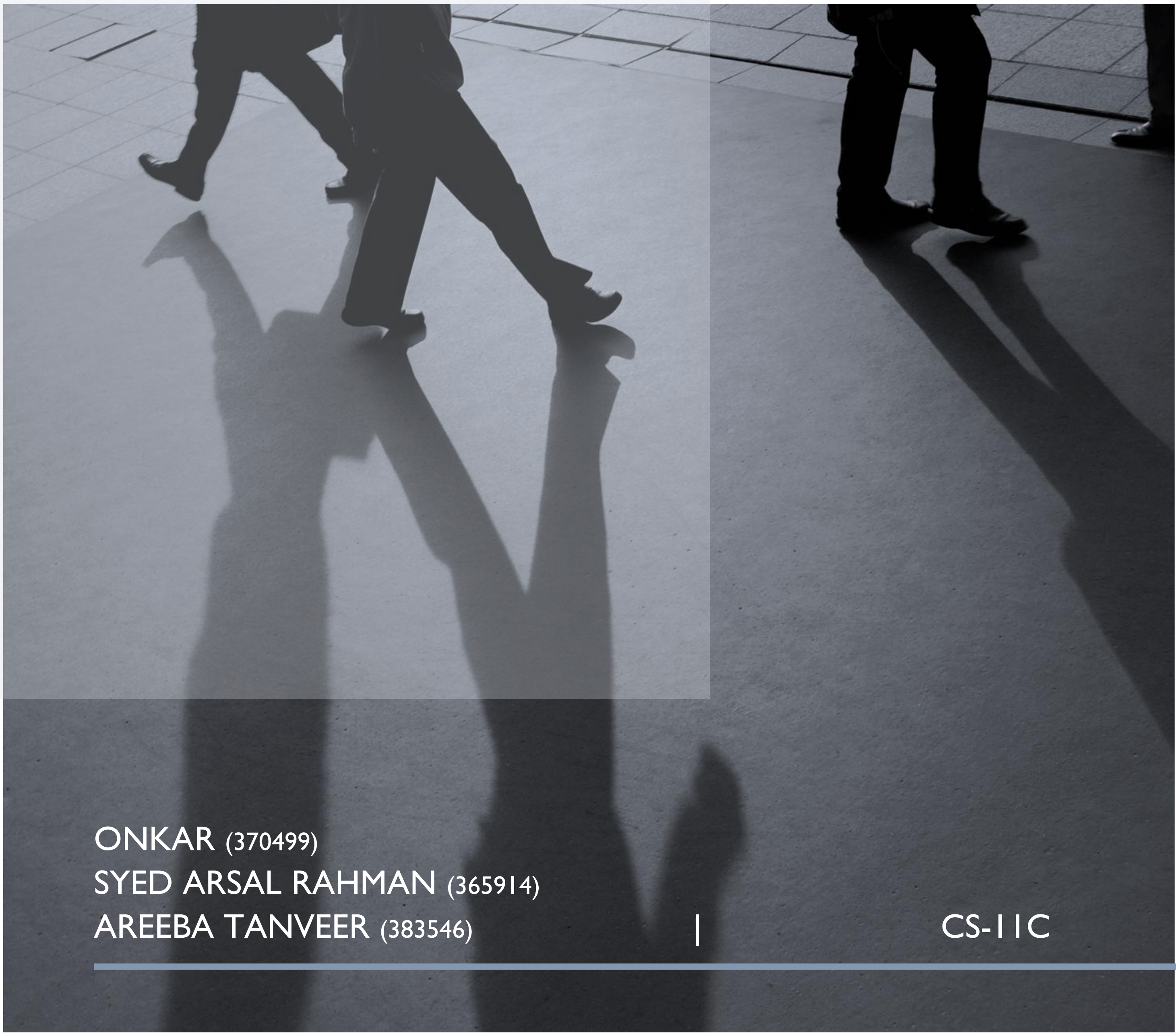
ASSIGNMENT 2 REPORT

DIGITAL IMAGE PROCESSING



ONKAR (370499)
SYED ARSAL RAHMAN (365914)
AREEBA TANVEER (383546) | CS-11C

EIGHT CONNECTIVITY USING OWN ALGORITHM

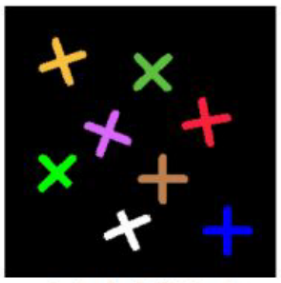
QUESTION:

Assignment 2 (Group of 2 students)

Due Date : 24 Oct 2023

Using 8-connectivity, write code in MatLab/Python to label each object and get the resulting image as shown on the right side. The input image is provided sep. Refer to the lecture notes for the algorithm. Do not use the built in commands.

Original Image


Labeled Objects

GITHUB LINK

ASSIGNMENT (CODE + DESCRIPTION):

1. **Import libraries:** numpy (as np) and cv2 (OpenCV) are imported to work with arrays and images.

```
import numpy as np
import cv2
```

2. **Define label_components Function:**
   a) This function takes a binary image (black and white) as input and labels its connected components.
   b) It initializes an empty labels matrix, eq_table to track equivalences, and label to assign labels to connected components.

```
def label_components(image):
    rows, cols = image.shape
    labels = np.zeros((rows, cols), dtype=np.uint8)
    label = 1
    eq_table = {}
```

3. **Define find_root Function:**
   a) A nested function that finds the root (representative) label for a given label in the eq_table.b
   b) Implements path compression for optimization in union-find operations.

```
def find_root(x):
        if x not in eq_table:
            eq_table[x] = x
            return x
        if eq_table[x] != x:
            eq_table[x] = find_root(eq_table[x])
        return eq_table[x]
```

4. **Iterate through the image pixels:**
   a) Nested loops iterate through each pixel of the binary image.
   b) If the pixel is white (255), it considers neighboring pixels to determine the label for the connected component.
   c) It checks left, top-left, top, and top-right neighbors (8-connectivity), assigning labels as appropriate.
   d) If there are no valid neighbors, a new label is assigned.
   e) If a pixel has valid neighbors with different labels, it updates the equivalence table to ensure that they all point to the same root label using find_root.

```
for row in range(rows):
        for col in range(cols):
            if image[row, col] == 255:
                neighbors = [
                    labels[row, col - 1],
                    labels[row - 1, col - 1],
                    labels[row - 1, col],
                ]

                if col + 1 < cols:
                    neighbors.append(labels[row - 1, col + 1])

                valid_neighbors = [n for n in neighbors if n != 0]

                if not valid_neighbors:
                    labels[row, col] = label
                    label += 1
                else:
                    labels[row, col] = min(valid_neighbors)
                    root = find_root(labels[row, col])
                    for neighbor in valid_neighbors:
                        if find_root(neighbor) != root:
                            eq_table[find_root(neighbor)] = root
```

5. **Final Label Assignment:**
   a) After iterating through all pixels, it goes through the image again and ensures that all pixels point to the same root label in the equivalence table.
   b) The function returns the equivalence table and the labeled image.

```
for row in range(rows):
        for col in range(cols):
            labels[row, col] = eq_table[find_root(labels[row, col])]

    return eq_table, labels
```

6. Read and Threshold Input Image:

a) The code reads an input binary image ("Image_01.bmp") and applies a threshold to convert it into a binary image using OpenCV's cv2.threshold function.
b) The binary image is displayed using OpenCV's cv2.imshow.

```
binary_image = cv2.imread("Image_01.bmp", 0)
threshold, image = cv2.threshold(binary_image, 120, 255, cv2.THRESH_BINARY)
cv2.imshow("Binary", image)
table, labelled_image = label_components(image)
rows, cols = labelled_image.shape

unique_labels = np.unique(labelled_image)
print(unique_labels)
num_objects = len(unique_labels) - 1
print(num_objects)
```

7. Label connected components:

a) The label_components function is called to label the connected components in the binary image.
b) A list of unique labels in the labeled image is obtained, and the number of connected components (excluding background) is calculated.
c) A color map is defined, mapping each connected component label to a specific BGR color.
d) Iterating through the labeled image, it assigns the color associated with each component to the corresponding pixels in the output image.
e) The final colored image, representing connected components, is displayed using cv2.imshow.
f) The code waits for user input (e.g., pressing a key) and then closes all OpenCV windows.

```python
color = [
    [0, 0, 0],
    [0, 0, 255],
    [0, 255, 0],
    [255, 0, 0],
    [0, 255, 255],
    [255, 0, 255],
    [255, 255, 0],
    [255, 255, 255],
    [50, 80, 100],
]

fin_image = cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR)


for i in range(1, num_objects + 1):
    for row in range(rows):
        for col in range(cols):
            if labelled_image[row, col] == unique_labels[i]:
                fin_image[row][col][0] = color[i][0]
                fin_image[row][col][1] = color[i][1]
                fin_image[row][col][2] = color[i][2]

cv2.imshow("Labelled", fin_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

CONCLUSION:

This Python code processes a binary image to label its connected components and then assigns different colors to each labeled component, effectively creating a visual representation of these components. It utilizes libraries like NumPy and OpenCV (cv2). This code demonstrates the process of labeling and visualizing connected components in a binary image, which can be useful in applications like image segmentation and object recognition. The labeled image visually highlights different objects in the input image by assigning distinct colors to them.