**COMSATS University Islamabad, Lahore Campus**

**Fall 2024 – Assignment 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Course Title: | Computer Vision | Course Code: | | CSC455 | Credit Hours: | 3(3,0) |
| Course Instructor/s: | Dr. Zulfiqar Habib, Professor | Programme Name: | | BS Computer Science | | |
| Topic | Fundamentals of Computer Imaging/Vision  Section: | | | Max Marks: | 10 | |
| Out Date: | 30-09-24 | **Due Date:** | | | **7-10-24** | |
| Student’s Name: | **SHAFIN-UZ-ZAMAN** | Reg. No. | SP22-BCS-063 | | | |
| **Instructions:**   1. You may use AI tools to help understand the concepts. 2. However, the answers must be in your own words and show your understanding of the topics. 3. Copying answers directly from any source, including AI tools, is not allowed. 4. Your assignment will be evaluated & graded through a leading Quiz   **Submission Guidelines:**  Submit your assignment on this sheet via Google Classroom. | | | | | | |

**Problem Statement:**

Take a picture of five transparent bottles, where some are partially filled with a coloured drink and others are fully filled, using your own camera. Then, write a program that identifies, counts, and labels the partially filled bottles.

**Bonus:** If you can calculate the exact percentage fill for each bottle and label it, you will receive extra credit.

**Program Code**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.filters import threshold\_multiotsu

# Convert the input image to grayscale

def convert\_to\_grayscale(image):

    return cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Display an image using matplotlib; if it's grayscale, show in gray colormap

def show\_image(image, is\_gray=False):

    plt.figure(figsize=(6, 6))

    if is\_gray:

        plt.imshow(image, cmap='gray')

    else:

        plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

    plt.axis('off')  # Turn off axis for a cleaner display

    plt.show()

# Sharpen the image by applying a kernel that enhances edges

def sharpen\_image(image):

    # Define sharpening kernel to enhance edges

    sharpening\_kernel = np.array([[0, -1, 0],

                                  [-1, 5, -1],

                                  [0, -1, 0]])

    return cv2.filter2D(image, -1, sharpening\_kernel)

# Apply multi-level Otsu thresholding to segment the image into three classes

def apply\_multi\_otsu\_threshold(image):

    # Get the thresholds using multi-Otsu for three classes

    thresholds = threshold\_multiotsu(image, classes=3)

    t1, t2 = thresholds  # Two thresholds for separating three regions

    # Create binary masks based on the thresholds

    bL = np.where(image > t1, 1, 0)  # Lower threshold for the lighter region (liquid)

    bH = np.where(image > t2, 1, 0)  # Higher threshold for darker regions

    # Subtract the high and low regions to isolate the middle region (liquid)

    liquid\_mask = bL - bH

    return (liquid\_mask \* 255).astype(np.uint8)  # Return as binary mask

# Apply morphological closing to remove small gaps in the image

def apply\_morphology(image):

    kernel = np.ones((5, 5), np.uint8)  # Define kernel for morphology

    return cv2.morphologyEx(image, cv2.MORPH\_CLOSE, kernel)  # Morphological closing

# Further refine contours using erosion and dilation for better shape detection

def refine\_contours(mask):

    # Step 1: Apply closing to fill small gaps in the mask

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

    closed\_mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

    # Step 2: Apply erosion to smooth the edges of the contours

    refined\_mask = cv2.erode(closed\_mask, kernel, iterations=2)

    return refined\_mask

# Detect and label glasses based on their liquid height percentage

def detect\_and\_label\_glasses(original\_image, liquid\_mask, glass\_height):

    # Find contours of liquid in the binary mask

    contours, \_ = cv2.findContours(liquid\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    glass\_count = 0  # Initialize glass count

    # Loop over each contour representing a glass

    for contour in contours:

        area = cv2.contourArea(contour)

        # Ignore small contours that may be noise

        if area < 1000:

            continue

        # Get bounding box of the detected glass contour

        x, y, w, h = cv2.boundingRect(contour)

        # Extract the region of interest within the bounding box

        glass\_region = liquid\_mask[y:y+h, x:x + w]

        # Find the highest and lowest points where liquid is detected

        non\_zero\_rows = np.any(glass\_region > 0, axis=1)

        if np.any(non\_zero\_rows):

            highest\_liquid\_level = np.argmax(non\_zero\_rows)  # First row with liquid

            lowest\_liquid\_level = len(non\_zero\_rows) - np.argmax(non\_zero\_rows[::-1])  # Last row with liquid

            liquid\_height = lowest\_liquid\_level - highest\_liquid\_level  # Calculate height of the liquid

        else:

            liquid\_height = 0  # If no liquid is detected

        # Calculate the fill percentage based on liquid height relative to glass height

        fill\_percentage = (liquid\_height / glass\_height) \* 100

        # Label the glass based on fill percentage

        if fill\_percentage >= 90:

            label = f"Fully Filled: {fill\_percentage:.1f}%"

        elif fill\_percentage == 0:

            label = "Empty"

        else:

            label = f"Partially Filled: {fill\_percentage:.1f}%"

        glass\_count += 1  # Increment glass count

        # Draw bounding box and label on the original image

        cv2.rectangle(original\_image, (x, y), (x + w, y + h), (0, 255, 0), 2)

        cv2.putText(original\_image, f"{label}", (x, y - 10),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2)

    # Display the final labeled image

    show\_image(original\_image)

    # Output the total number of glasses detected

    print(f"Total liquid Detected glasses : {glass\_count}")

# Get the average height of the glass based on contours from thresholded image

def get\_avg\_glass\_height(grayscale\_image):

    # Apply multi-Otsu threshold to detect glass regions

    thresholds = threshold\_multiotsu(grayscale\_image, classes=3)

    t1, t2 = thresholds

    glass\_mask = np.where(grayscale\_image > t1, 1, 0)

    glass\_mask = (glass\_mask \* 255).astype(np.uint8)

    # Find contours of glasses in the image

    glass\_contours, \_ = cv2.findContours(glass\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    total\_height = 0  # Sum of glass heights

    glass\_count = 0  # Count of detected glasses

    # Loop over each detected glass contour

    for contour in glass\_contours:

        area = cv2.contourArea(contour)

        if area < 1000:  # Filter out small, irrelevant contours

            continue

        x, y, w, glass\_height = cv2.boundingRect(contour)

        total\_height += glass\_height

        glass\_count += 1

    # Calculate the average height of the glasses

    if glass\_count == 0:  # Handle case where no glass is detected

        return 0

    avg\_glass\_height = total\_height / glass\_count

    print(f"Total Glasses Detected: {glass\_count}")

    print(f"Average Glass Height: {avg\_glass\_height} pixels")

    return avg\_glass\_height

# Main processing function to handle the image processing pipeline

def process\_image(image\_path):

    # Step 1: Load the image

    image = cv2.imread(image\_path)

    # Step 2: Convert the image to grayscale

    grayscale\_image = convert\_to\_grayscale(image)

    show\_image(grayscale\_image, True)

    # Step 3: Sharpen the grayscale image to enhance features

    sharpened\_image = sharpen\_image(grayscale\_image)

    show\_image(sharpened\_image, True)

    # Step 4: Apply Gaussian blur to smooth the image and reduce noise

    blurred = cv2.GaussianBlur(sharpened\_image, (3, 3), 0)

    show\_image(blurred, True)

    # Step 5: Use multi-Otsu thresholding to segment the liquid in the image

    liquid\_mask = apply\_multi\_otsu\_threshold(blurred)

    show\_image(liquid\_mask, True)

    # Step 6: Refine contours by applying further morphological operations

    refined\_mask = refine\_contours(liquid\_mask)

    show\_image(refined\_mask, True)

    # Step 7: Detect the average glass height from the image

    glass\_height = int(get\_avg\_glass\_height(grayscale\_image)) - 10 # Subtract a small buffer

    # Step 8: Detect and label glasses based on their liquid fill level

    detect\_and\_label\_glasses(image, refined\_mask, glass\_height)

# Run the image processing pipeline on the provided image file

process\_image('main4.png')

**Input**



**Output**

**Grayscale:**

**A group of glasses with liquid

Description automatically generated**

**Sharpen img:**

**A group of glasses with liquid in them

Description automatically generated**

**Blur img (For noice reduction):**

**A group of glasses with liquid

Description automatically generated**

**Thresholding (to detect glasses):**

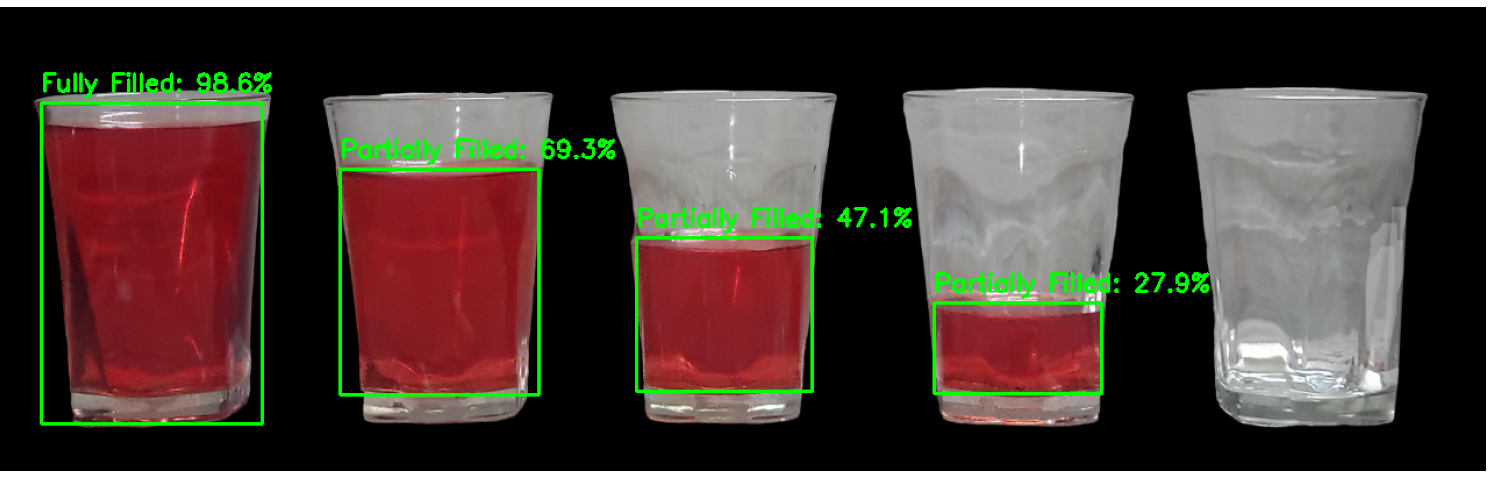
**A row of glasses with white liquid

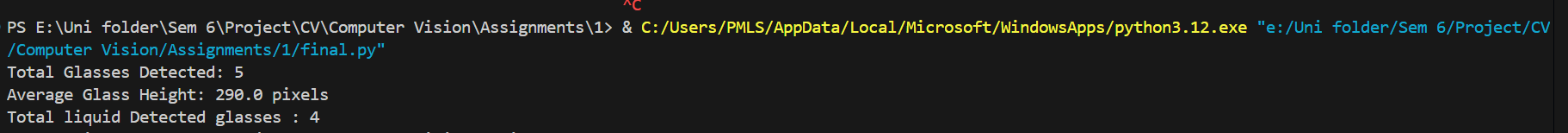
Description automatically generated**

**Morphological operation (to detect liquid):**

****

**Final result:**

****

****