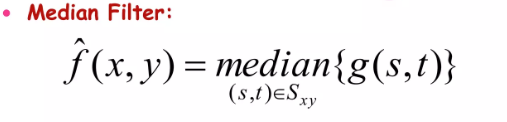
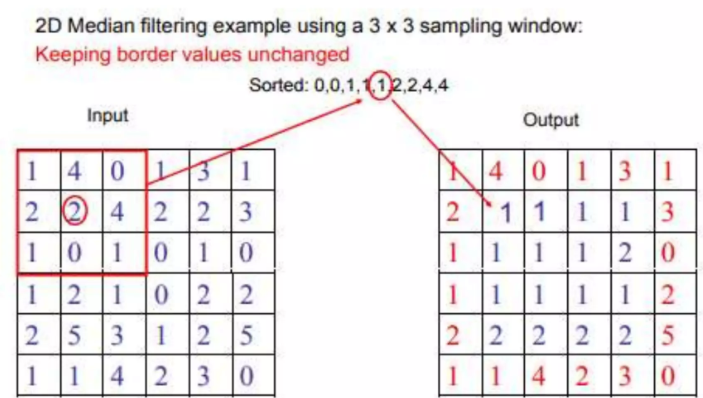
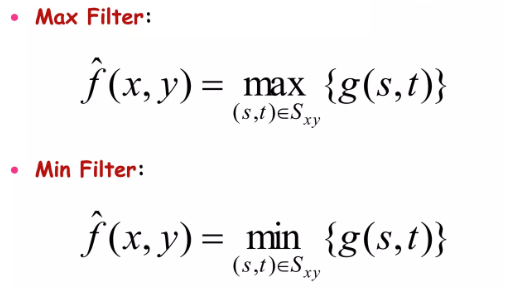
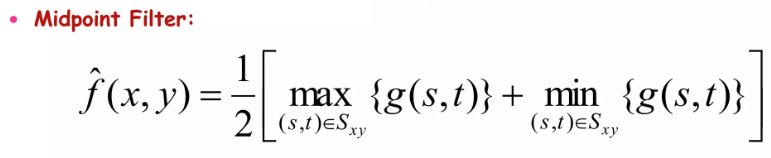
|  |
| --- |
| ปฏิบัติการ  Lab 5 – Order-statistics filters, Sharpening |

1. Order-statistics filters









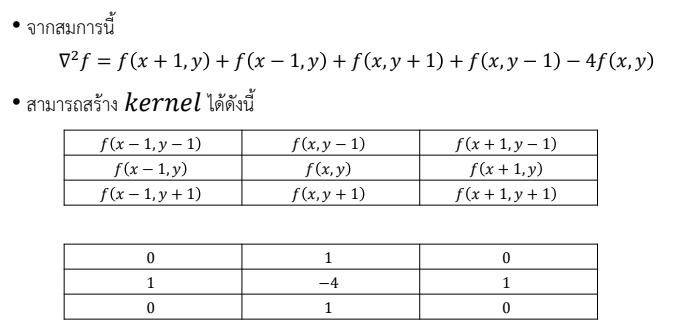
|  |
| --- |
| import numpy as np  from scipy.ndimage import median\_filter, maximum\_filter, minimum\_filter  import matplotlib.pyplot as plt  def apply\_median\_filter(image, size):  return median\_filter(image, size=size)  def apply\_max\_filter(image, size):  return maximum\_filter(image, size=size)  def apply\_min\_filter(image, size):  return minimum\_filter(image, size=size)  # Example usage  def main():  np.random.seed(0)  image = np.random.randint(0, 256, (10, 10), dtype=np.uint8)  print("Original Image:\n", image)  filter\_size = 3  median\_filtered = apply\_median\_filter(image, size=filter\_size)  max\_filtered = apply\_max\_filter(image, size=filter\_size)  min\_filtered = apply\_min\_filter(image, size=filter\_size)  print("\nMedian Filtered Image:\n", median\_filtered)  print("\nMax Filtered Image:\n", max\_filtered)  print("\nMin Filtered Image:\n", min\_filtered)  plt.figure(figsize=(10, 8))  plt.subplot(2, 2, 1)  plt.title("Original Image")  plt.imshow(image, cmap="gray")  plt.colorbar()  plt.subplot(2, 2, 2)  plt.title("Median Filtered")  plt.imshow(median\_filtered, cmap="gray")  plt.colorbar()  plt.subplot(2, 2, 3)  plt.title("Max Filtered")  plt.imshow(max\_filtered, cmap="gray")  plt.colorbar()  plt.subplot(2, 2, 4)  plt.title("Min Filtered")  plt.imshow(min\_filtered, cmap="gray")  plt.colorbar()  plt.tight\_layout()  plt.show()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

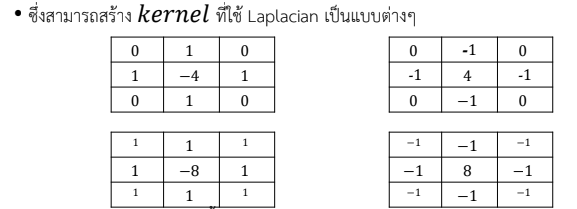
#1 จงปรับปรุงโค้ดเพื่อนำ Midpoint filter เพิ่มมาอีก 1 อัลกอริทึม และปรับปรุงให้โค้ดทำงานร่วมกับรูปภาพของตนเอง และปรับปรุงการแสดงผลให้แสดงภาพผลลัพธ์แทนอาเรย์ จากนั้นนำโค้ดที่แก้ไข และผลลัพธ์ที่ได้มาใส่ด้านล่าง

|  |
| --- |
| def apply\_filter(image, filter\_size, mode):  def get\_window(x, y):  window = []  for i in range(-half\_size, half\_size + 1):  for j in range(-half\_size, half\_size + 1):  xi = min(max(x + i, 0), rows - 1)  yj = min(max(y + j, 0), cols - 1)  window.append(image[xi][yj])  return window  # Validate inputs  if filter\_size % 2 == 0:  raise ValueError("Filter size must be an odd integer.")  half\_size = filter\_size // 2  rows, cols = len(image), len(image[0])  filtered\_image = [[0 for \_ in range(cols)] for \_ in range(rows)]  for x in range(rows):  for y in range(cols):  window = get\_window(x, y)  if mode == 'median':  filtered\_image[x][y] = sorted(window)[len(window) // 2]  elif mode == 'max':  filtered\_image[x][y] = max(window)  elif mode == 'min':  filtered\_image[x][y] = min(window)  else:  raise ValueError("Mode must be 'median', 'max', or 'min'.")  return filtered\_image  if \_\_name\_\_ == "\_\_main\_\_":  input\_image = [  [1, 2, 3, 4, 5],  [6, 7, 8, 9, 10],  [11, 12, 13, 14, 15],  [16, 17, 18, 19, 20],  [21, 22, 23, 24, 25]  ]  filter\_size = 3  print("Original Image:")  for row in input\_image:  print(row)  median\_filtered = apply\_filter(input\_image, filter\_size, 'median')  print("\nMedian Filtered Image:")  for row in median\_filtered:  print(row)  max\_filtered = apply\_filter(input\_image, filter\_size, 'max')  print("\nMax Filtered Image:")  for row in max\_filtered:  print(row)  min\_filtered = apply\_filter(input\_image, filter\_size, 'min')  print("\nMin Filtered Image:")  for row in min\_filtered:  print(row)    import numpy as np  from scipy.ndimage import median\_filter, maximum\_filter, minimum\_filter  import matplotlib.pyplot as plt  from PIL import Image  def apply\_median\_filter(image, size):      return median\_filter(image, size=size)  def apply\_max\_filter(image, size):      return maximum\_filter(image, size=size)  def apply\_min\_filter(image, size):      return minimum\_filter(image, size=size)  def apply\_midpoint\_filter(image, size):      min\_filtered = minimum\_filter(image, size=size)      max\_filtered = maximum\_filter(image, size=size)      return (min\_filtered + max\_filtered) / 2  def main():      image = np.array(Image.open('me.jpg').convert('L'))      filter\_size = 3      # Apply filters      median\_filtered = apply\_median\_filter(image, size=filter\_size)      max\_filtered = apply\_max\_filter(image, size=filter\_size)      min\_filtered = apply\_min\_filter(image, size=filter\_size)      midpoint\_filtered = apply\_midpoint\_filter(image, size=filter\_size)      # Display results      plt.figure(figsize=(10, 8))      plt.subplot(2, 3, 1)      plt.title("Original Image")      plt.imshow(image, cmap="gray")      plt.colorbar()      plt.subplot(2, 3, 2)      plt.title("Median Filtered")      plt.imshow(median\_filtered, cmap="gray")      plt.colorbar()      plt.subplot(2, 3, 3)      plt.title("Max Filtered")      plt.imshow(max\_filtered, cmap="gray")      plt.colorbar()      plt.subplot(2, 3, 4)      plt.title("Min Filtered")      plt.imshow(min\_filtered, cmap="gray")      plt.colorbar()      plt.subplot(2, 3, 5)      plt.title("Midpoint Filtered")      plt.imshow(midpoint\_filtered, cmap="gray")      plt.colorbar()      plt.tight\_layout()      plt.show()  if \_\_name\_\_ == "\_\_main\_\_":      main() |

2. Sharpening

2.1 Laplacian



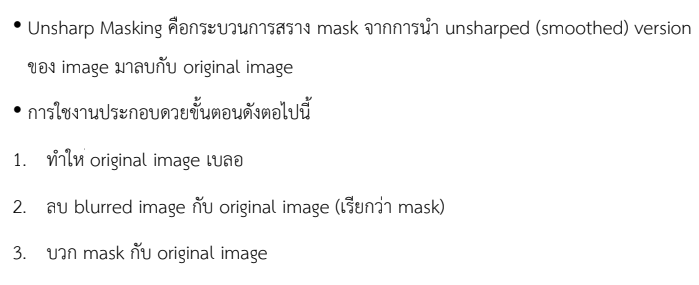


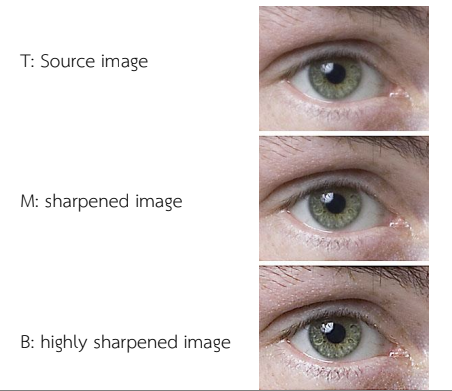
|  |
| --- |
| def apply\_laplacian\_sharpening(image):  kernel = [  [0, -1, 0],  [-1, 4, -1],  [0, -1, 0]  ]  rows, cols = len(image), len(image[0])  sharpened\_image = [[0 for \_ in range(cols)] for \_ in range(rows)]  def get\_pixel(x, y):  if x < 0 or x >= rows or y < 0 or y >= cols:  return 0  return image[x][y]  for x in range(rows):  for y in range(cols):  laplacian\_sum = 0  for i in range(-1, 2):  for j in range(-1, 2):  laplacian\_sum += kernel[i + 1][j + 1] \* get\_pixel(x + i, y + j)  sharpened\_image[x][y] = min(max(image[x][y] + laplacian\_sum, 0), 255)  return sharpened\_image  if \_\_name\_\_ == "\_\_main\_\_":  input\_image = [  [10, 20, 30, 40, 50],  [60, 70, 80, 90, 100],  [110, 120, 130, 140, 150],  [160, 170, 180, 190, 200],  [210, 220, 230, 240, 250]  ]  print("Original Image:")  for row in input\_image:  print(row)  sharpened\_image = apply\_laplacian\_sharpening(input\_image)  print("\nSharpened Image:")  for row in sharpened\_image:  print(row) |

#2 จงแก้ไขโค้ด และแคปภาพผลลัพธ์จากการทำ Laplacian โดยใช้ภาพตัวเองและเปลี่ยนแปลงขนาดของหน้ากากของ Laplacian ให้ครบทุกแบบ

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import cv2  def apply\_laplacian(image, kernel\_type):      # Convert image to grayscale      gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)        # Define different Laplacian kernels      kernels = {          'basic': np.array([[0, 1, 0],                            [1, -4, 1],                            [0, 1, 0]]),            'diagonal': np.array([[1, 1, 1],                              [1, -8, 1],                              [1, 1, 1]]),            'extended': np.array([[0, 0, -1, 0, 0],                              [0, -1, -2, -1, 0],                              [-1, -2, 16, -2, -1],                              [0, -1, -2, -1, 0],                              [0, 0, -1, 0, 0]])      }        # Apply custom kernel      laplacian = cv2.filter2D(gray, -1, kernels[kernel\_type])        # Normalize output      laplacian = cv2.normalize(laplacian, None, 0, 255, cv2.NORM\_MINMAX)        return laplacian.astype(np.uint8)  # Read your image  image = cv2.imread('me.jpg')  # Apply different Laplacian kernels  basic\_laplacian = apply\_laplacian(image, 'basic')  diagonal\_laplacian = apply\_laplacian(image, 'diagonal')  extended\_laplacian = apply\_laplacian(image, 'extended')  # Display results  plt.figure(figsize=(12, 8))  plt.subplot(221)  plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  plt.title('Original Image')  plt.axis('off')  plt.subplot(222)  plt.imshow(basic\_laplacian, cmap='gray')  plt.title('Basic Laplacian (3x3)')  plt.axis('off')  plt.subplot(223)  plt.imshow(diagonal\_laplacian, cmap='gray')  plt.title('Diagonal Laplacian (3x3)')  plt.axis('off')  plt.subplot(224)  plt.imshow(extended\_laplacian, cmap='gray')  plt.title('Extended Laplacian (5x5)')  plt.axis('off')  plt.tight\_layout()  plt.show() |

2.2 Unsharp masking และ Highboost Filtering





ตัวอย่างการทำ Unsharp masking และ Highboost Filtering

|  |
| --- |
| import numpy as np  def apply\_filter(image, kernel):  height, width = image.shape  kernel\_size = kernel.shape[0]  pad = kernel\_size // 2  padded\_image = np.pad(image, pad, mode='constant', constant\_values=0)  output = np.zeros\_like(image, dtype=np.float32)  # Convolution operation  for i in range(height):  for j in range(width):  region = padded\_image[i:i + kernel\_size, j:j + kernel\_size]  output[i, j] = np.sum(region \* kernel)    return output  def unsharp\_masking(image, kernel\_size=3, alpha=1.5):  kernel = np.ones((kernel\_size, kernel\_size)) / (kernel\_size \*\* 2)  blurred\_image = apply\_filter(image, kernel)  mask = image - blurred\_image  sharpened\_image = image + alpha \* mask    return sharpened\_image  def highboost\_filtering(image, kernel\_size=3, k=1.5):  kernel = np.ones((kernel\_size, kernel\_size)) / (kernel\_size \*\* 2)  blurred\_image = apply\_filter(image, kernel)  mask = image - blurred\_image  highboost\_image = image + k \* mask    return highboost\_image  if \_\_name\_\_ == "\_\_main\_\_":  image = np.array([  [50, 50, 50, 50, 50],  [50, 100, 100, 100, 50],  [50, 100, 150, 100, 50],  [50, 100, 100, 100, 50],  [50, 50, 50, 50, 50]  ], dtype=np.float32)  print("Original Image:")  print(image)  sharpened = unsharp\_masking(image, kernel\_size=3, alpha=1.5)  print("\nSharpened Image (Unsharp Masking):")  print(sharpened)  highboost = highboost\_filtering(image, kernel\_size=3, k=2.0)  print("\nHighboost Filtered Image:")  print(highboost) |

#3 จงหาตัวอย่างภาพที่นำมาทดสอบกับการทำ Unsharp masking และ Highboost Filteringของโค้ดตัวอย่างเพื่อให้เห็นผลลัพธ์ที่ชัดเจน และแคปภาพผลลัพธ์มาใส่ในคำตอบด้านล่าง [แก้ไขโค้ดให้แสดงภาพด้วย]

|  |
| --- |
| import numpy as np  import cv2  import matplotlib.pyplot as plt  def apply\_filter(image, kernel):      height, width = image.shape      kernel\_size = kernel.shape[0]      pad = kernel\_size // 2      padded\_image = np.pad(image, pad, mode='constant', constant\_values=0)      output = np.zeros\_like(image, dtype=np.float32)        for i in range(height):          for j in range(width):              region = padded\_image[i:i + kernel\_size, j:j + kernel\_size]              output[i, j] = np.sum(region \* kernel)        return output  def unsharp\_masking(image, kernel\_size=3, alpha=1.5):      kernel = np.ones((kernel\_size, kernel\_size)) / (kernel\_size \*\* 2)      blurred\_image = apply\_filter(image, kernel)      mask = image - blurred\_image      sharpened\_image = image + alpha \* mask      return np.clip(sharpened\_image, 0, 255)  def highboost\_filtering(image, kernel\_size=3, k=1.5):      kernel = np.ones((kernel\_size, kernel\_size)) / (kernel\_size \*\* 2)      blurred\_image = apply\_filter(image, kernel)      mask = image - blurred\_image      highboost\_image = image + k \* mask      return np.clip(highboost\_image, 0, 255)  # Read and process image  image = cv2.imread("eye.jpg", cv2.IMREAD\_GRAYSCALE)  image = cv2.resize(image, (400, 300))  # Apply filters  sharpened = unsharp\_masking(image, kernel\_size=3, alpha=1.5)  highboost = highboost\_filtering(image, kernel\_size=3, k=2.0)  # Display results  plt.figure(figsize=(15, 5))  plt.subplot(131)  plt.imshow(image, cmap='gray')  plt.title('Original Image')  plt.axis('off')  plt.subplot(132)  plt.imshow(sharpened, cmap='gray')  plt.title('Unsharp Masking')  plt.axis('off')  plt.subplot(133)  plt.imshow(highboost, cmap='gray')  plt.title('Highboost Filtering')  plt.axis('off')  plt.tight\_layout()  plt.show() |