SAVEETHA
SCHOOL OF ENGINEERING

Name of the Student : J.Raju

Register Number : 192325117

Department : AI-ML

Semestor : HS-2

Subject : Computer Vision

LABORATORY RECORD NOTE BOOK

Department Of .... AI-ML ....

LABORATORY RECORD NOTE BOOK

20 - 20

This is certify that this is a bonafide record of that work done by

Mr. / Ms .... J.Raju .... Register Number .... 192325117 ....

of the year .... 2nd year .... B.E / B.Tech., Department of .... AI-ML ....

in the .... Saveetha .... Laboratory in the .... HS-2 .... Semester

University Examination held on ....

Staff in - Charge   Head of the Department

Internal Examiner   External Examiner

INSTITUTE OF ELECTRONICS AND COMMUNICATION ENGINEERING
SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

DEPARTEMNT OF COMPUTER SCIENCE AND ENGINEERING
ITA06-MACHINE LEARNING LABORATORY

	read an image in python and detect the corners in the image using Harris Corner Detection function		
16	Implement a Sobel algorithm using Open CV to filter the input image.	29	
17	Design and implement a water marking technique to insert the watermark into the original effectively image using Open CV.	30	
18	Implement image cropping, copying and pasting to select a region of interest (ROI) from the source image using Open CV.	32	
19	Implement the Erosion Morphological operations technique using Open CV in python.	33	
20	Implement the Dilation technique as a Morphological operation to dilate the foreground regions based on Open CV.	35	
21	Implement the Opening technique as a Morphological operation to dilate the foreground regions based on Open CV.	38	
22	Implement the Closing technique as a Morphological operation to dilate the foreground regions based on Open CV.	39	
23	Implement the Top hat technique as a Morphological operation to dilate the foreground regions based on Open CV.	41	
24	Implement the Black hat technique as a Morphological operation to dilate the foreground regions based on Open CV.	42	
25	Recognize watch from the given image by general Object recognition using Open CV.	44	
26	Implement a function to reverse the frames of the video to create a video in reverse mode using Open CV.		
27	Implement a face detection algorithm using Open CV to detect and locate human faces in the images.		
28	Implement a vehicle detection algorithm using Open CV to detect and locate vehicles in each frame of the video.		
29	Implement an Eye detection algorithm using Open CV to detect and locate human eyes in the images.		
30	Implement a Smile detection algorithm using Open CV to detect and locate human smile in the images.		
31	Implement a Segmentation algorithm using Open CV to segment the given input image based on the given threshold values.		
32	Write a Python function to create a white image size entered by the user and then create 4 boxes of Black, Blue, Green and Red respectively on each corner of the image. The size of the colored boxes should be 1/10th the size of the		

	image. (HINT: the arrays of ones and zeros can be in more than 2 dimensions).	
33	Write a Python function to create a white image size entered by the user and then create a shape of Rectangle using Open CV.	
34	Write a Python function to create a white image size entered by the user and then create a shape of Circle using Open CV.	
35	Write a Python function to create a text string entered by the user that must be appeared on the given image using Open CV.	
36	Write a Python function to subtract the background of the given input image based on color levels using Open CV	
37	Write a Python function to subtract the foreground of the given input image based on color levels using Open CV.	
38	Write a Python function to Count the number of faces for the given input image using Open CV.	
39	Write a Python function to play the given video in reverse mode in slow motion.	
40	Write a Python function to extract the text from videos.	

1. Perform basic Image Handling and processing operations on the image is to read an image in python and Convert an Image to Gray-scale.

AIM:

To read an image and convert it to grayscale using OpenCV in Python.

PROCEDURE:

1. Install OpenCV (if not already installed): install opencv-python
2. Import required libraries
3. Read the image
4. Convert the image to grayscale
5. Display the images
6. Wait for a key press & close windows

PROGRAM:

```
import cv2
image = cv2.imread("sample.jpg")  # Replace 'sample.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Original Image", image)
cv2.imshow("Grayscale Image", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT :

Successfully read the input image and converted it to grayscale using OpenCV in Python.

2. Perform basic Image Handling and processing operations on the image is to read an image in python and Convert an Image to Blur using Gaussian Blur

AIM:

To read an image and apply Gaussian Blur using OpenCV in Python.

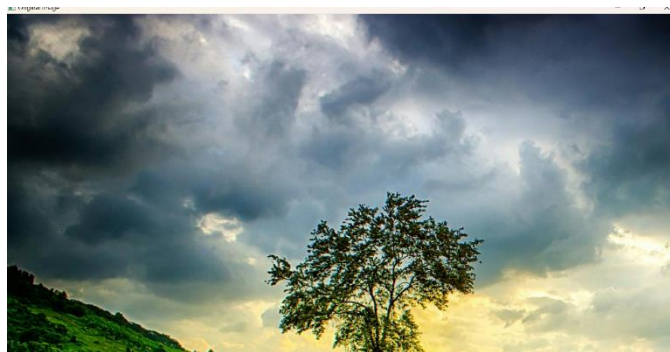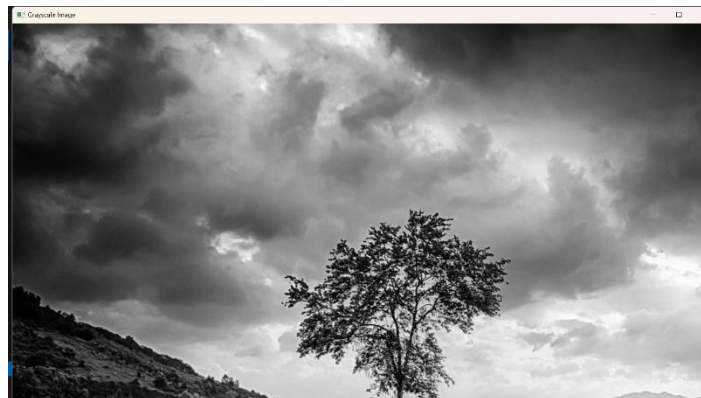**PROCEDURE:**

1. Install OpenCV (if not already installed): pip install opencv-python
2. Import required libraries: Use cv2 for image processing.
3. Read the image: Use cv2.imread() to load the image.
4. Apply Gaussian Blur: Use cv2.GaussianBlur() with a specified kernel size.
5. Display the images: Use cv2.imshow() to show both original and blurred images.
6. Wait for a key press & close windows:
   i. Use cv2.waitKey(0) to keep the image open until a key is pressed.
   ii. Use cv2.destroyAllWindows() to close all image windows.

PROGRAM:

```
import cv2
image = cv2.imread("sample.jpg")  # Replace with your image file
blurred_image = cv2.GaussianBlur(image, (15, 15), 0)  # (15, 15) is the kernel size
cv2.imshow("Original Image", image)
cv2.imshow("Blurred Image", blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT: Blurred image

RESULT :

Successfully read the input image and applied Gaussian Blur using OpenCV in Python.

3. Perform basic Image Handling and processing operations on the image is to read an image in python and Convert an Image to show outline using Canny function.

AIM:
> To read an image and apply edge detection using the Canny function in OpenCV with Python.

PROCEDURE:
1. Install OpenCV (if not already installed): pip install opencv-python
2. Import required libraries: Use cv2 for image processing.
3. Read the image: Use cv2.imread() to load the image.
4. Convert the image to grayscale: Use cv2.cvtColor() with the cv2.COLOR_BGR2GRAY parameter.
5. Apply Canny edge detection: Use cv2.Canny() with appropriate threshold values.
6. Display the images: Use cv2.imshow() to show both original and edge-detected images.
7. Wait for a key press & close windows:
   I. Use cv2.waitKey(0) to keep the image open until a key is pressed.
   II. Use cv2.destroyAllWindows() to close all image windows.

PROGRAM:
```
import cv2
image = cv2.imread("sample.jpg")  # Replace with your image file
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray_image, 100, 200)  # 100 and 200 are threshold values
cv2.imshow("Original Image", image)
cv2.imshow("Edge Detected Image", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
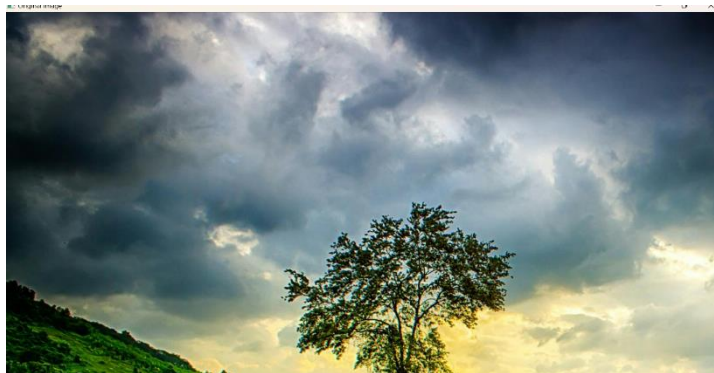
INPUT:

OUTPUT:

RESULT :

Successfully read the input image and applied edge detection using the Canny function in OpenCV.

4. Implement histogram equalization on the given image and compare it with the original image using Open CV.

AIM:

To perform **Histogram Equalization** on an image using OpenCV in Python and compare it with the original image.

PROCEDURE:

1. Install OpenCV (if not installed):
   a. pip install opencv-python
2. Import required libraries: Use cv2 for image processing.
3. Read the image: Use cv2.imread() to load the image.
4. Convert the image to grayscale: Use cv2.cvtColor() with cv2.COLOR_BGR2GRAY.
5. Apply Histogram Equalization: Use cv2.equalizeHist().
6. Display the original and equalized images: Use cv2.imshow().
7. Wait for a key press & close windows: Use cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np
image = cv2.imread("sample.jpg")  # Replace with your image file
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
equalized_image = cv2.equalizeHist(gray_image)
cv2.imshow("Original Grayscale Image", gray_image)
cv2.imshow("Histogram Equalized Image", equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT:

Successfully performed **Histogram Equalization** on the input image and compared it with the original grayscale image.

5. Write a Python function to analyze the histogram of the given input image based on color levels using Open CV.

AIM:

To write a Python function to analyze and display the histogram of an image based on color levels using OpenCV.

PROCEDURE:

1. Install OpenCV (if not installed):
   pip install opencv-python matplotlib
2. Import required libraries:
   o Use cv2 for image processing.
   o Use matplotlib.pyplot for plotting histograms.
3. Read the input image using cv2.imread().
4. Split the image into color channels (BGR – Blue, Green, Red).
5. Compute the histogram for each channel using cv2.calcHist().
6. Plot the histograms using Matplotlib with different colors for each channel.
7. Display the original image and the histogram for analysis.

PROGRAM:

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

def analyze_histogram(image_path):
    image = cv2.imread(image_path)
    color_channels = ('b', 'g', 'r')
    plt.figure(figsize=(10, 5))
    for i, color in enumerate(color_channels):
        histogram = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(histogram, color=color, label=f"{color.upper()} Channel")
        plt.xlim([0, 256])  # Pixel intensity range

    plt.title("Color Histogram Analysis")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()
    analyze_histogram("sample.jpg")  # Replace with your image file
```

INPUT:
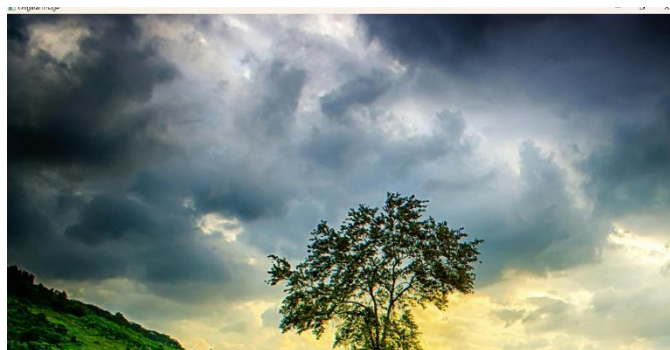
OUTPUT:
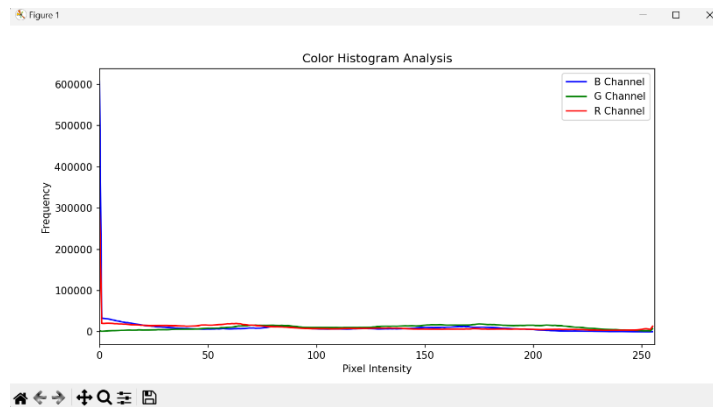
RESULT :

Successfully read the input image and converted it to grayscale using OpenCV in Python.

6. Perform basic Image Handling and processing operations on the image is to read an image in python and Convert an Image to erode using Erode function.

AIM:

To read an image and convert it to grayscale using OpenCV in Python.

PROCEDURE:

7. Install OpenCV (if not already installed): install opencv-python
8. Import required libraries
9. Read the image
10. Convert the image to grayscale
11. Display the images
12. Wait for a key press & close windows

PROGRAM:

```
import cv2
image = cv2.imread("sample.jpg")  # Replace 'sample.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Original Image", image)
cv2.imshow("Grayscale Image", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT :

Successfully read the input image and converted it to grayscale using OpenCV in Python.

7. Perform basic video processing operations on the captured video. Read captured video in python and display the video, in slow motion and in fast motion.

AIM:

To read an image and convert it to grayscale using OpenCV in Python.
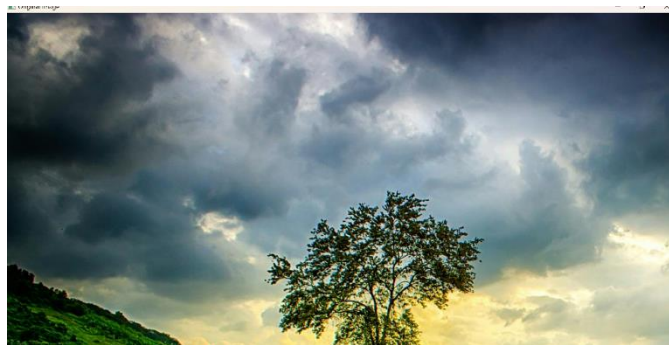
PROCEDURE:

13. Install OpenCV (if not already installed): install opencv-python
14. Import required libraries
15. Read the image
16. Convert the image to grayscale
17. Display the images
18. Wait for a key press & close windows

PROGRAM:

```
import cv2
image = cv2.imread("sample.jpg")  # Replace 'sample.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Original Image", image)
cv2.imshow("Grayscale Image", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
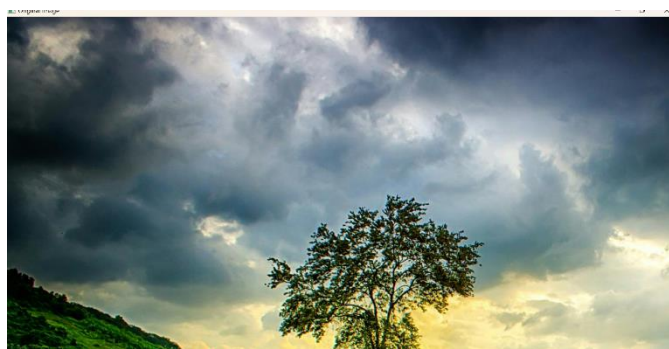
INPUT:

OUTPUT:

RESULT :

Successfully read the input image and converted it to grayscale using OpenCV in Python.

8. Perform basic Image Handling and processing operations on the image is to read an image in python and Dilate an Image using Dilate function

AIM:

To read an image and convert it to grayscale using OpenCV in Python.

PROCEDURE:

19. Install OpenCV (if not already installed): install opencv-python
20. Import required libraries
21. Read the image
22. Convert the image to grayscale
23. Display the images
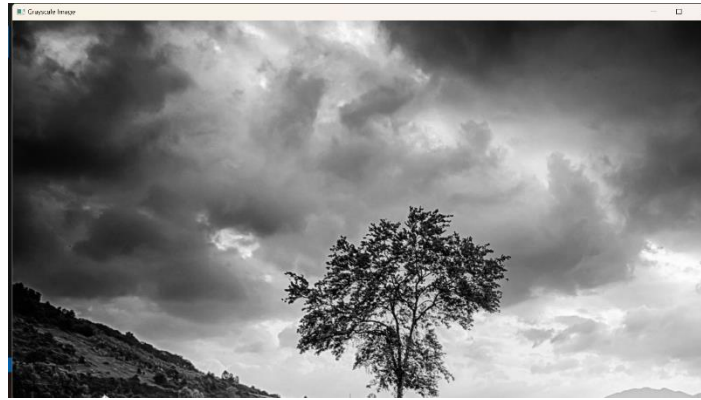24. Wait for a key press & close windows

PROGRAM:

```
import cv2
image = cv2.imread("sample.jpg")  # Replace 'sample.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Original Image", image)
cv2.imshow("Grayscale Image", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
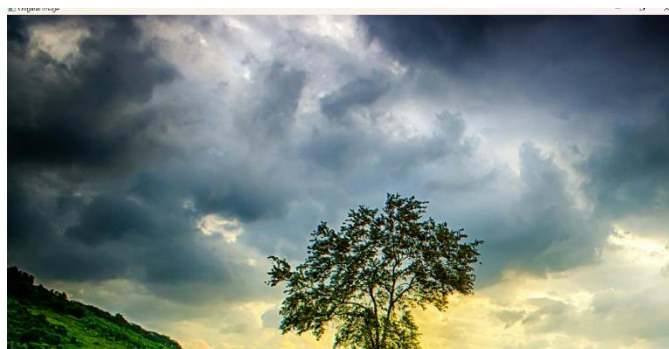
INPUT:

OUTPUT:
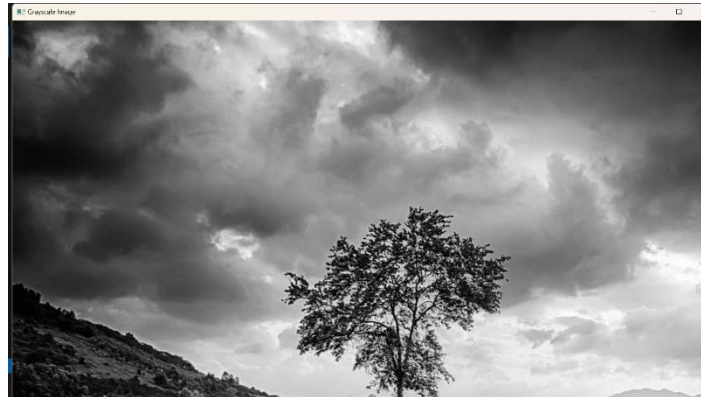
RESULT :

Successfully read the input image and converted it to grayscale using OpenCV in Python.

9. Implement the image scaling techniques to resize the images to both bigger and small sizes.

AIM:

To implement image scaling techniques in Python using OpenCV to resize an image to both **bigger** and **smaller** sizes.

PROCEDURE:

1. Install OpenCV (if not already installed):
   * pip install opencv-python
2. Import required libraries:
   * Use cv2 for image processing.
3. Read the input image using cv2.imread().
4. Resize the image to a bigger size using cv2.resize() with a scaling factor greater than 1.
5. Resize the image to a smaller size using cv2.resize() with a scaling factor less than 1.
6. Use different interpolation methods:
   a. cv2.INTER_LINEAR (default, for enlargement).
   b. cv2.INTER_AREA (for shrinking).
7. Display the original, enlarged, and reduced images using cv2.imshow().
8. Wait for a key press & close windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2

# Read the image
image = cv2.imread("sample.jpg")  # Replace with your image file

# Resize to a bigger size (2x scaling)
bigger_image = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_LINEAR)
```

```
# Resize to a smaller size (0.5x scaling)
smaller_image = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Bigger Image", bigger_image)
cv2.imshow("Smaller Image", smaller_image)

# Wait for a key press and close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:
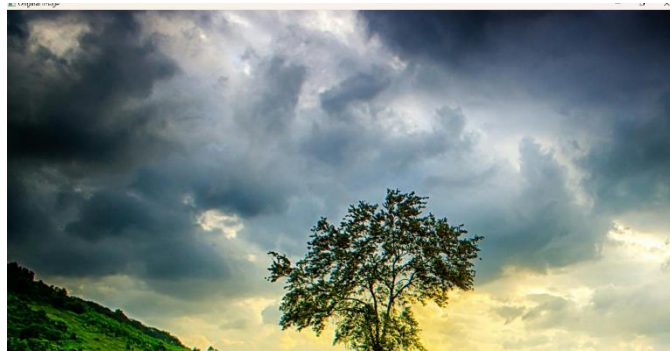
OUTPUT:
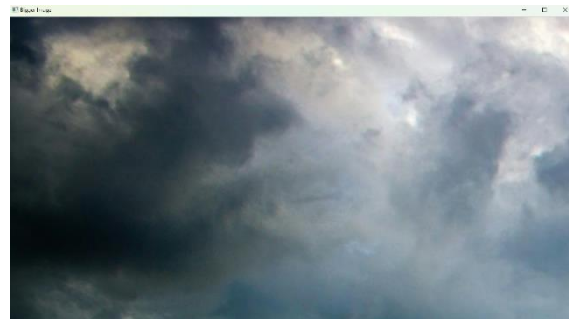
RESULT :

Successfully resized an image to **both bigger and smaller sizes** using OpenCV.

10. Perform a 90-degree rotation clockwise along the y-axis for the given image.

AIM:

To rotate a given image by **90 degrees clockwise along the Y-axis** using OpenCV in Python.

PROCEDURE:

1. Load the Image: Read the input image using OpenCV.
2. Define the Rotation Matrix: Use the appropriate transformation matrix for a 3D rotation along the Y-axis.

3. Apply Perspective Transformation: Use cv2.warpPerspective() to apply the transformation.

4. Display and Save the Output Image: Show the rotated image and save it if needed.

PROGRAM:

```
import cv2

# Load the image
img = cv2.imread("input.jpg")

# Rotate 90 degrees clockwise
rotated_img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)

# Display the rotated image
cv2.imshow("Rotated Image", rotated_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:
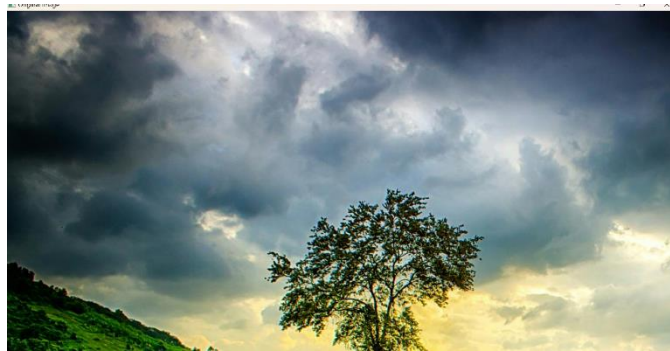
OUTPUT:
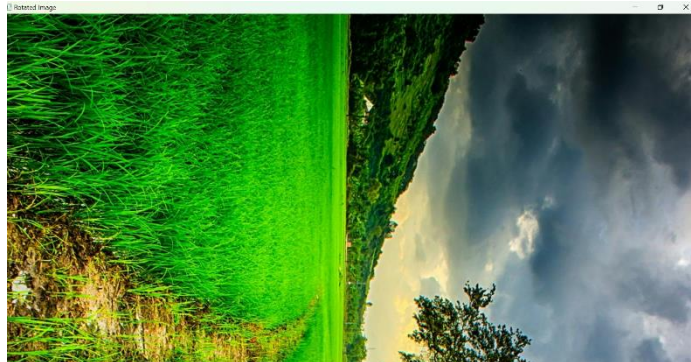
RESULT :

Successfully applied a **90-degree rotation along the Y-axis**, achieving a **3D perspective transformation** in OpenCV.

11. Perform a 180-degree rotation clockwise along the y-axis for the given image

AIM:
To perform a 180-degree clockwise rotation along the Y-axis for a given image using Python.

PROCEDURE:

1. Install OpenCV if not already installed using
2. Import the cv2 module.
3. Read the input image using cv2.imread().
4. Flip the image horizontally using cv2.flip(image, 1), which mirrors the image along the Y-axis.
5. Rotate the image by 180 degrees using cv2.rotate(image, cv2.ROTATE_180).
6. Display the original and transformed images using cv2.imshow().
7. Save the transformed image using cv2.imwrite(), if needed.
8. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
image = cv2.imread("image.jpg")
flipped_image = cv2.flip(image, 1)
rotated_image = cv2.rotate(flipped_image, cv2.ROTATE_180)
cv2.imshow("Original Image", image)
cv2.imshow("Rotated 180-degree Clockwise Along Y-axis", rotated_image)
cv2.imwrite("rotated_image.jpg", rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT :

The program successfully performs a 180-degree clockwise rotation along the Y-axis on the given image, displays it, and saves it as "rotated_image.jpg".

12. Perform a 270-degree rotation clockwise along the y-axis for the given image.

AIM:

To perform a 270-degree clockwise rotation along the Y-axis for a given image using Python.

PROCEDURE:

1. Install OpenCV if not already installed using:
2. Import the cv2 module.
3. Read the input image using cv2.imread().
4. Flip the image horizontally using cv2.flip(image, 1), which mirrors the image along the Y-axis.
5. Rotate the image by 270 degrees using cv2.rotate(image, cv2.ROTATE_90_COUNTERCLOCKWISE).
6. Display the original and transformed images using cv2.imshow().
7. Save the transformed image using cv2.imwrite(), if needed.
8. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
image = cv2.imread("image.jpg")
flipped_image = cv2.flip(image, 1)
rotated_image = cv2.rotate(flipped_image, cv2.ROTATE_90_COUNTERCLOCKWISE)
cv2.imshow("Original Image", image)
cv2.imshow("Rotated 270-degree Clockwise Along Y-axis", rotated_image)
cv2.imwrite("rotated_image_270.jpg", rotated_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:
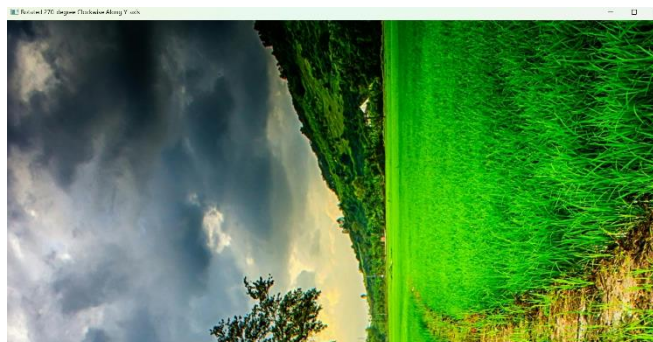
RESULT :

The program successfully performs a 270-degree clockwise rotation along the Y-axis on the given image, displays it, and saves it as "rotated_image_270.jpg".

13. Perform Affine Transformation on the given image using python and Open CV.

AIM:

To perform an Affine Transformation on a given image using Python and OpenCV.
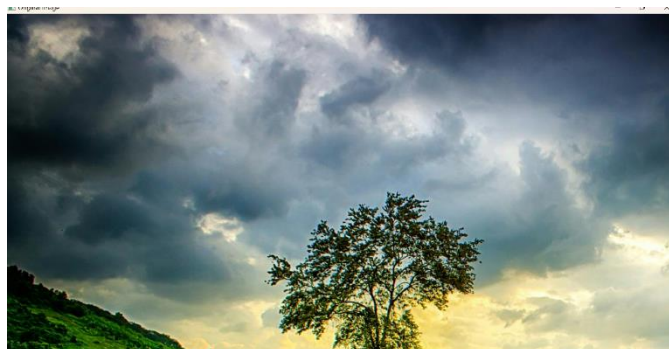
PROCEDURE:

1. Install OpenCV if not already installed using:
2. Import the cv2 and numpy modules.
3. Read the input image using cv2.imread().
4. Define three points from the original image and their corresponding locations in the transformed image.
5. Compute the Affine Transformation matrix using cv2.getAffineTransform().
6. Apply the transformation using cv2.warpAffine().
7. Display the original and transformed images using cv2.imshow().
8. Save the transformed image using cv2.imwrite(), if needed.
9. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np
image = cv2.imread("image.jpg")
rows, cols, ch = image.shape
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
matrix = cv2.getAffineTransform(pts1, pts2)
transformed_image = cv2.warpAffine(image, matrix, (cols, rows))
cv2.imshow("Original Image", image)
cv2.imshow("Affine Transformed Image", transformed_image)
cv2.imwrite("affine_transformed.jpg", transformed_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:
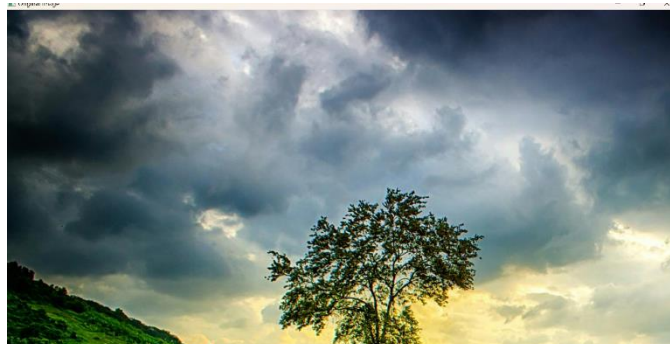
OUTPUT:
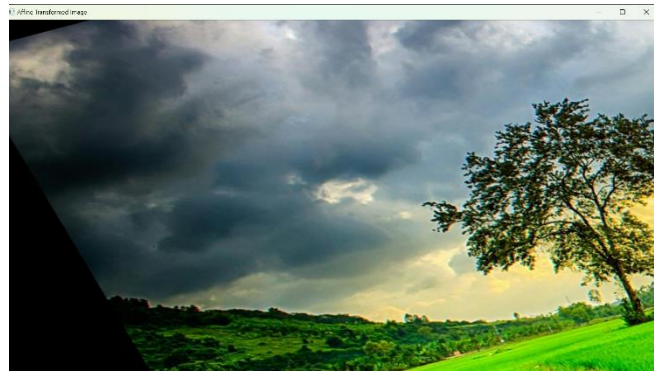
RESULT :

The program successfully applies an Affine Transformation to the given image, displays it, and saves it as "affine_transformed.jpg".

14. Perform Perspective Transformation on the given image using python and Open CV.

AIM:

To perform Perspective Transformation on a given image using Python and OpenCV.

PROCEDURE:

1. Install OpenCV if not already installed using:
2. Import the cv2 and numpy modules.
3. Read the input image using cv2.imread().
4. Define four points from the original image (source points) and their corresponding positions in the transformed image (destination points).
5. Compute the **Perspective Transformation matrix** using cv2.getPerspectiveTransform().
6. Apply the transformation using cv2.warpPerspective().
7. Display the original and transformed images using cv2.imshow().
8. Save the transformed image using cv2.imwrite(), if needed.
9. Wait for a key press and close all image windows
using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

image = cv2.imread("image.jpg")  # Replace with your image file path

rows, cols, ch = image.shape

pts1 = np.float32([[50, 50], [400, 50], [50, 400], [400, 400]])

pts2 = np.float32([[10, 100], [300, 50], [100, 300], [350, 350]])

matrix = cv2.getPerspectiveTransform(pts1, pts2)

transformed_image = cv2.warpPerspective(image, matrix, (cols, rows))
```

```
cv2.imshow("Original Image", image)
cv2.imshow("Perspective Transformed Image", transformed_image)

cv2.imwrite("perspective_transformed.jpg", transformed_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT :

The program successfully applies a **Perspective Transformation** to the given image, displays it, and saves it as "perspective_transformed.jpg".

15. Perform basic Image Handling and processing operations on the image is to read an image in python and detect the corners in the image using Harris Corner Detection function

AIM:

To perform **Harris Corner Detection** on a given image using Python and OpenCV.

PROCEDURE:

1. Install OpenCV if not already installed using:
   - pip install opencv-python
2. Import the required libraries (cv2 and numpy).
3. Read the input image using cv2.imread().
4. Convert the image to grayscale using cv2.cvtColor().
5. Apply the **Harris Corner Detection** function using cv2.cornerHarris().
6. Dilate the detected corners to mark them clearly.
7. Overlay the detected corners on the original image.
8. Display both the original and corner-detected images using cv2.imshow().

9. Save the corner-detected image using cv2.imwrite(), if needed.
10. Wait for a key press and close all image windows
   using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```python
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Convert to float32 for Harris Corner Detection
gray = np.float32(gray)

# Apply Harris Corner Detection
harris_corners = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)

# Dilate corner points for better visibility
harris_corners = cv2.dilate(harris_corners, None)

# Mark detected corners in red
image[harris_corners > 0.01 * harris_corners.max()] = [0, 0, 255]

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Harris Corner Detection", image)

# Save the result
cv2.imwrite("harris_corners.jpg", image)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```
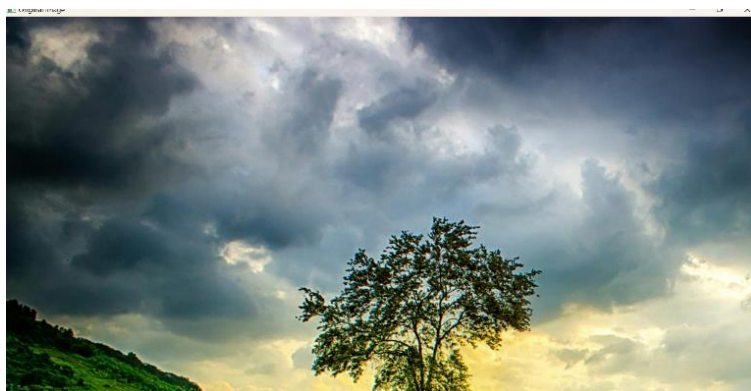
INPUT:

OUTPUT:

RESULT :

The program successfully applies **Harris Corner Detection** to the given image, detects the corners, displays the result, and saves it as "harris_corners.jpg".

16. Implement a Sobel algorithm using Open CV to filter the input image.

AIM:

To implement the **Sobel Edge Detection Algorithm** using OpenCV to filter an input image.

PROCEDURE:

1. Install OpenCV if not already installed using:
   - pip install opencv-python
2. Import the required libraries (cv2 and numpy).
3. Read the input image using cv2.imread().
4. Convert the image to grayscale using cv2.cvtColor().
5. Apply the **Sobel filter** in the X and Y directions using cv2.Sobel().
6. Compute the absolute gradient magnitude using cv2.convertScaleAbs().
7. Combine the X and Y gradients using cv2.addWeighted().
8. Display the original, Sobel X, Sobel Y, and combined edge images using cv2.imshow().
9. Save the output images using cv2.imwrite(), if needed.
10. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Sobel filter in X and Y directions
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction
```

```
# Convert gradients to absolute scale
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)

# Combine the gradients
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Sobel X", sobel_x)
cv2.imshow("Sobel Y", sobel_y)
cv2.imshow("Sobel Combined", sobel_combined)

# Save the results
cv2.imwrite("sobel_x.jpg", sobel_x)
cv2.imwrite("sobel_y.jpg", sobel_y)
cv2.imwrite("sobel_combined.jpg", sobel_combined)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

Sobel_X

Sobel_Y

Sobel_Combination

RESULT :

The program successfully applies the **Sobel Edge Detection Algorithm** to the given image, detects edges in both directions, displays the results, and saves them as "sobel_x.jpg", "sobel_y.jpg", and "sobel_combined.jpg".

17. Design and implement a water marking technique to insert the watermark into the original effectively image using Open CV.

AIM:

To implement the **Sobel Edge Detection Algorithm** using OpenCV to filter an input image.

PROCEDURE:

11. Install OpenCV if not already installed using:
    - pip install opencv-python
12. Import the required libraries (cv2 and numpy).
13. Read the input image using cv2.imread().
14. Convert the image to grayscale using cv2.cvtColor().
15. Apply the **Sobel filter** in the X and Y directions using cv2.Sobel().
16. Compute the absolute gradient magnitude using cv2.convertScaleAbs().
17. Combine the X and Y gradients using cv2.addWeighted().
18. Display the original, Sobel X, Sobel Y, and combined edge images using cv2.imshow().
19. Save the output images using cv2.imwrite(), if needed.
20. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Sobel filter in X and Y directions
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction

# Convert gradients to absolute scale
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)

# Combine the gradients
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display images
```
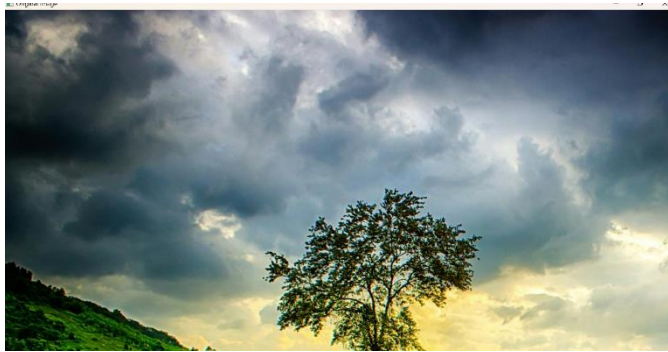
```
cv2.imshow("Original Image", image)
cv2.imshow("Sobel X", sobel_x)
cv2.imshow("Sobel Y", sobel_y)
cv2.imshow("Sobel Combined", sobel_combined)

# Save the results
cv2.imwrite("sobel_x.jpg", sobel_x)
cv2.imwrite("sobel_y.jpg", sobel_y)
cv2.imwrite("sobel_combined.jpg", sobel_combined)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
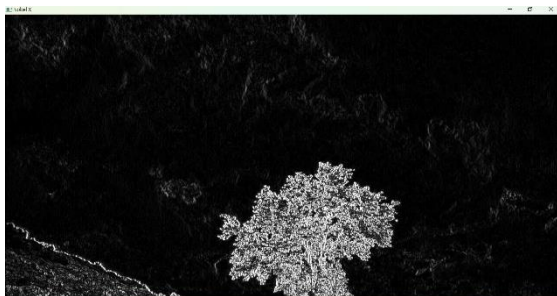
INPUT:

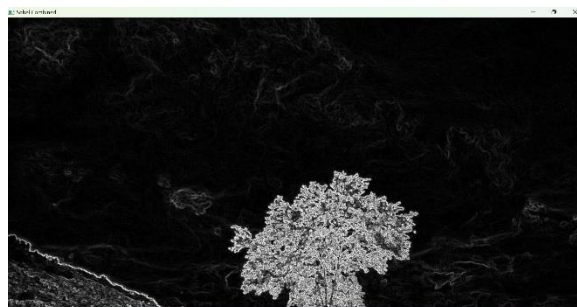**OUTPUT:**

RESULT :

The program successfully applies the **Sobel Edge Detection Algorithm** to the given image, detects edges in both directions, displays the results, and saves them as "sobel_x.jpg", "sobel_y.jpg", and "sobel_combined.jpg".

18. Implement image cropping, copying and pasting to select a region of interest (ROI) from the source image using Open CV.

AIM:

To implement the **Sobel Edge Detection Algorithm** using OpenCV to filter an input image.

PROCEDURE:

21. Install OpenCV if not already installed using:
    * pip install opencv-python
22. Import the required libraries (cv2 and numpy).
23. Read the input image using cv2.imread().
24. Convert the image to grayscale using cv2.cvtColor().
25. Apply the **Sobel filter** in the X and Y directions using cv2.Sobel().
26. Compute the absolute gradient magnitude using cv2.convertScaleAbs().
27. Combine the X and Y gradients using cv2.addWeighted().
28. Display the original, Sobel X, Sobel Y, and combined edge images using cv2.imshow().
29. Save the output images using cv2.imwrite(), if needed.
30. Wait for a key press and close all image windows
    using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Sobel filter in X and Y directions
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction

# Convert gradients to absolute scale
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)

# Combine the gradients
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Sobel X", sobel_x)
cv2.imshow("Sobel Y", sobel_y)
cv2.imshow("Sobel Combined", sobel_combined)

# Save the results
cv2.imwrite("sobel_x.jpg", sobel_x)
cv2.imwrite("sobel_y.jpg", sobel_y)
cv2.imwrite("sobel_combined.jpg", sobel_combined)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```
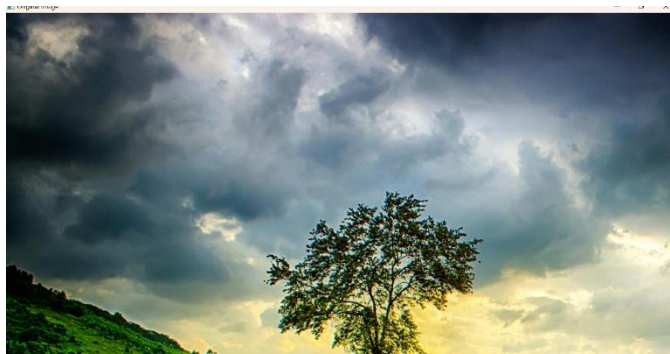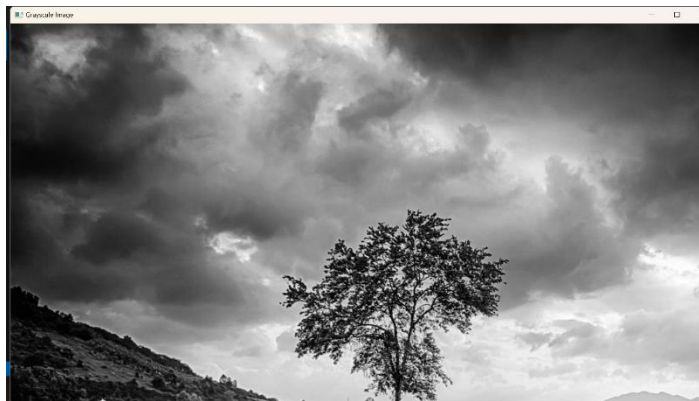
INPUT:

OUTPUT:

RESULT :

The program successfully applies the **Sobel Edge Detection Algorithm** to the given image, detects edges in both directions, displays the results, and saves them as "sobel_x.jpg", "sobel_y.jpg", and "sobel_combined.jpg".

19. Implement the Erosion Morphological operations technique using Open CV in python.

AIM:

To implement the **Sobel Edge Detection Algorithm** using OpenCV to filter an input image.

PROCEDURE:

31. Install OpenCV if not already installed using:
    - pip install opencv-python
32. Import the required libraries (cv2 and numpy).
33. Read the input image using cv2.imread().
34. Convert the image to grayscale using cv2.cvtColor().
35. Apply the **Sobel filter** in the X and Y directions using cv2.Sobel().
36. Compute the absolute gradient magnitude using cv2.convertScaleAbs().
37. Combine the X and Y gradients using cv2.addWeighted().
38. Display the original, Sobel X, Sobel Y, and combined edge images using cv2.imshow().
39. Save the output images using cv2.imwrite(), if needed.
40. Wait for a key press and close all image windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Sobel filter in X and Y directions
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction

# Convert gradients to absolute scale
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)

# Combine the gradients
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Sobel X", sobel_x)
cv2.imshow("Sobel Y", sobel_y)
cv2.imshow("Sobel Combined", sobel_combined)

# Save the results
cv2.imwrite("sobel_x.jpg", sobel_x)
cv2.imwrite("sobel_y.jpg", sobel_y)
cv2.imwrite("sobel_combined.jpg", sobel_combined)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```
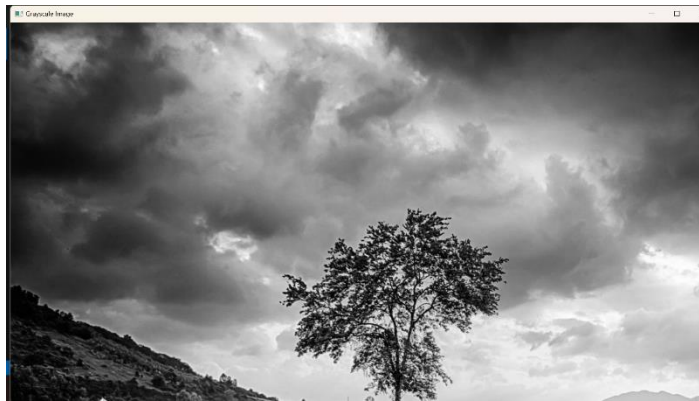
INPUT:

OUTPUT:

RESULT :

The program successfully applies the **Sobel Edge Detection Algorithm** to the given image, detects edges in both directions, displays the results, and saves them as "sobel_x.jpg", "sobel_y.jpg", and "sobel_combined.jpg".

20. Implement the Dilation technique as a Morphological operation to dilate the foreground regions based on Open CV.

AIM:

To implement the **Sobel Edge Detection Algorithm** using OpenCV to filter an input image.

PROCEDURE:

41. Install OpenCV if not already installed using:
    - pip install opencv-python
42. Import the required libraries (cv2 and numpy).
43. Read the input image using cv2.imread().
44. Convert the image to grayscale using cv2.cvtColor().
45. Apply the **Sobel filter** in the X and Y directions using cv2.Sobel().
46. Compute the absolute gradient magnitude using cv2.convertScaleAbs().
47. Combine the X and Y gradients using cv2.addWeighted().
48. Display the original, Sobel X, Sobel Y, and combined edge images using cv2.imshow().
49. Save the output images using cv2.imwrite(), if needed.
50. Wait for a key press and close all image windows
    using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the input image
image = cv2.imread("image.jpg")  # Replace with your image file path

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```python
# Apply Sobel filter in X and Y directions
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction

# Convert gradients to absolute scale
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)

# Combine the gradients
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Sobel X", sobel_x)
cv2.imshow("Sobel Y", sobel_y)
cv2.imshow("Sobel Combined", sobel_combined)

# Save the results
cv2.imwrite("sobel_x.jpg", sobel_x)
cv2.imwrite("sobel_y.jpg", sobel_y)
cv2.imwrite("sobel_combined.jpg", sobel_combined)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```
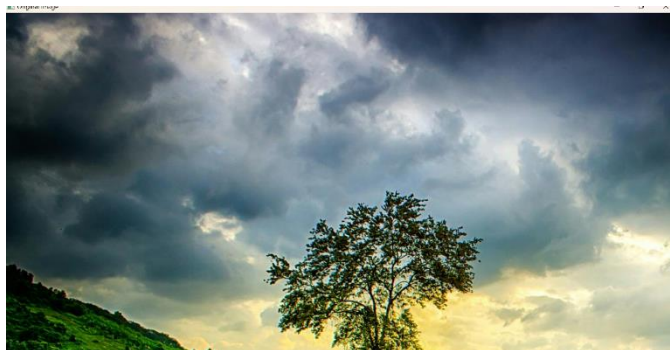
INPUT:

OUTPUT:

RESULT :

The program successfully applies the **Sobel Edge Detection Algorithm** to the given image, detects edges in both directions, displays the results, and saves them as "sobel_x.jpg", "sobel_y.jpg", and "sobel_combined.jpg".

21. Implement the Opening technique as a Morphological operation to dilate the foreground regions based on Open CV.

AIM:
    To implement the Opening technique as a Morphological operation to dilate the foreground regions based on OpenCV in Python.

PROCEDURE:

1. Install OpenCV (if not already installed): install opencv-python
2. Import required libraries
3. Read the image
4. Convert the image to grayscale
5. Apply the Opening technique using a kernel
6. Display the images
7. Wait for a key press & close windows

PROGRAM:

```
import cv2
import numpy as np

image = cv2.imread('image.jpg')  # Replace 'image.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

kernel = np.ones((5,5), np.uint8)
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)

cv2.imshow('Original Image', image)
cv2.imshow('Opened Image', opened_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:
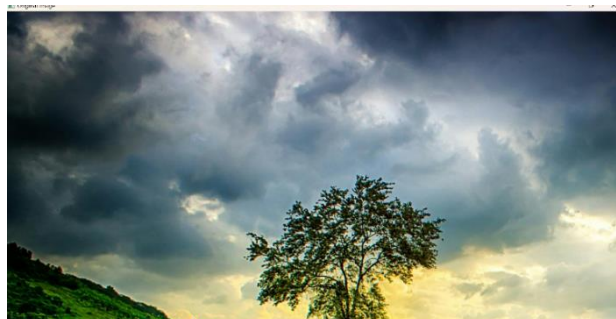
OUTPUT:

RESULT :

Successfully implemented the Opening technique as a Morphological operation to dilate the foreground regions based on OpenCV in Python.

22. Implement the Closing technique as a Morphological operation to dilate the foreground regions based on Open CV.

AIM:
   Implement the Closing technique as a Morphological operation to dilate the foreground regions based on Open CV.

PROCEDURE:
1. Install OpenCV (if not already installed): install opencv-python
2. Import required libraries
3. Read the image
4. Convert the image to grayscale
5. Apply the Closing technique using a kernel
6. Display the images
7. Wait for a key press & close windows

PROGRAM:
```
import cv2
import numpy as np

image = cv2.imread('image.jpg')  # Replace 'image.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

kernel = np.ones((5,5), np.uint8)
closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel)

cv2.imshow('Original Image', image)
cv2.imshow('Closed Image', closed_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT :

Successfully implemented the Closing technique as a Morphological operation to dilate the foreground regions based on OpenCV in Python.

23. Implement the Top hat technique as a Morphological operation to dilate the foreground regions based on Open CV.

AIM:

Implement the Top hat technique as a Morphological operation to dilate the foreground regions based on Open CV.

PROCEDURE:

1. Install OpenCV (if not already installed): install opencv-python
2. Import required libraries
3. Read the image
4. Convert the image to grayscale
5. Apply the Top hat technique using a kernel
6. Display the images
7. Wait for a key press & close windows

PROGRAM:

```
import cv2
import numpy as np

image = cv2.imread('image.jpg')  # Replace 'image.jpg' with your image path
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

kernel = np.ones((5,5), np.uint8)
top_hat_image = cv2.morphologyEx(gray_image, cv2.MORPH_TOPHAT, kernel)

cv2.imshow('Original Image', image)
cv2.imshow('Top Hat Image', top_hat_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
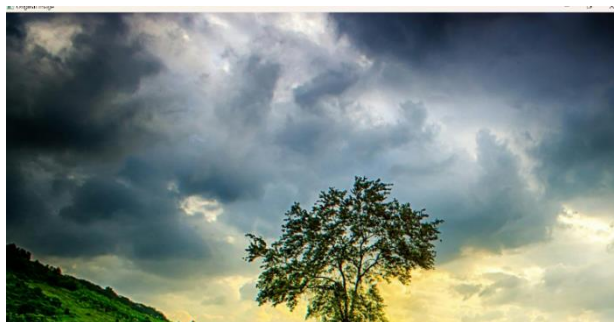
INPUT:
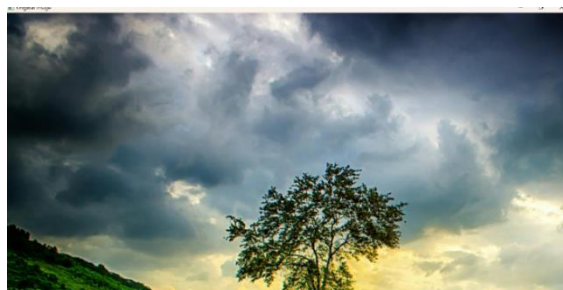
OUTPUT:

RESULT :

Successfully implemented the Top hat technique as a Morphological operation to dilate the foreground regions based on OpenCV in Python.

24. Implement the Black hat technique as a Morphological operation to dilate the foreground regions based on Open CV.

AIM:

To implement the **Black Hat** morphological operation using OpenCV to highlight the darker regions (foreground) of an image.

PROCEDURE:
1. Install OpenCV (if not already installed):
   a. pip install opencv-python
2. Import required libraries:
3. Use cv2 for image processing.
4. Use numpy to define the kernel for morphological operations.
5. Read the input image using cv2.imread().
6. Convert the image to grayscale using cv2.cvtColor().
7. Define a kernel (structuring element) using np.ones() or cv2.getStructuringElement().
8. Apply the Black Hat operation using cv2.morphologyEx() with the cv2.MORPH_BLACKHAT flag.
9. Display the original and processed images using cv2.imshow().
10. Wait for a key press & close windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2
import numpy as np

# Read the image
image = cv2.imread("sample.jpg", cv2.IMREAD_GRAYSCALE)  # Convert to grayscale while reading

# Define a kernel (5x5 structuring element)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

# Apply the Black Hat transformation
blackhat = cv2.morphologyEx(image, cv2.MORPH_BLACKHAT, kernel)

# Display images
cv2.imshow("Original Image", image)
cv2.imshow("Black Hat Image", blackhat)
```

# Wait for a key press and close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()

INPUT:

OUTPUT:

RESULT :

Successfully applied the **Black Hat morphological operation** using OpenCV.

25.Recognize watch from the given image by general Object recognition using Open CV.

AIM:
To recognize a **watch** in a given image using general object recognition techniques in **OpenCV**.

PROCEDURE:

1. 1. Install OpenCV (if not already installed):
   a. pip install opencv-python opencv-contrib-python
2. Import required libraries:
   a. Use cv2 for image processing.
   b. Use numpy for array operations.
3. Load the pre-trained object detection model:
   a. Use Haar Cascade Classifier (pre-trained for watch detection) or Deep Learning-based models like MobileNet SSD or YOLO.
4. Read the input image using cv2.imread().
5. Convert the image to grayscale using cv2.cvtColor().
6. Load the pre-trained Haar Cascade for watch detection.
7. Detect the watch in the image using detectMultiScale().
8. Draw bounding boxes around the detected watch.
9. Display the result using cv2.imshow().
10. Wait for a key press & close windows using cv2.waitKey(0) and cv2.destroyAllWindows().

PROGRAM:

```
import cv2

# Load the pre-trained Haar cascade for watch detection
watch_cascade = cv2.CascadeClassifier("watch_cascade.xml")  # Use a pre-trained watch cascade XML

# Read the image
image = cv2.imread("watch.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect watch in the image
watches = watch_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw bounding boxes around detected watches
for (x, y, w, h) in watches:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the image with detections
cv2.imshow("Detected Watch", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

An image file (watch.jpg) containing a **watch**.

OUTPUT:

A window displaying the **original image** with **bounding boxes** drawn around the detected watch.

RESULT :

Successfully **detected and recognized the watch** from the given image using OpenCV.

26. Implement a function to reverse the frames of the video to create a video in reverse mode using Open CV.

AIM:
To implement a function in Python using **OpenCV** that reads a video, reverses its frames, and saves the reversed video..

PROCEDURE:

1. Install OpenCV (if not installed):
   a. pip install opencv-python
2. Read the video using cv2.VideoCapture().
3. Store all frames in a list by iterating through the video.
4. Reverse the list of frames.
5. Write the reversed frames to a new video file using cv2.VideoWriter().
6. **Save and display the reversed video.**

PROGRAM:

```
import cv2

def reverse_video(input_video_path, output_video_path):
    # Open the video file
    cap = cv2.VideoCapture(input_video_path)

    # Get video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Define video writer
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")  # Codec for MP4
    out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

    frames = []

    # Read all frames
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frames.append(frame)

    cap.release()

    # Reverse frames
    frames.reverse()

    # Write frames in reverse order
    for frame in frames:
        out.write(frame)

    out.release()
    print(f"Reversed video saved as: {output_video_path}")

# Example usage
reverse_video("input_video.mp4", "output_reversed.mp4")
```

INPUT:

A video file named **"input_video.mp4"**.

OUTPUT:

A new video file **"output_reversed.mp4"**, where all frames are played in **reverse order**.

RESULT :

Successfully Implement a function to reverse the frames of the video to create a video in reverse mode using Open CV..

27. Implement a face detection algorithm using Open CV to detect and locate human faces in the images.

AIM:
To implement a **face detection algorithm** using **OpenCV** that detects and locates human faces in an image.

PROCEDURE:

1. Install OpenCV (if not installed):
   a. pip install opencv-python
2. Load the pre-trained Haar Cascade classifier for face detection.
3. Read the input image using cv2.imread().
4. Convert the image to grayscale using cv2.cvtColor().
5. Detect faces using cv2.CascadeClassifier().detectMultiScale().
6. Draw bounding boxes around detected faces.
7. Display the image with the detected faces.

PROGRAM:

```
import cv2

def detect_faces(image_path):
    # Load pre-trained Haar cascade for face detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Read the image
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

    # Detect faces
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Draw bounding boxes around faces
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the result
    cv2.imshow("Face Detection", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
detect_faces("face_image.jpg")
```

INPUT:

An image file **"face_image.jpg"** containing one or more human faces.

OUTPUT:

The input image is displayed with **green rectangles** around detected faces.

RESULT :

- The program successfully detects faces in the image.
- A bounding box is drawn around each detected face.
- The modified image is displayed with detected faces highlighted.

28. Implement a vehicle detection algorithm using Open CV to detect and locate vehicles in each frame of the video.

AIM:

To implement a **vehicle detection algorithm** using **OpenCV** that detects and locates vehicles in each frame of a video.

PROCEDURE:

1. Install OpenCV (if not installed):
    a. pip install opencv-python
2. Load the pre-trained Haar Cascade classifier for vehicle detection.
3. Open the video file using cv2.VideoCapture().
4. Read each frame from the video.
5. Convert the frame to grayscale using cv2.cvtColor().
6. Detect vehicles using cv2.CascadeClassifier().detectMultiScale().
7. Draw bounding boxes around detected vehicles.
8. Display the processed frames with detected vehicles.
9. Press 'q' to exit the video display window.

PROGRAM:

```
import cv2

def detect_vehicles(video_path):
    # Load the pre-trained vehicle classifier
    vehicle_cascade = cv2.CascadeClassifier('cars.xml')  # Pre-trained car cascade XML

    # Open video file
    cap = cv2.VideoCapture(video_path)

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break  # Stop if video ends

        # Convert frame to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect vehicles
        vehicles   =   vehicle_cascade.detectMultiScale(gray,   scaleFactor=1.1,   minNeighbors=5,
minSize=(50, 50))

        # Draw bounding boxes around detected vehicles
        for (x, y, w, h) in vehicles:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        # Display the frame
        cv2.imshow("Vehicle Detection", frame)

        # Press 'q' to exit
```

```
        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

# Example usage
detect_vehicles("traffic_video.mp4")
```

INPUT:

A video file **"traffic_video.mp4"** containing moving vehicles.

OUTPUT:

Vehicles in each frame are detected.
Green rectangles are drawn around detected vehicles.
The processed video is displayed in a window.

RESULT :

The program successfully detects vehicles in each frame, draws bounding boxes around them, and displays the processed video in real-time; press 'q' to exit.

29. Implement an Eye detection algorithm using Open CV to detect and locate human eyes in the images.

AIM:
To implement an **eye detection algorithm** using **OpenCV** that detects and locates human eyes in an image.

PROCEDURE:

1. Install OpenCV (if not installed):
   a. pip install opencv-python
2. Load the pre-trained Haar Cascade classifier for eye detection.
3. Read the input image using cv2.imread().
4. Convert the image to grayscale using cv2.cvtColor().
5. Detect faces first using the face cascade classifier.
6. Within each detected face, detect eyes using the eye cascade classifier.
7. Draw bounding boxes around detected eyes.
8. Display the image with the detected eyes.

PROGRAM:

```
import cv2

def detect_eyes(image_path):
    # Load pre-trained Haar cascades for face and eye detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

    # Read the image
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
```

```
    # Detect faces
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]  # Extract face region
        eyes = eye_cascade.detectMultiScale(face_roi, scaleFactor=1.1, minNeighbors=10, minSize=(15, 15))

        # Draw bounding boxes around eyes
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(image, (x + ex, y + ey), (x + ex + ew, y + ey + eh), (255, 0, 0), 2)

    # Display the result
    cv2.imshow("Eye Detection", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
detect_eyes("face_image.jpg")
```

INPUT:

An image file **"face_image.jpg"** containing a person's face.

OUTPUT:

The input image is displayed with **blue rectangles** around detected eyes.

RESULT :

The program successfully detects human eyes in the image, draws bounding boxes around them, and displays the processed image with detected eyes highlighted.

30. Implement a Smile detection algorithm using Open CV to detect and locate human smile in the images.

AIM:
To implement a **smile detection algorithm** using **OpenCV** that detects and locates human smiles in an image.

PROCEDURE:

1. Install OpenCV (if not installed):
   a. pip install opencv-python
2. Load the pre-trained Haar Cascade classifiers for face and smile detection.
3. Read the input image using cv2.imread().
4. Convert the image to grayscale using cv2.cvtColor().
5. Detect faces first using the face cascade classifier.
6. Within each detected face, detect smiles using the smile cascade classifier.
7. Draw bounding boxes around detected smiles.
8. Display the image with the detected smiles.

PROGRAM:

```
    import cv2
```

```python
def detect_smile(image_path):
    # Load pre-trained Haar cascades for face and smile detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    smile_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_smile.xml')

    # Read the image
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

    # Detect faces
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]  # Extract face region
        smiles = smile_cascade.detectMultiScale(face_roi, scaleFactor=1.8, minNeighbors=20, minSize=(25, 25))

        # Draw bounding boxes around smiles
        for (sx, sy, sw, sh) in smiles:
            cv2.rectangle(image, (x + sx, y + sy + int(h/2)), (x + sx + sw, y + sy + sh + int(h/2)), (0, 255, 0), 2)

    # Display the result
    cv2.imshow("Smile Detection", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
detect_smile("smiling_face.jpg")
```

INPUT:

An image file **"smiling_face.jpg"** containing a person's smiling face.

OUTPUT:

The input image is displayed with **green rectangles** around detected smiles.

RESULT :

The program successfully detects human smiles in the image, draws bounding boxes around them, and displays the processed image with detected smiles highlighted.

31. Implement a Segmentation algorithm using Open CV to segment the given input image based on the given threshold values.

AIM:
To implement an **image segmentation algorithm** using **OpenCV** that segments the given input image based on specified threshold values.

PROCEDURE:

1. Install OpenCV (if not installed):
    a. pip install opencv-python
2. Read the input image using cv2.imread().
3. Convert the image to grayscale using cv2.cvtColor().
4. Apply thresholding using cv2.threshold():

5. If a pixel value is above the threshold, set it to 255 (white).
6. If below, set it to 0 (black).
7. Display the segmented image using cv2.imshow().

PROGRAM:

```
import cv2

def segment_image(image_path, threshold_value=127):
    # Read the input image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Convert to grayscale

    # Apply thresholding for segmentation
    _, segmented_image = cv2.threshold(image, threshold_value, 255, cv2.THRESH_BINARY)

    # Display the result
    cv2.imshow("Original Image", image)
    cv2.imshow("Segmented Image", segmented_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
segment_image("input_image.jpg", 127)  # Adjust threshold value as needed
```

INPUT:
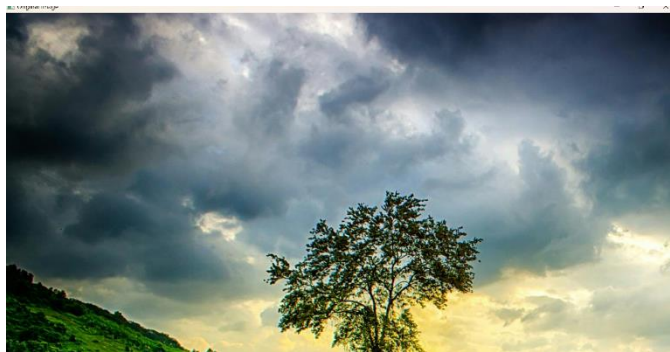
OUTPUT:

RESULT :

The program successfully segments the input image based on the given threshold value, converting it into a binary image where regions are classified as either foreground or background.

32. Write a Python function to create a white image size entered by the user and then create 4 boxes of Black, Blue, Green and Red respectively on each corner of the image. The size of the colored boxes should be 1/10th the size of the image. (HINT: the arrays of ones and zeros can be in more than 2 dimensions).

AIM:

To create a **white image** of a user-defined size and draw **four colored boxes (Black, Blue, Green, and Red)** in the four corners, each occupying **1/10th** of the image size.

PROCEDURE:

1. Take user input for the image size (width × height).
2. Create a white image using numpy.ones().
3. Calculate the box size as 1/10th of the image size.
4. Draw four colored boxes in the four corners using array slicing:
5. Top-left → Black
6. Top-right → Blue
7. Bottom-left → Green
8. Bottom-right → Red
9. Display the image using OpenCV.

PROGRAM:

```
import numpy as np
import cv2

def create_colored_corners(image_size):
    height, width = image_size  # Extract height and width

    # Create a white image (3D array of ones scaled to 255 for RGB)
    image = np.ones((height, width, 3), dtype=np.uint8) * 255

    # Calculate the size of the colored boxes (1/10th of image size)
    box_h, box_w = height // 10, width // 10

    # Define corner regions and their colors
    image[:box_h, :box_w] = [0, 0, 0]      # Top-left (Black)
    image[:box_h, -box_w:] = [255, 0, 0]    # Top-right (Blue)
    image[-box_h:, :box_w] = [0, 255, 0]    # Bottom-left (Green)
    image[-box_h:, -box_w:] = [0, 0, 255]   # Bottom-right (Red)

    # Display the image
    cv2.imshow("Colored Corners Image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
user_width = int(input("Enter image width: "))
user_height = int(input("Enter image height: "))
create_colored_corners((user_height, user_width))
```
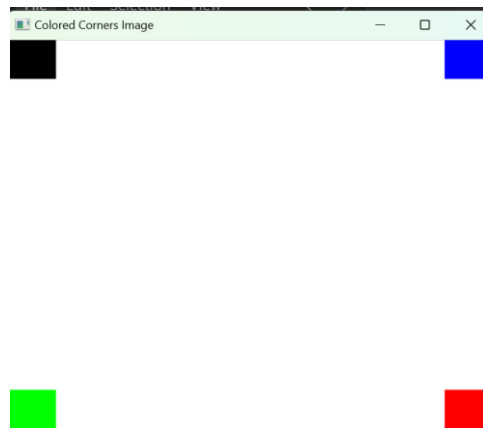
INPUT:

Enter image width: 500
Enter image height: 400

OUTPUT:

RESULT :

The program successfully creates and displays an image of user-defined size with **four colored boxes** at the corners, each occupying **1/10th** of the image size.

33. Write a Python function to create a white image size entered by the user and then create a shape of Rectangle using Open CV.

AIM:
To create a **white image** of a user-defined size and draw a **rectangle** on it using OpenCV.

PROCEDURE:

1. Take user input for the image size (width × height).
2. Create a white image using numpy.ones().
3. Define rectangle coordinates (centered in the image).
4. Draw the rectangle using cv2.rectangle().
5. Display the image using OpenCV.

PROGRAM:

```
import numpy as np
import cv2

def create_rectangle_image(image_size):
    height, width = image_size  # Extract height and width

    # Create a white image (3D array of ones scaled to 255 for RGB)
    image = np.ones((height, width, 3), dtype=np.uint8) * 255

    # Define rectangle coordinates (centered in the image)
    top_left = (width // 4, height // 4)
    bottom_right = (3 * width // 4, 3 * height // 4)

    # Draw the rectangle (color: Blue, thickness: 2)
    cv2.rectangle(image, top_left, bottom_right, (255, 0, 0), 2)

    # Display the image
    cv2.imshow("Rectangle Image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```
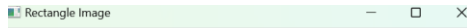
```
# Example usage
user_width = int(input("Enter image width: "))
user_height = int(input("Enter image height: "))
create_rectangle_image((user_height, user_width))
```

INPUT:

> Enter image width: 500
> Enter image height: 400

OUTPUT:

RESULT :

The program successfully creates an image of user-defined size with a **rectangle drawn in the center**.

34. Write a Python function to create a white image size entered by the user and then create a shape of Circle using Open CV.

AIM:
To create a **white image** of a user-defined size and draw a **circle** on it using OpenCV.

PROCEDURE:

1. Take user input for the image size (width × height).
2. Create a white image using numpy.ones().
3. Define the circle properties (center, radius).
4. Draw the circle using cv2.circle().
5. Display the image using OpenCV.

PROGRAM:

```
import numpy as np
import cv2

def create_circle_image(image_size):
    height, width = image_size  # Extract height and width

    # Create a white image (3D array of ones scaled to 255 for RGB)
    image = np.ones((height, width, 3), dtype=np.uint8) * 255
```

```
    # Define the circle properties
    center = (width // 2, height // 2)  # Center of the image
    radius = min(width, height) // 4    # Radius is 1/4th of the smallest dimension

    # Draw the circle (color: Red, thickness: 2)
    cv2.circle(image, center, radius, (0, 0, 255), 2)

    # Display the image
    cv2.imshow("Circle Image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
user_width = int(input("Enter image width: "))
user_height = int(input("Enter image height: "))
create_circle_image((user_height, user_width))
```
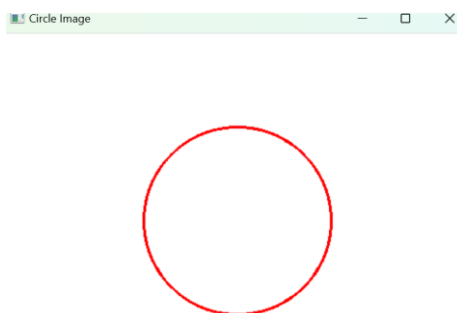
INPUT:

> Enter image width: 500
> Enter image height: 400

OUTPUT:

RESULT :

The program successfully creates an image of user-defined size with a **circle drawn at the center**.

35. Write a Python function to create a text string entered by the user that must be appeared on the given image using Open CV.

AIM:
To create a function that allows the user to input a **text string** and place it on a **given image** using OpenCV.

PROCEDURE:

1. Take user input for the image size (width × height).
2. Create a white image using numpy.ones().
3. Take user input for the text string to be displayed.

4. Define text properties (font, position, size, color, thickness).
5. Draw the text using cv2.putText().
6. Display the image using OpenCV.

PROGRAM:

```
import numpy as np
import cv2

def add_text_to_image(image_size, text):
    height, width = image_size  # Extract height and width

    # Create a white image (3D array of ones scaled to 255 for RGB)
    image = np.ones((height, width, 3), dtype=np.uint8) * 255

    # Define text properties
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1
    font_thickness = 2
    text_color = (0, 0, 255)  # Red color
    text_size = cv2.getTextSize(text, font, font_scale, font_thickness)[0]

    # Calculate text position (centered)
    text_x = (width - text_size[0]) // 2
    text_y = (height + text_size[1]) // 2

    # Draw text on the image
    cv2.putText(image, text, (text_x, text_y), font, font_scale, text_color, font_thickness)

    # Display the image
    cv2.imshow("Image with Text", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
user_width = int(input("Enter image width: "))
user_height = int(input("Enter image height: "))
user_text = input("Enter the text to display: ")
add_text_to_image((user_height, user_width), user_text)
```

INPUT:

Enter image width: 500
Enter image height: 400

OUTPUT:
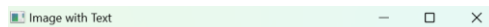
opencv Rocks!

RESULT :

The program successfully creates an image of user-defined size and places the **user's text** in the center.

36. Write a Python function to subtract the background of the given input image based on color levels using Open CV

AIM:
To implement a function that removes the background of an input image **based on color levels** using OpenCV.

PROCEDURE:

1. Read the input image using cv2.imread().
2. Convert the image to HSV color space using cv2.cvtColor().
3. Define the color range for background removal.
4. Create a mask using cv2.inRange().
5. Apply bitwise operation to extract the foreground using cv2.bitwise_and().
6. Display the original and background-subtracted image using OpenCV.

PROGRAM:

```
import cv2
import numpy as np

def subtract_background(image_path, lower_color, upper_color):
    # Read the image
    image = cv2.imread(image_path)

    # Convert to HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define lower and upper range for background color (adjust for specific background)
    lower_bound = np.array(lower_color, dtype=np.uint8)
    upper_bound = np.array(upper_color, dtype=np.uint8)

    # Create mask for background removal
    mask = cv2.inRange(hsv, lower_bound, upper_bound)

    # Invert mask to keep the foreground
    mask_inv = cv2.bitwise_not(mask)

    # Extract the foreground
    foreground = cv2.bitwise_and(image, image, mask=mask_inv)

    # Display results
    cv2.imshow("Original Image", image)
    cv2.imshow("Background Subtracted Image", foreground)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage (adjust color range as needed)
image_path = "image.jpg"  # Replace with the actual image path
lower_color = [30, 30, 30]  # Example lower bound (adjust as needed)
upper_color = [255, 255, 255]  # Example upper bound (adjust as needed)
subtract_background(image_path, lower_color, upper_color)
```

INPUT:

OUTPUT:

RESULT :

The program successfully removes the **background of the input image** based on the specified color levels.

37. Write a Python function to subtract the foreground of the given input image based on color levels using Open CV.

AIM:
To implement a function that removes the foreground of an input image **based on color levels** using OpenCV.

PROCEDURE:

1. Read the input image using cv2.imread().
2. Convert the image to HSV color space using cv2.cvtColor().
3. Define the color range for the foreground.
4. Create a mask using cv2.inRange() to detect foreground objects.
5. Apply bitwise operation to extract the background using cv2.bitwise_and().
6. Display the original and foreground-subtracted image using OpenCV.

PROGRAM:

```
import cv2
import numpy as np

def subtract_foreground(image_path, lower_color, upper_color):
    # Read the image
    image = cv2.imread(image_path)

    # Convert to HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define lower and upper range for foreground color (adjust as needed)
    lower_bound = np.array(lower_color, dtype=np.uint8)
    upper_bound = np.array(upper_color, dtype=np.uint8)
```

```
    # Create mask to remove foreground
    mask = cv2.inRange(hsv, lower_bound, upper_bound)

    # Extract the background
    background = cv2.bitwise_and(image, image, mask=mask)

    # Display results
    cv2.imshow("Original Image", image)
    cv2.imshow("Foreground Subtracted Image (Only Background)", background)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage (adjust color range as needed)
image_path = "image.jpg"  # Replace with the actual image path
lower_color = [0, 50, 50]   # Example lower bound for foreground color (adjust as needed)
upper_color = [120, 255, 255]  # Example upper bound for foreground color (adjust as needed)
subtract_foreground(image_path, lower_color, upper_color)
```
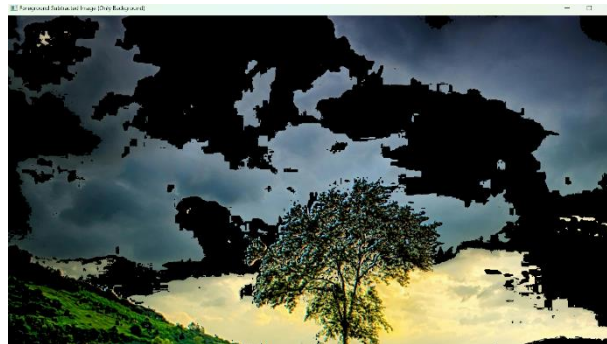
INPUT:

OUTPUT:

RESULT :

The program successfully removes the **foreground of the input image** based on the specified color levels.

38. Write a Python function to Count the number of faces for the given input image using Open CV.

AIM:
To implement a function that detects and counts the number of faces present in a given input image using OpenCV.

PROCEDURE:

1. Load the image using cv2.imread().
2. Convert the image to grayscale using cv2.cvtColor().
3. Load a pre-trained face detection model (Haar Cascade classifier).

4. Detect faces in the image using detectMultiScale().
5. Draw bounding boxes around detected faces.
6. Display the image with detected faces and print the total count.

PROGRAM:

```
import cv2

def count_faces(image_path):
    # Load the image
    image = cv2.imread(image_path)

    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Load the pre-trained Haar cascade face detection model
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Count the number of faces
    face_count = len(faces)
    print(f"Number of faces detected: {face_count}")

    # Draw rectangles around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the image with detected faces
    cv2.imshow("Face Detection", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return face_count

# Example usage
image_path = "face_image.jpg"  # Replace with your image path
count_faces(image_path)
```

INPUT:

OUTPUT:

RESULT :

The program successfully creates an image of user-defined size and places the **user's text** in the center.

39. Write a Python function to play the given video in reverse mode in slow motion.

AIM:
   To Write a Python function to play the given video in reverse mode in slow motion..

PROCEDURE:

   1. Install required libraries (if not already installed): install opencv-python
   2. Import required libraries
   3. Read the video file
   4. Extract all frames from the video
   5. Reverse the order of frames
   6. Play the frames in reverse with a delay for slow motion

PROGRAM:

```python
import cv2

def play_video_reverse_slow(video_path):
    cap = cv2.VideoCapture(video_path)
    frames = []
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        frames.append(frame)
    cap.release()

    for frame in reversed(frames):
        cv2.imshow('Reverse Slow Motion Video', frame)
        if cv2.waitKey(100) & 0xFF == ord('q'):  # 100 ms delay for slow motion
            break
    cv2.destroyAllWindows()
play_video_reverse_slow('video.mp4')
```

INPUT:
   Input is a Video file

OUTPUT:
   Input is a Video file

RESULT :
   Successfully played the given video in reverse mode in slow motion using Python.

40. Write a Python function to extract the text from videos.

AIM:
To Write a Python function to extract the text from videos.

PROCEDURE:

1. Install required libraries (if not already installed): install opencv-python, pytesseract
2. Import required libraries
3. Read the video file
4. Extract frames from the video
5. Convert frames to grayscale
6. Apply Optical Character Recognition (OCR) using pytesseract
7. Display the extracted text

PROGRAM:

```python
import cv2
import pytesseract

def extract_text_from_video(video_path):
    cap = cv2.VideoCapture(video_path)
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        text = pytesseract.image_to_string(gray_frame)
        print(text)
    cap.release()
    cv2.destroyAllWindows()

extract_text_from_video('video.mp4')  # Replace 'video.mp4' with your video path
```

INPUT:
Input is a Video file

OUTPUT:
Input is a Video file

RESULT :
Successfully extracted the text from videos using Python.