Dr B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY CSPC-423

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY: SUBMITTED TO:

NAME: YENNI SHASHANK DR. SHVETA MAHAJAN

ROLL NO: 20103167 ASSISTANT PROFESSOR

GROUP: G3 DEPARTMENT OF CSE

INDEX

S.NO.	**AIM**	**DATE**	**REMARKS/ SIGNATURE**
1	Implementation of various line drawing algorithms: DDA, Bresenham's, Midpoint.	11-08-23	
2	Implementation of various circle drawing algorithms: Bresenham's , Midpoint.	25-08-23	
3	Implementation of Midpoint Ellipse drawing Algorithm	01-09-23	
4	Implementation of various line clipping algorithms: Cohen Sutherland, Cyrus beck line clipping.	08-09-23	
5	Implementation of various translation, rotation and scaling techniques in the 2D plane.	29-09-23	
6	Implementation of various composite transformation techniques on an object.	06-10-23	
7	Simulation and Display of an Image, Negative of an image (Binary & Grey Scale).	13-10-23	
8	Computation of Mean, Median, Variance and Standard Deviation of the given Image.	20-10-23	
9	Display of colour images and conversion between colour spaces.	03-11-23	
10	Implement image segmentation using histogram thresholding.	17-11-23	

LAB – 1

Implementation of various line drawing algorithms: DDA, Bresenham's, Midpoint.

1) DDA Line Drawing Algorithm:

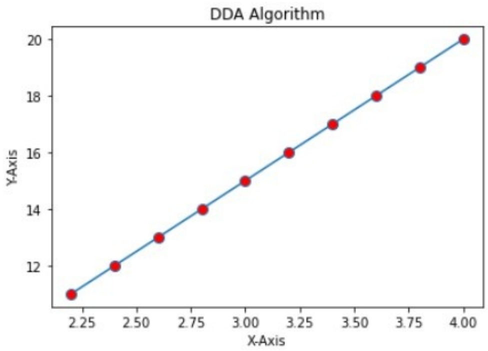
Code:

```
import matplotlib.pyplot as plt
x1 = int(input("Enter the value of x1: "))
y1 = int(input("Enter the value of y1: "))
x2 = int(input("Enter the value of x2: "))
y2 = int(input("Enter the value of y2: "))
dx = x2 - x1
dy = y2 - y1
if abs(dx) > abs(dy):
steps = abs(dx)
else:
steps = abs(dy)
xincrement = dx/steps
yincrement = dy/steps
i = 0
xcoordinates = []
ycoordinates = []
while i < steps:
i +=1
x1 = x1 + xincrement
y1 = y1 + yincrement
print("X1:",x1, " Y1:", y1)
xcoordinates.append(x1)
ycoordinates.append(y1)
plt.plot(xcoordinates, ycoordinates,marker="o", markersize=8, markerfacecolor="red")
plt.xlabel("X-Axis") #Naming the Axis
plt.ylabel("Y-Axis")
plt.title("DDA Algorithm") #Graph title
```

plt.show() #show the plot

Output:

```
Enter the value of x1: 2
Enter the value of y1: 10
Enter the value of x2: 4
Enter the value of y2: 20
X1: 2.2    Y1: 11.0
X1: 2.4000000000000004    Y1: 12.0
X1: 2.6000000000000005    Y1: 13.0
X1: 2.8000000000000007    Y1: 14.0
X1: 3.000000000000001    Y1: 15.0
X1: 3.200000000000001    Y1: 16.0
X1: 3.4000000000000012    Y1: 17.0
X1: 3.6000000000000014    Y1: 18.0
X1: 3.8000000000000016    Y1: 19.0
X1: 4.000000000000002    Y1: 20.0
```



2) Bresenham's Line Drawing Algorithm:

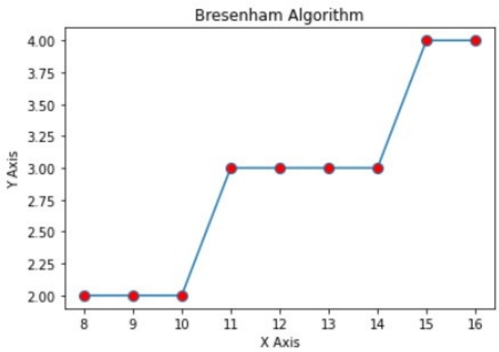
Code:

```
import matplotlib.pyplot as plt
plt.title("Bresenham Algorithm")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
def bres(x1,y1,x2,y2):
x,y = x1,y1
dx = abs(x2 - x1)
dy = abs(y2 -y1)
gradient = dy/float(dx)
if gradient > 1:
dx, dy = dy, dx
x, y = y, x
x1, y1 = y1, x1
x2, y2 = y2, x2
p = 2*dy - dx
print(f"x = {x}, y = {y}")
# Initialize the plotting points
xcoordinates = [x]
ycoordinates = [y]
for k in range(2, dx + 2):
if p > 0:
y = y + 1 if y < y2 else y - 1
p = p + 2 * (dy - dx)
else:
p = p + 2 * dy
x = x + 1 if x < x2 else x - 1
print(f"x = {x}, y = {y}")
xcoordinates.append(x)
ycoordinates.append(y)
plt.plot(xcoordinates, ycoordinates,marker="o", markersize=8, markerfacecolor="red")
plt.show()
def main():
x1 = int(input("Enter the Starting point of x: "))
y1 = int(input("Enter the Starting point of y: "))
```

```
x2 = int(input("Enter the end point of x: "))
y2 = int(input("Enter the end point of y: "))
bres(x1, y1, x2, y2)
if __name__ == "__main__":
main()
```

Output:

```
Enter the Starting point of x: 2
Enter the Starting point of y: 8
Enter the end point of x: 4
Enter the end point of y: 16
x = 8, y = 2
x = 9, y = 2
x = 10, y = 2
x = 11, y = 3
x = 12, y = 3
x = 13, y = 3
x = 14, y = 3
x = 15, y = 4
x = 16, y = 4
```



3) Midpoint Line Drawing Algorithm:

Code:

```
import matplotlib.pyplot as plt
plt.title("Midpoint Line Algorithm")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
def midpoint(x1, y1, x2, y2):
dx = x2 - x1
dy = y2 - y1
# Initialize the decision parameter
d = dy - (dx/2)
x = x1
y = y1
print(f"x = {x}, y = {y}")
# Initialize the plotting points
xcoordinates = [x]
ycoordinates = [y]
while (x<x2):
x = x + 1
# East is Chosen
if (d<0):
d = d + dy
# North East is Chosen
else:
d = d + (dy - dx)
y = y + 1
xcoordinates.append(x)
ycoordinates.append(y)
print(f"x = {x}, y = {y}")
plt.plot(xcoordinates, ycoordinates,marker="o", markersize=8, markerfacecolor="red")
plt.show()
if __name__=="__main__":
```

x1 = int(input("Enter the starting point of x: "))

y1 = int(input("Enter the starting point of y: "))

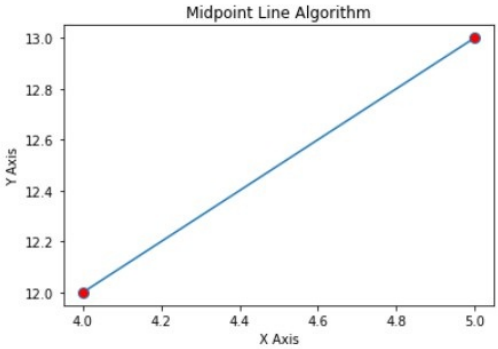
x2 = int(input("Enter the end point of x: "))

y2 = int(input("Enter the end point of y: "))

midpoint(x1, y1, x2, y2)

Output:

```
Enter the starting point of x: 4
Enter the starting point of y: 12
Enter the end point of x: 5
Enter the end point of y: 15
x = 4, y = 12
x = 5, y = 13
```



LAB – 2

Implementation of various circle drawing algorithms: Bresenham's, Midpoint.

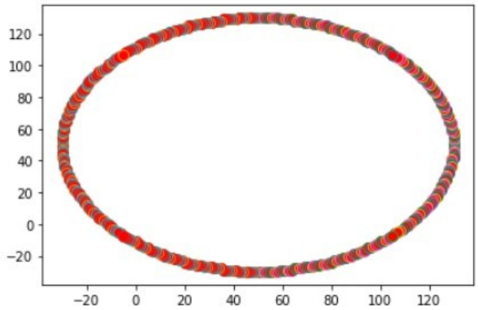
1) Bresenham's Circle Drawing Algorithm:

Code:

```
import matplotlib.pyplot as plt

def draw(xc,yc,x,y):

plt.plot(xc+x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc+x, yc-y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc-y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc+y, yc+x, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-y, yc+x, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc+y, yc-x, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-y, yc-x, marker="o", markersize=8, markerfacecolor="red")

def circle(xc,yc,r):

x=0

y=r

d=3-2*r;

draw(xc,yc,x,y);

while(y>=x):

x=x+1;

if(d>0):

y=y-1;

d=d+4*(x-y)+10;

else:

d=d+4*x+6;

draw(xc,yc,x,y);

x0 = int(input("Enter x0: "))

y0 = int(input("Enter y0: "))

radius = int(input("Enter Radius : "))

circle(x0,y0,radius)

plt.show()
```

Output:

```
Enter x0: 50
Enter y0: 50
Enter Radius : 80
```



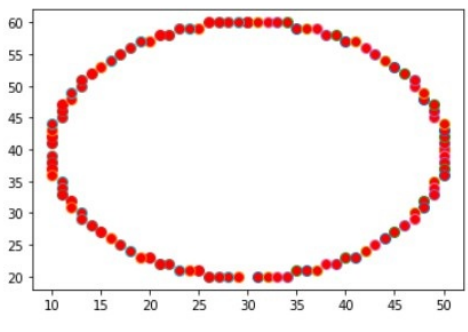
2) Midpoint Circle Drawing Algorithm:

Code:

```
import matplotlib.pyplot as plt
def midpoint(xc,yc,r):
x=r
y=0
plt.plot(xc+x, yc+y, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc+x, yc-y, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc+y, yc+x, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc-y, yc+x, marker="o", markersize=8, markerfacecolor="red")
p=1-r
while x>y:
y+=1
if p<=0:
p=p+2*y+1
else:
x-=1
p=p+2*y-2*x+1
if(x<y):
break
plt.plot(xc+x, yc+y, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc-x, yc+y, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc+x, yc-y, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc-x, yc-y, marker="o", markersize=8, markerfacecolor="red")
if(x!=y):
plt.plot(xc+y, yc+x, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc-y, yc+x, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc+y, yc-x, marker="o", markersize=8, markerfacecolor="red")
plt.plot(xc-y, yc-x, marker="o", markersize=8, markerfacecolor="red")
x0 = int(input("Enter x0: "))
y0 = int(input("Enter y0: "))
radius = int(input("Enter Radius : "))
midpoint(x0, y0, radius)
```

Output:

```
Enter x0: 30
Enter y0: 40
Enter Radius : 20
```



LAB – 3

Implementation of Mid Point Ellipse drawing Algorithm

Mid Point Ellipse Drawing Algorithm:

Code:

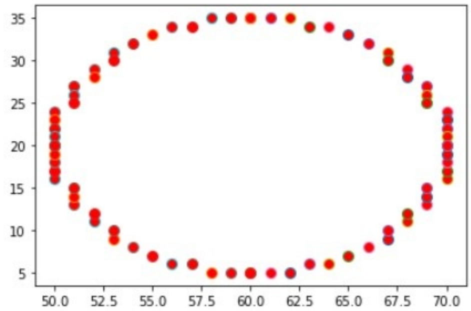
```
import matplotlib.pyplot as plt

def midptellipse(rx, ry, xc, yc):

x = 0;

y = ry;

# Initial decision parameter of region 1

d1 = ((ry * ry) - (rx * rx * ry) + (0.25 * rx * rx));

dx = 2 * ry * ry * x;

dy = 2 * rx * rx * y;

# For region 1

while (dx < dy): # Print points based on 4-way symmetry

plt.plot(xc+x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc+x, yc-y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc-y, marker="o", markersize=8, markerfacecolor="red")

# Checking and updating value of

# decision parameter based on algorithm

if (d1 < 0):

x += 1;

dx = dx + (2 * ry * ry);

d1 = d1 + dx + (ry * ry);

else:

x += 1;

y -= 1;

dx = dx + (2 * ry * ry);

dy = dy - (2 * rx * rx);

d1 = d1 + dx - dy + (ry * ry);

# Decision parameter of region 2

d2 = (((ry * ry) * ((x + 0.5) * (x + 0.5))) + ((rx * rx) * ((y - 1) * (y - 1))) - (rx * rx * ry * ry));

# Plotting points of region 2

while (y >= 0):

# printing points based on 4-way symmetry

plt.plot(xc+x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc+y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc+x, yc-y, marker="o", markersize=8, markerfacecolor="red")

plt.plot(xc-x, yc-y, marker="o", markersize=8, markerfacecolor="red")

# Checking and updating parameter

# value based on algorithm

if (d2 > 0):

y -= 1;

dy = dy - (2 * rx * rx);

d2 = d2 + (rx * rx) - dy;

else:

y -= 1;

x += 1;

dx = dx + (2 * ry * ry);

dy = dy - (2 * rx * rx);

d2 = d2 + dx - dy + (rx * rx);

rx = int(input("Enter radius along x-axis: "))

ry = int(input("Enter radius along y-axis: "))

xc = int(input("Enter x coordinate of center: "))
```

yc = int(input("Enter y coordinate of center: "))

midptellipse(rx, ry, xc, yc);

Output:

```
Enter radius along x-axis: 10
Enter radius along y-axis: 15
Enter x coordinate of center: 60
Enter y coordinate of center: 20
```



LAB – 4

Implementation of various line clipping algorithms: Cohen Sutherland, Cyrus beck line clipping.

1) Cohen Sutherland Line Clipping Algorithm:

Code:

```
# Defining region codes

INSIDE = 0 # 0000

LEFT = 1 # 0001

RIGHT = 2 # 0010

BOTTOM = 4 # 0100

TOP = 8 # 1000

# Defining x_max, y_max and x_min, y_min for rectangle

#Since diagonal points are enough to define a rectangle

x_max = 10.0

y_max = 8.0

x_min = 4.0

y_min = 4.0

# Function to compute region code for a point(x, y)

def computeCode(x, y):

code = INSIDE

if x < x_min: # to the left of rectangle

code |= LEFT

elif x > x_max: # to the right of rectangle

code |= RIGHT

if y < y_min: # below the rectangle

code |= BOTTOM

elif y > y_max: # above the rectangle

code |= TOP

return code

# Implementing Cohen-Sutherland algorithm

# Clipping a line from P1 = (x1, y1) to P2 = (x2, y2)

def cohenSutherlandClip(x1, y1, x2, y2):

# Compute region codes for P1, P2

code1 = computeCode(x1, y1)

code2 = computeCode(x2, y2)

accept = False

while True:

# If both endpoints lie within rectangle

if code1 == 0 and code2 == 0:

accept = True

break

# If both endpoints are outside rectangle
```

```
elif (code1 & code2) != 0:
break
# Some segment lies within the rectangle
else:
# Line needs clipping
# At least one of the points is outside,
# select it
x = 1.0
y = 1.0
if code1 != 0:
code_out = code1
else:
code_out = code2
# Find intersection point
# using formulas y = y1 + slope * (x - x1),
# x = x1 + (1 / slope) * (y - y1)
if code_out & TOP:
# Point is above the clip rectangle
x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
y = y_max
elif code_out & BOTTOM:
# Point is below the clip rectangle
x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
y = y_min
elif code_out & RIGHT:
# Point is to the right of the clip rectangle
y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
x = x_max
elif code_out & LEFT:
# Point is to the left of the clip rectangle
y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
x = x_min
# Now intersection point (x, y) is found
# We replace point outside clipping rectangle
# by intersection point
if code_out == code1:
x1 = x
y1 = y
code1 = computeCode(x1, y1)
else:
x2 = x
y2 = y
code2 = computeCode(x2, y2)
if accept:
print("Line accepted from %.2f, %.2f to %.2f, %.2f" % (x1, y1, x2, y2))
# Here the user can add code to display the rectangle
# along with the accepted (portion of) lines
else:
print("Line rejected")
cohenSutherlandClip(2, 1, 9, 8)
cohenSutherlandClip(11, 11, 5, 9)
cohenSutherlandClip(1, 6, 7, 3)
```

Output:

```
Line accepted from 5.00, 4.00 to 9.00, 8.00
Line rejected
Line accepted from 4.00, 4.50 to 5.00, 4.00
```

2) Cyrus Beck Line Clipping Algorithm:

Code:

```
import numpy as np

# Define the polygon as a list of vertices in clockwise order

polygon = [(100, 100), (200, 50), (300, 100), (250, 200), (150, 200)]

def cyrus_beck_line_clip(p1, p2, polygon):

def normalize(vector):

return vector / np.linalg.norm(vector)

def dot_product(a, b):

return sum(x * y for x, y in zip(a, b))

def clip_t(p, d):

num = dot_product(polygon[0] - p1, d)

den = dot_product(p2 - p1, d)

if den == 0:

if num < 0:

return float('-inf'), float('inf')

else:

return float('-inf'), float('-inf')

t = num / den

return t, t

n = len(polygon)

d = np.array([0, 0])

for i in range(n):

e1 = np.array(polygon[(i + 1) % n]) - np.array(polygon[i])

e2 = p1 - np.array(polygon[i])

ni = np.array([-e1[1], e1[0]]) # Normal vector of the polygon edge

if dot_product(e1, d) == 0:

# Line is parallel to this edge, check if it's outside or inside

if dot_product(e2, ni) < 0:

return None # Line is outside the polygon

else:

t1, t2 = clip_t(np.array(polygon[i]), d)

t3, t4 = clip_t(np.array(polygon[i]) + e1, d)

t1, t2 = max(t1, t3), min(t2, t4)

if t1 > t2:

return None # Line is outside the polygon

else:

d = normalize(e1)

p1 = np.array(p1) + t1 * (np.array(p2) - np.array(p1))

p2 = np.array(p1) + (t2 - t1) * (np.array(p2) - np.array(p1))

return tuple(p1), tuple(p2)

p1 = (18, 36)

p2 = (54, 27)

# Clip the line segment against the polygon

clipped_line = cyrus_beck_line_clip(p1, p2, polygon)

if clipped_line:

print(f"Clipped Line: {clipped_line}")

else:

print("Line is completely outside the polygon, rejected.")
```

Output:

```
Line is completely outside the polygon, rejected.
```

LAB – 5

Implementation of various translation, rotation and scaling techniques in the 2D plane.

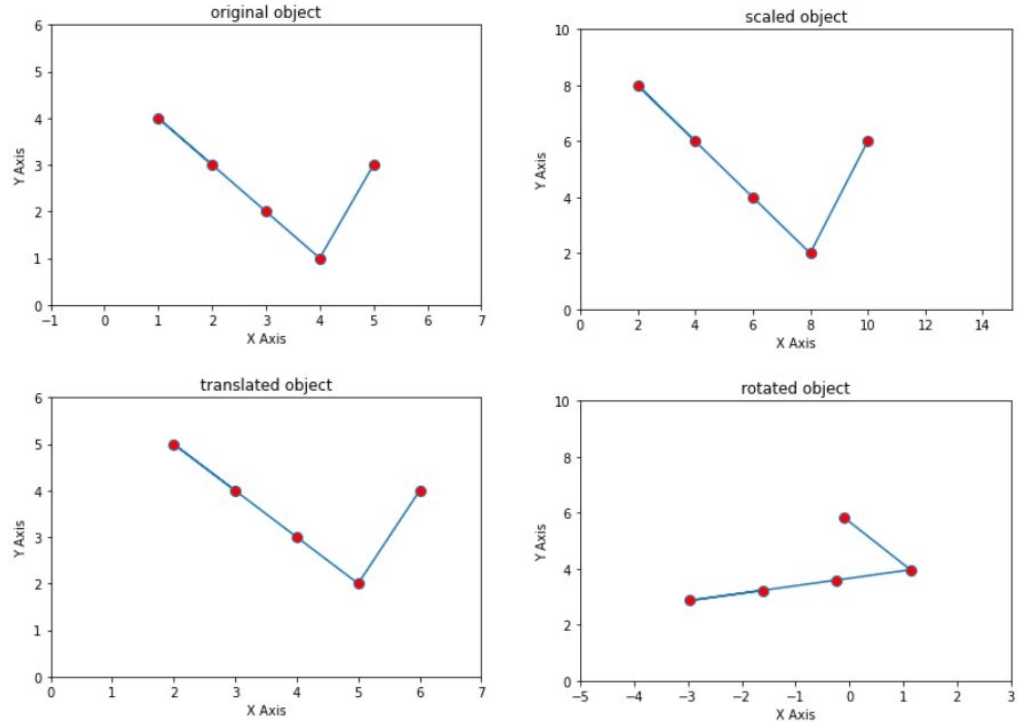
Translation, Rotation, and Scaling techniques in the 2D plane:

Code:

```
import matplotlib.pyplot as plt

import numpy as np

# Define the 2D object as a list of points (x, y)

xcordinates=[2,1,3,4,5]

ycordinates=[3,4,2,1,3]

#plot the original object

plt.title("original object")

plt.xlim(-1, 7)

plt.ylim(0,6)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(xcordinates, ycordinates,marker="o", markersize=8, markerfacecolor="red")

plt.show()

# Function to apply translation to the object

def translate(xcordinates,ycordinates, tx, ty):

translated_x = [x + tx for x in xcordinates]

translated_y=[y+ty for y in ycordinates]

#plot the translated object

plt.title("translated object")

plt.xlim(0, 7)

plt.ylim(0,6)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(translated_x, translated_y,marker="o", markersize=8, markerfacecolor="red")

plt.show()

# Function to apply scaling to the object

def scale(xcordinates,ycordinates, sx, sy):

scale_x = [x*sx for x in xcordinates]

scale_y=[y*sy for y in ycordinates]

#plot the scaled object

plt.title("scaled object")

plt.xlim(0, 15)

plt.ylim(0,10)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(scale_x,scale_y,marker="o", markersize=8, markerfacecolor="red")

plt.show()

# Function to apply rotation to the object

def rotate(xcordinates,ycordinates, angle_degrees):

angle_rad = np.deg2rad(angle_degrees)

object_points = [(2, 3), (1, 4), (3, 2), (4, 1),(5,3)]

rotate_x = [x* np.cos(angle_rad) - y * np.sin(angle_rad) for x,y in object_points]

rotate_y=[x * np.sin(angle_rad) + y * np.cos(angle_rad) for x,y in object_points]

#plot the rotated object

plt.title("rotated object")

plt.xlim(-5, 3)

plt.ylim(0,10)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(rotate_x, rotate_y,marker="o", markersize=8, markerfacecolor="red")

plt.show()
```

```
# Apply transformations

translate(xcordinates,ycordinates, 1, 1)

scale(xcordinates,ycordinates, 2, 2)

rotate(xcordinates,ycordinates, 60)
```

Output:



LAB – 6

Implementation of various composite transformation techniques on an object

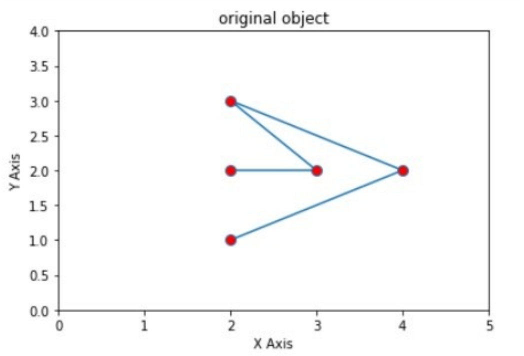
Composite transformation by applying rotation and reflection on an object:

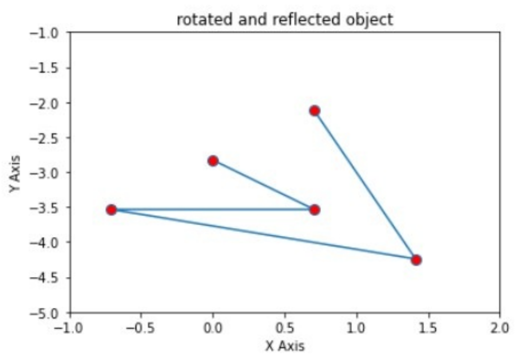
Code:

```
import matplotlib.pyplot as plt

import numpy as np

# Define the 2D object as a list of points (x, y)

xcordinates=[2,3,2,4,2]

ycordinates=[2,2,3,2,1]

# Function to plot the original object

plt.title("original object")

plt.xlim(0,5)

plt.ylim(0,4)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(xcordinates, ycordinates, marker="o", markersize=8, markerfacecolor="red")

plt.show()

# Function to apply rotation and reflection to the object

def rotateandreflect(xcordinates,ycordinates, angle_degrees,axis):

angle_rad = np.deg2rad(angle_degrees)

object_points = [(2, 2), (3, 2), (2, 3), (4, 2),(2,1)]

rotate_x = [x* np.cos(angle_rad) - y * np.sin(angle_rad) for x,y in object_points]

rotate_y=[x * np.sin(angle_rad) + y * np.cos(angle_rad) for x,y in object_points]

if axis == 'x':

reflected_x = [x for x in rotate_x]

reflected_y=[-y for y in rotate_y]

elif axis == 'y':

reflected_x = [-x for x in rotate_x]

reflected_y=[y for y in rotate_y]

else: # No reflection for invalid axis

reflected_x = [x for x in rotate_x]
```

```
reflected_y=[y for y in rotate_y]

#plot the transformed object

plt.title("rotated and reflected object")

plt.xlim(-1,2)

plt.ylim(-5,-1)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.plot(reflected_x, reflected_y, marker="o", markersize=8, markerfacecolor="red")

plt.show()

#apply transformation

rotateandreflect(xcordinates,ycordinates,45,'x')
```

Output:





LAB – 7

Simulation and Display of an Image, Negative of an image (Binary & Grey Scale).

a) Negative of an image (Grey Scale).

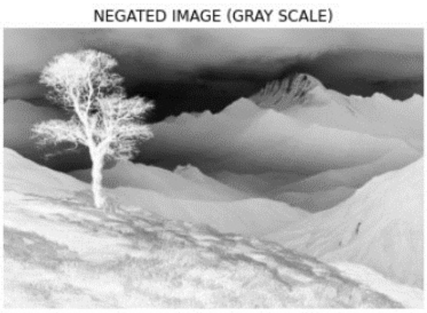
Code:

```
#Step 1: Import the required libraries

import cv2

import numpy as np

import matplotlib.pyplot as plt

#Step 2: Load and display the grayscale image

image_path="img.jpg"

color_image = cv2.imread(image_path)

color_image_rgb = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)

plt.imshow(color_image_rgb)

plt.title("ORIGINAL IMAGE")

plt.axis('off')

plt.show()

# Replace with the path to your grayscale image

gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

#Step 3: Store the grayscale image in a matrix

gray_matrix = np.array(gray_image)

print("GRAY SCALE MATRIX")

print(gray_matrix)

#Step 4: Negate the matrix

negated_matrix = 255 - gray_matrix #Simple negation by subtracting from 255
```

```
print("\nGRAY NEGATED MATRIX")
print(negated_matrix)
#Step 5: Display the grayscale image
plt.imshow(gray_image, cmap='gray')
plt.title("GRAYSCALE IMAGE")
plt.axis('off')
plt.show()
#Step 6: Display the negated image
negated_image = negated_matrix
plt.imshow(negated_image, cmap = 'gray')
plt.title("NEGATED IMAGE (GRAY SCALE)")
plt.axis('off')
plt.show()
```

Output:


```
GRAY SCALE MATRIX
[[63 63 63 ... 63 63 63]
 [63 63 63 ... 63 63 63]
 [63 63 63 ... 64 64 64]
 ...
 [13 13 13 ... 24 28 25]
 [19 16 11 ... 21 25 22]
 [23 20 14 ... 16 20 17]]

GRAY NEGATED MATRIX
[[192 192 192 ... 192 192 192]
 [192 192 192 ... 192 192 192]
 [192 192 192 ... 191 191 191]
 ...
 [242 242 242 ... 231 227 230]
 [236 239 244 ... 234 230 233]
 [232 235 241 ... 239 235 238]]
```



b) Negative of an image (Binary).

Code:

```
# Step 1: Import the required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Step 2: Load and display the original color image
image_path = "img.jpg"
color_image = cv2.imread(image_path)
# Convert the color image to grayscale
gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
# Step 3: Threshold the grayscale image to create a binary image
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
# Step 4: Store the binary image in a matrix
binary_matrix = np.array(binary_image)
print("\nBINARY SCALE MATRIX")
print(binary_matrix)
# Step 5: Negate the binary matrix
binary_negated_matrix = 255 - binary_matrix
print("\nBINARY NEGATED MATRIX")
print(binary_negated_matrix)
# Step 6: Display the binary image
```

plt.imshow(binary_image, cmap='binary')

plt.title("BINARY IMAGE")

plt.axis('off')

plt.show()

Step 7: Display the negated binary image

plt.imshow(binary_negated_matrix, cmap='binary')

plt.title("NEGATED BINARY IMAGE")

plt.axis('off')

plt.show()

Output:


```
BINARY SCALE MATRIX
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

BINARY NEGATED MATRIX
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
```

LAB – 8

Computation of Mean, Median, Variance and Standard Deviation of the given Image.

Code:

import numpy as np

import matplotlib.pyplot as plt

Load image

img = plt.imread('img.jpg')

Compute mean, median, variance, and standard deviation

mean = np.mean(img)

median = np.median(img)

variance = np.var(img)

std_dev = np.std(img)

Print results

print('Mean: ', mean)

print('Median: ', median)

print('Variance: ', variance)

print('Standard Deviation: ', std_dev)

Output:

```
Mean:  89.62031425891182
Median:  77.0
Variance:  3667.774505091983
Standard Deviation:  60.56215406581889
```

LAB – 9

Display of colour images and conversion between colour spaces.

Code:

import cv2

import matplotlib.pyplot as plt

Load and Display Image

image = cv2.imread('img.jpg')

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.title('Original Image')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

Display the original and converted images

plt.figure(figsize=(12, 4))

#Grayscale image

plt.subplot(1, 3, 1)

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

plt.title('Grayscale')

#HSV color space

plt.subplot(1, 3,2)

plt.imshow(cv2.cvtColor(hsv_image, cv2.COLOR_HSV2RGB))

plt.title('HSV Color Space')

#LAB color space

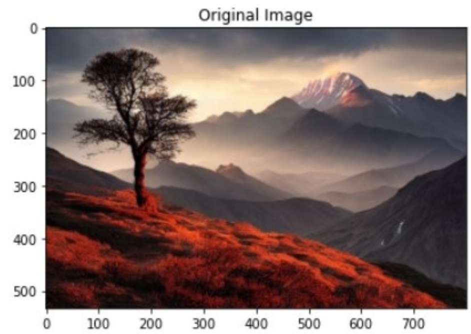
plt.subplot(1, 3,3)

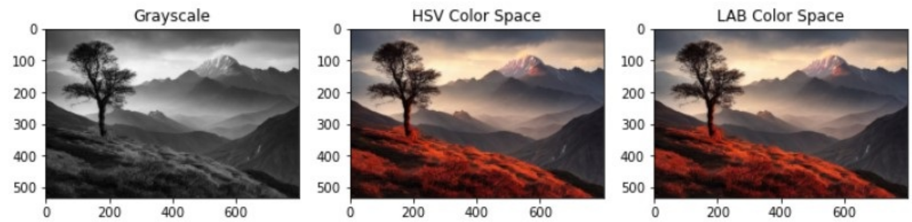
plt.imshow(cv2.cvtColor(lab_image, cv2.COLOR_LAB2RGB))

plt.title('LAB Color Space')

plt.show()

Output:





LAB – 10

Implement image segmentation using histogram thresholding.

Code:

import cv2

import numpy as np

import matplotlib.pyplot as plt

#Load the image

image = cv2.imread('shapes.jpg')

#Convert the image to grayscale

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Calculate histogram

hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256])

Plot the histogram

plt.plot(hist)

plt.title('Histogram')

plt.xlabel('Pixel Value')

plt.ylabel('Frequency')

plt.show()

```
# Apply thresholding (adjust the threshold value as needed)

_, segmented_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

#Invert the pixel values to make background black and object white

segmented_image = cv2.bitwise_not(segmented_image)

# Display the original and segmented images

plt.subplot(1, 2, 1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.title('Original Image')

plt.subplot(1, 2, 2)

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

plt.title('Segmented Image')

plt.show()
```

Output: