Assignment-1: Image Effect Implementation (Part-1)

Md. Abu Ammar
Student ID: 2315189650
email: abu.ammar.2315189@northsouth.edu

August 15, 2023

I. Introduction

Digital Image Processing is a fundamental field that involves various techniques and algorithms to manipulate and enhance digital images. In this assignment, we explore the implementation of several image effects, including Posterization, Nightvision, Photocopy, Vignetting, and Mirror Image. These effects showcase different aspects of image processing, from basic pixel-level operations to more complex transformations.

The purpose of this assignment is to develop a hands-on understanding of image processing techniques and to demonstrate the ability to implement these effects using MATLAB. By applying these effects, we gain insights into how image manipulation can alter visual perception and create artistic or functional enhancements.

Each effect will be implemented step by step, showcasing the transition from the original input image to the final output image. For each effect, we will provide explanations of the underlying techniques, code implementation, and visualizations of the intermediate steps. This assignment not only assesses our programming skills but also evaluates our comprehension of the image processing concepts and our ability to convey them effectively.

In the following sections, we will delve into the implementation of each image effect, explaining the approach, presenting the code, and displaying the results. Through this assignment, we aim to demonstrate a practical grasp of digital image processing and its applications.

II. Method

In this section, we describe the method adopted for implementing each of the specified image effects: Posterization, Nightvision, Photocopy, Vignetting, and Mirror Image.

A. Posterization

Posterization is a technique that reduces the number of colors in an image to create a simplified and stylized effect. To implement posterization, we divide the intensity range of the image into a fixed number of levels. Each pixel's intensity value is then quantized to the nearest level, resulting in a reduced-color output image. The following steps outline the methodology for posterization:

1) Load the input image.
2) Determine the desired number of intensity levels (*num_levels*).
3) Calculate the quantized value for each pixel based on its intensity and *num_levels*.
4) Assign the quantized value to the corresponding color channel of the output image.

B. Nightvision

The Nightvision effect simulates the green hue commonly associated with night vision devices. This effect involves enhancing the green channel of the image while reducing the intensity of other channels. Here's the methodology for implementing the Nightvision effect:

1) Load the input image.
2) Extract the individual red, green, and blue channels.
3) Enhance the green channel by dividing its intensity values by 2.
4) Multiply the red channel by 2 and the blue channel by 4 to create the night vision green hue.
5) Combine the modified channels to create the final Nightvision image.

C. Photocopy

The Photocopy effect converts an image to grayscale and applies a threshold-based transformation to create a sketch-like appearance. The methodology for implementing the Photocopy effect is as follows:

1) Load the input image.
2) Convert the image to grayscale.
3) Set a threshold value.
4) Iterate through each pixel:
   - If the pixel intensity is above the threshold, set the output pixel value to 255.
   - Otherwise, calculate the adjusted pixel value based on the threshold and input pixel intensity.

D. Vignetting

Vignetting creates a gradual darkening towards the corners of the image, simulating an effect observed in photography. The methodology for implementing the Vignetting effect is outlined as follows:

1) Load the input image.
2) Calculate the center pixel and maximum distance from the center to any corner.
3) For each pixel, calculate the distance from the center and darkness weight.
4) Multiply the input pixel value by the darkness weight to create the vignette effect.

E. Mirror Image

The Mirror Image effect flips the image horizontally by reversing the order of pixels in each row. The methodology for implementing the Mirror Image effect is as follows:

1) Load the input image.
2) Iterate through each row:
   - Reverse the order of pixels in the row to create the mirror effect.

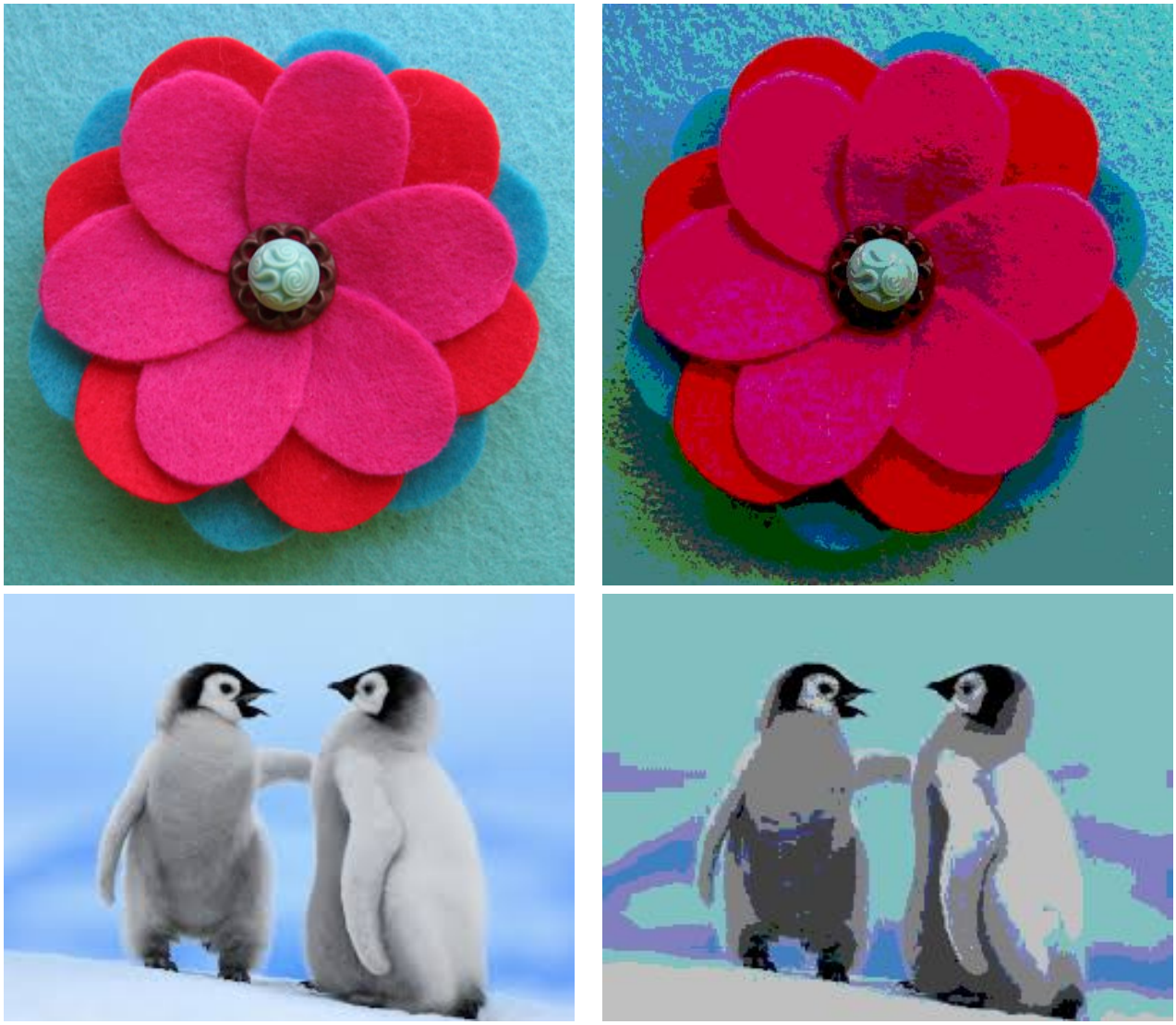
In the following sections, we will provide the MATLAB code for each effect and present the results along with relevant explanations and visualizations.

III. Experiment

In this section, we present the experimental results of implementing the specified image effects: Posterization, Nightvision, Photocopy, Vignetting, and Mirror Image. For each effect, we provide an explanation of each step, and visualizations of the input and final output images.
We provide the MATLAB code used for implementation in section V.

A. Posterize

To implement the Posterize effect, we followed the methodology outlined in II-A. We loaded the input image and applied the quantization process to reduce the number of colors in the image. We experimented with different values of *num_levels* to observe the impact on the image's appearance. As we increased the number of levels, the image transitioned towards a more cartoon-like appearance with reduced color variations. Conversely, reducing the number of levels led to a more stylized and simplified visual effect. Figure 1 showcases the transition from the input image to the final Posterization output.

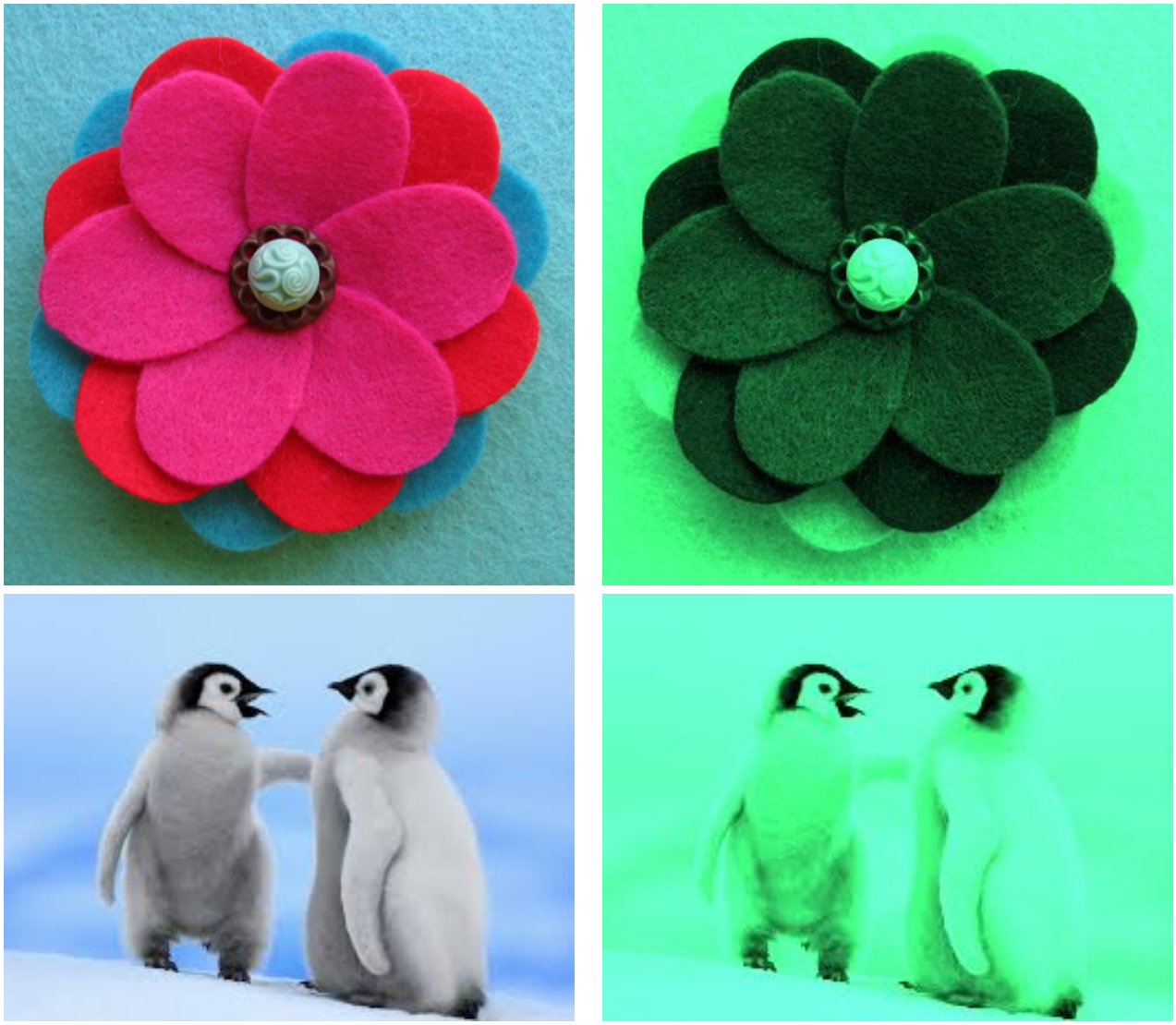


(a) Input Image (b) Posterization Output

Fig. 1: Transition from input to Posterization effect

B. Nightvision

The Nightvision effect was implemented following the methodology described in II-B. We loaded the input image and applied enhancements to the green channel while adjusting the red and blue channels. The result was a visually striking green hue that simulated the characteristic appearance of night vision. We observed that increasing the enhancement of the green channel intensified the night vision effect, creating a distinct and recognizable green-toned output. Figure 2 showcases the transition from the input image to the final night vision output.



(a) Input Image (b) Nightvision Image

Fig. 2: Transition from input to Night vision effect

C. *Photocopy*

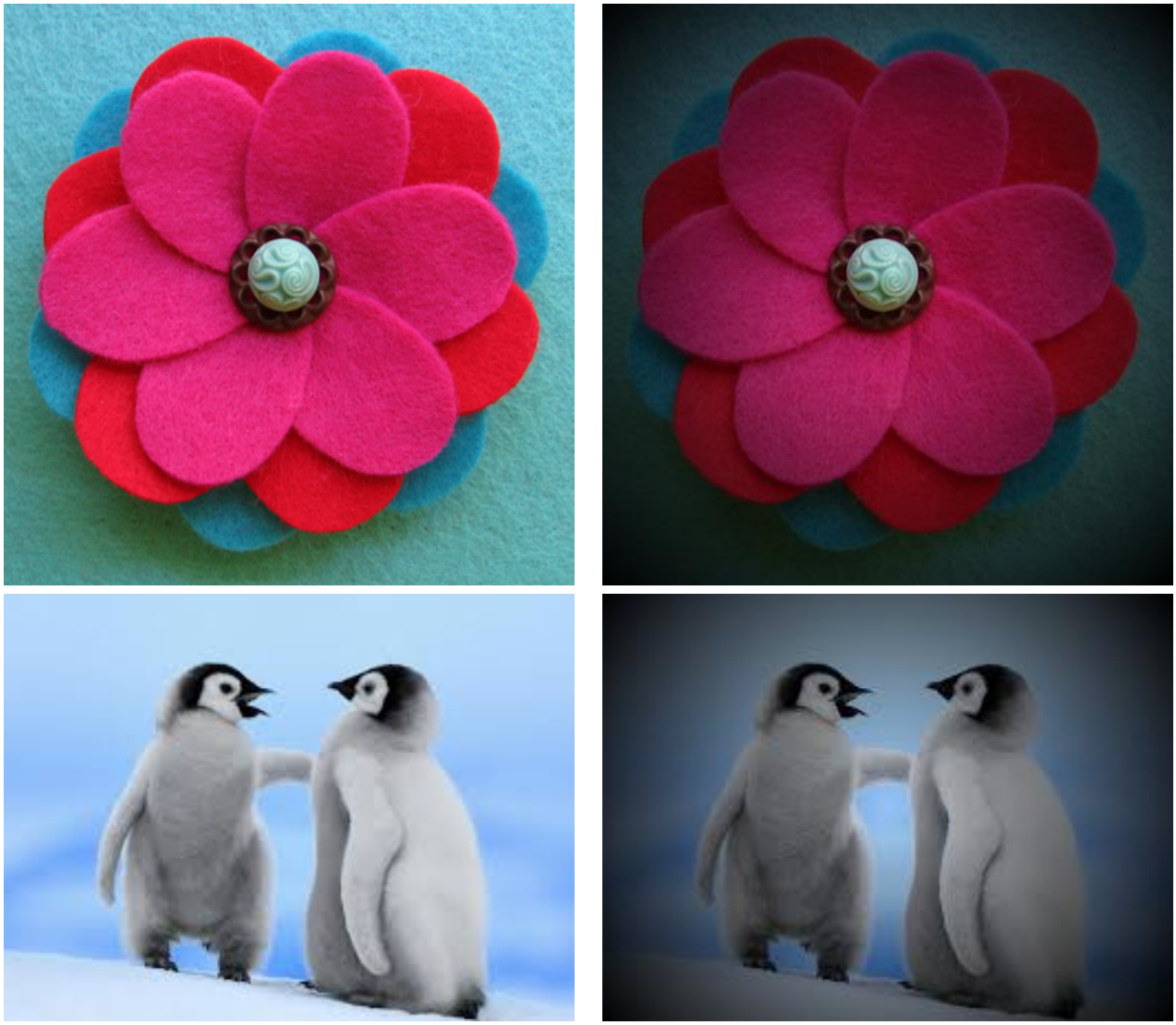
We implemented the Photocopy effect as outlined in II-C. We converted the input image to grayscale and applied a threshold-based transformation to create a sketch-like appearance. Experimenting with different threshold values, we noticed that higher thresholds resulted in more pronounced line art-like outputs, while lower thresholds retained more grayscale detail. This effect successfully simulated the appearance of photocopy. Figure 3 showcases the transition from the input image to the final photocopy output.

(a) Input Image (b) Photocopy Image

Fig. 3: Transition from input to Photocopy effect

D. Vignetting

Following the methodology in II-D, we applied the Vignetting effect to the input image. We calculated the center pixel and maximum distance and used darkness weights to create a gradual darkening towards the corners. Experimenting with different images, we observed that images with more uniform backgrounds displayed a more noticeable vignette effect. The Vignetting effect successfully added an artistic touch to the images, drawing attention to the center. Figure 4 showcases the transition from the input image to the final vignetting output.

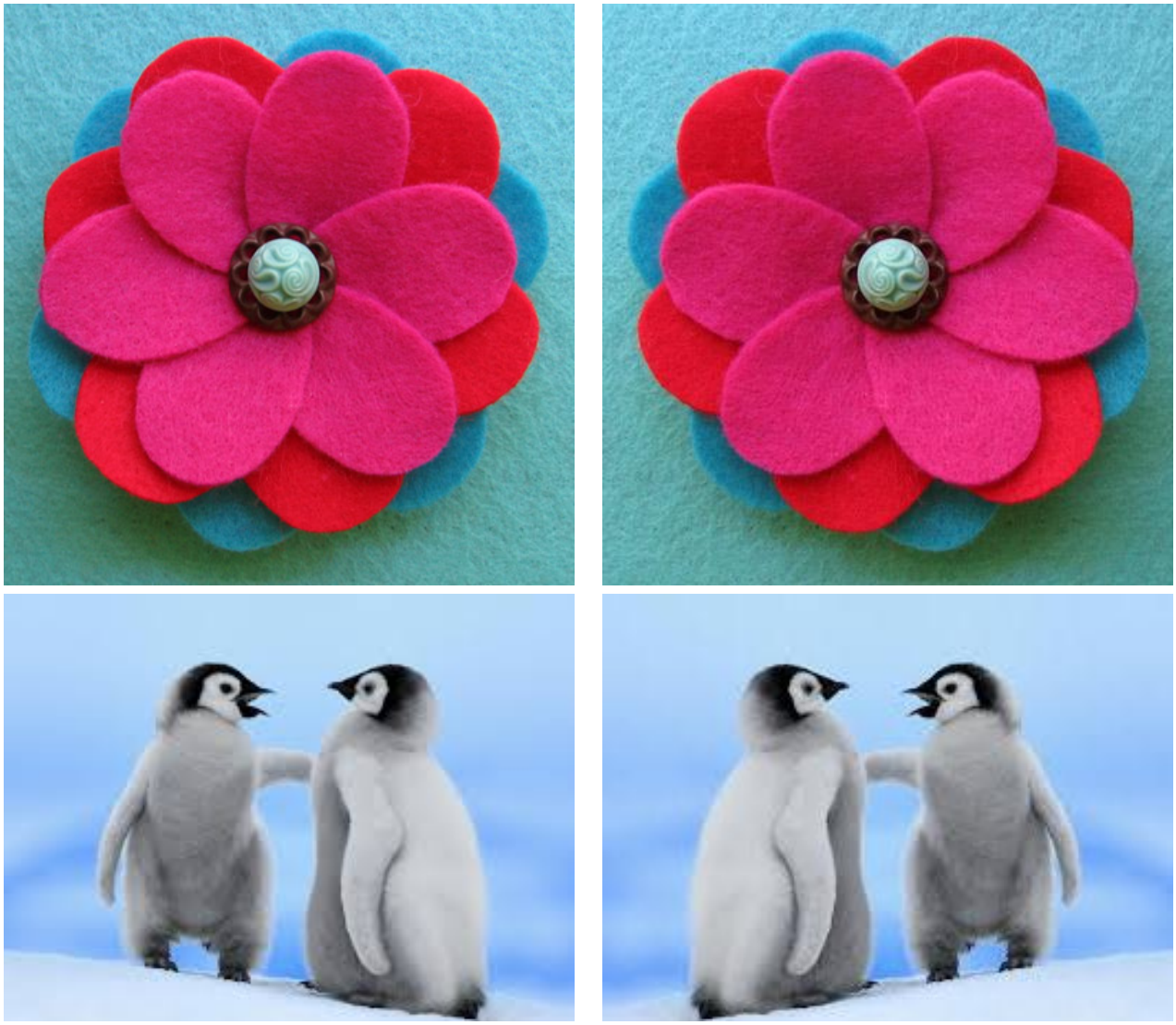


(a) Input Image (b) Vignetting Image

Fig. 4: Transition from input to Vignetting effect

E. Mirror Image

To create the Mirror Image effect, we adopted the process detailed in II-E. We reversed the order of pixels in each row to create a horizontally flipped version of the input image. We found that this effect worked particularly well with symmetrical images or objects. By observing the mirror images of various inputs, we noted how this effect provided an interesting visual transformation that was often quite distinct from the original. Figure 5 showcases the transition from the input image to the final vignetting output



(a) Input Image (b) Mirror Image

Fig. 5: Transition from input to Mirror effect

In the next section, we present the MATLAB code for each effect, along with the results showcasing the transition from the input image to the final output image.

IV. Conclusion

The experimentation with various image effects underscores the versatility and power of digital image processing techniques. Through this assignment, we gained hands-on experience in implementing these effects and deepened our understanding of their underlying concepts.

V. Appendix

Code for Posterize:

```
clc
clear all
close all

% load the image
input_image = imread('input/flower.jpg');

% Define the number of levels for posterization
```

```
num_levels = 4;

% Get the size of the image
[height, width, channels] = size(input_image);

% Initialize the posterized image
posterized_image = zeros(size(input_image));

% Calculate the step size for each color level
step_size = 256 / num_levels;

% Loop through each pixel
for y = 1:height
    for x = 1:width
        for c = 1:channels
            % Get the original pixel value
            pixel_value = double(input_image(y, x, c));

            % Calculate the quantized value
            quantized_value = floor(pixel_value / step_size) * step_size;

            % Assign the quantized value to the color channel
            posterized_image(y, x, c) = quantized_value;
        end
    end
end

posterized_image = uint8(posterized_image);

% Display and save the result
subplot(1,2, 1), imshow(input_image), title('Input Image');
subplot(1, 2, 2), imshow(posterized_image), title('Posterize Image');
imwrite(posterized_image, 'output/flower_posterized_image.jpg');
```

Code for Nightvision:

```

% Load the image
input_image = imread('input/flower.jpg');

% Extract the individual color channels
red_channel = input_image(:, :, 1);
green_channel = input_image(:, :, 2);
blue_channel = input_image(:, :, 3);


% Display the input image
subplot(2, 3, 1), imshow(input_image), title('Input Image');

% Apply Nightvision transformation
output_red = green_channel / 2;

% Combine the modified channels to create the intermediate nightvision image
intermediate_nightvision_image = cat(3, output_red, green_channel, blue_channel);

% Display the intermediate nightvision image
subplot(2, 3, 2), imshow(intermediate_nightvision_image), title('Intermediate - Green
    Enhanced');

```

```
% Apply Nightvision transformation
output_blue = 2 * output_red;

% Combine the modified channels to create the intermediate nightvision image
intermediate_nightvision_image = cat(3, output_red, output_blue, blue_channel);

% Display the intermediate nightvision image
subplot(2, 3, 3), imshow(intermediate_nightvision_image), title('Intermediate - Blue
    Enhanced');

% Apply Nightvision transformation
output_green = 2 * output_blue;

% Combine the modified channels to create the final nightvision image
nightvision_image = cat(3, output_red, output_green, output_blue);

% Display the final nightvision image
subplot(2, 3, 4), imshow(nightvision_image), title('Final Nightvision Image');

% Save the result
imwrite(nightvision_image, 'output/flower_nightvision_image.jpg');
```

Code for Photocopy:

```

% Load the image
input_image = imread('input/flower.jpg');

% Convert the image to grayscale
gray_image = rgb2gray(input_image);

% Set the threshold value
threshold = 100;

% Initialize the output image
output_image = zeros(size(gray_image));

% Apply Photocopy effect
for y = 1:size(gray_image, 1)
    for x = 1:size(gray_image, 2)
        if gray_image(y, x) > threshold
            output_image(y, x) = 255;
        else
            output_image(y, x) = gray_image(y, x) * (threshold - gray_image(y, x)) /
                (threshold^2);
        end
    end
end

photo_copy_image = uint8(output_image);

% Display  the result
subplot(1,3, 1), imshow(input_image), title('Input Image');
subplot(1, 3, 2), imshow(gray_image), title('Grayscale Image');
subplot(1, 3, 3), imshow(photo_copy_image), title('Photocopy Image');

% Save  the result
imwrite(output_image, 'output/photocopy_image.jpg');
```

Code for Vignetting:

```

% Load the image
input_image = imread('input/flower.jpg');

% Calculate center pixel and maximum distance
center_x = size(input_image, 2) / 2;
center_y = size(input_image, 1) / 2;
max_distance = sqrt(center_x^2 + center_y^2);

% Initialize the vignette image
vignette_image = zeros(size(input_image));

% Apply Vignetting effect
for row = 1:size(input_image, 1)
    for col = 1:size(input_image, 2)
        distance = sqrt((col - center_x)^2 + (row - center_y)^2);
        darkness_weight = 1 - distance / max_distance;
        vignette_image(row, col, :) = input_image(row, col, :) * darkness_weight;
    end
end

vignette_image = uint8(vignette_image);

% Display the input image
subplot(1, 2, 1), imshow(input_image), title('Input Image');

% Display the vignette image
subplot(1, 2, 2), imshow(vignette_image), title('Vignette Effect');


% Save the result
imwrite(enhanced_vignette_image, 'output/flower_vignette_image.jpg');
```

Code for Mirror Effect:

```

% Load the image
input_image = imread('input/flower.jpg');

% Create the mirror image
mirror_image = zeros(size(input_image));

for row = 1:size(input_image, 1)
    mirror_image(row, :, :) = input_image(row, end:-1:1, :);
end

mirror_image = uint8(mirror_image);

% Display the input image
subplot(1, 2, 1), imshow(input_image), title('Input Image');
% Display the mirror image
subplot(1, 2, 2), imshow(mirror_image), title('Mirror Image');

% Save the result
imwrite(mirror_image, 'output/mirror_image.jpg');
```