_threshold(non_max_suppresed_img,.1,.30)
```

After the threshold image is created, weak and strong images can be made by separating the edges based on the weak and strong filters identified.

```python
#Create black (black=0) images for each threshold processed image for comparison
weak_img = np.zeros_like(thresh_img, dtype=np.uint8)
strong_img = np.zeros_like(thresh_img, dtype=np.uint8)

#copy weak edges from array to weak image
weak_img[thresh_img == weak] = weak
#copy strong edges from array to strong images
strong_img[thresh_img == strong] = strong
```

The final Canny Edge Detection based images for strong and weak images are illustrated below which demonstrate the effectiveness of multistage image segmentation techniques such as Canny Edge Detection and their usefulness in computer vision and image processing.



References

[1] Guo, Hongyu. Computer Vision & Image Processing. COSC 6349 – Module 4: Color Spaces.
[2] Gonzalez, R. C., & Wintz, P. (2008). *Digital image processing* (3rd ed.). Addison-Wesley.
[3] Active Contours. (n.d.). MATLAB.
https://www.mathworks.com/help/images/ref/activecontour.html
[4] Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6), 679–698. https://doi.org/10.1109/tpami.1986.4767851
[5] Sahir, S. (n.d.). Canny Edge Detection step by step in Python — Computer Vision.
https://medium.com/data-science/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123