|  |
| --- |
| ปฏิบัติการ  Lab 7 – Edge |

1. Sobel Operator

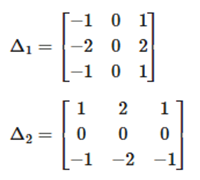
Edge Detection ในภาพดิจิทัลใช้เทคนิคที่ใช้คำนวณการเปลี่ยนแปลงของความเข้มของพิกเซลในแนวแกน X และ Y เพื่อหาเส้นขอบหรือโครงสร้างที่มีการเปลี่ยนแปลงอย่างเด่นชัดในภาพ เช่น ขอบของวัตถุ หรือรายละเอียดที่สำคัญ

หลักการทำงานของ Sobel Operator

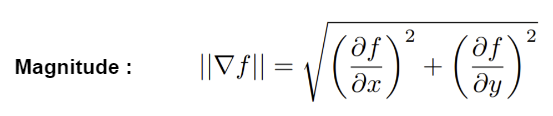
1. Sobel Operator ใช้คอนโวลูชัน (Convolution) กับภาพต้นฉบับ โดยใช้ Kernel หรือ Filter สองตัว:

* Gx: สำหรับตรวจจับการเปลี่ยนแปลงในแนวนอน (Horizontal edges)
* Gy: สำหรับตรวจจับการเปลี่ยนแปลงในแนวตั้ง (Vertical edges)

1. Kernel หรือ Filter ของ Sobel Operator เป็นเมทริกซ์ขนาด 3×3 ที่นิยามดังนี้



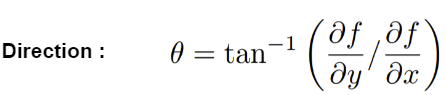
1. ผลลัพธ์จาก Gx และ Gy จะถูกนำมาคำนวณความเข้มของขอบ (Edge Magnitude) ด้วยสมการ



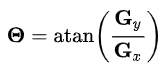
หรือ

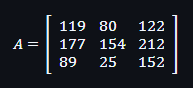


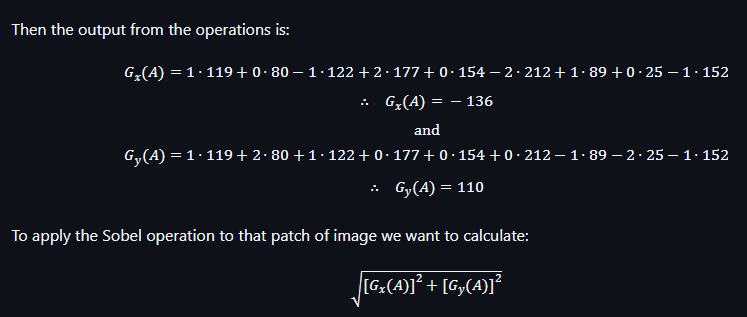
1. หาทิศทางของขอบ (Gradient Direction) สามารถคำนวณได้ด้วยสมการ



หรือ







|  |
| --- |
| from matplotlib.image import imread  import matplotlib.pyplot as plt  import numpy as np  image\_file = 'original\_image.PNG'  input\_image = imread(image\_file)  [nx, ny, nz] = np.shape(input\_image)  r\_img, g\_img, b\_img = input\_image[:, :, 0], input\_image[:, :, 1], input\_image[:, :, 2]  gamma = 1.400  r\_const, g\_const, b\_const = 0.2126, 0.7152, 0.0722  grayscale\_image = r\_const \* r\_img \*\* gamma + g\_const \* g\_img \*\* gamma + b\_const \* b\_img \*\* gamma  fig1 = plt.figure(1)  ax1, ax2 = fig1.add\_subplot(121), fig1.add\_subplot(122)  ax1.imshow(input\_image)  ax2.imshow(grayscale\_image, cmap=plt.get\_cmap('gray'))  fig1.show()  """  The kernels Gx and Gy can be thought of as a differential operation in the "input\_image" array in the directions x and y  respectively. These kernels are represented by the following matrices:  \_ \_ \_ \_  | | | |  | 1.0 0.0 -1.0 | | 1.0 2.0 1.0 |  Gx = | 2.0 0.0 -2.0 | and Gy = | 0.0 0.0 0.0 |  | 1.0 0.0 -1.0 | | -1.0 -2.0 -1.0 |  |\_ \_| |\_ \_|  """  Gx = np.array([[1.0, 0.0, -1.0], [2.0, 0.0, -2.0], [1.0, 0.0, -1.0]])  Gy = np.array([[1.0, 2.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -2.0, -1.0]])  [rows, columns] = np.shape(grayscale\_image)  sobel\_filtered\_image = np.zeros(shape=(rows, columns))  for i in range(rows - 2):  for j in range(columns - 2):  gx = np.sum(np.multiply(Gx, grayscale\_image[i:i + 3, j:j + 3]))  gy = np.sum(np.multiply(Gy, grayscale\_image[i:i + 3, j:j + 3]))  sobel\_filtered\_image[i + 1, j + 1] = np.sqrt(gx \*\* 2 + gy \*\* 2)  fig2 = plt.figure(2)  ax1, ax2 = fig2.add\_subplot(121), fig2.add\_subplot(122)  ax1.imshow(input\_image)  ax2.imshow(sobel\_filtered\_image, cmap=plt.get\_cmap('gray'))  fig2.show() |

#1 จงหาโค้ดการทำงานของ Sobel โดยใช้ Library นำโค้ดมาใส่ในช่องด้านล่าง และแสดงผลลัพธ์โดยให้แคปหน้าผลลัพธ์จากโค้ดเก่า และโค้ดใหม่มาเทียบกัน รวมถึงวัดความเร็วในการทำงานของทั้งสองแบบและนำมาแสดงผล

|  |
| --- |
|  |

2. Canny Edge

Canny Edge Detection เป็นหนึ่งในอัลกอริธึมที่ใช้ในการตรวจจับขอบ (Edge Detection) ที่ได้รับความนิยมและทรงพลังมากใน Image Processing เนื่องจากมันผสมผสานเทคนิคที่ซับซ้อนเพื่อให้ได้ผลลัพธ์ที่มีความแม่นยำและลดผลกระทบจากสัญญาณรบกวนในภาพได้ดี

หลักการทำงานของ Canny Edge Detection

1. Noise Reduction (การลดสัญญาณรบกวน): โดยก่อนที่จะตรวจจับขอบ อัลกอริธึมจะใช้ Gaussian Blur เพื่อลดสัญญาณรบกวนในภาพ โดยใช้ Kernel (ฟิลเตอร์) แบบ Gaussian เพื่อทำให้ภาพเรียบเนียน ซึ่งช่วยลดผลกระทบจากสัญญาณรบกวนที่อาจทำให้เกิด False Edges
2. Gradient Calculation: ใช้ Sobel Operator หรือฟิลเตอร์ที่คล้ายกันในการคำนวณ Gradient ในแกน X และ Y (Gx และ Gy) รวมถึงคำนวณขนาดของ Gradient และทิศทางของ Gradient
3. Non-Maximum Suppression (การลดขนาดของขอบ): ลดขนาดของขอบที่ตรวจจับได้เพื่อให้ขอบบางที่สุด (Thin Edge) โดยเก็บเฉพาะพิกเซลที่เป็นขอบสูงสุดในทิศทางของ Gradient และพิกเซลที่ไม่ใช่ขอบสูงสุดในทิศทางของ Gradient จะถูกกำหนดค่าเป็น 0
4. Double Thresholding: ใช้ High Threshold และ Low Threshold ในการแยกขอบที่มีความสำคัญ: โดยพิกเซลที่มีค่ามากกว่า High Threshold จะถูกระบุว่าเป็น "Strong Edge" พิกเซลที่มีค่าระหว่าง Low Threshold และ High Threshold จะถูกระบุว่าเป็น "Weak Edge"พิกเซลที่มีค่าต่ำกว่า Low Threshold จะถูกลบออก
5. Edge Tracking by Hysteresis (การติดตามขอบ): ขอบที่เป็น "Weak Edge" จะถูกพิจารณาว่าเป็นขอบสุดท้ายก็ต่อเมื่อมันเชื่อมต่อกับ "Strong Edge" กระบวนการนี้ช่วยลดการตรวจจับขอบที่ไม่จำเป็นและทำให้ผลลัพธ์มีความต่อเนื่องมากขึ้น

|  |
| --- |
| import numpy as np  import cv2  from scipy.ndimage import convolve  import matplotlib.pyplot as plt  # Gaussian Filter  def gaussian\_filter(image, kernel\_size=5, sigma=1.0):  ax = np.linspace(-(kernel\_size // 2), kernel\_size // 2, kernel\_size)  gauss = np.exp(-0.5 \* np.square(ax) / np.square(sigma))  kernel = np.outer(gauss, gauss)  kernel /= np.sum(kernel)  return convolve(image, kernel)  # Sobel Filter  def sobel\_filters(image):  Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) # Horizontal  Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]) # Vertical  Gx = convolve(image, Kx)  Gy = convolve(image, Ky)  magnitude = np.sqrt(Gx\*\*2 + Gy\*\*2)  direction = np.arctan2(Gy, Gx) # Gradient direction  return magnitude, direction  # Non-Maximum Suppression  def non\_maximum\_suppression(magnitude, direction):  rows, cols = magnitude.shape  suppressed = np.zeros((rows, cols), dtype=np.float32)  direction = np.rad2deg(direction)  direction[direction < 0] += 180  for i in range(1, rows-1):  for j in range(1, cols-1):  # Direction quantization  if (0 <= direction[i, j] < 22.5) or (157.5 <= direction[i, j] <= 180):  neighbors = (magnitude[i, j+1], magnitude[i, j-1])  elif 22.5 <= direction[i, j] < 67.5:  neighbors = (magnitude[i-1, j+1], magnitude[i+1, j-1])  elif 67.5 <= direction[i, j] < 112.5:  neighbors = (magnitude[i+1, j], magnitude[i-1, j])  else: # 112.5 <= direction < 157.5  neighbors = (magnitude[i-1, j-1], magnitude[i+1, j+1])  # Suppress non-maximum values  if magnitude[i, j] >= neighbors[0] and magnitude[i, j] >= neighbors[1]:  suppressed[i, j] = magnitude[i, j]  else:  suppressed[i, j] = 0  return suppressed  # Double Threshold  def double\_threshold(suppressed, low\_threshold, high\_threshold):  strong = 255  weak = 75  strong\_i, strong\_j = np.where(suppressed >= high\_threshold)  weak\_i, weak\_j = np.where((suppressed <= high\_threshold) & (suppressed >= low\_threshold))  result = np.zeros\_like(suppressed, dtype=np.uint8)  result[strong\_i, strong\_j] = strong  result[weak\_i, weak\_j] = weak  return result, weak, strong  # Edge Tracking by Hysteresis  def edge\_tracking\_by\_hysteresis(image, weak, strong):  rows, cols = image.shape  for i in range(1, rows-1):  for j in range(1, cols-1):  if image[i, j] == weak:  if (strong in [image[i+1, j-1], image[i+1, j], image[i+1, j+1],  image[i, j-1], image[i, j+1],  image[i-1, j-1], image[i-1, j], image[i-1, j+1]]):  image[i, j] = strong  else:  image[i, j] = 0  return image  # Main Function for Canny Edge Detection  def canny\_edge\_detection(image, low\_threshold, high\_threshold):  # 1. Step 1: Gaussian Blur  smoothed = gaussian\_filter(image)  # 2. Step 2: Sobel Filters  magnitude, direction = sobel\_filters(smoothed)  # 3. Step 3: Non-Maximum Suppression  suppressed = non\_maximum\_suppression(magnitude, direction)  # 4. Step 4: Double Thresholding  thresholded, weak, strong = double\_threshold(suppressed, low\_threshold, high\_threshold)  # 5. Step 5: Edge Tracking by Hysteresis  final\_edges = edge\_tracking\_by\_hysteresis(thresholded, weak, strong)  return final\_edges  image = cv2.imread('Lenna.png', cv2.IMREAD\_GRAYSCALE)  edges = canny\_edge\_detection(image, low\_threshold=50, high\_threshold=150)  plt.figure(figsize=(10, 5))  plt.subplot(1, 2, 1)  plt.title('Original Image')  plt.imshow(image, cmap='gray')  plt.subplot(1, 2, 2)  plt.title('Canny Edge Detection')  plt.imshow(edges, cmap='gray')  plt.show() |

#2 จงหาข้อผิดพลาดจากโค้ดข้างต้น และทำการปรับปรุงให้ได้ผลลัพธ์ที่ถูกต้อง

|  |
| --- |
|  |

3. Sobel vs Canny 

ตัวอย่างการเรียกใช้ Canny Edge ผ่าน Library

|  |
| --- |
| import cv2  import matplotlib.pyplot as plt  image = cv2.imread('Lenna.png', cv2.IMREAD\_GRAYSCALE)  edges = cv2.Canny(image, threshold1=100, threshold2=200)  plt.figure(figsize=(10, 5))  plt.subplot(1, 2, 1)  plt.title('Original Image')  plt.imshow(image, cmap='gray')  plt.subplot(1, 2, 2)  plt.title('Canny Edge Detection')  plt.imshow(edges, cmap='gray')  plt.show() |

#3 จงแก้ไชโค้ดข้างต้นเพื่อนำมาใช้กับวิดีโอ

|  |
| --- |
|  |