**2110431 Introduction to Digital Imaging**

**2147329 Digital Image Processing and Vision Systems**

**Homework #3**

**Deadline: November 21, 2023 @23:59**

**Submissions: (1) PDF version of this file ONLY problem 1 and 3 will be graded.**

**Submissions: (1) PDF version of this file**

**(2) .ipynb file; template in this link**

**All images are in the hw3 folder.**

**IMPORTANT! (1) Before submitting the python file, please make sure it can be successfully compiled and correctly in its format name**

**(2) The scores will be 0 for all students whose source codes are very similar to each other.**

1. (10 points) Reading a (very) simple clock

Use image processing to read a simple clock provided below and write a program using python library to provide output in the format displayed “HH:MM”, such as “04:00” for the most left clock, “10:00” for the second clock, and so on. (HH in the range [01,12], MM in [00,59])

|  |  |  |  |
| --- | --- | --- | --- |
| Icon  Description automatically generated | Icon  Description automatically generated | Icon  Description automatically generated | Icon  Description automatically generated |
| 04:00 | 10:00 | 09:00 | 09:10 |

**Note:** your algorithm does not have to be 100% accurate; you should explain your results.

1.1) Describe steps of your algorithm

|  |  |
| --- | --- |
| **Steps** | **Description and purposes** |
| 1 | Convert RGB image to gray for computer to easy understand |
| 2 | Make circle mask.   * Black circle with r=140 to remove outer circle of clock * White circle with r=15 to remove black circle at center |
| 3 | Threshold the image to make binary image |
| 4 | Dilate the black area to make clock hand thinner |
| 5 | Apply the created mask |
| 6 | Apply canny to detect edge |
| 7 | Using Hough Line Transform to detect clock hand |
| 8 | If the clock hand is thick, we need to combine the two line that detected |
| 9 | Getting x2, y2 that return from Hough Line Transform and calculate the angle |
| 10 | Convert angle to time |
| 11 | Show the result |

1.2) Write down the results from your program:

|  |  |  |  |
| --- | --- | --- | --- |
| Icon  Description automatically generated | Icon  Description automatically generated | Icon  Description automatically generated | Icon  Description automatically generated |
| 04:00 | 10:00 | 09:00 | 09:10 |

1.3) Analyze the results.

*Hint: in terms of how accurate is your technique, any further improvement can be done?*

My result is 100% accurate.

A screenshot of a clock

Description automatically generatedA screenshot of a clock

Description automatically generatedA screenshot of a clock

Description automatically generatedA screenshot of a clock

Description automatically generated

import cv2

import numpy as np

import matplotlib.pyplot as plt

import math

def detect\_hands\_and\_calculate\_time(image\_path):

# Load the image

img = cv2.imread(image\_path)

# Convert to gray

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Make circle mask

m,n,\_ = img.shape

m,n = m//2,n//2

mask1,mask2 = np.ones\_like(gray),np.ones\_like(gray)

cv2.circle(mask1,(m,n),140,0,-1)

cv2.circle(mask1,(m,n),15,1,-1)

# Threshold

\_,gray = cv2.threshold(gray,128,1,cv2.THRESH\_BINARY)

# Dilate the white area to make clock hand thinner

kernel = np.ones((5, 5), np.uint8)

gray = cv2.dilate(gray, kernel, iterations=1)

# Apply mask

gray = cv2.bitwise\_or(mask1,gray)

# Apply Canny to detect the edge

edges = cv2.Canny(gray, 0, 1, apertureSize=3)

# Hough Line Transform

lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=40, minLineLength=30, maxLineGap=5)

# Assuming the longest line is the minute hand and second longest is the hour hand

if lines is not None:

lines = sorted(lines, key=lambda x: np.linalg.norm(x[0][:2] - x[0][2:4]), reverse=True)

minute\_hand = lines[0][0]

if len(lines) > 3:

hour\_hand = np.array([(lines[2][0][0] + lines[3][0][0])/2, (lines[2][0][1] + lines[3][0][1])/2, (lines[2][0][2] + lines[3][0][2])/2, (lines[2][0][3] + lines[3][0][3])/2])

elif len(lines) > 2:

hour\_hand = lines[len(lines) - 1][0]

else:

hour\_hand = None

# Calculate angles and time

center = (img.shape[0] // 2, img.shape[1] // 2)

minute\_angle = calculate\_angle(minute\_hand, center)

hour\_angle = calculate\_angle(hour\_hand, center) if hour\_hand is not None else None

minute = math.floor(minute\_angle / 360 \* 60) % 60

hour = math.ceil(hour\_angle / 360 \* 12) if hour\_angle is not None else None

for line in lines:

for x1,y1,x2,y2 in line:

cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

plt.imshow(img)

plt.show()

return hour, minute

else:

plt.imshow(img)

plt.show()

return None, None

def calculate\_angle(line, center):

x1, y1, x2, y2 = line

x = x2 - 150

y = y2 - 152

angle = math.atan( abs(x) / abs(y) )

angle = np.degrees(angle)

if x < 0 and y >= 0:

angle = 180 + angle

elif x < 0 and y < 0:

angle = 360 - angle

elif x >= 0 and y >= 0:

angle = 180 - angle

return angle

hour, minute = detect\_hands\_and\_calculate\_time('clock0900.png')

if hour is not None and minute is not None:

print(f"Detected time: {hour}:{minute:02d}")

else:

print("Could not detect clock hands")

2. (Optional – for practice) Pyricularia Oryzae, rice blast fungus can cause rice blast disease. To identify the possibility of the occurrence of rice blast disease, the density of the spores of Pyricularia Oryzae can be calculated. Plant pathologist knows that you studied image processing, so they have asked you to help them automatically count the number of spores using image processing. They have provided two image samples below for you to develop an algorithm to count them. You should provide your results in terms of num\_count and resulted\_image (labeled count) (you can use cv2.rectangle(…) and cv2.putText(…) functions) as the example shown below

**Note:** your algorithm does not have to be 100% accurate; you should explain your results.

|  |  |
| --- | --- |
| **Original image** | **Your results / number of counted spores** |
| A picture containing outdoor  Description automatically generated  pyri02.png | **EXAMPLE**  **A picture containing text, several  Description automatically generated**  **num\_count = 12** |

2.1) Describe steps of your algorithm

|  |  |
| --- | --- |
| **Steps** | **Description and purposes** |
| 1 |  |
| 2 |  |
| 3 |  |
|  |  |
|  |  |

2.2) Results

|  |  |
| --- | --- |
| **Original image** | **Your results / number of counted spores** |
| A picture containing outdoor  Description automatically generated  pyri02.png |  |
| A picture containing outdoor, flying, nature  Description automatically generated  pyri02.png |  |

2.3) Analyze the results.

*Hint: in terms of how accurate is your technique, any further improvement can be done?*

3. (10 points) Separate and segment the oil in the beaker by distinguishing between solid (darker) and liquid oil. The container has a width and height, as shown in Figure 3.1. The equation for volume is , where represents the radius and is the height of the beaker, respectively.

|  |
| --- |
|  |
| Fig 3.1 Crude oil (Oil.png) |

3.1 Find the volume of the oil in the liquid state.

Please use image enhancement, such as, Log transform, Power Law before apply segmentation. Then, you can use Otsu’s, Adaptive Thresholding, Region Growing, and Manual Threshold to find the volume. Put your image results in the blank areas below.

**Optional Enhancement image**

|  |
| --- |
|  |
| Enhanced image |

|  |
| --- |
| A cup with brown liquid and a straw  Description automatically generated |
| Segmented regions of the liquid oil and show width and height of the segmented image |

Explain your steps and techniques used briefly:

|  |
| --- |
| 1. Use Power Law transform to make image brighter and easy to thresholding. 2. Use manual threshold to threshold the image. 3. Use find contour and get the second largest contour. 4. Use bounding rectangle to bound the largest contour. 5. Get x, y, w, h of bounded rectangle and calculate the volume.   import cv2  import numpy as np  import matplotlib.pyplot as plt  # Load the image  image = cv2.imread('Oil.png', cv2.IMREAD\_COLOR)  image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # Apply log transform for image enhancement  img\_log = (np.log(image + 1)/(np.log(1 + np.max(image)))) \* 255  img\_log = np.array(img\_log, dtype=np.uint8)  # Apply gamma correction for power law transform  gamma = 0.5  img\_gamma = np.array(255\*(image / 255) \*\* gamma, dtype='uint8')  # Display the images  fig, ax = plt.subplots(1, 3, figsize=(20, 10))  ax[0].imshow(image)  ax[0].set\_title('Original Image')  ax[0].axis('off')  ax[1].imshow(img\_log)  ax[1].set\_title('Log Transform')  ax[1].axis('off')  ax[2].imshow(img\_gamma)  ax[2].set\_title('Power Law Transform')  ax[2].axis('off')  plt.show()  from skimage.filters import threshold\_otsu  from skimage import measure  import cv2  import numpy as np  import matplotlib.pyplot as plt  # Using power-law image and convert to grayscale  gray\_power\_law = cv2.cvtColor(img\_gamma, cv2.COLOR\_RGB2GRAY)  # Apply Otsu's thresholding  thresh\_val\_otsu = threshold\_otsu(gray\_power\_law)  otsu\_mask = gray\_power\_law > thresh\_val\_otsu  # Apply Adaptive Thresholding  adaptive\_thresh = cv2.adaptiveThreshold(gray\_power\_law, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)  # Manual Thresholding - we select a threshold value that might be suitable  manual\_thresh\_value = 100  \_, manual\_mask = cv2.threshold(gray\_power\_law, manual\_thresh\_value, 255, cv2.THRESH\_BINARY)  # Display the segmentation results  fig, ax = plt.subplots(1, 3, figsize=(20, 10))  ax[0].imshow(otsu\_mask, cmap='gray')  ax[0].set\_title('Otsu’s Thresholding')  ax[0].axis('off')  ax[1].imshow(adaptive\_thresh, cmap='gray')  ax[1].set\_title('Adaptive Thresholding')  ax[1].axis('off')  ax[2].imshow(manual\_mask, cmap='gray')  ax[2].set\_title('Manual Thresholding')  ax[2].axis('off')  plt.show()  def segment\_dark\_area(image\_binary, image\_rgb):  # Find the contours of the dark area.  contours, \_ = cv2.findContours(image\_binary, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  areaArray = []  # print(contours)  for i, c in enumerate(contours):  area = cv2.contourArea(c)  areaArray.append(area)  sorteddata = sorted(zip(areaArray, contours), key=lambda x:x[0], reverse=True)  # Find the largest contour.  largest\_contour = sorteddata[1][1]  # Draw a rectangle around the largest contour.  x, y, w, h = cv2.boundingRect(largest\_contour)  cv2.rectangle(image\_rgb, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Calculate the height of the rectangle.  height = h  width = w  return image\_rgb, height, width  def volumeCal(height\_pixel,width\_pixel):  cm\_per\_pix = 18 / width\_pixel  width\_cm = 18  height\_cm = cm\_per\_pix \* height\_pixel  volumn = np.pi \* (width\_cm/2)\*\*2 \* height\_cm  return volumn, width\_cm, height\_cm  img\_con = np.copy(img\_gamma)  im, height\_pixel, width\_pixel = segment\_dark\_area(manual\_mask,img\_con)  plt.imshow(im)  plt.show()  print(f"The volumn is {volumeCal(height\_pixel,width\_pixel)[0]:.2f} cm3")  print(f"Width is {volumeCal(height\_pixel,width\_pixel)[1]:.2f} cm")  print(f"Height is {volumeCal(height\_pixel,width\_pixel)[2]:.2f} cm") |

3.2 Segment the liquid oil again using Connected-component-with-stats method and compare the segmented result and calculated volume with 3.1.

*Hint: Don’t forget to use image Enhancement and connectivity either 4 or 8*

|  |
| --- |
| A glass with brown liquid and a straw  Description automatically generated |
| Segmented region of the liquid oil using Connected Component with stats method. |

Explain your steps and techniques used briefly:

|  |
| --- |
| 1. Convert the image to gray. 2. Thresholding the image. 3. Find the connected components with stats. 4. Sort the stats to find the second largest component. 5. Get w, y, w, h of the second largest component. 6. Calculate the volume.   img\_concom\_rgb = cv2.imread('Oil.png', cv2.IMREAD\_COLOR)  img\_concom = cv2.cvtColor(img\_concom\_rgb, cv2.COLOR\_BGR2GRAY)  ret,thresh1 = cv2.threshold(img\_concom,28,255,cv2.THRESH\_BINARY)  # Connectivity type  connectivity = 4  # Get Connected Components With Stats  output = cv2.connectedComponentsWithStats(thresh1, connectivity, cv2.CV\_32S)  # The the labels matrix  labels = output[1]  # The the stat matrix  stats = output[2]  # Sort the component to get second largest component  sortedstat = sorted(stats, key=lambda x: x[4], reverse=True)  # Get w, y, w, h of the second largest component  x,y,w,h,\_ = sortedstat[1]  # Draw the rectangle  cv2.rectangle(img\_concom\_rgb, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Show lables image  plt.imshow(labels, cmap="seismic")  plt.show()  # Show result image  plt.imshow(cv2.cvtColor(img\_concom\_rgb, cv2.COLOR\_BGR2RGB))  plt.show()  print(f"The volume is {volumeCal(h,w)[0]:.2f} cm3")  print(f"Width is {volumeCal(h,w)[1]:.2f} cm")  print(f"Height is {volumeCal(h,w)[2]:.2f} cm") |
| From 3.1: The calculated height is 22.58 cm., and the volume is 5745.78 cm3.  From 3.2: The calculated height is 22.00 cm., and the volume is 5598.32 cm3.  So, the different between calculated height is less than 1 cm. I think these 2 results are very similar. |

Note: You will get full score if the calculated volume for both 3.1 and 3.2 are within 10% error from our reference volume.