Lab Report 6 : Image Conversion(RGB to HSI, RGB to Grayscale, RGB to Red, Green, Blue Channel)

Course Title: Digital Image Processing Lab

Course Code: CSE-406

Date of Performance: September 29, 2024

Date of Submission: October 06, 2024

Submitted to:

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University

Submitted by:

Class Roll	Exam Roll	Name
370	202182	Rubayed All Islam

Experiment Name

a) **RGB to HSI Conversion**

b) **RGB to Grayscale Conversion**

c) **RGB Channel Separation**

Objectives

a) To convert an RGB image to the HSI (Hue, Saturation, Intensity) color space and visualize the individual components.

b) To convert an RGB image to a grayscale image using both a formula-based approach and a built-in function, and to visualize the results.

c) To extract and visualize the individual Red, Green, and Blue channels from an RGB image and observe how each channel contributes to the overall color.

Methodology

1. **RGB to HSI Conversion:**
   - Read the RGB image using a suitable library in both MATLAB and Python.
   - Normalize the RGB values to the range [0, 1].
   - Compute the intensity (I) by averaging the RGB values.
   - Calculate the saturation (S) using the formula that considers the minimum RGB value.
   - Compute the hue (H) using the arccosine function, ensuring the correct quadrant is accounted for.
   - Combine the H, S, and I components to form the HSI image.
   - Visualize the original RGB image and the individual H, S, and I components.
2. **RGB to Grayscale Conversion:**
   - Read the RGB image and convert it to a grayscale image using a formula or built-in function.
   - For the formula-based approach, apply the weighted sum of RGB values to get the luminance.
   - Use a built-in function (e.g., ‘rgb2gray‘ in MATLAB or Pillow in Python) for comparison.
   - Display both the original RGB image and the grayscale image for visual comparison.
3. **RGB Channel Separation:**
   - Read the RGB image and extract the individual Red, Green, and Blue channels.

- Create separate images for each channel, ensuring that only the corresponding channel is visible while others are set to zero.
- Visualize the original RGB image alongside the separated Red, Green, and Blue channel images for comparison.

Experiment 1: RGB to HSI Conversion

MATLAB Code:

Listing 1: RGB_To_HSI.m

```
% Read the RGB image
rgbImage = imread('flower.jpg'); % Replace with your image file

% Display the original RGB image
figure;
subplot(2, 2, 1);
imshow(rgbImage);
title('Original RGB Image');

% Convert RGB to HSI
hsiImage = rgb2hsi(rgbImage);

% Extract H, S, and I components
H = hsiImage(:,:,1);
S = hsiImage(:,:,2);
I = hsiImage(:,:,3);

% Normalize H, S, and I for display purposes
H_disp = H / 360; % Normalize Hue to [0, 1] for display
S_disp = S; % Saturation is already in [0, 1]
I_disp = I; % Intensity is already in [0, 1]

% Display the HSI components separately

% Display Hue component using 'hsv' colormap
subplot(2, 2, 2);
imshow(H_disp);
colormap(gca, gray); % Apply the hsv colormap to the current axis
colorbar; % Show colorbar to indicate Hue range
caxis([0 1]); % Set the color axis to match [0, 1] (normalized)
title('Hue Component');

% Display Saturation component
subplot(2, 2, 3);
imshow(S_disp);
colormap(gca, gray); % Apply the gray colormap to the current
    axis
colorbar; % Show colorbar to indicate Saturation range
caxis([0 1]); % Set the color axis to match [0, 1]
```

```
title('Saturation Component');

% Display Intensity component
subplot(2, 2, 4);
imshow(I_disp);
colormap(gca, gray); % Apply the gray colormap to the current
   axis
colorbar; % Show colorbar to indicate Intensity range
caxis([0 1]); % Set the color axis to match [0, 1]
title('Intensity Component');

% Function to convert RGB to HSI
function hsi = rgb2hsi(rgb)
    % Normalize the RGB values
    rgb = double(rgb) / 255; % Normalize to [0, 1]
    R = rgb(:,:,1);
    G = rgb(:,:,2);
    B = rgb(:,:,3);

    % Compute Intensity
    I = (R + G + B) / 3;

    % Compute Saturation
    min_val = min(min(R, G), B);
    S = 1 - (3 ./ (R + G + B + eps)) .* min_val;
    S(R + G + B == 0) = 0; % Handle zero-intensity case

    % Compute Hue
    num = 0.5 * ((R - G) + (R - B));
    den = sqrt((R - G).^2 + (R - B) .* (G - B)) + eps;
    theta = acos(num ./ den);
    H = zeros(size(I));
    H(B <= G) = theta(B <= G);
    H(B > G) = 2 * pi - theta(B > G);

    % Convert radians to degrees
    H = H * (180 / pi);

    % Normalize H to [0, 360]
    H(H < 0) = H(H < 0) + 360;
    H = mod(H, 360);

    % Combine H, S, I into HSI
    hsi = cat(3, H, S, I);
end
```

Python Code:

Listing 2: RGB_To_HSI.py

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from PIL import Image

# Function to convert RGB to HSI
def rgb_to_hsi(rgb):
    # Normalize the RGB values to [0, 1]
    rgb = np.array(rgb, dtype=np.float64) / 255.0
    R = rgb[:, :, 0]
    G = rgb[:, :, 1]
    B = rgb[:, :, 2]

    # Compute Intensity
    I = (R + G + B) / 3

    # Compute Saturation
    min_rgb = np.minimum(np.minimum(R, G), B)
    S = 1 - (3 / (R + G + B + 1e-8)) * min_rgb
    S[R + G + B == 0] = 0  # Handle zero-intensity case

    # Compute Hue
    num = 0.5 * ((R - G) + (R - B))
    den = np.sqrt((R - G)**2 + (R - B) * (G - B)) + 1e-8
    theta = np.arccos(num / den)
    H = np.zeros_like(I)
    H[B <= G] = theta[B <= G]
    H[B > G] = 2 * np.pi - theta[B > G]

    # Convert radians to degrees and normalize to [0, 360]
    H = np.degrees(H)
    H[H < 0] += 360
    H = H % 360

    # Combine H, S, I into HSI
    hsi = np.stack((H, S, I), axis=-1)
    return hsi

# Read the RGB image
rgb_image = Image.open('flower.jpg').convert('RGB')

# Display the original RGB image
plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.imshow(rgb_image)
plt.title('Original RGB Image')

# Convert RGB to HSI
hsi_image = rgb_to_hsi(rgb_image)

# Extract H, S, and I components
H = hsi_image[:, :, 0]
S = hsi_image[:, :, 1]
```

```
I = hsi_image[:, :, 2]

# Display the HSI components separately

# Display Hue component using a 'hsv' colormap for better visualization
plt.subplot(2, 2, 2)
plt.imshow(H, cmap='gray', vmin=0, vmax=360)  # Set vmin and vmax to match Hue range
plt.title('Hue Component')
plt.colorbar(label='Hue (degrees)')

# Display Saturation component in grayscale
plt.subplot(2, 2, 3)
plt.imshow(S, cmap='gray', vmin=0, vmax=1)
plt.title('Saturation Component')
plt.colorbar(label='Saturation')

# Display Intensity component in grayscale
plt.subplot(2, 2, 4)
plt.imshow(I, cmap='gray', vmin=0, vmax=1)
plt.title('Intensity Component')
plt.colorbar(label='Intensity')

# Show the plot
plt.tight_layout()
plt.show()
```

Output:

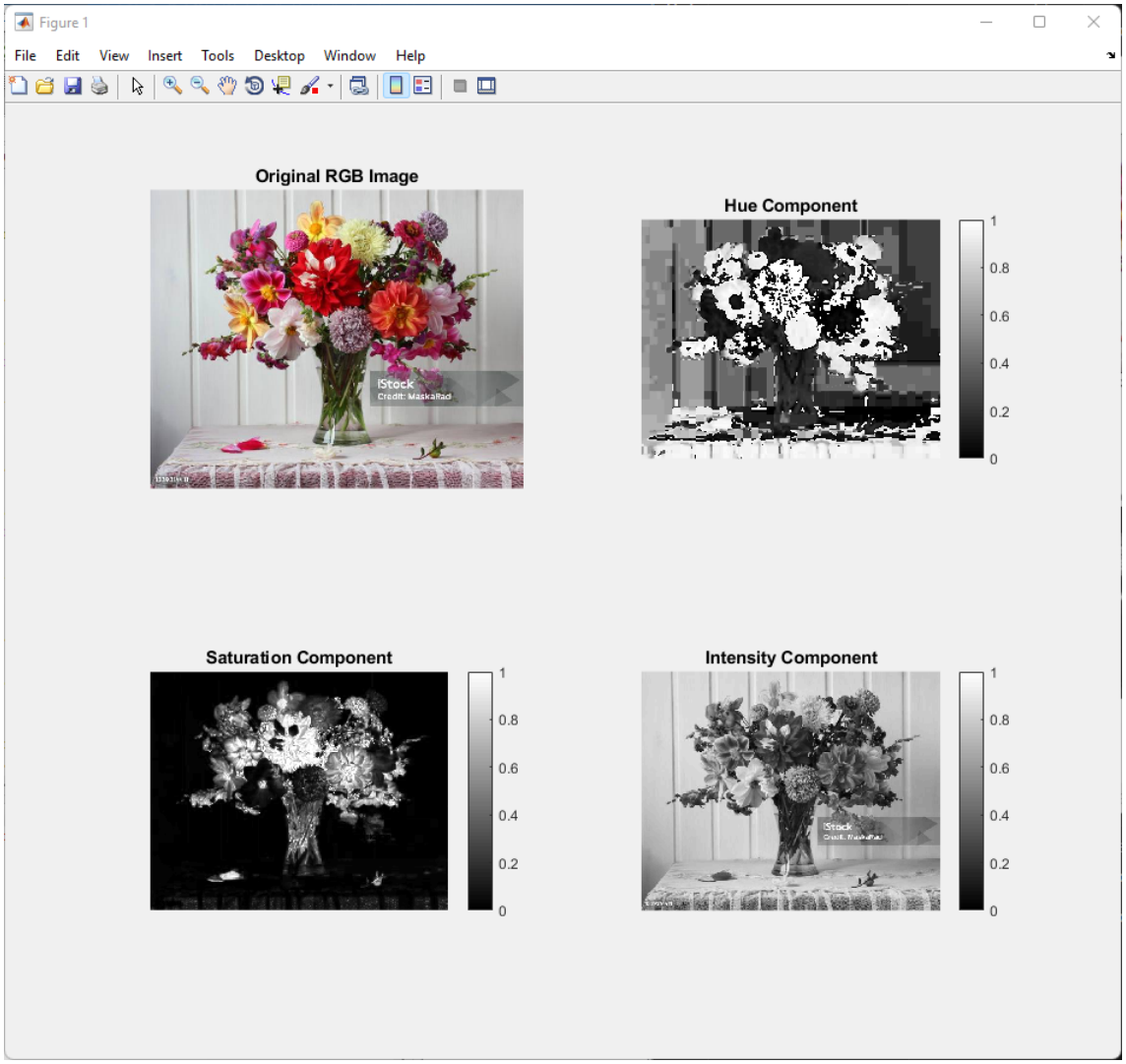

Figure 1: RGB to HSI Conversion Output for Matlab Code

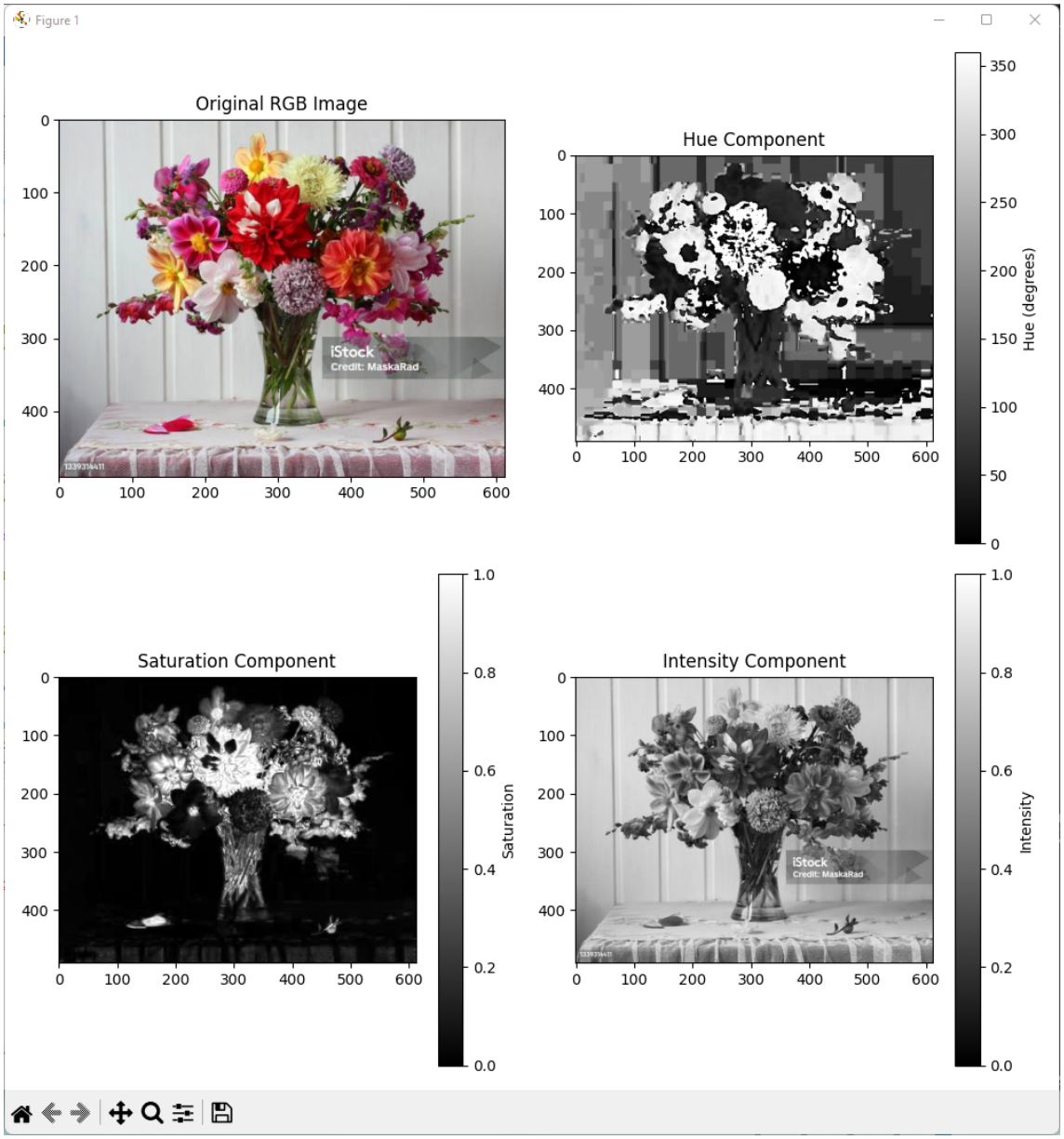

Figure 2: RGB to HSI Conversion Output for Python Code

Experiment 2: RGB to Grayscale Conversion

MATLAB Code:

Listing 3: RGB_To_Grayscale.m

```
% Read the RGB image
rgbImage = imread('flower.jpg');

% Method 1: Using the Formula
% Convert to grayscale using the weighted sum formula
grayscaleImage_formula = 0.2989 * rgbImage(:,:,1) + 0.5870 *
   rgbImage(:,:,2) + 0.1140 * rgbImage(:,:,3);

% Display the original RGB image and grayscale image
figure;
subplot(1, 2, 1);
imshow(rgbImage);
title('Original RGB Image');
axis off;

subplot(1, 2, 2);
imshow(grayscaleImage_formula, []);
title('Grayscale Image (Formula)');
axis off;

% Method 2: Using MATLAB's built-in function
% Convert to grayscale using rgb2gray
grayscaleImage_builtin = rgb2gray(rgbImage);

% Display the grayscale image using built-in function
figure;
subplot(1, 2, 1);
imshow(rgbImage);
title('Original RGB Image');
axis off;

subplot(1, 2, 2);
imshow(grayscaleImage_builtin);
title('Grayscale Image (Built-in)');
axis off;
```

Python Code:

Listing 4: RGB_To_Grayscale.py

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Method 1: Using the Formula
```

```
6
7  # Read the RGB image
8  rgb_image = Image.open('flower.jpg').convert('RGB')
9  rgb_array = np.array(rgb_image)
10
11 # Convert to grayscale using the weighted sum formula
12 grayscale_image = 0.2989 * rgb_array[:, :, 0] + 0.5870 *
       rgb_array[:, :, 1] + 0.1140 * rgb_array[:, :, 2]
13
14 # Display the grayscale image
15 plt.figure(figsize=(8, 4))
16
17 plt.subplot(1, 2, 1)
18 plt.imshow(rgb_image)
19 plt.title('Original RGB Image')
20 plt.axis('off')
21
22 plt.subplot(1, 2, 2)
23 plt.imshow(grayscale_image, cmap='gray')
24 plt.title('Grayscale Image (Formula)')
25 plt.axis('off')
26
27 plt.tight_layout()
28 plt.show()
29
30
31 # Method 2: Using PIL
32
33 # Convert to grayscale using PIL
34 grayscale_image_pil = rgb_image.convert('L')
35
36 # Display the grayscale image
37 plt.figure(figsize=(8, 4))
38
39 plt.subplot(1, 2, 1)
40 plt.imshow(rgb_image)
41 plt.title('Original RGB Image')
42 plt.axis('off')
43
44 plt.subplot(1, 2, 2)
45 plt.imshow(grayscale_image_pil, cmap='gray')
46 plt.title('Grayscale Image (PIL)')
47 plt.axis('off')
48
49 plt.tight_layout()
50 plt.show()
```

Output:

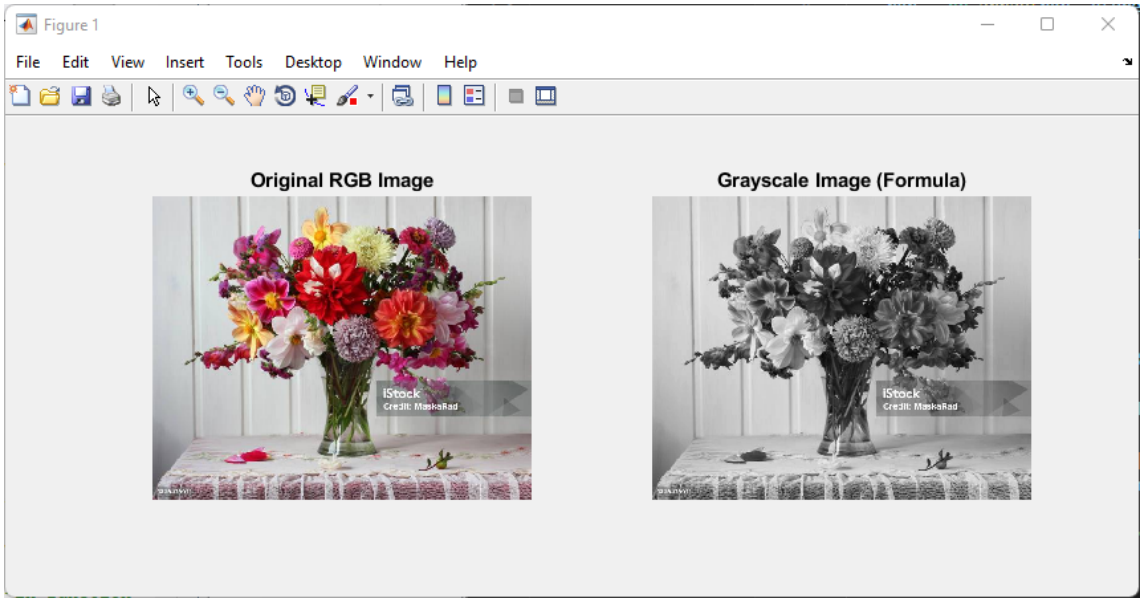

Figure 3: RGB to Grayscale Conversion Output for Matlab Code

Figure 4: RGB to Grayscale Conversion Output for Python Code

Experiment 3: RGB Channel Separation

MATLAB Code:

Listing 5: RGB_Channel_Separation.m

```
% Read the RGB image
rgbImage = imread('flower.jpg'); % Replace with your image file

% Extract the Red, Green, and Blue channels
RedChannel = rgbImage(:,:,1);
GreenChannel = rgbImage(:,:,2);
BlueChannel = rgbImage(:,:,3);

% Display the original RGB image and the individual channels
figure('Position', [100, 100, 800, 600]); % Set figure size

% Display original RGB image
subplot(2, 2, 1);
imshow(rgbImage);
title('Original RGB Image');
axis off;

% Display Red channel
subplot(2, 2, 2);
imshow(RedChannel);
title('Red Channel');
axis off;

% Display Green channel
subplot(2, 2, 3);
imshow(GreenChannel);
title('Green Channel');
axis off;

% Display Blue channel
subplot(2, 2, 4);
imshow(BlueChannel);
title('Blue Channel');
axis off;

% Adjust layout
tight_layout();
```

Python Code:

Listing 6: RGB_Channel_Separation.py

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

```
4
5  # Read the RGB image
6  rgb_image = Image.open('flower.jpg').convert('RGB')
7  rgb_array = np.array(rgb_image)
8
9  # Extract the Red, Green, and Blue channels
10 RedChannel = rgb_array[:, :, 0]
11 GreenChannel = rgb_array[:, :, 1]
12 BlueChannel = rgb_array[:, :, 2]
13
14 # Display the original RGB image and the individual channels
15 plt.figure(figsize=(10, 8))
16
17 # Display original RGB image
18 plt.subplot(2, 2, 1)
19 plt.imshow(rgb_image)
20 plt.title('Original RGB Image')
21 plt.axis('off')
22
23 # Display Red channel
24 plt.subplot(2, 2, 2)
25 plt.imshow(RedChannel, cmap='Reds')
26 plt.title('Red Channel')
27 plt.axis('off')
28
29 # Display Green channel
30 plt.subplot(2, 2, 3)
31 plt.imshow(GreenChannel, cmap='Greens')
32 plt.title('Green Channel')
33 plt.axis('off')
34
35 # Display Blue channel
36 plt.subplot(2, 2, 4)
37 plt.imshow(BlueChannel, cmap='Blues')
38 plt.title('Blue Channel')
39 plt.axis('off')
40
41 plt.tight_layout()
42 plt.show()
```

Output:

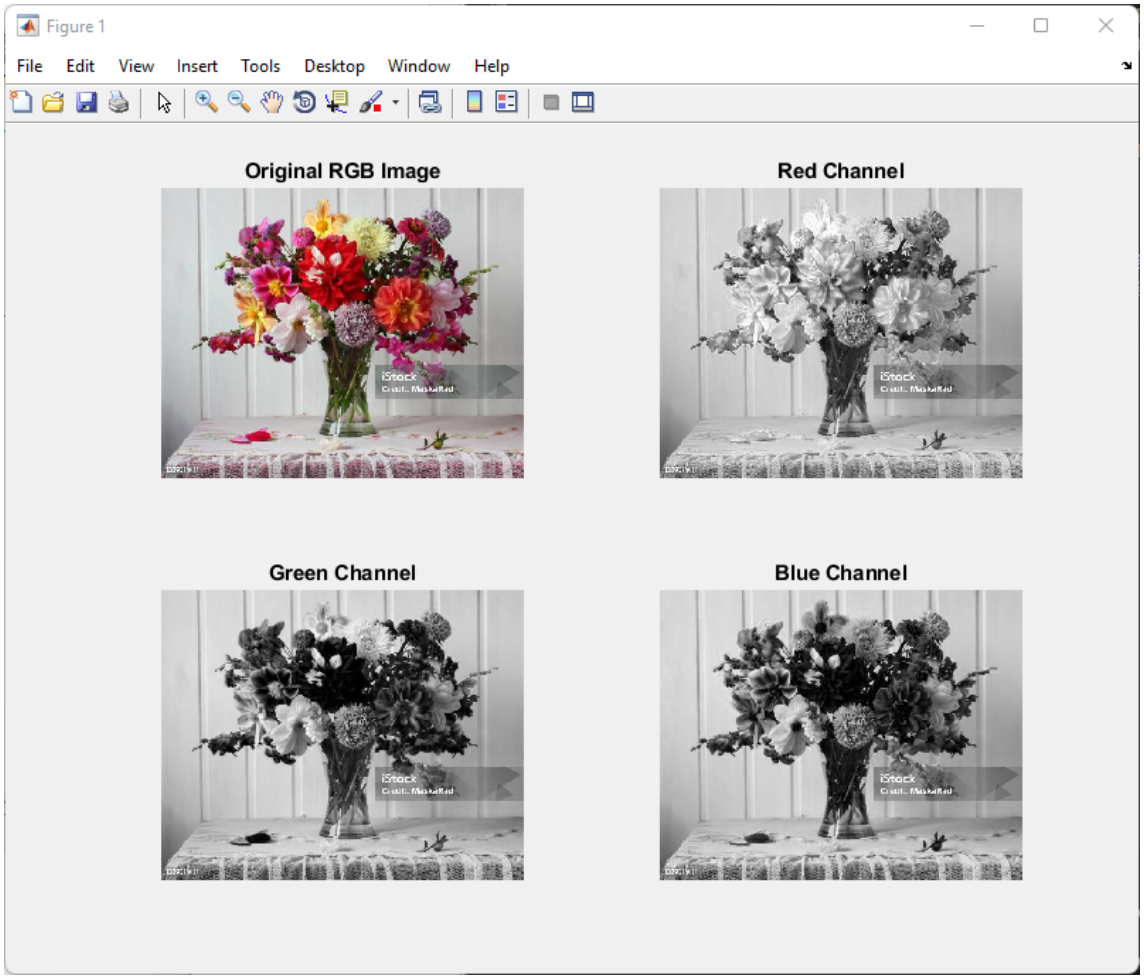

Figure 5: RGB Channel Separation Output for Matlab Code

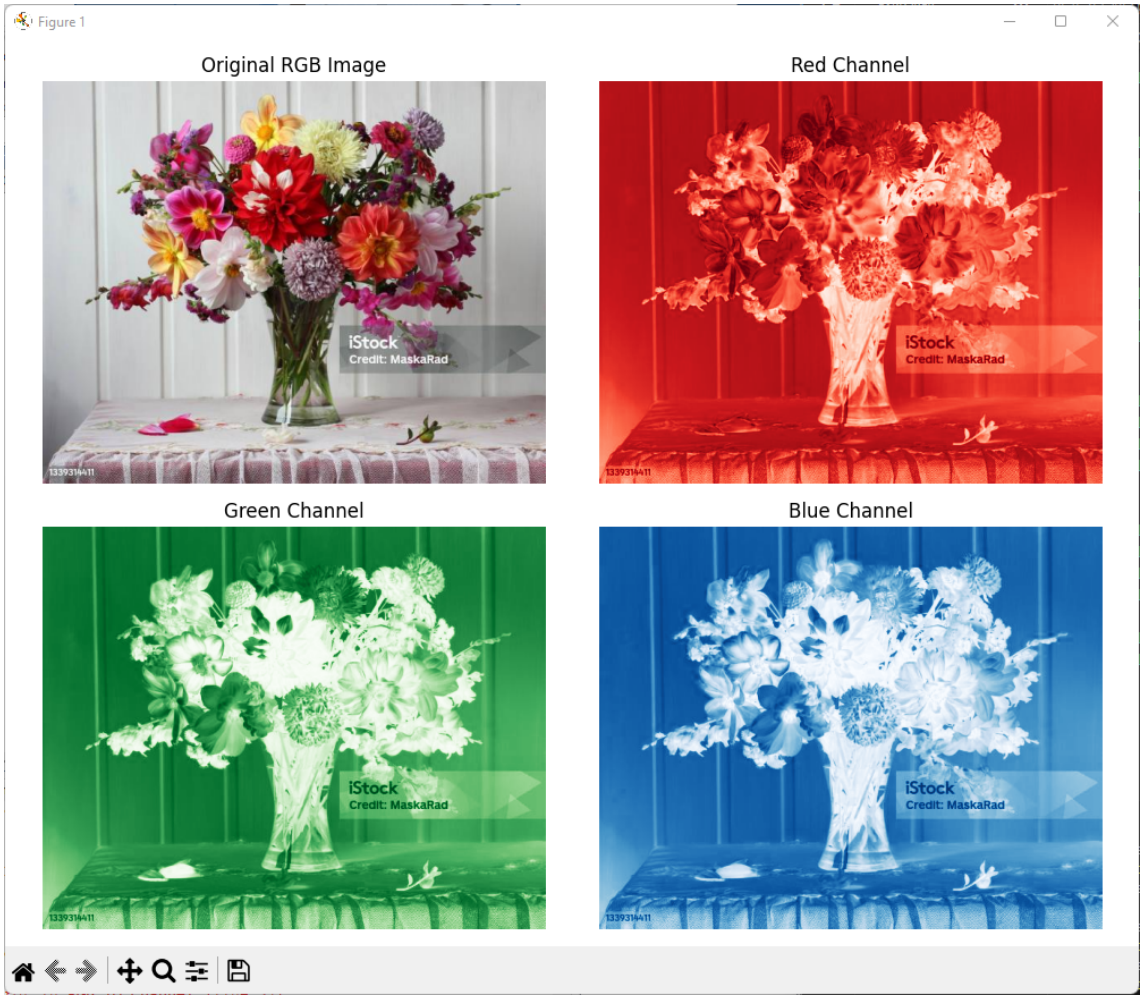

Figure 6: RGB Channel Separation Output for Python Code

Results

1. **RGB to HSI Conversion:**
    - The RGB image was successfully converted to HSI, revealing distinct Hue, Saturation, and Intensity components.
    - The hue component effectively represented the image's colors, while saturation indicated color vividness, and intensity showed brightness levels.
2. **RGB to Grayscale Conversion:**
    - The RGB image was converted to a grayscale image, retaining luminance information.
    - Both the formula-based method and the built-in function produced similar results, confirming the effectiveness of the luminance calculation.
3. **RGB Channel Separation:**
    - The Red, Green, and Blue channels were extracted from the RGB image, clearly showing the contribution of each color.
    - The isolated channels highlighted the structure of color information within the original image.

In summary, the experiments effectively demonstrated various color space transformations and channel manipulations, enhancing the understanding of image representation.

Conclusion

In this lab report, we explored three fundamental image processing techniques: RGB to HSI conversion, RGB to grayscale conversion, and RGB channel separation.

The successful conversion to HSI provided valuable insights into color representation, allowing for a clearer understanding of the image's color components. The grayscale conversion effectively highlighted luminance while maintaining essential visual information, demonstrating the versatility of grayscale images in further analysis. Lastly, the extraction of individual color channels revealed the contribution of each color in the RGB model, reinforcing the understanding of color composition in digital images.

Overall, these experiments underscored the importance of different color models and transformations in image processing, paving the way for advanced applications in computer vision, graphics, and image analysis.