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

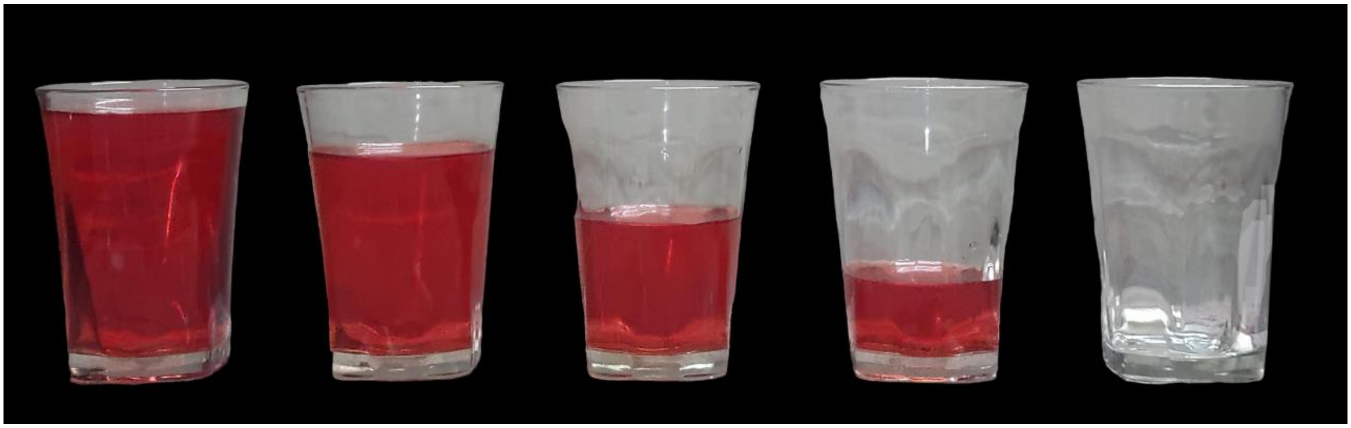
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.filters import threshold_multiotsu

# 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
```

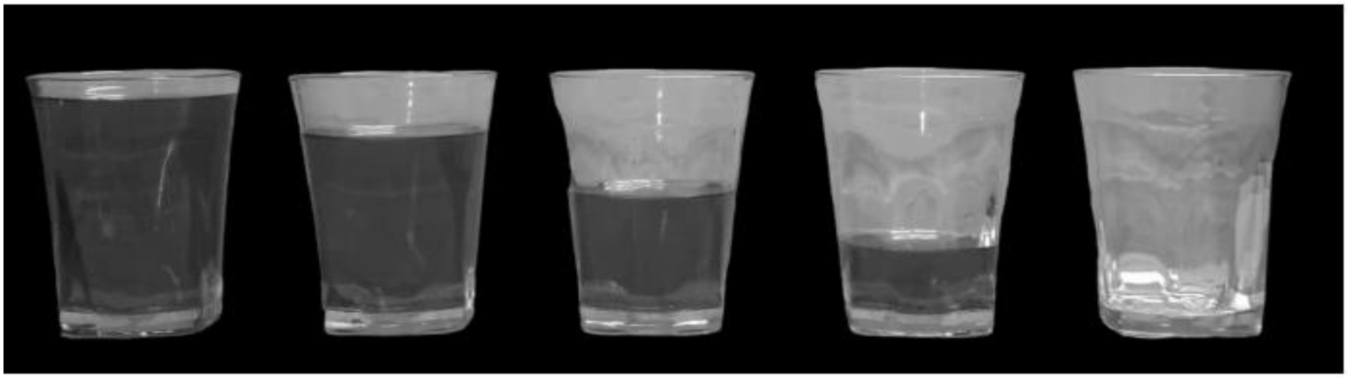
```
    # Step 8: Detect and label glasses based on their liquid fill level
    detect_and_label_glasses(image, refined_mask, glass_height)

# Run the image processing pipeline on the provided image file
process_image('main4.png')
```

Input

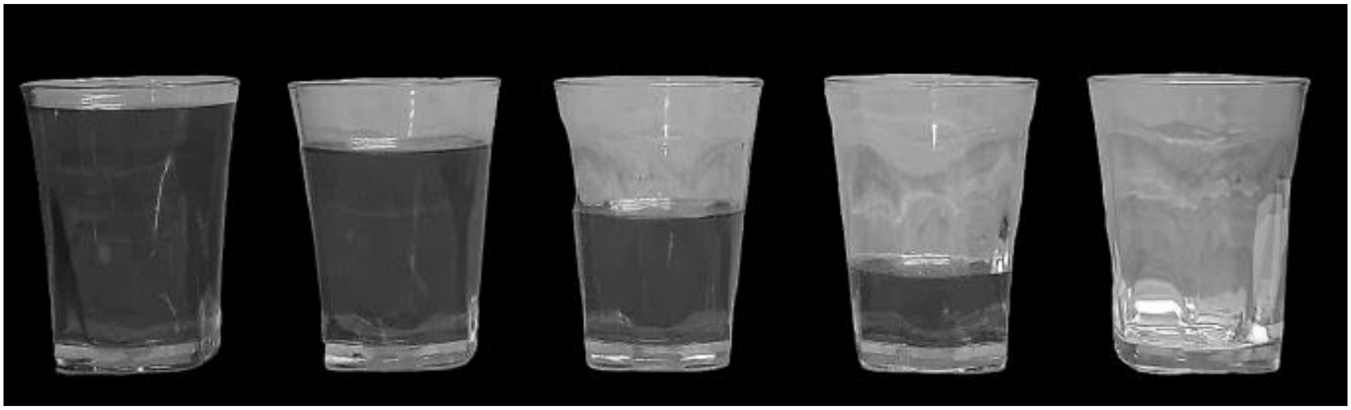


Output

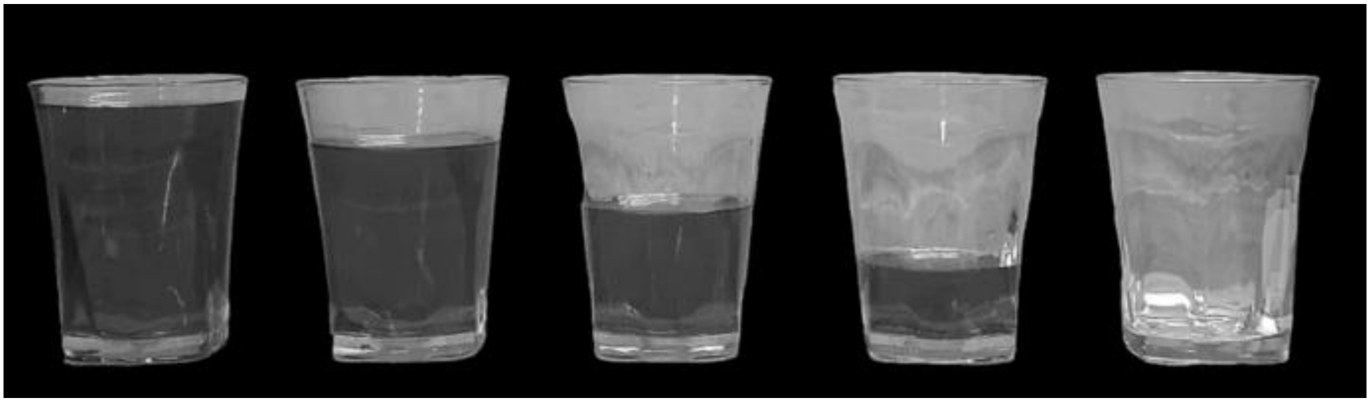
Grayscale:



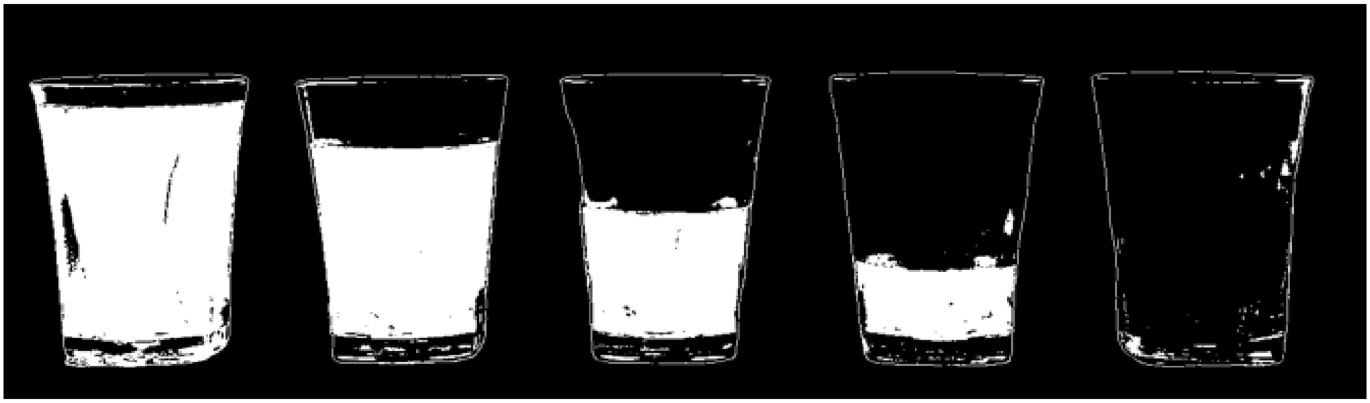
Sharpen img:



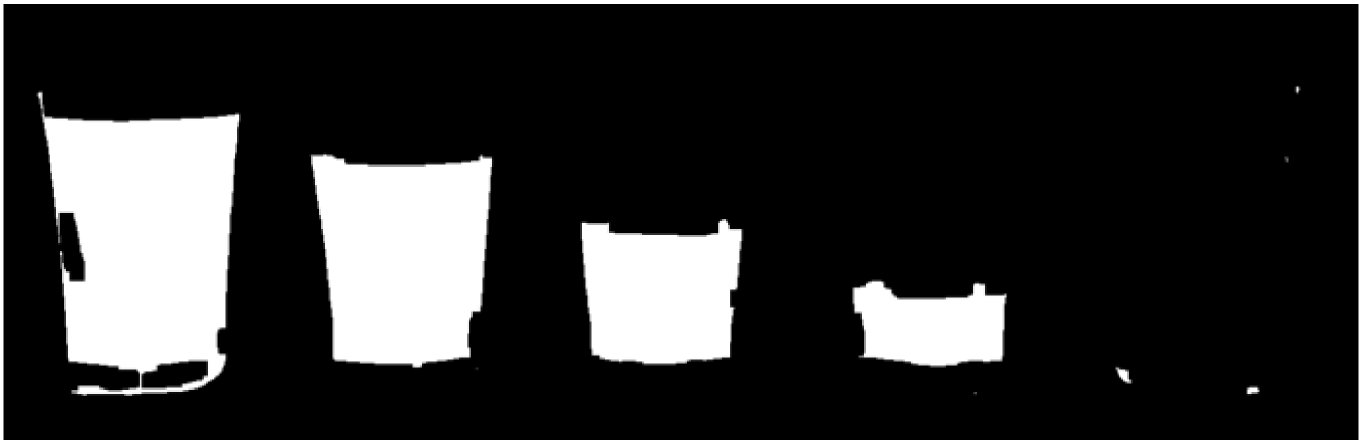
Blur img (For noice reduction):



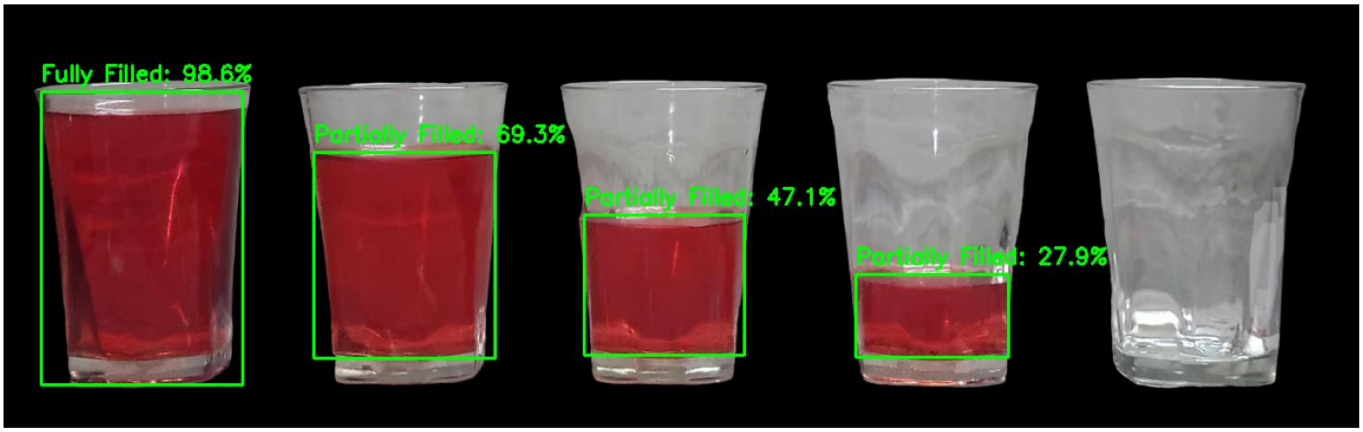
Thresholding (to detect glasses):

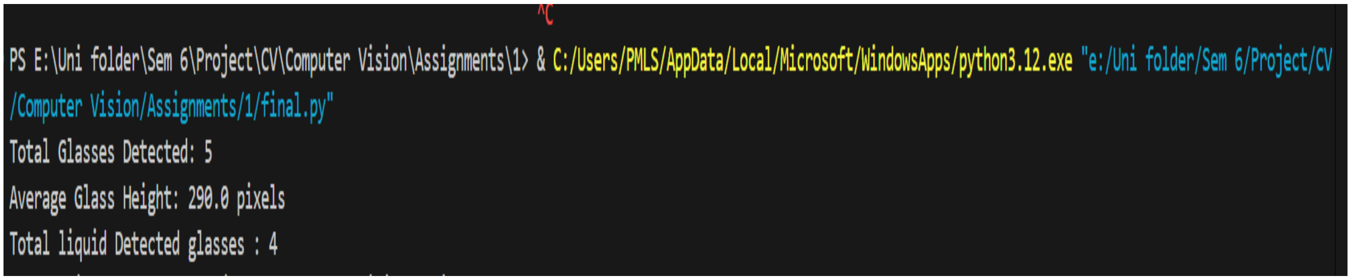


Morphological operation (to detect liquid):



Final result:





```
^C
PS E:\Uni folder\Sem 6\Project\CV\Computer Vision\Assignments\1> & C:/Users/PMLS/AppData/Local/Microsoft/WindowsApps/python3.12.exe "e:/Uni folder/Sem 6/Project/CV
/Computer Vision/Assignments/1/final.py"
Total Glasses Detected: 5
Average Glass Height: 290.0 pixels
Total liquid Detected glasses : 4
```