**รายงาน**

**เรื่อง**

Computer Assignment 3
Wormhole counting

**โดย**

**นางสาววินิตรา แสงสร้อย  รหัส 640612097**

**เสนอ**

**รศ.ดร. ศันสนีย์  เอื้อพันธ์วิริยะกุล**

**รายงานนี้เป็นส่วนหนึ่งของวิชา**
**261453 การประมวลผลภาพเชิงเลข**
**(Digital Image Processing)**
**ภาควิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่**
**ปีการศึกษา 2566**

# สารบัญ

# วัตถุประสงค์

รายงานนี้ เป็นส่วนหนึ่งของวิชา 261453 การประมวลผลภาพเชิงเลข (Digital Image Processing) ซึ่งจัดทำเพื่อศึกษาการทำงานของการทำการแบ่งส่วนของภาพ (Image Segmentation) เพื่อนับวัตถุบนภาพ ซึ่งเป็นภาพที่มีความซับซ้อนสูง และจากการเรียนรู้รายงานฉบับนี้ จะสามารถนำองค์ความรู้นี้ไปประยุกต์ใช้กับการนับวัตถุบนภาพอื่นๆได้

# ทฤษฎีที่เกี่ยวข้อง

ในการนับจำนวนวัตถุจากภาพ ซึ่งในรายงานนี้จะเป็นการนับจำนวนรูหนอนสีดำจากมะเขือยาวสีเขียว โดยใช้หลายองค์ความรู้ประกอบกัน เช่น การทำ segmentation การ Filter ภาพ การหาขอบของภาพ และการทำ Morphological Operator

**การทำ Segmentation**

การแบ่งส่วนภาพเป็นกระบวนการแยกวัตถุที่สนใจออกจากพื้นหลัง ซึ่งเป็นขั้นตอนแรกในการวิเคราะห์ภาพ ในการทำ segmentation จะใช้การกำหนดค่า Threshold ซึ่งเป็นการแปลงภาพให้เป็น binary image โดยพิกเซลที่มีค่ามากกว่าหรือเท่ากับค่า Threshold จะกลายเป็นสีขาว (วัตถุ) และที่น้อยกว่าจะกลายเป็นสีดำ (พื้นหลัง)

**การกรองภาพ (Filtering)**

เพื่อปรับปรุงคุณภาพภาพโดยการลดสัญญาณรบกวน (noise) ในที่นี้จะใช้ Median filter ซึ่งทำงานโดยการจัดเรียงค่าความเข้มของพิกเซลในหน้าต่างที่กำหนด และกำหนดค่าความเข้มของพิกเซลกลางด้วยค่า median (ค่ากลาง)

**การหาขอบของภาพ (Edge Detection)**

เพื่อระบุตำแหน่งที่มีการเปลี่ยนแปลงความเข้มของพิกเซลอย่างรวดเร็วในภาพ ซึ่งมักบ่งบอกถึงขอบเขตของวัตถุ

**Morphological Operators**

ใช้กับภาพ binary image เพื่อเน้นหรือลบขนาดวัตถุ ซึ่งจะมีหลายๆเทคนิคที่ใช้ เช่น erosion ( การกัดกร่อน) dilation (การขยาย) opening (การเปิด) และ closing (การปิด) เพื่อที่จะเชื่อมหรือลบภาพที่ต้องการ

ในการทำงาน ได้ทำเป็น 2 วิธี

**วิธีที่ 1**

**ฟังก์ชัน custom_median_filter**

ใช้สำหรับการกรองภาพด้วยเทคนิค Median Filter ที่กำหนดค่าต่างๆเอง โดยขั้นตอนการทำงานคือ

1. สร้างภาพขนาดเดียวกับภาพป้อนเข้าและเตรียมให้ทุกค่าเป็น 0
2. ใช้ลูปสองชั้นสำหรับการวนรอบผ่านทุกพิกเซลในภาพป้อนเข้า
3. สร้าง window ขนาดเท่ากับ window_size x window_size โดยให้พิกเซลที่ต้องการฟิลเตอร์เป็นตัวกลางของหน้าต่าง
4. หาค่า Mean ของพิกเซลในหน้าต่างและกำหนดค่านั้นให้เป็นค่าของพิกเซลที่ต้องการฟิลเตอร์ในภาพผลลัพธ์

**ฟังก์ชัน segment_and_count_holes**

ฟังก์ชันนี้ใช้สำหรับการแยกวัตถุจากพื้นหลังและนับจำนวนรู (holes) ในวัตถุนั้นๆเพื่อประมวลผลภาพและนับจำนวนรูหรือช่องว่างในภาพ โดยฟังก์ชันจะคืนค่าเป็นภาพขาวดำที่ผ่านการแยกวัตถุแล้ว ภาพที่ผ่านการกรองเพื่อลด noise และจำนวนรูที่นับได้ในภาพ

มีขั้นตอนการทำงานคือ

1. แปลงภาพสีเป็นภาพขาวดำ (grayscale) โดยใช้ฟังก์ชัน mean(axis=2) เพื่อหาค่าเฉลี่ยของสีในทุกๆพิกเซล
2. นำภาพขาวดำที่ได้มาทำการส่วนแบ่ง (segmentation) โดยใช้ค่า threshold ที่คำนวณจากค่าเฉลี่ยของสีในภาพ
3. นำภาพที่ได้มาทำการกรองด้วยฟังก์ชัน custom_median_filter เพื่อลบ noise ออกจากภาพ
4. นับจำนวนรู (holes) ในวัตถุที่ได้จากการใช้ cv2.connectedComponentsWithStats เพื่อนับจำนวนของ object ที่เชื่อมโยงกันในภาพ

**การปรับประสิทธิภาพของฟังก์ชัน segment_and_count_holes**

โดยการทำ thresholding อีกครั้งด้วยค่าที่สูงขึ้นและการใช้ขนาดของหน้าต่างในการกรองภาพ (window size) ใหญ่ขึ้น เพื่อปรับปรุงการตรวจจับรู และสุดท้ายมีการแสดงผลภาพโดยใช้ OpenCV (cv2) เพื่อแสดงผลลัพธ์ที่ได้ในรูปแบบภาพสีจริง

**วิธีที่ 2**

**ฟังก์ชัน count_holes**

เป็นฟังก์ชันที่รับภาพเป็น Argument และทำการนับจำนวนรูสีดำในภาพดังนี้

1. แปลงเป็นภาพขาวดำ โดยใช้ฟังก์ชัน color.rgb2gray() เพื่อแปลงเป็นภาพขาวดำ
2. กลับสีภาพ โดยการหาค่าสูงสุดของพิกเซลแล้วลบค่าพิกเซลนั้นออกจากค่าสูงสุดนั้น เพื่อเน้นที่รูสีดำในภาพ
3. การแบ่งสีให้แตกออกเป็นภาพขาวดำ โค้ดจะใช้ thresholding แบบ Otsu เพื่อแบ่งสีให้แตกออกเป็นภาพขาวดำ เพื่อแยกแยะรูสีดำในภาพ
4. **Morphological Operators** เพื่อลบส่วนขยายของรู เพื่อลดขนาดของรูและตัดขอบของรู

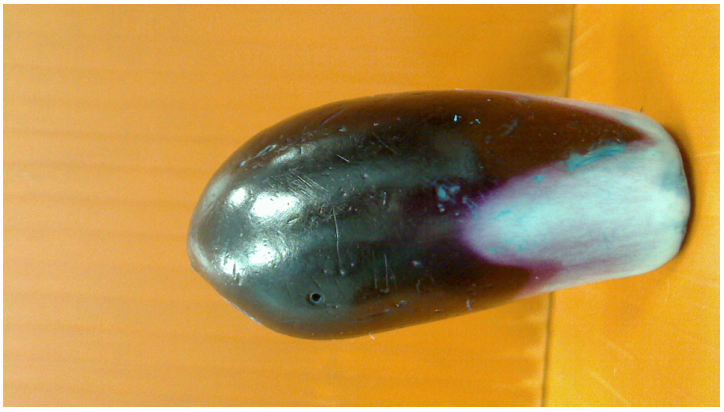
**การประมวลผลภาพและการนับรู**

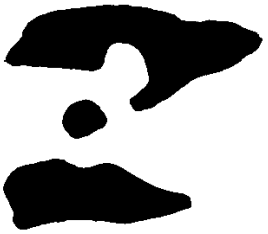
ใช้ฟังก์ชัน count_holes เพื่อประมวลผลภาพและนับจำนวนรูสีดำในแต่ละภาพ โดยคืนค่าเป็นจำนวนรูสีดำทั้งหมดและข้อมูลของรูแต่ละรู และจะแสดงผลลัพธ์การนับจำนวนรูสีดำบนภาพ โดยพล็อตภาพและเน้นแสดงตำแหน่งของรูด้วยจุดสีแดง

# ผลการทดลอง

วิธีที่ 1

WormHole_1H :

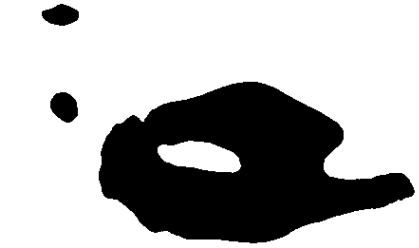




ผลลัพธ์

```
    image = imageio.imread('WormHole_1H.tif')
Number of refined holes detected: 1
```

WormHole_2H :



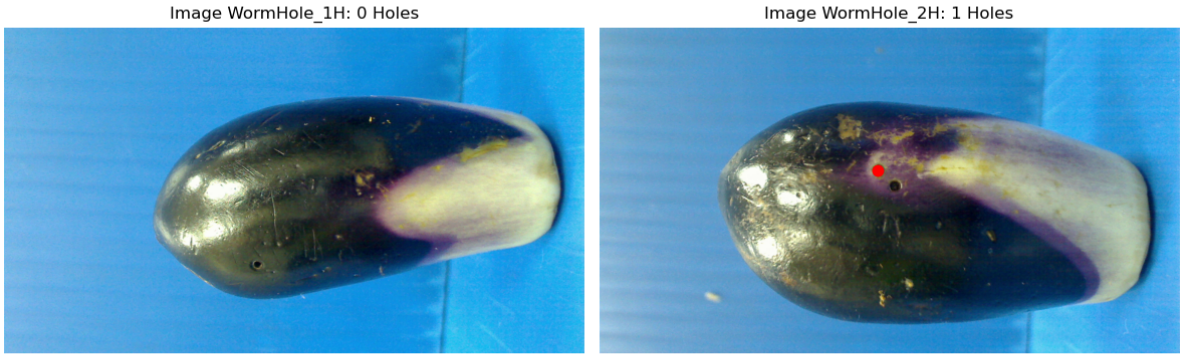
ผลลัพธ์

```
image = imageio.imread('WormHole_2H.tif')
Number of refined holes detected: 2
```

ซึ่งผลลัพธ์นั้น มีความตรงถูกต้องตามที่ควรจะเป็น แต่ก็ยังไม่อาจพิสูจน์ได้ว่าผลลัพธ์นั้นถูกต้องจริงๆหรือเปล่า ดังนั้น จึงได้ทำวิธีที่ 2 ใหม่ เพื่อที่จะชี้ให้เห็นว่าตรงนี้คือการ Detect รูหนอนจริงๆ

วิธีที่ 2

ผลลัพธ์




และจะเห็นว่าได้มีการ detect ส่วนที่เป็นรูหนอนจริงๆ แต่ยังติดปัญหาในด้านของการ implement code ให้สามารถตรวจสอบรูหนอนทั้งหมดได้ เนื่องจากภาพนั้นยังมีความซับซ้อน และเกิดการ Detect ในส่วนที่ไม่ใช่รูหนอนอยู่บ่อยครั้ง

# บทสรุป

การนับจำนวนรูหรือช่องว่างในภาพทั้งสองวิธี มีความแตกต่างกันในขั้นตอนการนับและการปรับปรุงความถูกต้องของผลลัพธ์ วิธีที่สองได้ทำการปรับปรุงและปรับให้มีประสิทธิภาพสูงขึ้นเพื่อให้ได้ผลลัพธ์ที่มีความถูกต้องและเชื่อถือได้มากขึ้น โดยสามารถปรับแต่งพารามิเตอร์และวิธีการในแต่ละขั้นตอนเพื่อให้เหมาะสมกับข้อมูลและความต้องการของผู้ใช้งานได้ การปรับปรุงนี้ช่วยเพิ่มความแม่นยำและความเชื่อถือในการนับจำนวนรูหรือช่องว่างในภาพได้อย่างมีประสิทธิภาพ

และในการทำรายงานฉบับนี้ ถึงแม้ว่าจะเกิดปัญหาที่ยังไม่สามารถแก้ได้ ทางผู้จัดทำก็ได้รับประสบการณ์และความรู้จำนวนมากในด้านของการทำ Image processing และสามารถนำความรู้ที่ได้ทั้งหมด ไปปรับใช้ในงานอื่นๆในอนาคตได้

# ภาคผนวก

Github : https://github.com/Donteatpineappleonpizza/DIPHW3_640612097

วิธีที่ 1

```
import imageio
import cv2
import numpy as np


def custom_median_filter(image, window_size):
 """Implements a custom median filter on a grayscale image.

 Args:
     image: A numpy array representing the grayscale image.
     window_size: The size of the median filter window (must be odd).

 Returns:
     A numpy array representing the filtered image.
 """
 filtered_image = np.zeros_like(image)
 height, width = image.shape

 for y in range(height):
   for x in range(width):
     window_y_min = max(0, y - window_size // 2)
     window_y_max = min(height, y + window_size // 2 + 1)
     window_x_min = max(0, x - window_size // 2)
     window_x_max = min(width, x + window_size // 2 + 1)

     window = image[window_y_min:window_y_max,
window_x_min:window_x_max]
     filtered_image[y, x] = np.median(window)

 return filtered_image


def segment_and_count_holes(image):
 """Segments the eggplant from the background and counts holes (dark
regions).
```

```
        Args:
            image: A numpy array representing the TIF image.

        Returns:
            A tuple containing the segmented eggplant image and the number
of holes.
        """
        # Convert the image to grayscale
        grayscale_image = image.mean(axis=2)

        # Apply thresholding to segment the eggplant
        threshold = np.mean(grayscale_image) / 2
        segmented_image = (grayscale_image > threshold).astype(np.uint8) *
255

        # Apply custom median filter to remove noise
        filtered_image = custom_median_filter(segmented_image, 20)

        # Count the number of holes (connected black regions)
        _, labels, stats, _ =
cv2.connectedComponentsWithStats(filtered_image, connectivity=4)
        holes = np.sum(stats[1:, cv2.CC_STAT_AREA] > 20)

        return grayscale_image, segmented_image, holes


      # Read the TIF image
      image = imageio.imread('WormHole_1H.tif')

      grayscale_image, segmented_image, hole_count =
segment_and_count_holes(image.copy())

      # Refine segmentation and hole detection
      more_selective_threshold = 0.3 * np.mean(grayscale_image)
      refined_segmented_image = (grayscale_image >
more_selective_threshold).astype(np.uint8) * 255
      filtered_image = custom_median_filter(refined_segmented_image, 30)  #
Slightly larger window
      _, labels, stats, _ =
cv2.connectedComponentsWithStats(filtered_image, connectivity=4)
```

```
    holes = np.sum(stats[1:, cv2.CC_STAT_AREA] > 25)  # Increased area
threshold

    print(f"Number of refined holes detected: {holes}")
    # Visualize results (optional)
    cv2.imshow('Original Image', image)
    cv2.imshow('Refined Segmented Image', refined_segmented_image)
    cv2.imshow('Filtered Image', filtered_image)
    cv2.waitKey(0)
```

วิธีที่ 2

```
from skimage import io, color, morphology, measure, filters
import numpy as np
import matplotlib.pyplot as plt

# Load images
image_1h = io.imread('WormHole_1H.tif')
image_2h = io.imread('WormHole_2H.tif')

# Define the hole counting function using previously defined logic
def count_holes(image):
   # Convert to grayscale
   gray_image = color.rgb2gray(image)

   # Invert the image to highlight the holes
   inverted_image = np.max(gray_image) - gray_image

   # Threshold the image to isolate the holes
   thresh = filters.threshold_otsu(inverted_image)
   binary = inverted_image > thresh

   # Perform morphological operations to clean up the image
   opened = morphology.binary_opening(binary, morphology.disk(3))
   closed = morphology.binary_closing(opened, morphology.disk(1))

   # Label the image to find connected regions
   labeled = measure.label(closed)
   properties = measure.regionprops(labeled)
```

```
    # Define a method to calculate circularity
    def circularity(region):
        return (4 * np.pi * region.area) / (region.perimeter ** 2)

    # Define criteria for what you consider a hole
    def is_hole(region):
        return (circularity(region) > 0.8 and region.area >= 100)

    # Filter regions based on the defined criteria
    holes = [prop for prop in properties if is_hole(prop)]

    # Return the number of holes
    return len(holes), holes

# Process images and count holes
num_holes_1h, holes_1h = count_holes(image_1h)
num_holes_2h, holes_2h = count_holes(image_2h)

# Plot the results for both images
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Display image 1H with marked holes
ax[0].imshow(image_1h, cmap='gray')
for prop in holes_1h:
    y0, x0 = prop.centroid
    ax[0].plot(x0, y0, '.r', markersize=15)
ax[0].set_title(f'Image WormHole_1H: {num_holes_1h} Holes')
ax[0].axis('off')

# Display image 2H with marked holes
ax[1].imshow(image_2h, cmap='gray')
for prop in holes_2h:
    y0, x0 = prop.centroid
    ax[1].plot(x0, y0, '.r', markersize=15)
ax[1].set_title(f'Image WormHole_2H: {num_holes_2h} Holes')
ax[1].axis('off')

plt.tight_layout()
plt.show()

(num_holes_1h, num_holes_2h)
```