AI VISIONS LAB

7(A)

```
def decimal_to_binary(decimal_number):
    # Convert to integer before converting to binary
    binary_number = bin(int(decimal_number))[2:]  # Remove the '0b' prefix
    return binary_number.zfill(10)

def convert_and_display():
    decimal_number = float(input("Enter a decimal number (between 0 and 1023): "))
    binary_number = decimal_to_binary(decimal_number)
    print(f"The binary representation of {decimal_number} is: {binary_number}")

# Example usage
convert_and_display()
```

9(B)

```
import cv2
import numpy as np

def box_filter(img, kernel_size):
    height, width = img.shape
    output = np.zeros((height, width), dtype=np.uint8)

    for i in range(height - kernel_size + 1):
        for j in range(width - kernel_size + 1):
            output[i, j] = np.mean(img[i:i+kernel_size, j:j+kernel_size])

    return output

def median_filter(img, kernel_size):
    height, width = img.shape
    output = np.zeros((height, width), dtype=np.uint8)

    for i in range(height - kernel_size + 1):
        for j in range(width - kernel_size + 1):
            output[i, j] = np.median(img[i:i+kernel_size, j:j+kernel_size])
    return output

# Read an image
image = cv2.imread('IMG_2325.jpg', cv2.IMREAD_GRAYSCALE)

# Apply box filter
box_filtered = box_filter(image, kernel_size=3)

# Apply median filter
median_filtered = median_filter(image, kernel_size=3)

# Display original, box-filtered, and median-filtered images
cv2.imshow('Original Image', image)
cv2.imshow('Box Filtered', box_filtered)
cv2.imshow('Median Filtered', median_filtered)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9(a)

```
import cv2
import numpy as np

def remove_glare(image_path):
    # Read the input image
    original_image = cv2.imread(image_path)

    # Convert the image to grayscale
    gray = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

    # Apply thresholding to create a binary mask of the glare
    _, binary_mask = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)

    # Inpainting: Fill in the glare region using nearby pixel values
    inpainted_image = cv2.inpaint(original_image, binary_mask, inpaintRadius=3,
flags=cv2.INPAINT_TELEA)

    # Display the original and enhanced images
    cv2.imshow("Original Image", original_image)
    cv2.imshow("Enhanced Image", inpainted_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage:
image_path = "path/to/your/image_with_glare.jpg"
remove_glare(image_path)
```

14.

```
# Import necessary libraries
import cv2
# Read the image from file
image = cv2.imread('Screenshot 2024-02-04 at 7.02.01 PM.png')
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to the grayscale image
blur = cv2.GaussianBlur(gray, (11, 11), 0)

# Use Canny edge detector to find edges in the blurred image
canny = cv2.Canny(blur, 30, 150, 3)

# Dilate the edges to connect nearby edges and close gaps
dilated = cv2.dilate(canny, (1, 1), iterations=2)

# Find contours in the dilated image
(cnt, hierarchy) = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
# Convert image to RGB for visualization
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)z
# Draw contours on the RGB image
cv2.drawContours(rgb, cnt, -1, (0, 255, 0), 2)

# Print the number of detected coins
print('Coins in the image: ', len(cnt))
# Wait for a key press and close all OpenCV windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

1.

```
# Python program to illustrate # Python program to illustrate # template matching
import cv2
import numpy as np
# Read the main image
img_rgb = cv2.imread('TSITP.jpg')
# Convert it to grayscale
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY) # Read the template
template = cv2.imread('TSITP1.jpg', 0)
# Store width and height of template in w and h
w, h = template.shape[::-1]
# Perform match operations.
res = cv2.matchTemplate(img_gray, template, cv2.TM_CCOEFF_NORMED) # Specify a threshold
threshold = 0.8
# Store the coordinates of matched area in a numpy array loc = np.where(res >= threshold)
# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2) # Show the final image with
the matched area. cv2.imshow('Detected', img_rgb)
cv2.waitKey(0)
```

2.

```
import cv2
import numpy as np

def find_predefined_shape(image_path, templates):
    # Read the input image
    img = cv2.imread(image_path)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Initialize a list to store results
    results = []

    # Loop through each template
    for template_path in templates:
        # Read the template image
        template = cv2.imread(template_path, 0)
        w, h = template.shape[::-1]

        # Perform template matching
        res = cv2.matchTemplate(gray_img, template, cv2.TM_CCOEFF_NORMED)
        threshold = 0.8
        loc = np.where(res >= threshold)

        # Draw rectangles around the matched areas
        for pt in zip(*loc[::-1]):
            cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)
            results.append(pt)

    # Display the result
    cv2.imshow('Result', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return results

# Define the image path and templates
image_path = 'image_to_search.png'
templates = ['template1.png', 'template2.png', 'template3.png']

# Find the predefined shape in the image
matched_points = find_predefined_shape(image_path, templates)

# Print the coordinates of the matched points
print("Matched points:", matched_points)

3.import cv2
import numpy as np

def template_matching(image, template, method=cv2.TM_CCOEFF_NORMED):
    result = cv2.matchTemplate(image, template, method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        match_loc = min_loc
    else:
        match_loc = max_loc
    return match_loc

# Generate a random larger matrix
larger_matrix = np.random.randint(0, 255, (300, 300)).astype(np.uint8)

# Define a predefined sub-matrix (template)
template = np.random.randint(0, 255, (50, 50)).astype(np.uint8)

# Find the predefined sub-matrix within the larger matrix using different template matching
methods
methods = [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED, cv2.TM_CCORR,
cv2.TM_CCORR_NORMED, cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED]

for method in methods:
    # Make a copy of the larger matrix to draw the rectangle
    img_display = larger_matrix.copy()

    # Perform template matching
    match_loc = template_matching(larger_matrix, template, method)

    # Draw a rectangle around the matched region
    w, h = template.shape[::-1]
    top_left = match_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv2.rectangle(img_display, top_left, bottom_right, 255, 2)

    # Display the result
    cv2.imshow('Result using Method: {}'.format(method), img_display)
    cv2.waitKey(0)

cv2.destroyAllWindows()
```

6

With

```
import cv2
import numpy as np

# Define the video capture object
cap = cv2.VideoCapture(0)

while True:
    # Capture the video frame by frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Define the Sobel, Prewitt, and Canny filters
    sobel_x = cv2.Sobel(gray,cv2.CV_64F, 1, 0, ksize=5)
    sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0,1, ksize=5)
    prewitt_x_kernel = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    prewitt_y_kernel = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
    prewitt_x = cv2.filter2D(gray, cv2.CV_64F, prewitt_x_kernel)
    prewitt_y = cv2.filter2D(gray,cv2.CV_64F, prewitt_y_kernel)
    canny = cv2.Canny(gray, 30,100)

    # Display the resulting frames
    cv2.imshow('Sobel X', sobel_x)
    cv2.imshow('Sobel Y', sobel_y)
    cv2.imshow('Prewitt X', prewitt_x)
    cv2.imshow('Prewitt Y', prewitt_y)
    cv2.imshow('Canny', canny)

    # Exit the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the capture and destroy all windows
cap.release()
cv2.destroyAllWindows()
```

Without

```
import cv2
import numpy as np

# Define the video capture object
cap = cv2.VideoCapture(0)

while True:
    # Capture the video frame by frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Define the Sobel, Prewitt, and Canny filters
    sobel_x_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    prewitt_x_kernel = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    prewitt_y_kernel = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
    canny = cv2.Canny(gray,30, threshold2=100)

    # Apply the Sobel, Prewitt, and Canny filters
    sobel_x = cv2.filter2D(src=gray, ddepth=cv2.CV_64F, kernel=sobel_x_kernel)
    sobel_y = cv2.filter2D(src=gray, ddepth=cv2.CV_64F, kernel=sobel_y_kernel)
    prewitt_x = cv2.filter2D(src=gray, ddepth=cv2.CV_64F, kernel=prewitt_x_kernel)
    prewitt_y = cv2.filter2D(src=gray, ddepth=cv2.CV_64F, kernel=prewitt_y_kernel)

    # Display the resulting frames
    cv2.imshow('Sobel X', sobel_x)
    cv2.imshow('Sobel Y', sobel_y)
    cv2.imshow('Prewitt X', prewitt_x)
    cv2.imshow('Prewitt Y', prewitt_y)
    cv2.imshow('Canny', canny)

    # Exit the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the capture and destroy all windows
```

```
cap.release()
cv2.destroyAllWindows()
```

17,2

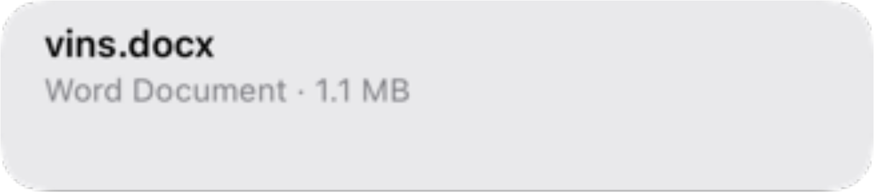
AIVL INDIVIDUAL REPORT.docx
Word Document · 274 KB

17,3

```
import numpy as np

# Given pixel values
pixel_values = np.array([
    [180, 160, 160, 140, 120],
    [110, 110, 120, 140, 120],
    [110, 140, 120, 120, 140],
    [120, 160, 160, 170, 170],
    [170, 120, 110, 140, 110]
])

# Linear Scaling
min_val = np.min(pixel_values)
max_val = np.max(pixel_values)
linear_scaled_values = ((pixel_values - min_val) / (max_val - min_val)) * 255

# Logarithmic Transformation
log_transformed_values = np.log1p(pixel_values)

# Power-law (Gamma) Transformation
gamma = 0.5
gamma_transformed_values = np.power(pixel_values, gamma)

# Display the original and transformed pixel values
print("Original Pixel Values:")
print(pixel_values)
print("\nLinear Scaled Values:")
print(linear_scaled_values.astype(int))
print("\nLogarithmic Transformed Values:")
print(log_transformed_values.astype(int))
print("\nGamma Transformed Values:")
print(gamma_transformed_values.astype(int))
```

```
19)import cv2
import numpy as np
import matplotlib.pyplot as plt

def display_histogram_and_equalize(matrix):
    # Display the pixel values in a histogram
    plt.hist(matrix.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7)
    plt.title('Original Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()

    # Apply histogram equalization
    equalized_matrix = cv2.equalizeHist(matrix)

    # Display the pixel values in a histogram after equalization
    plt.hist(equalized_matrix.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7)
    plt.title('Equalized Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()

    # Display the original and equalized matrices
    print("Original Matrix:")
    print(matrix)
    print("\nEqualized Matrix:")
    print(equalized_matrix)

# Example usage with an 8x8 matrix
input_matrix = np.array([[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, 20, 30, 40, 50, 60, 70, 80],
                         [90, 100, 110, 120, 130, 140, 150, 160],
                         [170, 180, 190, 200, 210, 220, 230, 240],
                         [250, 20, 30, 40, 50, 60, 70, 80],
                         [90, 100, 110, 120, 130, 140, 150, 160]], dtype=np.uint8)

display_histogram_and_equalize(input_matrix)
```

5.



vins.docx

Word Document · 1.1 MB

```
22.import cv2
import numpy as np

def generate_matrix(name, register_number):
    # Convert name and register number to ASCII representation
    name_ascii = [ord(char) for char in name]
    register_number_ascii = [int(digit) for digit in str(register_number)]

    # Ensure proper capitalization for the first letter of each word and initials
    name = name.title()

    # Create a 5x5 matrix to hold the generated data
    matrix = np.zeros((5, 5), dtype=np.uint8)

    # Fill the matrix with ASCII values of name and register number
    idx = 0
    for i in range(5):
        for j in range(5):
            if idx < len(name_ascii):
                matrix[i, j] = name_ascii[idx]
                idx += 1
            elif idx < len(name_ascii) + len(register_number_ascii):
                matrix[i, j] = register_number_ascii[idx - len(name_ascii)]
                idx += 1
            else:
                last_digit = register_number_ascii[-1]
                matrix[i, j] = 2 * last_digit

    return matrix

# Generate the matrix
your_name = "shudhi"  # Change this to your name with proper capitalization
your_register_number = 241  # Change this to your register number
generated_matrix = generate_matrix(your_name, your_register_number)

# Perform thresholding
_, thresholded_matrix = cv2.threshold(generated_matrix, 127, 255, cv2.THRESH_BINARY)

# Display the matrices
print("Generated Matrix:")
print(generated_matrix)
print("\nThresholded Matrix:")
print(thresholded_matrix)
```

8.

21R241 INDIVIDUAL REPORT 2.docx

Word Document · 305 KB