SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Winter Semester 24-25

DIGITAL ASSIGNMENT – I

COURSE : DIGITAL IMAGE PROCESSING

COURSE CODE : SWE1010

SLOT : F1+TF1

FACULTY : Dr. JAGANNATHAN J

SUMBITTED BY :

NAME : SWATHI.S

REGISTER NUMBER : 22MIS0060

1) Extract and Visualize Individual Color Channels (R, G, B) from an Image

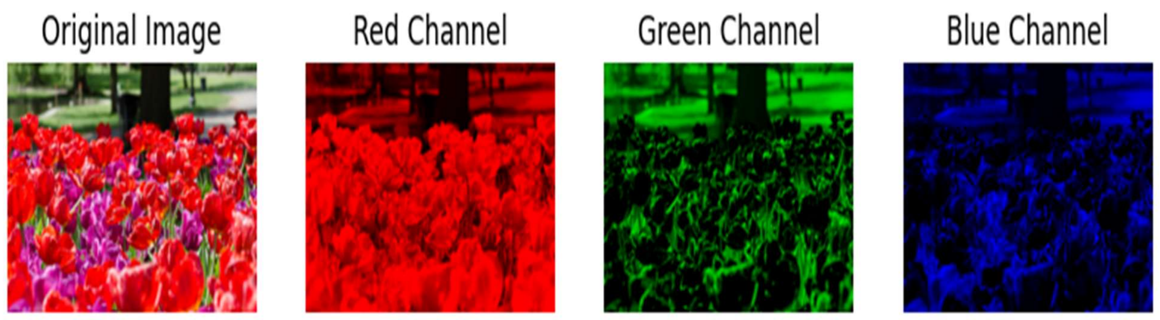
1. Title

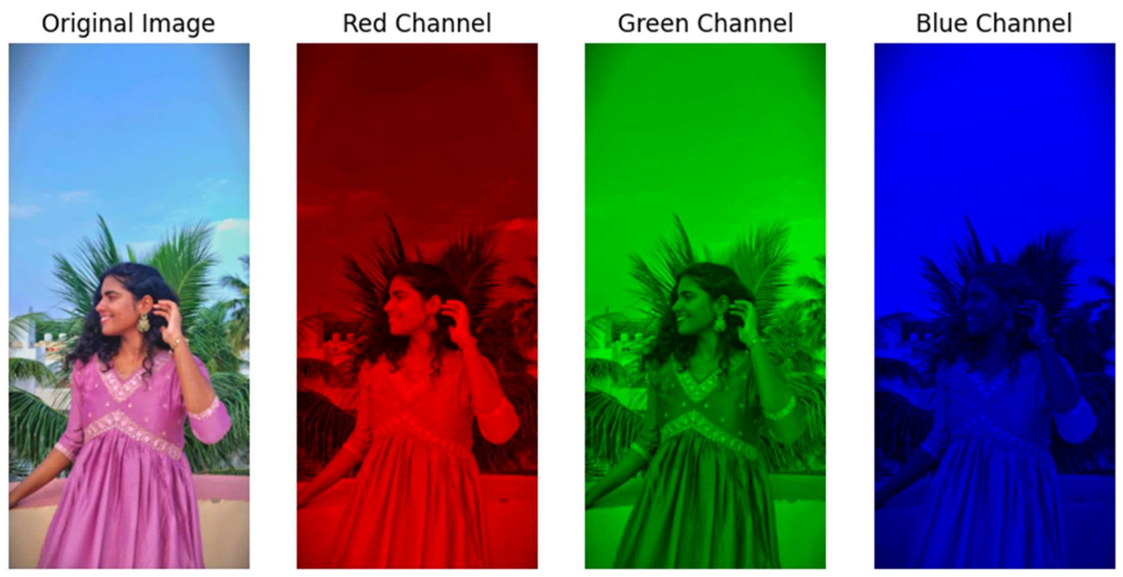
Extracting and Visualizing Red, Green, and Blue Color Channels from an Image

2. Implementation (Python using OpenCV & Matplotlib)

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

# Load the image
image = cv2.imread("/content/Screenshot 2025-03-24 194904.png")  # Replace with your image path
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

# Split the image into Red, Green, and Blue channels
R, G, B = cv2.split(image)

# Create blank images for visualization
zeros = np.zeros_like(R)

# Create individual color channel images
red_image = cv2.merge([R, zeros, zeros])   # Red channel
green_image = cv2.merge([zeros, G, zeros]) # Green channel
blue_image = cv2.merge([zeros, zeros, B])  # Blue channel

# Display the original and individual channel images
plt.figure(figsize=(10, 5))
plt.subplot(1, 4, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 4, 2)
plt.imshow(red_image)
plt.title("Red Channel")
plt.axis("off")

plt.subplot(1, 4, 3)
plt.imshow(green_image)
plt.title("Green Channel")
plt.axis("off")

plt.subplot(1, 4, 4)
plt.imshow(blue_image)
plt.title("Blue Channel")
plt.axis("off")

plt.show()
```

3. Output:





4. Algorithm

1. Load the input image using OpenCV.
2. Convert the BGR image to RGB format.
3. Split the image into three color channels (R, G, B).
4. Create blank images to isolate each channel.
5. Merge each channel separately with zeros to highlight only that color.
6. Display the original and processed images using Matplotlib.

5. Application

- Used in medical imaging to analyze specific color components.
- Helps in object detection and segmentation.
- Used in image processing tasks like color-based filtering.

2: Feature Matching Using ORB Descriptors

1. Title

Feature Matching using ORB (Oriented FAST and Rotated BRIEF)

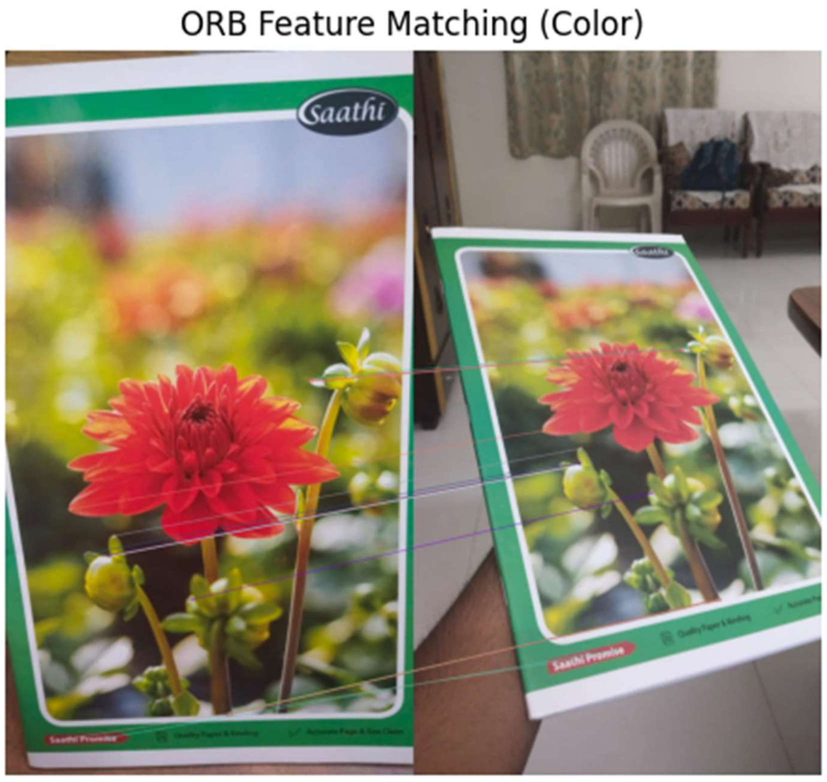
2. Implementation (Python using OpenCV)

```
import cv2
import matplotlib.pyplot as plt

# Load the images
image1 = cv2.imread("/content/veh 19.jpg", 0)  # Query Image
image2 = cv2.imread("/content/veh 19 neww.jpg", 0)  # Train Image

# Create ORB detector
orb = cv2.ORB_create()

# Detect keypoints and compute descriptors
keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
keypoints2, descriptors2 = orb.detectAndCompute(image2, None)

# Create BFMatcher (Brute-Force Matcher) with Hamming distance
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors
matches = bf.match(descriptors1, descriptors2)

# Sort matches by distance (lower distance = better match)
matches = sorted(matches, key=lambda x: x.distance)
```

```
# Draw the top 20 matches

matched_image = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches[:20],
None, flags=2)


# Display the output

plt.figure(figsize=(10, 5))

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

plt.title("ORB Feature Matching")

plt.axis("off")

plt.show()
```

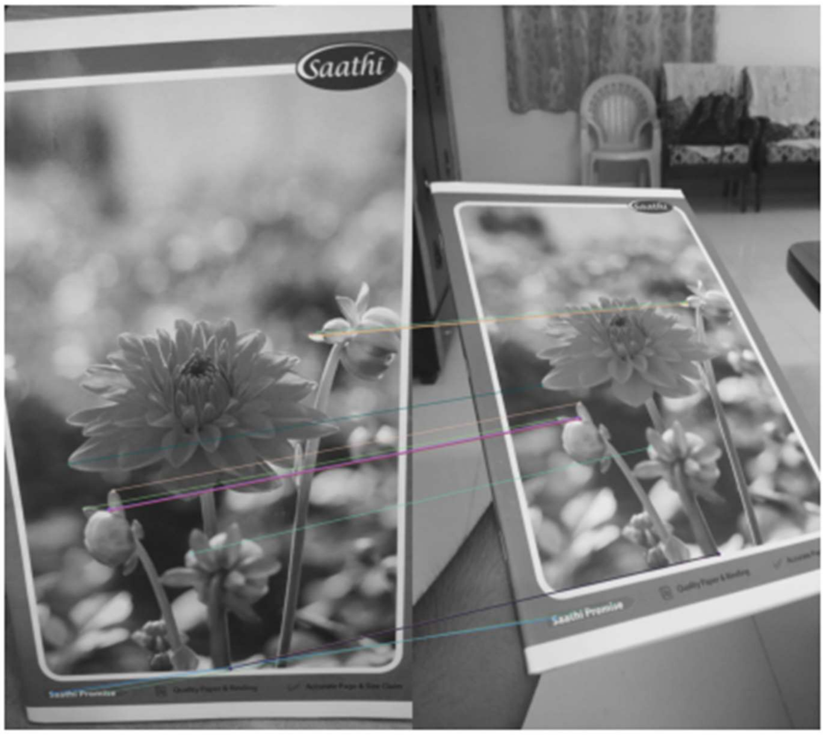
3. Output:

In colour image:



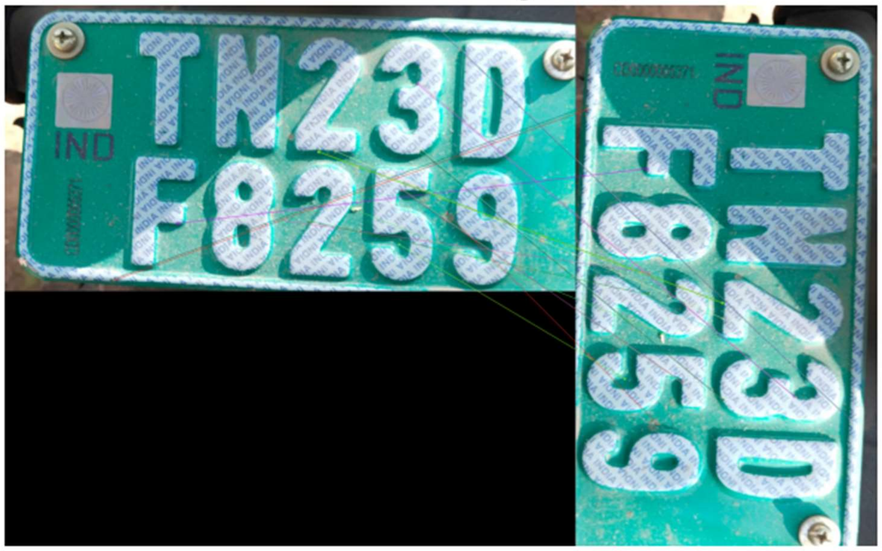
In Black and White:

ORB Feature Matching

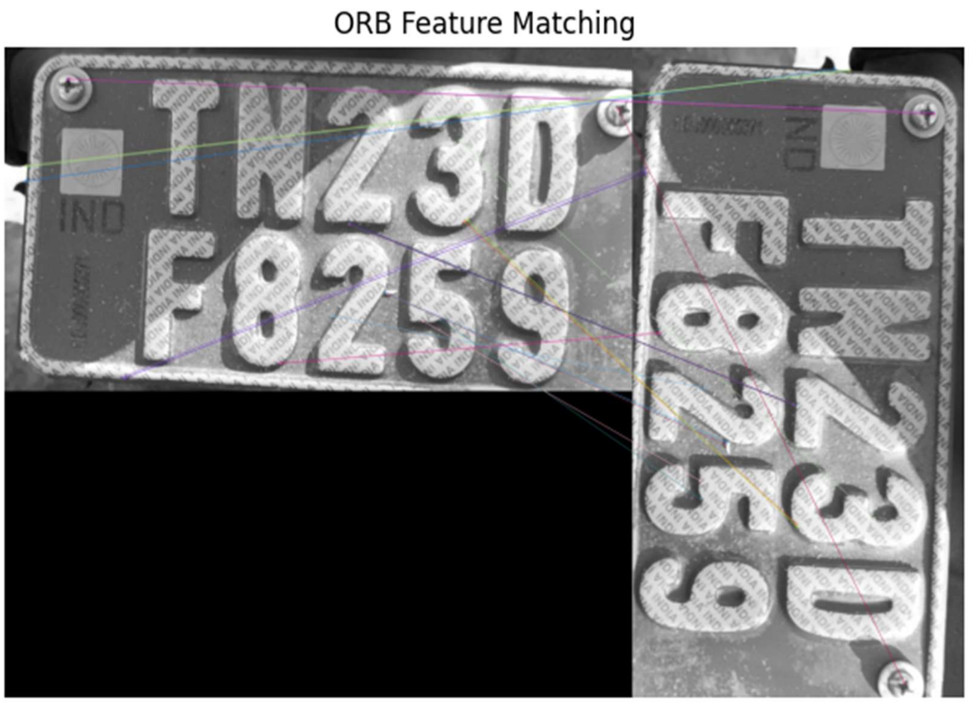


In

ORB Feature Matching (Color)



In black and white:



4. Algorithm

1. Load two grayscale images.
2. Initialize the ORB feature detector.
3. Detect keypoints and compute descriptors for both images.
4. Use a Brute-Force Matcher with Hamming distance to find feature matches.
5. Sort matches based on distance.
6. Draw and display the best matches.

5. Application

- Used in object recognition and image stitching (e.g., panorama creation).
- Helps in tracking objects across frames in videos.
- Useful in biometric applications like fingerprint and face recognition.