Name: Raj Koyani

Reg No:21MIS1017

Subject: DIP

DA:2

- **DATASET(5):Basic Transformation**

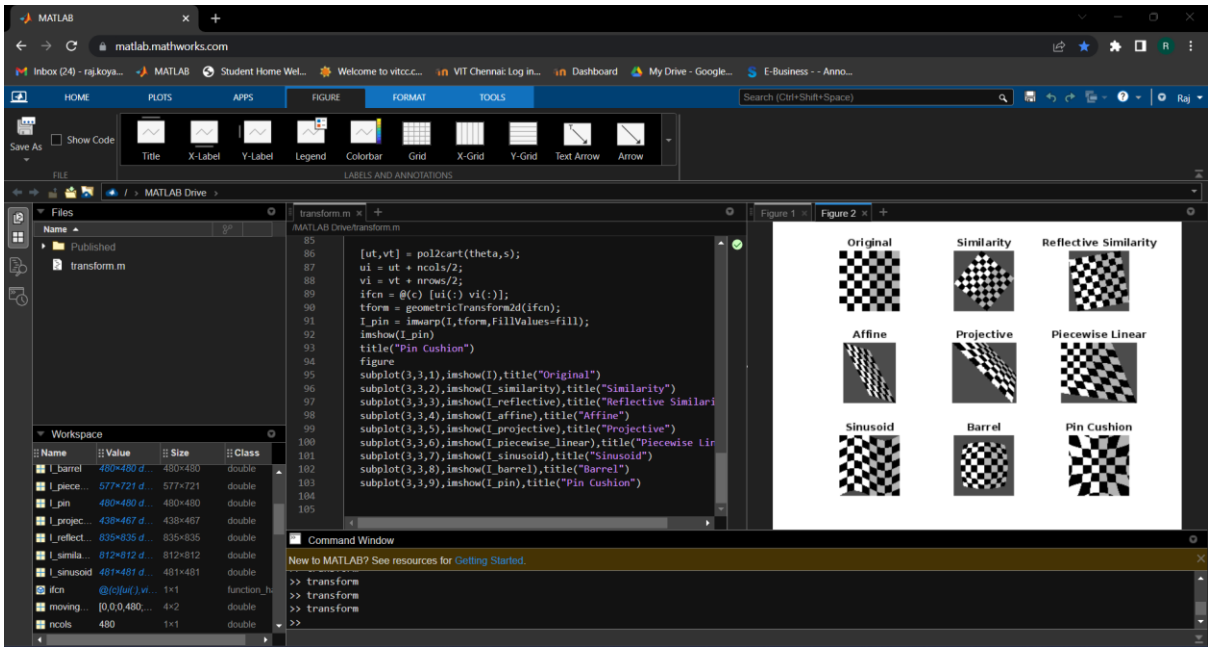
MatLab Code:

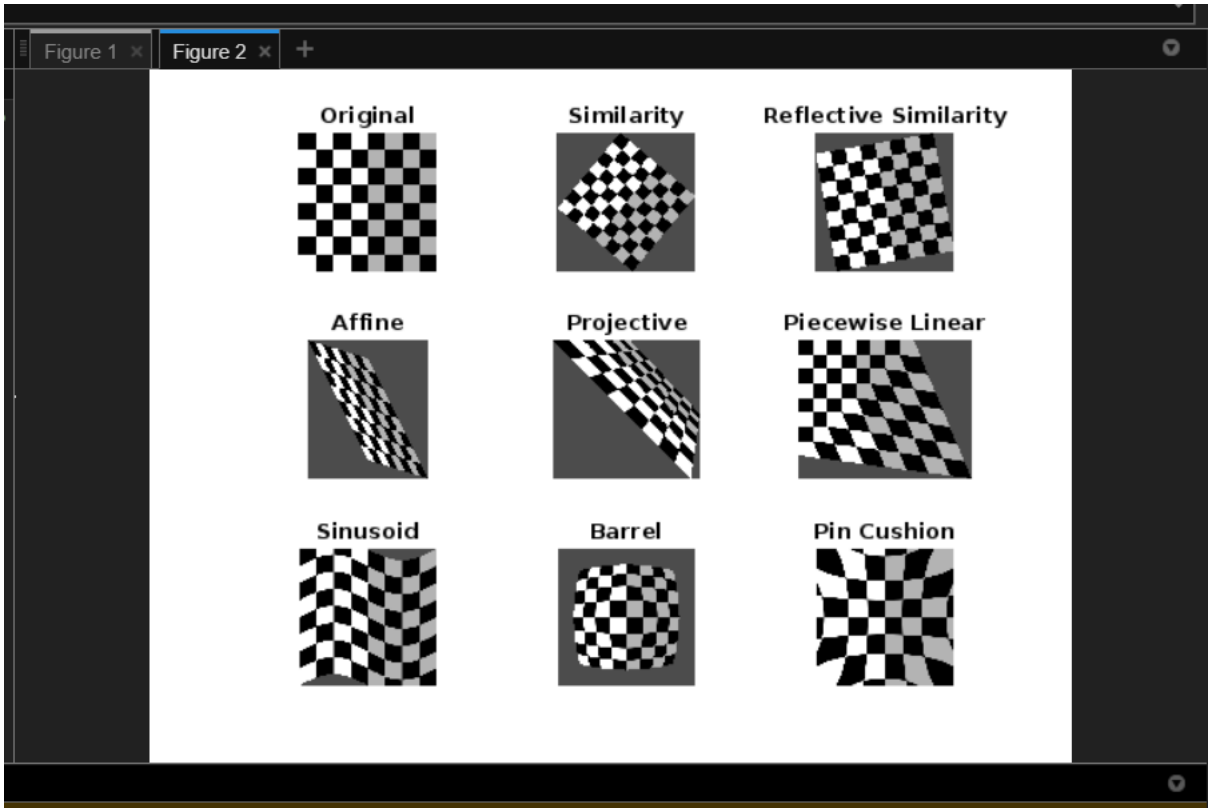
```
sqsize = 60;
I = checkerboard(sqsize,4,4); //Input Image
imshow(I)
title("Original")
nrows = size(I,1);
ncols = size(I,2);
fill = 0.3;
scale = 1.2;
angle = 40;
tx = 0;
ty = 0;
t_sim = simtform2d(scale,angle,[tx ty]);
I_similarity = imwarp(I,t_sim,FillValues=fill);

imshow(I_similarity);
title("Similarity")
[I_similarity,RI] = imwarp(I,t_sim,FillValues=fill);

imshow(I_similarity,RI)
axis on
title("Similarity (Spatially Referenced)")
scale = 1.5;
angle = 10;
tx = 0;
ty = 0;
```

```
r = -1;

sc = scale*cosd(angle);
ss = scale*sind(angle);

A = [ sc  r*ss  tx;
     -ss  r*sc  ty;
       0    0   1];
t_reflective = affinetform2d(A);
[I_reflective,RI] = imwarp(I,t_reflective,FillValues=fill);
imshow(I_reflective,RI)
axis on
title("Reflective Similarity")
A = [  1  1  0;
     0.3  2  0;
       0  0  1];
t_aff = affinetform2d(A);
I_affine = imwarp(I,t_aff,FillValues=fill);
imshow(I_affine)
title("Affine")
A = [    1       1  0;
         0       1  0;
     0.002  0.0002  1];
t_proj = projtform2d(A);
I_projective = imwarp(I,t_proj,FillValues=fill);
imshow(I_projective)
title("Projective")
movingPoints = [0 0; 0 nrows; ncols 0; ncols nrows;];
fixedPoints  = [0 0; 0 nrows; ncols 0; ncols*1.5 nrows*1.2];
t_piecewise_linear = fitgeotform2d(movingPoints,fixedPoints,"pwl");

I_piecewise_linear = imwarp(I,t_piecewise_linear,FillValues=fill);
imshow(I_piecewise_linear)
title("Piecewise Linear")
a = ncols/12; % Try varying the amplitude of the sinusoid
ifcn = @(xy) [xy(:,1), xy(:,2) + a*sin(2*pi*xy(:,1)/nrows)];
tform = geometricTransform2d(ifcn);

I_sinusoid = imwarp(I,tform,FillValues=fill);
imshow(I_sinusoid);
title("Sinusoid")
[xi,yi] = meshgrid(1:ncols,1:nrows);
xt = xi - ncols/2;
yt = yi - nrows/2;
[theta,r] = cart2pol(xt,yt);
a = 1; % Try varying the amplitude of the cubic term.
rmax = max(r(:));
s1 = r + r.^3*(a/rmax.^2);
[ut,vt] = pol2cart(theta,s1);
ui = ut + ncols/2;
vi = vt + nrows/2;
ifcn = @(c) [ui(:) vi(:)];
tform = geometricTransform2d(ifcn);

I_barrel = imwarp(I,tform,FillValues=fill);
```

```
imshow(I_barrel)
title("Barrel")
b = 0.4; % Try varying the amplitude of the cubic term.
s = r - r.^3*(b/rmax.^2);

[ut,vt] = pol2cart(theta,s);
ui = ut + ncols/2;
vi = vt + nrows/2;
ifcn = @(c) [ui(:) vi(:)];
tform = geometricTransform2d(ifcn);
I_pin = imwarp(I,tform,FillValues=fill);
imshow(I_pin)
title("Pin Cushion")
figure
subplot(3,3,1),imshow(I),title("Original")
subplot(3,3,2),imshow(I_similarity),title("Similarity")
subplot(3,3,3),imshow(I_reflective),title("Reflective Similarity")
subplot(3,3,4),imshow(I_affine),title("Affine")
subplot(3,3,5),imshow(I_projective),title("Projective")
subplot(3,3,6),imshow(I_piecewise_linear),title("Piecewise Linear")
subplot(3,3,7),imshow(I_sinusoid),title("Sinusoid")
subplot(3,3,8),imshow(I_barrel),title("Barrel")
subplot(3,3,9),imshow(I_pin),title("Pin Cushion")
```

output:





Python code:

```
import cv2
import numpy as np

# Create a checkerboard image
sqsize = 60
I = np.kron([[1, 0] * 4, [0, 1] * 4] * 4, np.ones((sqsize, sqsize),
dtype=np.uint8))
cv2.imshow("Original", I)

# Similarity transformation
fill = 0.3
scale = 1.2
angle = 40
tx = 0
ty = 0
t_sim = cv2.getRotationMatrix2D((I.shape[1] / 2, I.shape[0] / 2), angle,
scale)
I_similarity = cv2.warpAffine(I, t_sim, (I.shape[1], I.shape[0]),
borderValue=fill)
```

```
cv2.imshow("Similarity", I_similarity)

# Reflective similarity transformation
scale = 1.5
angle = 10
tx = 0
ty = 0
r = -1
sc = scale * np.cos(np.radians(angle))
ss = scale * np.sin(np.radians(angle))
A = np.array([[sc, r * ss, tx], [-ss, r * sc, ty], [0, 0, 1]])
t_reflective = cv2.warpAffine(I, A[:2], (I.shape[1], I.shape[0]),
borderValue=fill)
cv2.imshow("Reflective Similarity", t_reflective)

# Affine transformation
A = np.array([[1, 1, 0], [0.3, 2, 0], [0, 0, 1]])
t_aff = cv2.warpAffine(I, A[:2], (I.shape[1], I.shape[0]), borderValue=fill)
cv2.imshow("Affine", t_aff)

# Projective transformation
A = np.array([[1, 1, 0], [0, 1, 0], [0.002, 0.0002, 1]])
t_proj = cv2.warpPerspective(I, A, (I.shape[1], I.shape[0]), borderValue=fill)
cv2.imshow("Projective", t_proj)

# Piecewise Linear transformation
movingPoints = np.array([[0, 0], [0, I.shape[0]], [I.shape[1], 0],
[I.shape[1], I.shape[0]]], dtype=np.float32)
fixedPoints = np.array([[0, 0], [0, I.shape[0]], [I.shape[1], 0], [I.shape[1]
* 1.5, I.shape[0] * 1.2]], dtype=np.float32)
t_piecewise_linear = cv2.getPerspectiveTransform(movingPoints, fixedPoints)
I_piecewise_linear = cv2.warpPerspective(I, t_piecewise_linear, (I.shape[1],
I.shape[0]), borderValue=fill)
cv2.imshow("Piecewise Linear", I_piecewise_linear)

# Sinusoid transformation
a = I.shape[1] / 12
t_sinusoid = np.array([[1, 0, 0], [0, 1, a * np.sin(2 * np.pi *
np.arange(I.shape[0]) / I.shape[0])], [0, 0, 1]])
I_sinusoid = cv2.warpAffine(I, t_sinusoid[:2], (I.shape[1], I.shape[0]),
borderValue=fill)
cv2.imshow("Sinusoid", I_sinusoid)

# Barrel and Pin Cushion transformations
xt, yt = np.meshgrid(np.arange(I.shape[1]), np.arange(I.shape[0]))
xt = xt - I.shape[1] / 2
yt = yt - I.shape[0] / 2
theta, r = np.arctan2(yt, xt), np.hypot(xt, yt)
```

```
a = 1
rmax = np.max(r)
s1 = r + r**3 * (a / rmax**2)
ut, vt = r * np.cos(theta), r * np.sin(theta)
ui, vi = ut + I.shape[1] / 2, vt + I.shape[0] / 2
t_barrel = cv2.getPerspectiveTransform(np.float32([xt, yt]).transpose(1, 2,
0), np.float32([ui, vi]).transpose(1, 2, 0))
I_barrel = cv2.warpPerspective(I, t_barrel, (I.shape[1], I.shape[0]),
borderValue=fill)
cv2.imshow("Barrel", I_barrel)

b = 0.4
s = r - r**3 * (b / rmax**2)
ut, vt = r * np.cos(theta), r * np.sin(theta)
ui, vi = ut + I.shape[1] / 2, vt + I.shape[0] / 2
t_pin = cv2.getPerspectiveTransform(np.float32([xt, yt]).transpose(1, 2, 0),
np.float32([ui, vi]).transpose(1, 2, 0))
I_pin = cv2.warpPerspective(I, t_pin, (I.shape[1], I.shape[0]),
borderValue=fill)
cv2.imshow("Pin Cushion", I_pin)

# Display all images together
cv2.imshow("Transformations", np.hstack([
    I, I_similarity, t_reflective, t_aff, t_proj, I_piecewise_linear,
I_sinusoid, I_barrel, I_pin
]))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

output:

Input Image:

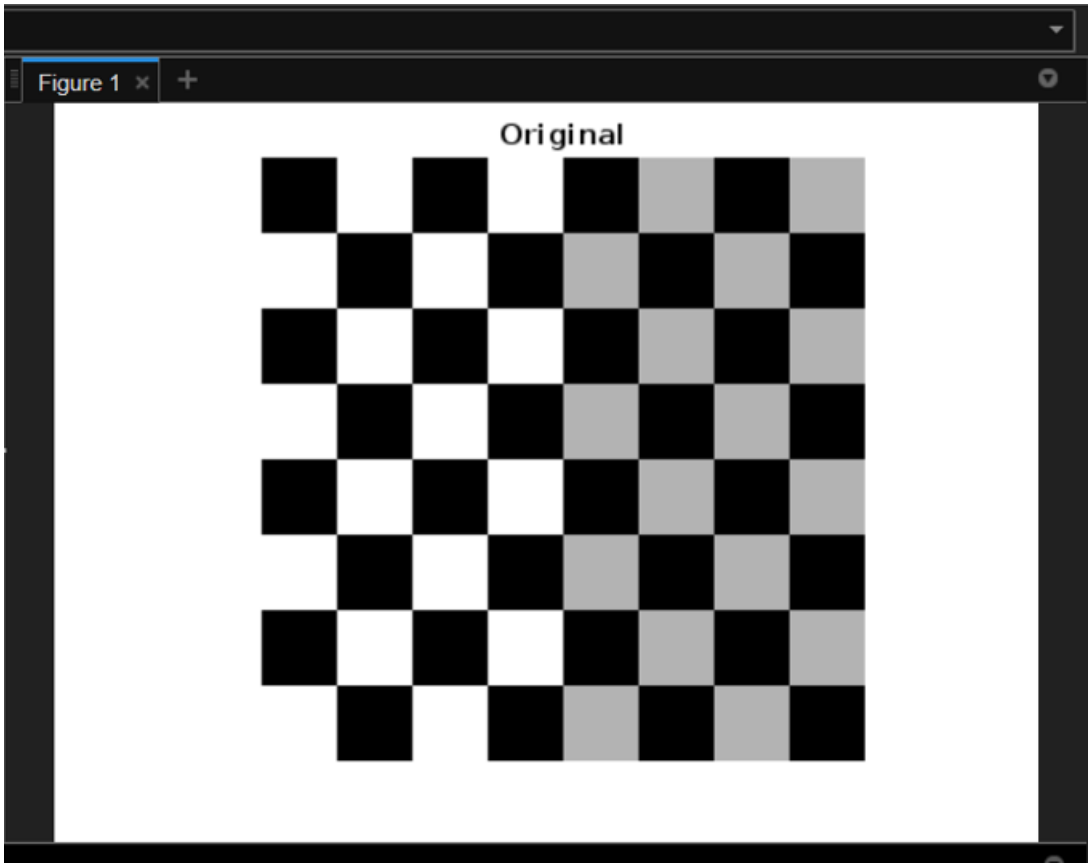


Figure 1
Original

Output:

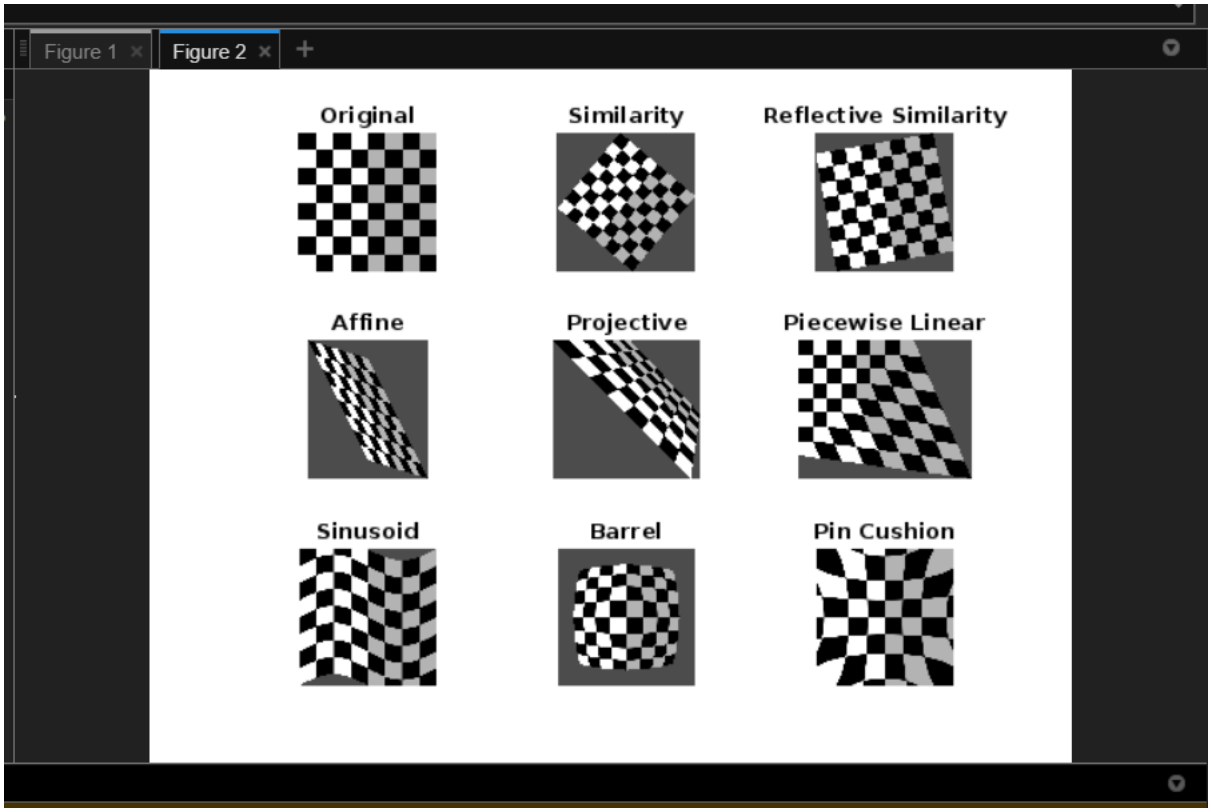
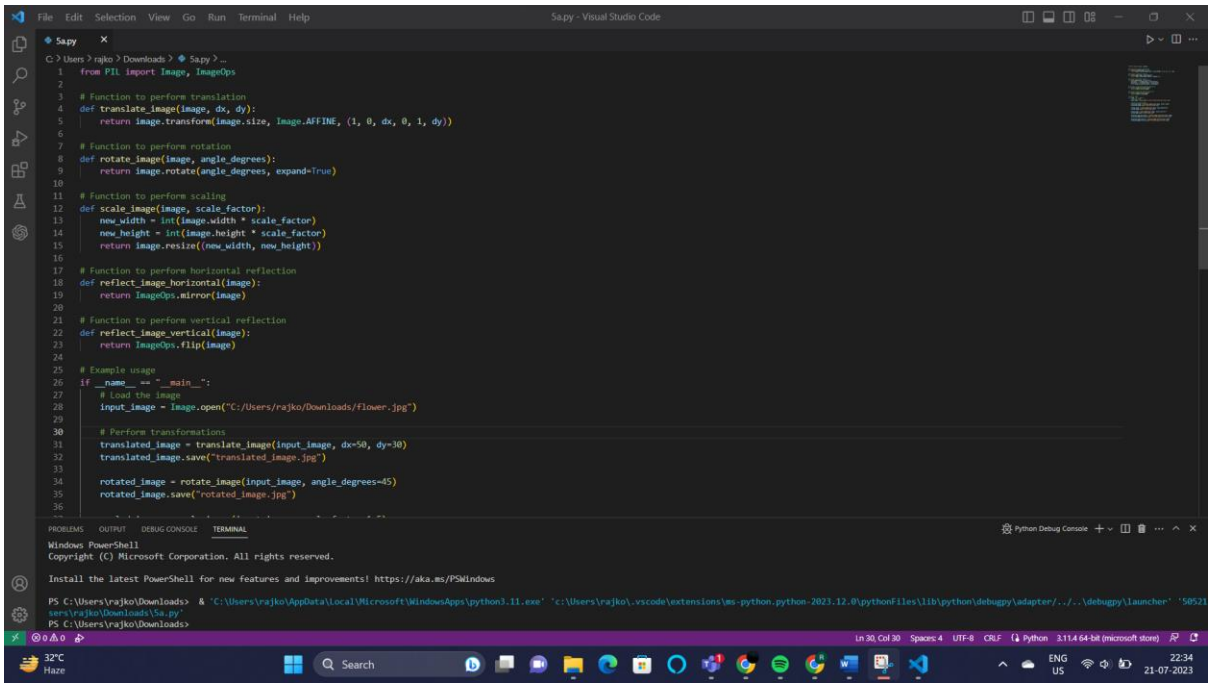


Figure 1
Figure 2
Original
Similarity
Reflective Similarity
Affine
Projective
Piecewise Linear
Sinusoid
Barrel
Pin Cushion

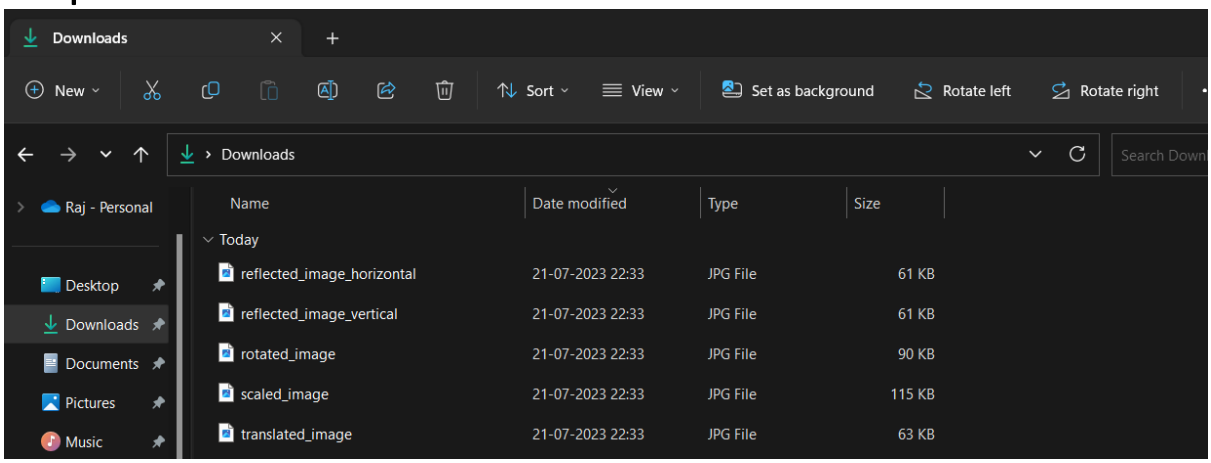
Python code for rotating:

```python
from PIL import Image, ImageOps

# Function to perform translation
def translate_image(image, dx, dy):
    return image.transform(image.size, Image.AFFINE, (1, 0, dx, 0, 1, dy))

# Function to perform rotation
def rotate_image(image, angle_degrees):
    return image.rotate(angle_degrees, expand=True)

# Function to perform scaling
def scale_image(image, scale_factor):
    new_width = int(image.width * scale_factor)
    new_height = int(image.height * scale_factor)
    return image.resize((new_width, new_height))

# Function to perform horizontal reflection
def reflect_image_horizontal(image):
    return ImageOps.mirror(image)

# Function to perform vertical reflection
def reflect_image_vertical(image):
    return ImageOps.flip(image)

# Example usage
if __name__ == "__main__":
    # Load the image
    input_image = Image.open("C:/Users/rajko/Downloads/flower.jpg")

    # Perform transformations
    translated_image = translate_image(input_image, dx=50, dy=30)
    translated_image.save("translated_image.jpg")

    rotated_image = rotate_image(input_image, angle_degrees=45)
    rotated_image.save("rotated_image.jpg")

    scaled_image = scale_image(input_image, scale_factor=1.5)
    scaled_image.save("scaled_image.jpg")

    reflected_image_horizontal = reflect_image_horizontal(input_image)
    reflected_image_horizontal.save("reflected_image_horizontal.jpg")
```

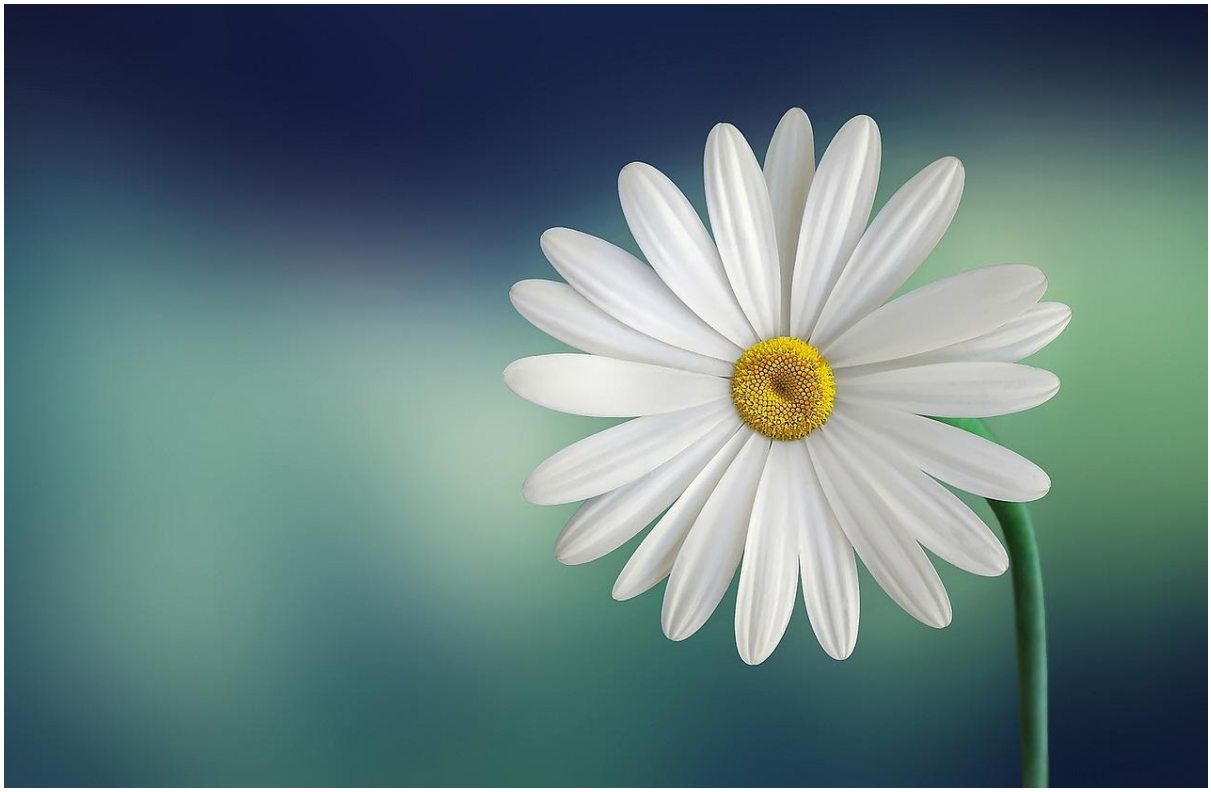
```
reflected_image_vertical = reflect_image_vertical(input_image)
reflected_image_vertical.save("reflected_image_vertical.jpg")
```

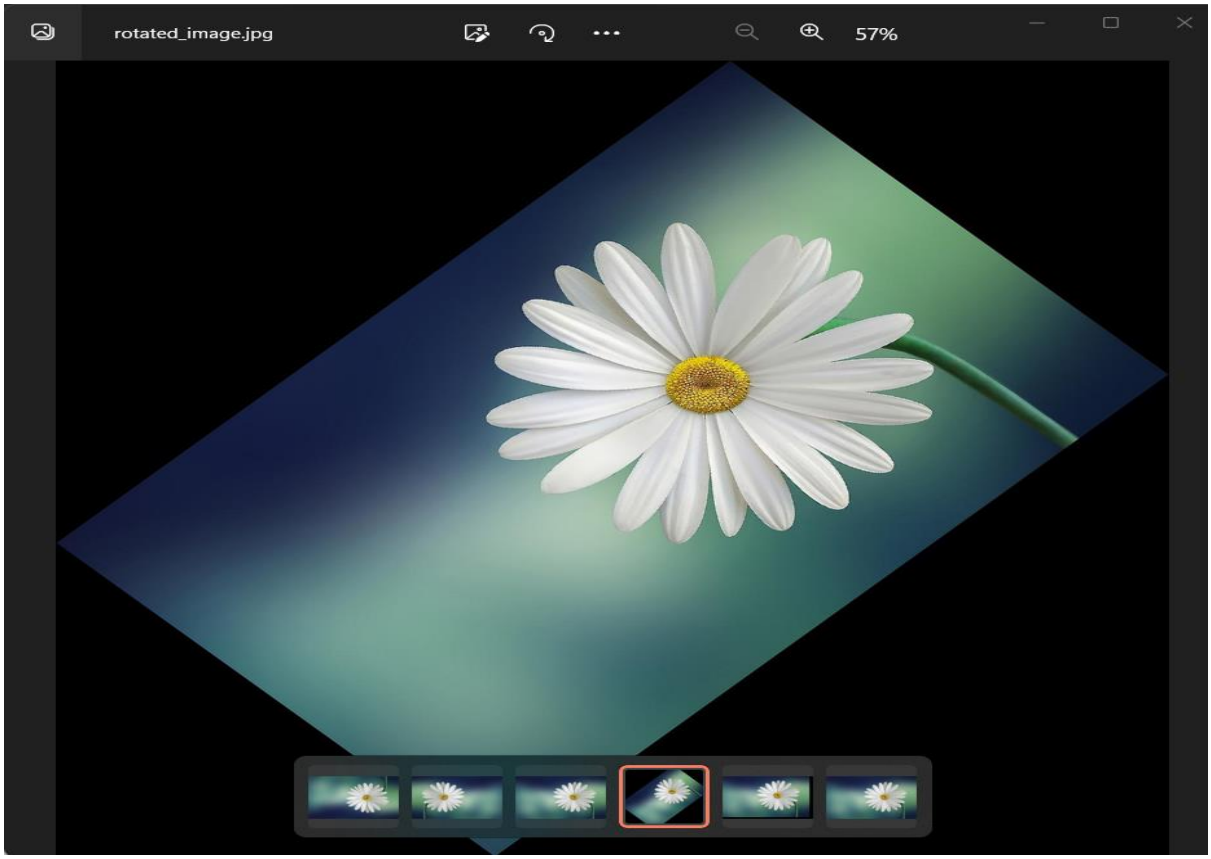


Ouput:



Input Image:




rotated_image.jpg
57%