Computer Graphics & Multimedia

[COURSE CODE : CICPE06]

PRACTICAL RECORD FILE

Submitted By:

Name : CHANDAN

Roll No : 2021UCI8002

Branch : CSE(IOT)

Semester : 6

Submitted To:

Dr.

(Department of CSE)

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Geeta Colony, New Delhi

Delhi-110031

EXPERIMENT - 01

AIM : Write a program to draw an eclipse.

System Requirements:

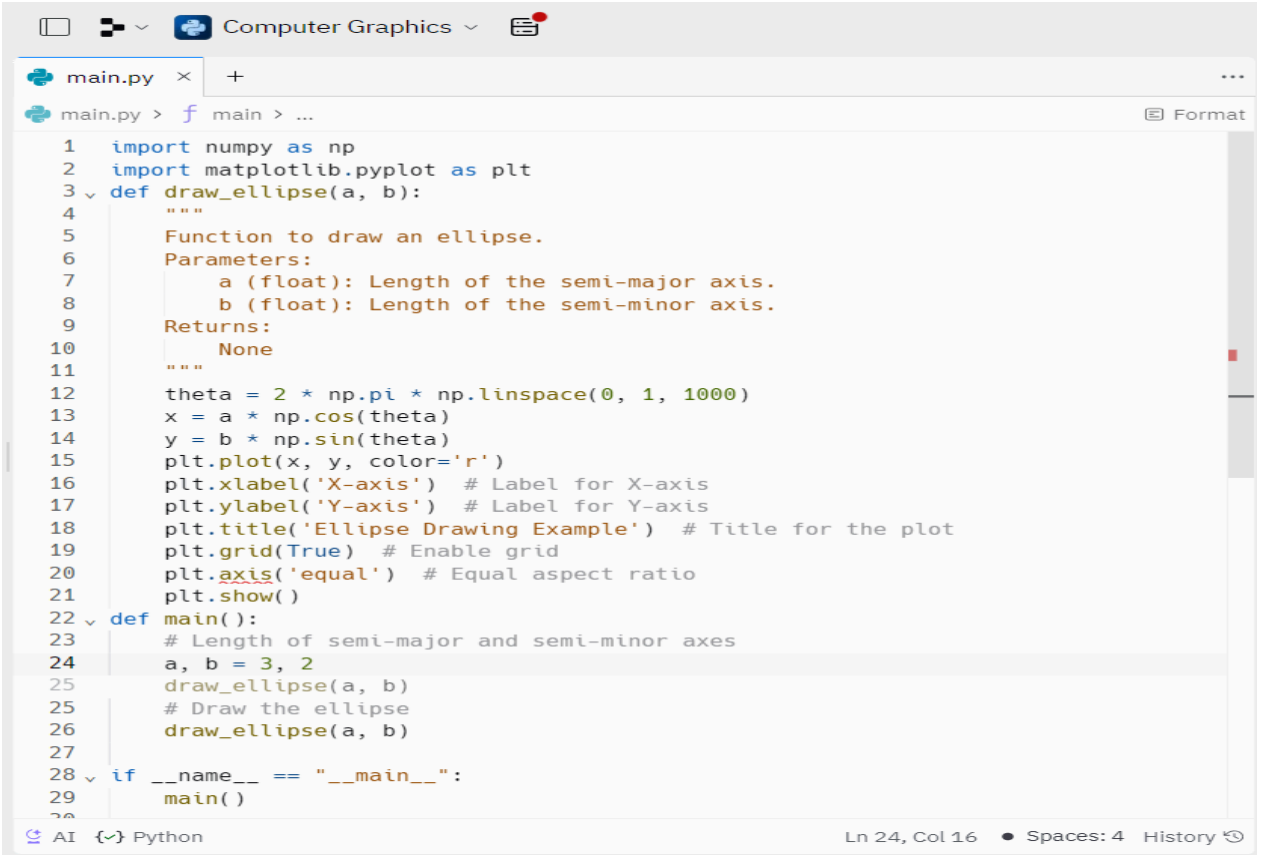
- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

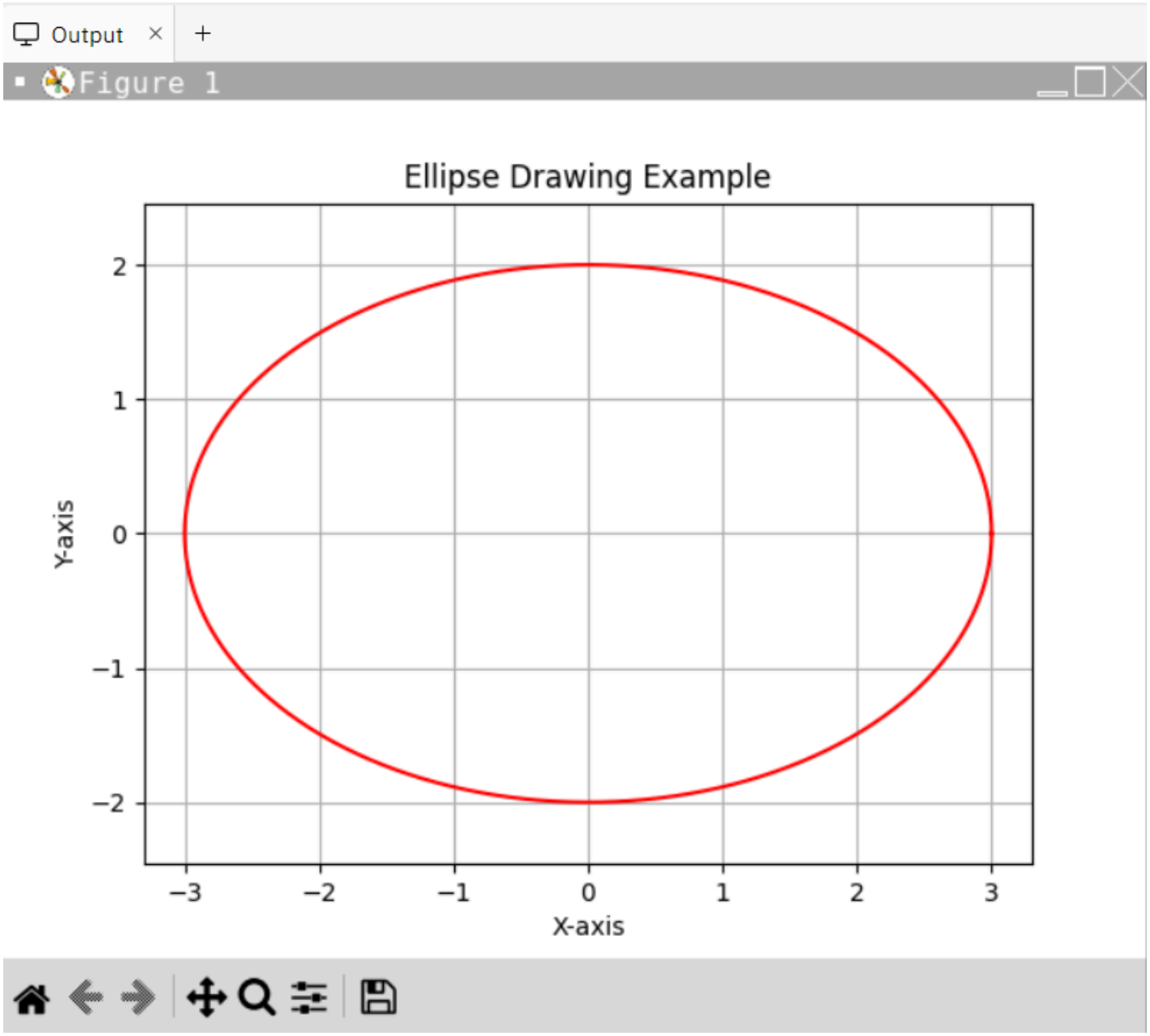
An ellipse is a closed curve that resembles a flattened circle. It is defined by two radii, a major axis (a) and a minor axis (b).

The center of the ellipse is represented by the point (h, k). The equation of an ellipse centered at (h, k) is given by $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ ı this program, we define a function `draw_ellipse` that takes the center coordinates (h, k), major and minor axes (a, b) as input and plots the ellipse using Matplotlib's `Ellipse` function.

Source Code :



Output :


Output
Figure 1
Ellipse Drawing Example
Y-axis
X-axis
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

EXPERIMENT - 02

AIM : Write a program to perform 2D Transformation.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

This program demonstrates how to draw a line using Python's Matplotlib library. A line is a straight path connecting two points. In Cartesian coordinates, it's represented by the equation y = mx + c, where m is the slope of the line, and c is the y-intercept. The slope (m) defines the angle of inclination of the line with respect to the x-axis. When m is positive, the line slopes upwards from left to right, and when m is negative, the line slopes downwards from left to right. The y-intercept (c) is the point where the line intersects the y-axis. In this program, we define a function `draw_line` that takes the coordinates of two points (x1, y1) and (x2, y2) as input and plots a line between them using Matplotlib's `plot` function.

Source Code :

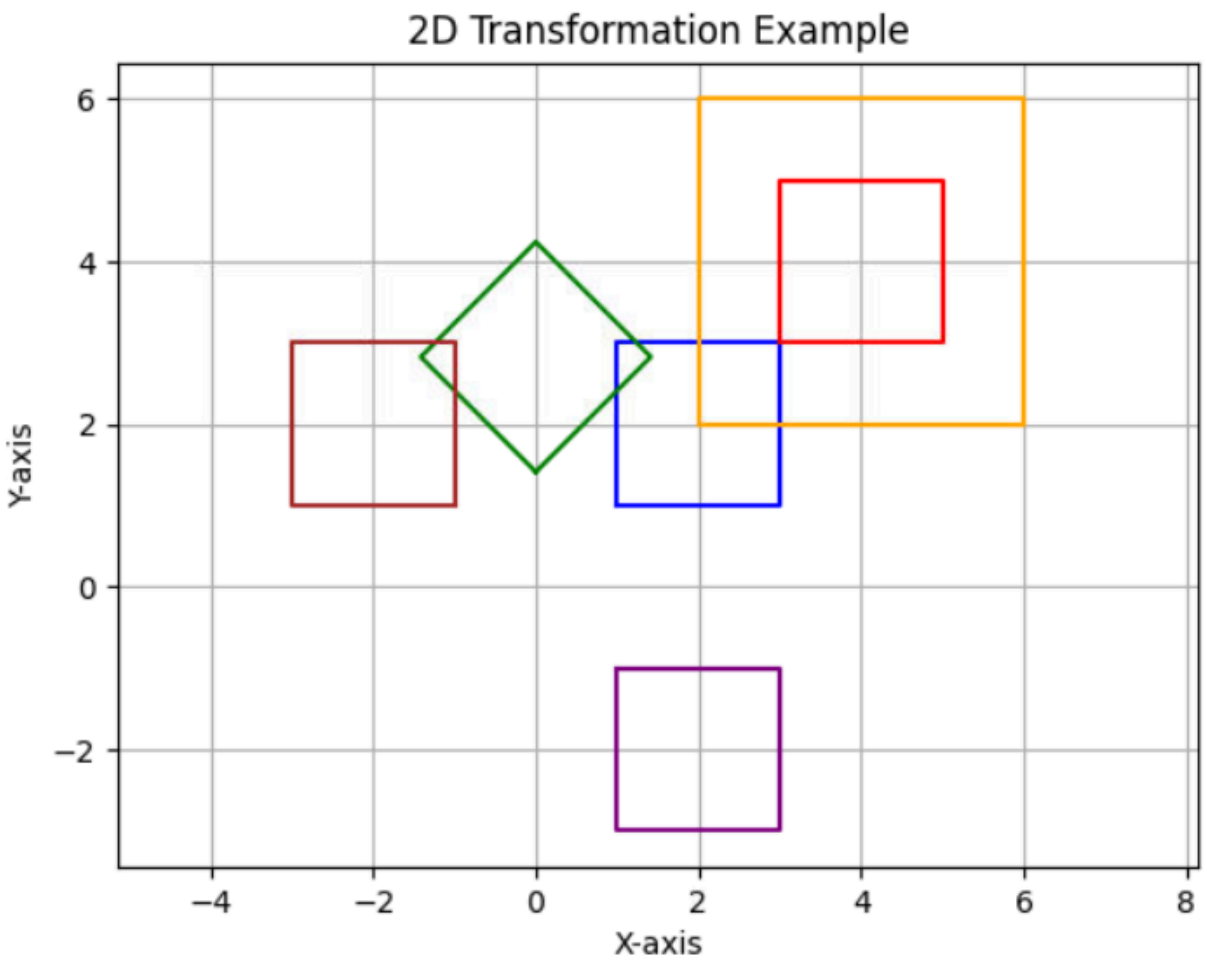
```
1. import matplotlib.pyplot as plt
2. from math import cos, sin, radians
3.
4. # Function to plot the original shape
5. def plot_shape(vertices, color):
6.     x = [vertex[0] for vertex in vertices]
7.     y = [vertex[1] for vertex in vertices]
8.     plt.plot(x, y, color=color)
9.
10.# Function to perform translation
11.def translate(vertices, tx, ty):
12.    translated_vertices = [[vertex[0] + tx, vertex[1] + ty] for vertex in vertices]
13.    return translated_vertices
14.
15.# Function to perform rotation
16.def rotate(vertices, angle):
17.    radians_angle = radians(angle)
18.    rotated_vertices = [[vertex[0] * cos(radians_angle) - vertex[1] *
   sin(radians_angle),
19.                         vertex[0] * sin(radians_angle) + vertex[1] *
   cos(radians_angle)] for vertex in vertices]
```

```
    return rotated_vertices

# Function to perform scaling
def scale(vertices, sx, sy):
    scaled_vertices = [[vertex[0] * sx, vertex[1] * sy] for vertex in vertices]
    return scaled_vertices

# Function to perform reflection
def reflect(vertices, axis):
    if axis == 'x':
        reflected_vertices = [[vertex[0], -vertex[1]] for vertex in vertices]
    elif axis == 'y':
        reflected_vertices = [[-vertex[0], vertex[1]] for vertex in vertices]
    return reflected_vertices

def main():
    # Original shape vertices
    vertices = [[1, 1], [1, 3], [3, 3], [3, 1], [1, 1]]

    # Plot the original shape
    plot_shape(vertices, 'blue')

    # Perform transformations
    translated_vertices = translate(vertices, 2, 2)
    rotated_vertices = rotate(vertices, 45)
    scaled_vertices = scale(vertices, 2, 2)
    reflected_vertices_x = reflect(vertices, 'x')
    reflected_vertices_y = reflect(vertices, 'y')

    # Plot the transformed shapes
    plot_shape(translated_vertices, 'red')  # Translated shape
    plot_shape(rotated_vertices, 'green')  # Rotated shape
    plot_shape(scaled_vertices, 'orange')  # Scaled shape
    plot_shape(reflected_vertices_x, 'purple')  # Reflected shape along x-axis
    plot_shape(reflected_vertices_y, 'brown')  # Reflected shape along y-axis

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('2D Transformation Example')
    plt.grid(True)
    plt.axis('equal')
    plt.show()

if __name__ == "__main__":
    main()
```

Output :

Output

Figure 1



EXPERIMENT - 03

AIM : Write a program to perform 3D Transformations.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)
- NumPy library

Description:

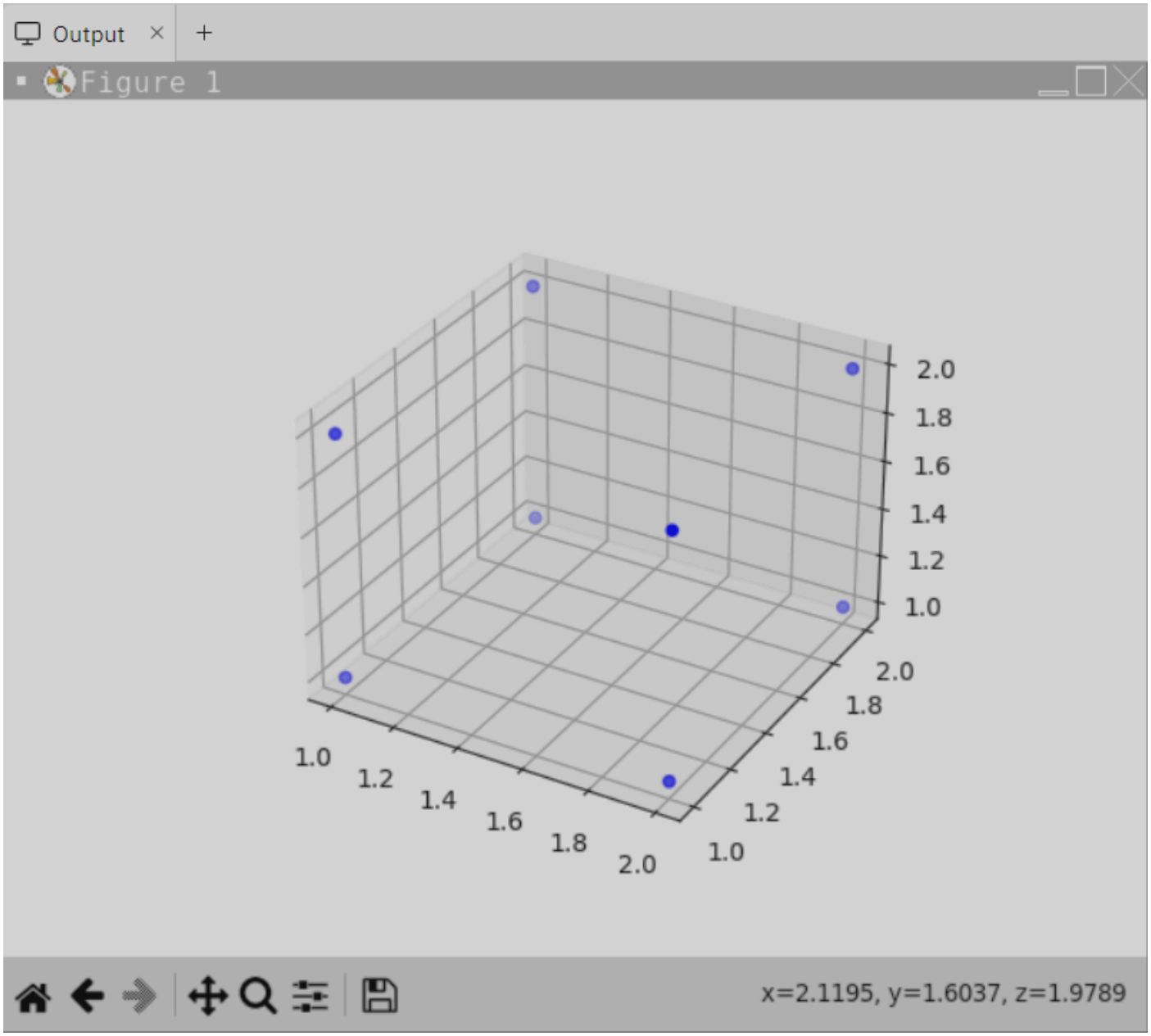
This program demonstrates how to perform basic 3D transformations such as translation, rotation, and scaling on a 3D object. The transformations are applied to a set of points representing the vertices of the object.

Source Code :

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from mpl_toolkits.mplot3d import Axes3D
4.
5. # Function to plot the original 3D object
6. def plot_3d_object(vertices, color):
7.     fig = plt.figure()
8.     ax = fig.add_subplot(111, projection='3d')
9.     x = [vertex[0] for vertex in vertices]
10.    y = [vertex[1] for vertex in vertices]
11.    z = [vertex[2] for vertex in vertices]
12.    ax.scatter(x, y, z, color=color)
13.
14.# Function to perform translation in 3D space
15.def translate_3d(vertices, tx, ty, tz):
16.    translated_vertices = [[vertex[0] + tx, vertex[1] + ty, vertex[2] + tz] for
   vertex in vertices]
17.    return translated_vertices
18.
19.# Function to perform rotation around the x-axis in 3D space
20.def rotate_x_3d(vertices, angle):
21.    radians = np.radians(angle)
22.    rotation_matrix = np.array([[1, 0, 0],
23.                                [0, np.cos(radians), -np.sin(radians)],
24.                                [0, np.sin(radians), np.cos(radians)]])
25.    rotated_vertices = np.dot(vertices, rotation_matrix)
26.    return rotated_vertices
27.
28.# Function to perform rotation around the y-axis in 3D space
29.def rotate_y_3d(vertices, angle):
```

```
    radians = np.radians(angle)
    rotation_matrix = np.array([[np.cos(radians), 0, np.sin(radians)],
                                [0, 1, 0],
                                [-np.sin(radians), 0, np.cos(radians)]])
    rotated_vertices = np.dot(vertices, rotation_matrix)
    return rotated_vertices

# Function to perform rotation around the z-axis in 3D space
def rotate_z_3d(vertices, angle):
    radians = np.radians(angle)
    rotation_matrix = np.array([[np.cos(radians), -np.sin(radians), 0],
                                [np.sin(radians), np.cos(radians), 0],
                                [0, 0, 1]])
    rotated_vertices = np.dot(vertices, rotation_matrix)
    return rotated_vertices

# Function to perform scaling in 3D space
def scale_3d(vertices, sx, sy, sz):
    scaled_vertices = [[vertex[0] * sx, vertex[1] * sy, vertex[2] * sz] for vertex
  in vertices]
    return scaled_vertices

def main():
    # Original 3D object vertices
    vertices = np.array([[1, 1, 1],
                         [1, 2, 1],
                         [2, 2, 1],
                         [2, 1, 1],
                         [1, 1, 2],
                         [1, 2, 2],
                         [2, 2, 2],
                         [2, 1, 2]])

    # Plot the original 3D object
    plot_3d_object(vertices, 'blue')

    # Perform transformations
    translated_vertices = translate_3d(vertices, 2, 2, 2)
    rotated_x_vertices = rotate_x_3d(vertices, 45)
    rotated_y_vertices = rotate_y_3d(vertices, 45)
    rotated_z_vertices = rotate_z_3d(vertices, 45)
    scaled_vertices = scale_3d(vertices, 2, 2, 2)

    # Plot the transformed 3D objects
    plot_3d_object(translated_vertices, 'red')  # Translated object
    plot_3d_object(rotated_x_vertices, 'green')  # Rotated object around x-axis
    plot_3d_object(rotated_y_vertices, 'orange')  # Rotated object around y-axis
    plot_3d_object(rotated_z_vertices, 'purple')  # Rotated object around z-axis
    plot_3d_object(scaled_vertices, 'brown')  # Scaled object

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('3D Transformation Example')
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

Output :


Output
Figure 1
2.0
1.8
1.6
1.4
1.2
1.0
1.0
1.2
1.4
1.6
1.8
2.0
x=2.1195, y=1.6037, z=1.9789

EXPERIMENT - 04

AIM : Implement Polygon Drawing Algorithm.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

In computer graphics, a polygon drawing algorithm is used to draw polygons, which are geometric shapes with straight sides. One common algorithm for drawing polygons is the Bresenham's Line Drawing Algorithm. This algorithm calculates the coordinates of each pixel along the edges of the polygon to effectively draw the entire polygon.

Bresenham's Line Drawing Algorithm:

Bresenham's algorithm is primarily used to draw lines but can be adapted to draw polygons by connecting multiple lines. The algorithm calculates the coordinates of each pixel along the line segment between two given points efficiently, without needing to use floating-point arithmetic.

Formula:

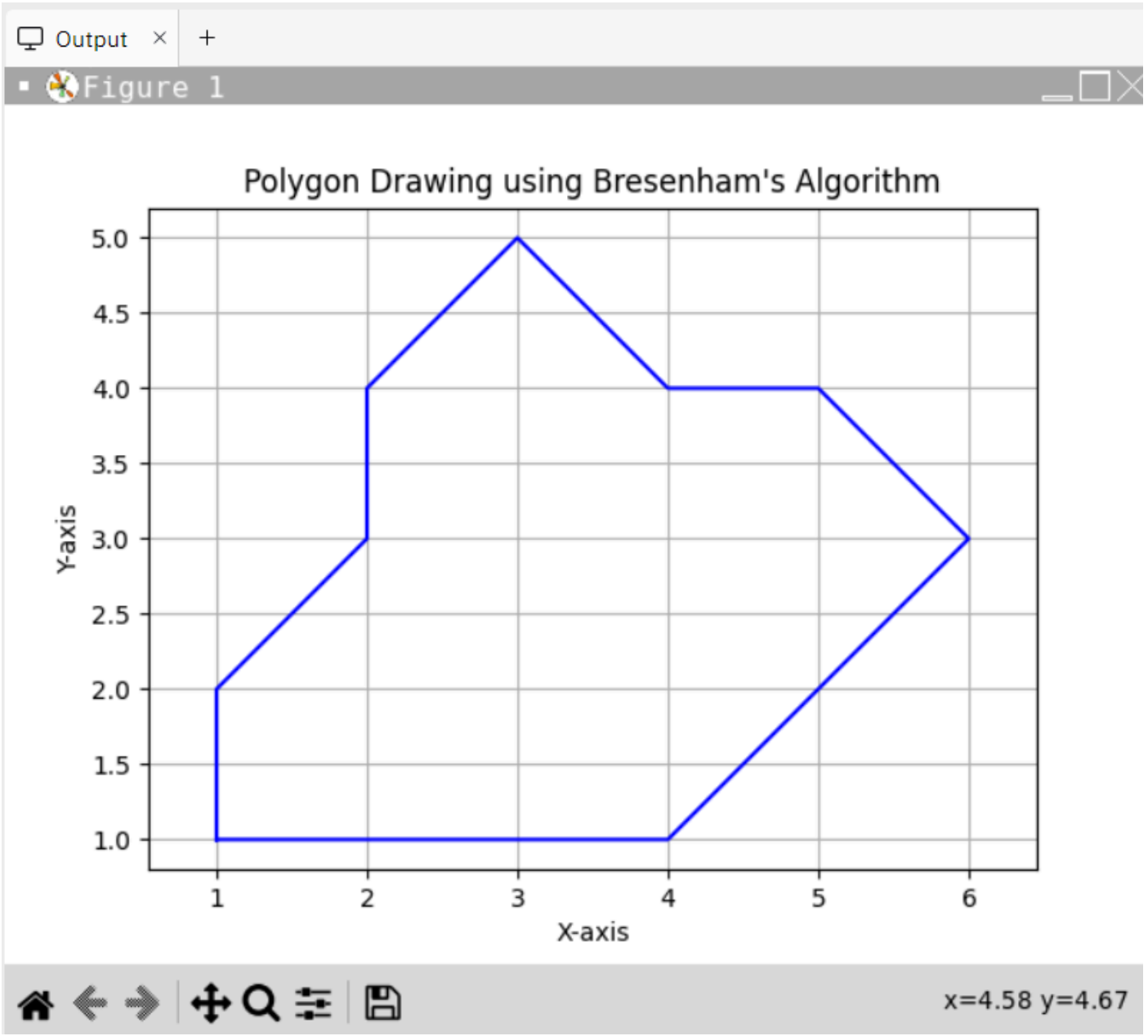
The main idea behind Bresenham's algorithm is to determine which pixel to choose at each step between two points based on the pixel's distance to the ideal line path. The algorithm decides whether to move horizontally or vertically to the next pixel, based on the slope of the line segment.

Source Code :

```
1. import matplotlib.pyplot as plt
2.
3. def draw_polygon(vertices):
4.     """
5.     Function to draw a polygon using Bresenham's Line Drawing Algorithm.
6.
7.     Parameters:
8.         vertices (list of tuples): List of (x, y) coordinates of polygon vertices.
9.
10.    Returns:
11.        None
12.    """
13.    # Initialize lists to store x and y coordinates of points on the polygon edges
14.    x_coords = []
```

```
    y_coords = []

    # Iterate through vertices to draw edges of the polygon
    num_vertices = len(vertices)
    for i in range(num_vertices):
        x0, y0 = vertices[i]
        x1, y1 = vertices[(i + 1) % num_vertices]  # Wrap around to the first
  vertex for the last edge

        # Apply Bresenham's Line Drawing Algorithm
        dx = abs(x1 - x0)
        dy = abs(y1 - y0)
        sx = 1 if x0 < x1 else -1
        sy = 1 if y0 < y1 else -1
        err = dx - dy

        while True:
            # Plot the current point (x0, y0)
            x_coords.append(x0)
            y_coords.append(y0)

            # Check if we have reached the end point (x1, y1)
            if x0 == x1 and y0 == y1:
                break

            # Calculate next point along the line
            e2 = 2 * err
            if e2 > -dy:
                err -= dy
                x0 += sx
            if e2 < dx:
                err += dx
                y0 += sy

    # Plot the polygon
    plt.plot(x_coords, y_coords, color='blue')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Polygon Drawing using Bresenham\'s Algorithm')
    plt.grid(True)
    plt.axis('equal')
    plt.show()

def main():
    # Define vertices of the polygon
    vertices = [(1, 1), (3, 5), (6, 3), (4, 1), (1, 1)]

    # Draw the polygon
    draw_polygon(vertices)

if __name__ == "__main__":
    main()
```

Output :


Output
Figure 1
Polygon Drawing using Bresenham's Algorithm
Y-axis
X-axis
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6
x=4.58 y=4.67

EXPERIMENT - 05

AIM : Write a program for Spline Drawing.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

In computer graphics, a spline is a smooth curve that is defined by a set of control points. Spline curves are widely used for representing complex shapes and smooth curves in computer graphics applications. There are various types of splines, such as Bezier splines, B-splines, and NURBS (Non-Uniform Rational B-Splines), each with its own properties and advantages.

Spline Drawing Algorithm:

One commonly used algorithm for drawing splines is the De Casteljau's algorithm for Bezier curves. Bezier curves are defined by a set of control points, and the De Casteljau's algorithm recursively calculates intermediate control points along the curve to draw a smooth spline passing through these control points.

Steps of De Casteljau's Algorithm:

1. Given a set of control points, draw straight lines connecting consecutive control points.
2. Divide each line segment into smaller segments by interpolating points along the line.
3. Repeat the process recursively until a single point is obtained for each line segment.
4. Connect the final points obtained from each segment to draw the spline curve.

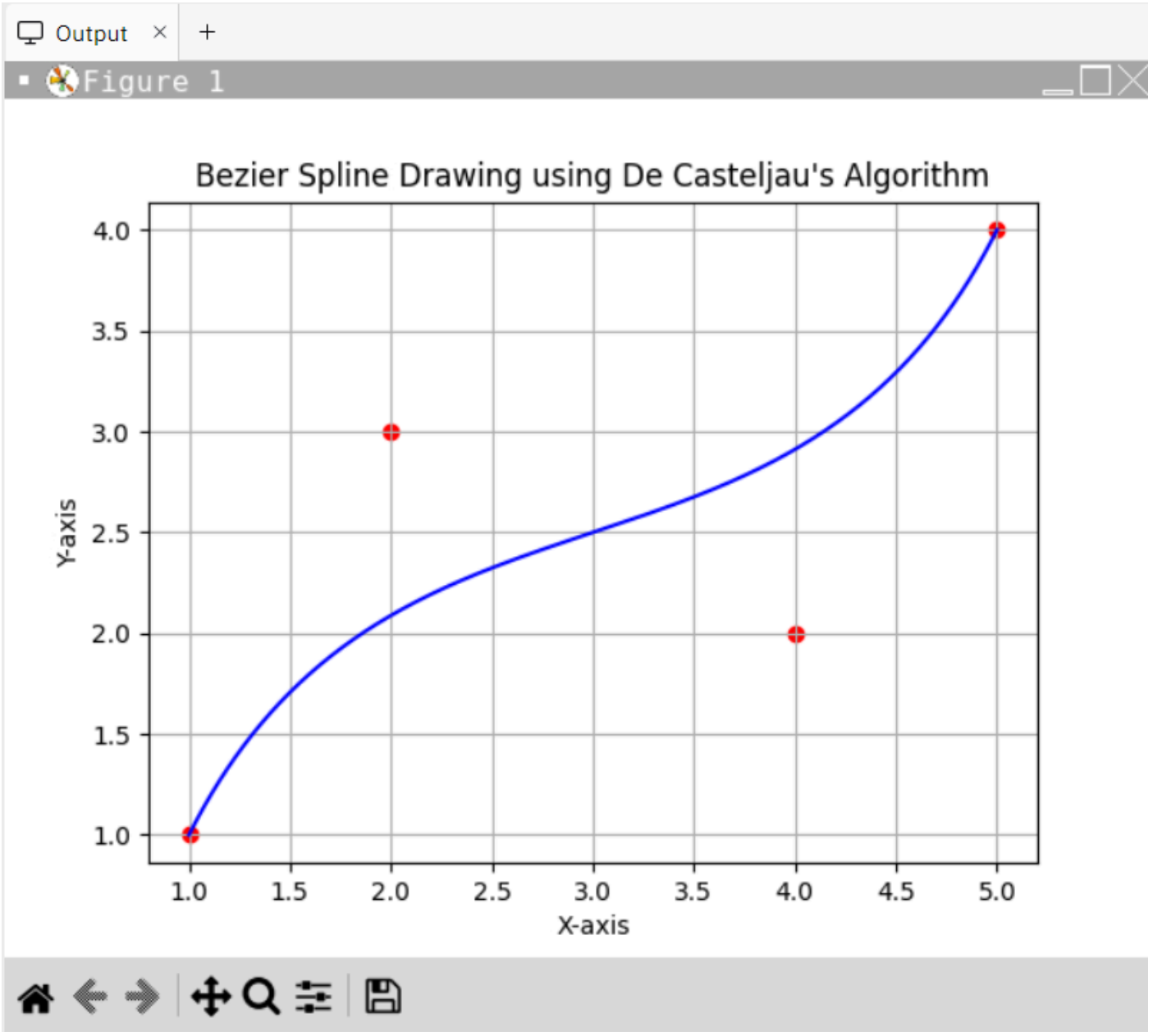
Source Code :

Here's an implementation of De Casteljau's algorithm for drawing a Bezier spline in Python:

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3.
4. def de_casteljau(control_points, t):
5.     """
6.     Function to compute De Casteljau's algorithm for Bezier curve.
7.
8.     Parameters:
9.         control_points (list of tuples): List of (x, y) coordinates of control
   points.
```

```
        t (float): Parameter value between 0 and 1.

    Returns:
        tuple: (x, y) coordinates of the point on the curve at parameter t.
    """
    if len(control_points) == 1:
        return control_points[0]

    new_control_points = []
    for i in range(len(control_points) - 1):
        x = (1 - t) * control_points[i][0] + t * control_points[i+1][0]
        y = (1 - t) * control_points[i][1] + t * control_points[i+1][1]
        new_control_points.append((x, y))

    return de_casteljau(new_control_points, t)

def draw_bezier_spline(control_points):
    """
    Function to draw a Bezier spline using De Casteljau's algorithm.

    Parameters:
        control_points (list of tuples): List of (x, y) coordinates of control points.

    Returns:
        None
    """
    t_values = np.linspace(0, 1, 1000)
    x_values = []
    y_values = []

    for t in t_values:
        point = de_casteljau(control_points, t)
        x_values.append(point[0])
        y_values.append(point[1])

    plt.plot(x_values, y_values, color='blue')
    plt.scatter([point[0] for point in control_points], [point[1] for point in control_points], color='red')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Bezier Spline Drawing using De Casteljau\'s Algorithm')
    plt.grid(True)
    plt.axis('equal')
    plt.show()

def main():
    # Define control points for the Bezier spline
    control_points = [(1, 1), (2, 3), (4, 2), (5, 4)]

    # Draw the Bezier spline
    draw_bezier_spline(control_points)

if __name__ == "__main__":
    main()

```

Output :


Output
Figure 1
Bezier Spline Drawing using De Casteljau's Algorithm
Y-axis
X-axis
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

EXPERIMENT - 06

AIM : Write a program for Image Analysis.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)
- OpenCV library

Description:

Image analysis in computer graphics involves processing and analyzing digital images to extract useful information or features from them. This can include tasks such as edge detection, object recognition, image segmentation, and more. Image analysis is widely used in various fields, including medical imaging, remote sensing, computer vision, and more.

Source Code :

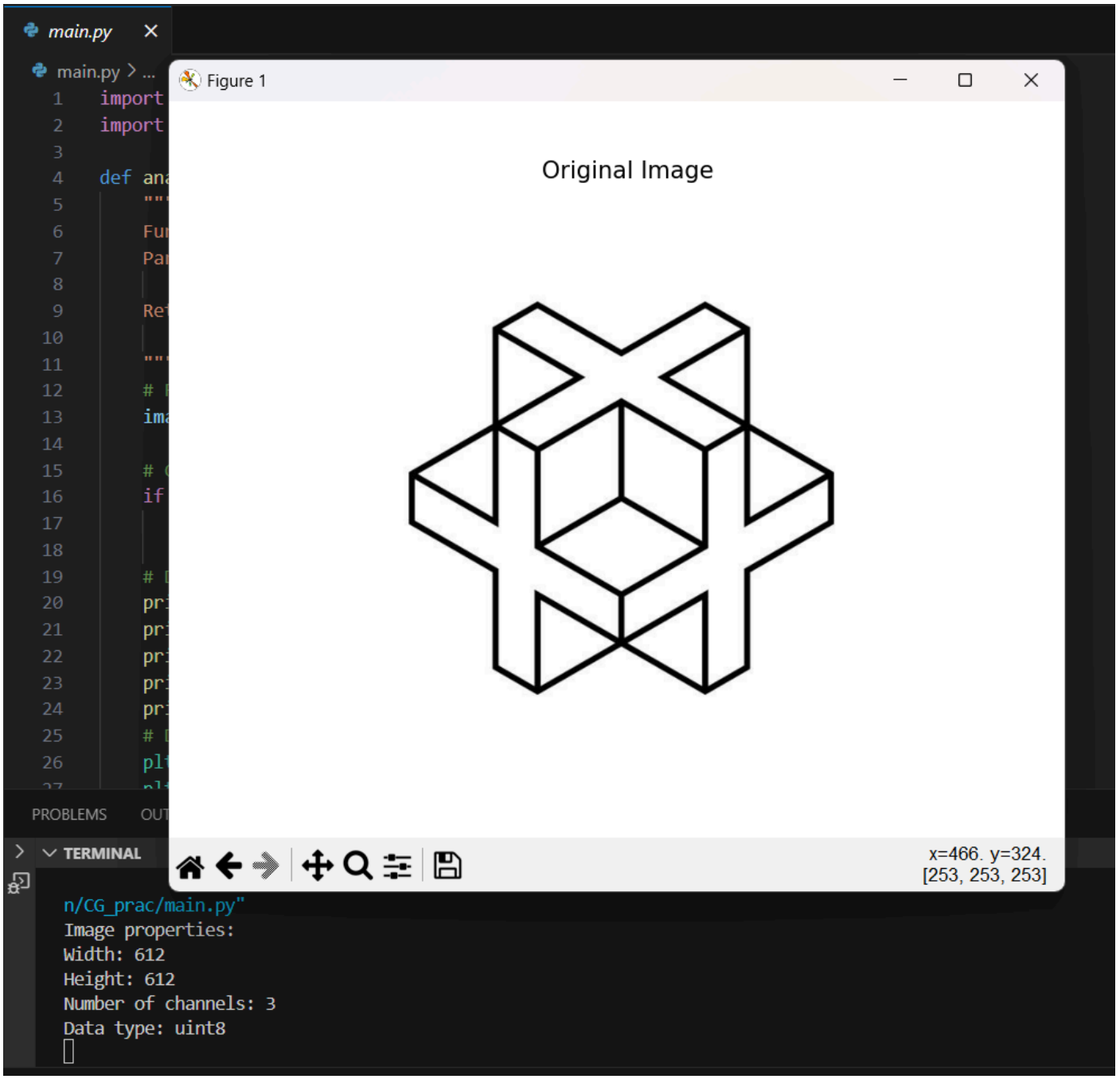
```
EXPLORER
CG_PRAC
  image.jpg
  main.py
main.py
main.py > ...
```

```python
import cv2
import matplotlib.pyplot as plt

def analyze_image(image_path):
    """
    Function to analyze an image and display its properties.
    Parameters:
        image_path (str): Path to the image file.
    Returns:
        None
    """
    # Read the image
    image = cv2.imread(image_path)

    # Check if image is successfully loaded
    if image is None:
        print("Error: Unable to load image.")
        return
    # Display image properties
    print("Image properties:")
    print("Width:", image.shape[1])
    print("Height:", image.shape[0])
    print("Number of channels:", image.shape[2] if len(image.shape) == 3 else 1)
    print("Data type:", image.dtype)
    # Display the image
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title('Original Image')
    plt.show()
def main():
    # Path to the sample image
    image_path = 'image.jpg'
    # Analyze the image
    analyze_image(image_path)
if __name__ == "__main__":
    main()
```

OUTLINE
TIMELINE
PROJECT COMPONENTS

Output :

```
PROBLEMS  OUTPUT  TERMINAL  PORTS
TERMINAL
n/CG_prac/main.py"
Image properties:
Width: 612
Height: 612
Number of channels: 3
Data type: uint8
```


main.py
Figure 1
Original Image
x=466. y=324.
[253, 253, 253]
PROBLEMS
TERMINAL
n/CG_prac/main.py"
Image properties:
Width: 612
Height: 612
Number of channels: 3
Data type: uint8

EXPERIMENT - 07

AIM : Write a program for fructual motion detection from video.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)
- OpenCV library

Description:

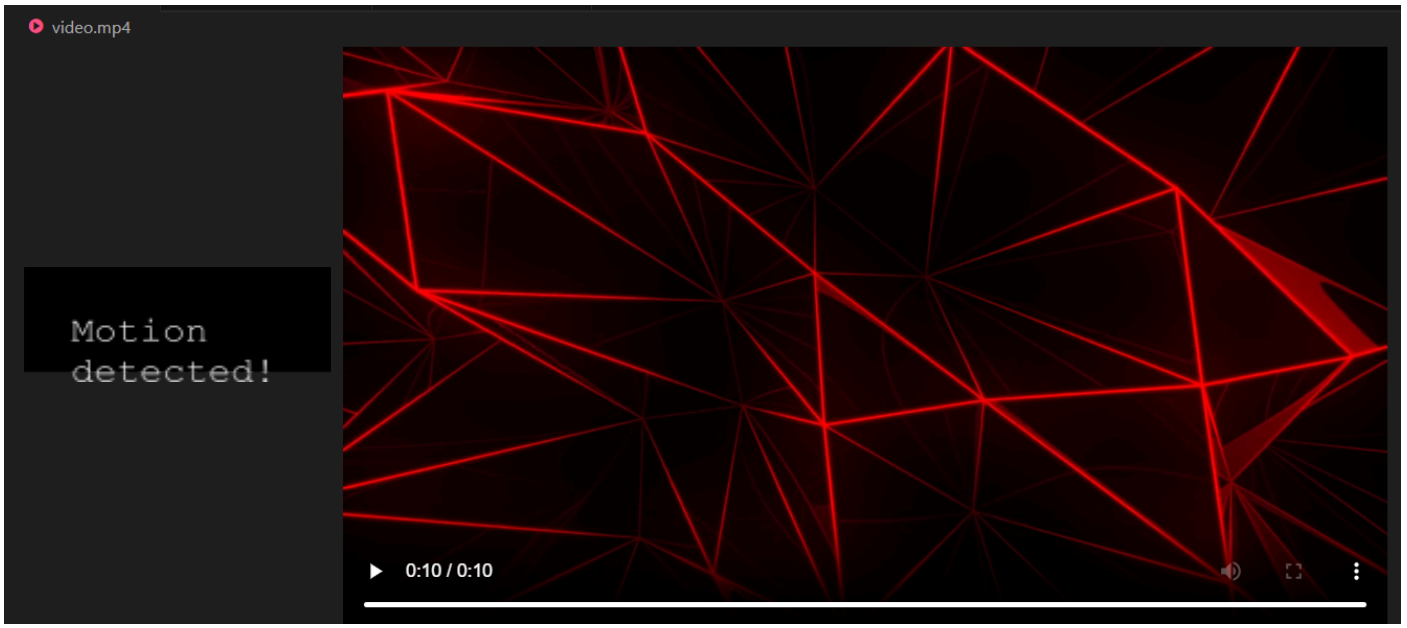
Fractal motion detection is a technique used to detect motion in video sequences by comparing fractal patterns between consecutive frames. Fractals are geometric shapes that exhibit self-similarity at different scales. In fractal motion detection, the fractal dimension of regions in consecutive frames is compared to determine if there is any motion.

Source Code :

```
import cv2
import numpy as np

def calculate_fractal_dimension(image):
    """
    Function to calculate the fractal dimension of an image.

    Parameters:
        image (numpy.ndarray): Input image.

    Returns:
        float: Fractal dimension of the image.
    """
    # Convert image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply edge detection
    edges = cv2.Canny(gray_image, 100, 200)

    # Calculate fractal dimension using box-counting method
    num_white_pixels = np.sum(edges == 255)
    box_sizes = np.arange(1, min(edges.shape) // 2)
    num_boxes = []
    for size in box_sizes:
        num_boxes.append(np.sum(cv2.dilate(edges, np.ones((size, size))) == 255))
    coeffs = np.polyfit(np.log(box_sizes), np.log(num_boxes), 1)
    fractal_dimension = coeffs[0]
```

```
28.
29.     return fractal_dimension
30.
31.def detect_motion(video_path):
32.     """
33.     Function to detect motion from a video using fractal motion detection.
34.
35.     Parameters:
36.         video_path (str): Path to the input video file.
37.
38.     Returns:
39.         None
40.     """
41.     # Open video capture object
42.     cap = cv2.VideoCapture(video_path)
43.
44.     # Check if video is successfully opened
45.     if not cap.isOpened():
46.         print("Error: Unable to open video.")
47.         return
48.
49.     # Read the first frame
50.     ret, prev_frame = cap.read()
51.
52.     # Check if frame is successfully read
53.     if not ret:
54.         print("Error: Unable to read frame.")
55.         return
56.
57.     # Loop through the video frames
58.     while True:
59.         # Read the next frame
60.         ret, next_frame = cap.read()
61.
62.         # Break the loop if video ends
63.         if not ret:
64.             break
65.
66.         # Calculate fractal dimensions of consecutive frames
67.         fractal_dimension_prev = calculate_fractal_dimension(prev_frame)
68.         fractal_dimension_next = calculate_fractal_dimension(next_frame)
69.
70.         # Compare fractal dimensions to detect motion
71.         if abs(fractal_dimension_next - fractal_dimension_prev) > 0.1:
72.             print("Motion detected!")
73.
74.         # Update previous frame
75.         prev_frame = next_frame
76.
77.     # Release video capture object
78.     cap.release()
79.
80.def main():
81.     # Path to the input video file
82.     video_path = 'video.mp4'
83.
84.     # Detect motion in the video
85.     detect_motion(video_path)
86.
87.if __name__ == "__main__":
88.     main()
```

Output :



Prints "Motion detected!" to the console if motion is detected between consecutive frames in the input video.

EXPERIMENT - 08

AIM : Write a program to apply Animation.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

Animation is the process of creating a sequence of images or frames and displaying them in rapid succession to create the illusion of motion. In computer graphics, animation involves generating and displaying a series of frames, typically at a high frame rate, to create the perception of movement.

1. Animation in Computer Graphics:In computer graphics, animation is widely used for various purposes, including entertainment, education, simulation, and visualization. Some common applications of animation in computer graphics include:
2. Entertainment: Animation is extensively used in movies, TV shows, and video games to bring characters and scenes to life, creating immersive and engaging experiences for viewers and players.
3. Education: Animation is used in educational videos, interactive simulations, and e-learning platforms to visualize complex concepts, making learning more interactive and understandable.
4. Simulation: Animation is employed in simulations for training purposes, such as flight simulators, driving simulators, and medical simulations, to provide realistic experiences without the risks associated with real-world scenarios.
5. Visualization: Animation is utilized in data visualization, scientific visualization, and architectural visualization to illustrate trends, patterns, and relationships in data, as well as to showcase architectural designs and concepts.

Source Code :

main.py | sin_anime.py U

sin_anime.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
def animate(i):
    """
    Function to update the plot for animation.
    Parameters:
        i (int): Frame number.
    Returns:
        None
    """
    x = np.linspace(0, 2 * np.pi, 100)
    y = np.sin(x + i / 10)
    line.set_data(x, y)
    return line,
# Create a figure and axis
fig, ax = plt.subplots()
# Plot an empty line
line, = ax.plot([], [], lw=2)
# Set plot limits
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1, 1)
# Set title and labels
ax.set_title('Sine Wave Animation')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
# Create animation
ani = animation.FuncAnimation(fig, animate, frames=200, interval=50, blit=True)
# Show animation
plt.show()

```

Output :

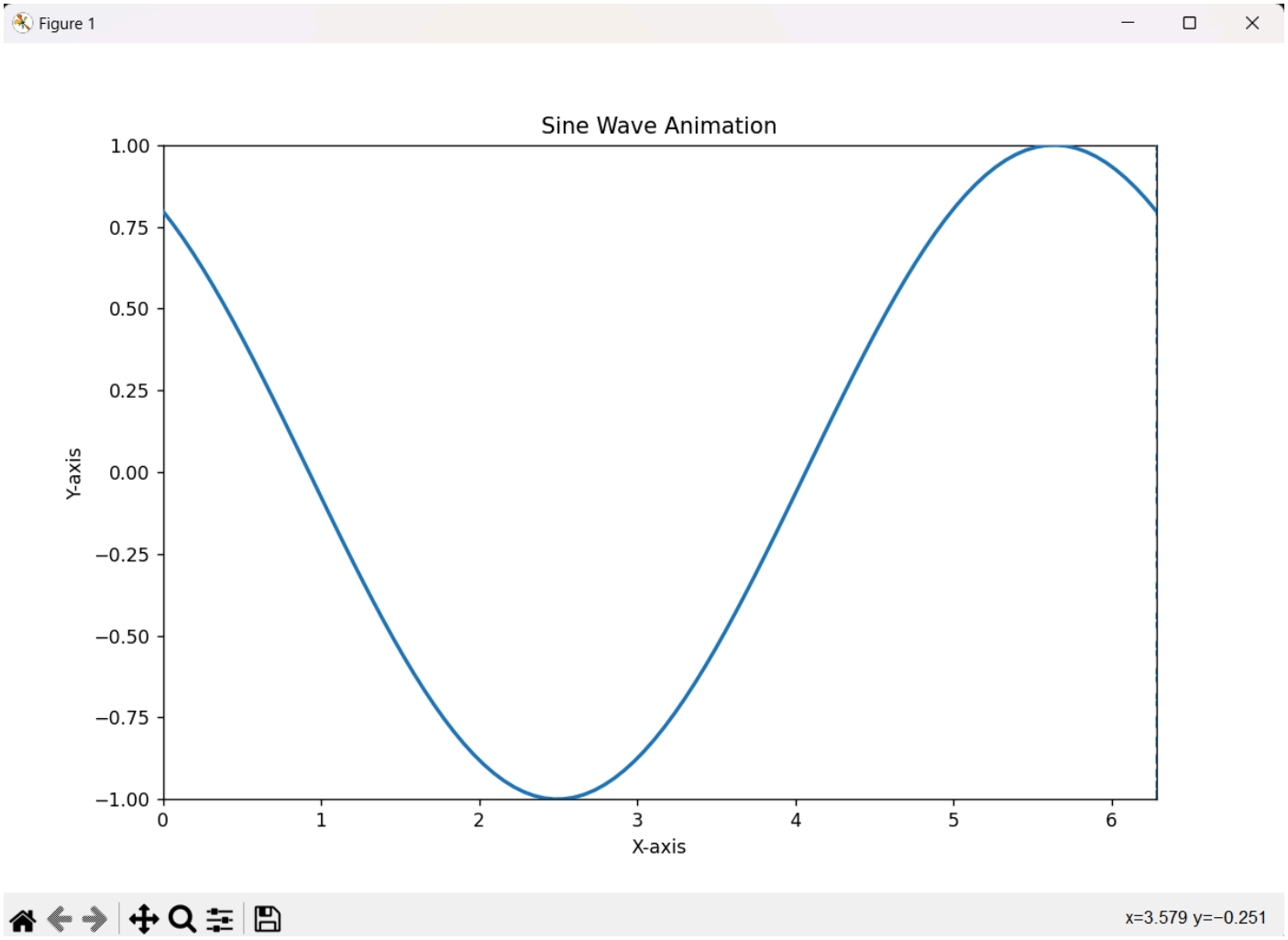


Figure 1
Sine Wave Animation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y-axis
0
1
2
3
4
5
6
X-axis
x=3.579 y=−0.251

EXPERIMENT - 09

AIM : Write a program for Speech Recognition.

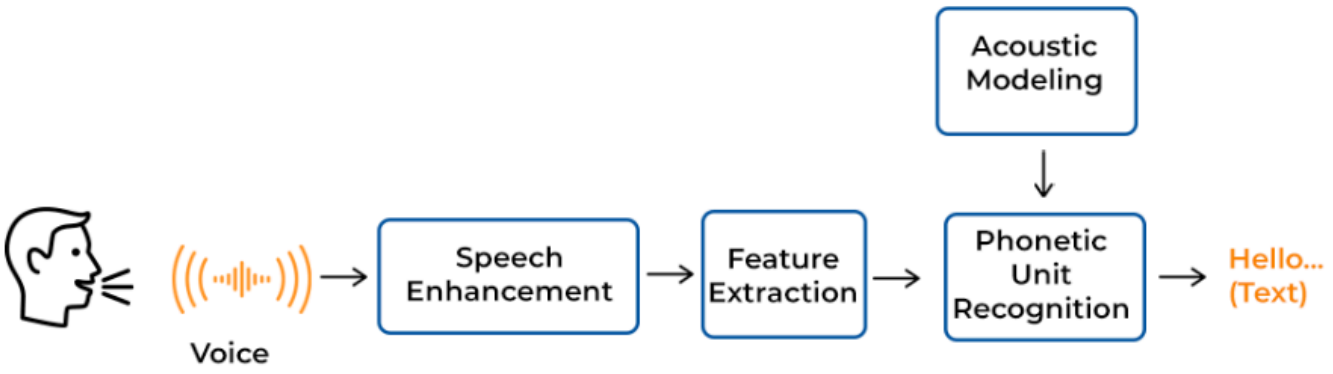
System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- pyttsx3 library
- speech_recognition library

Description:

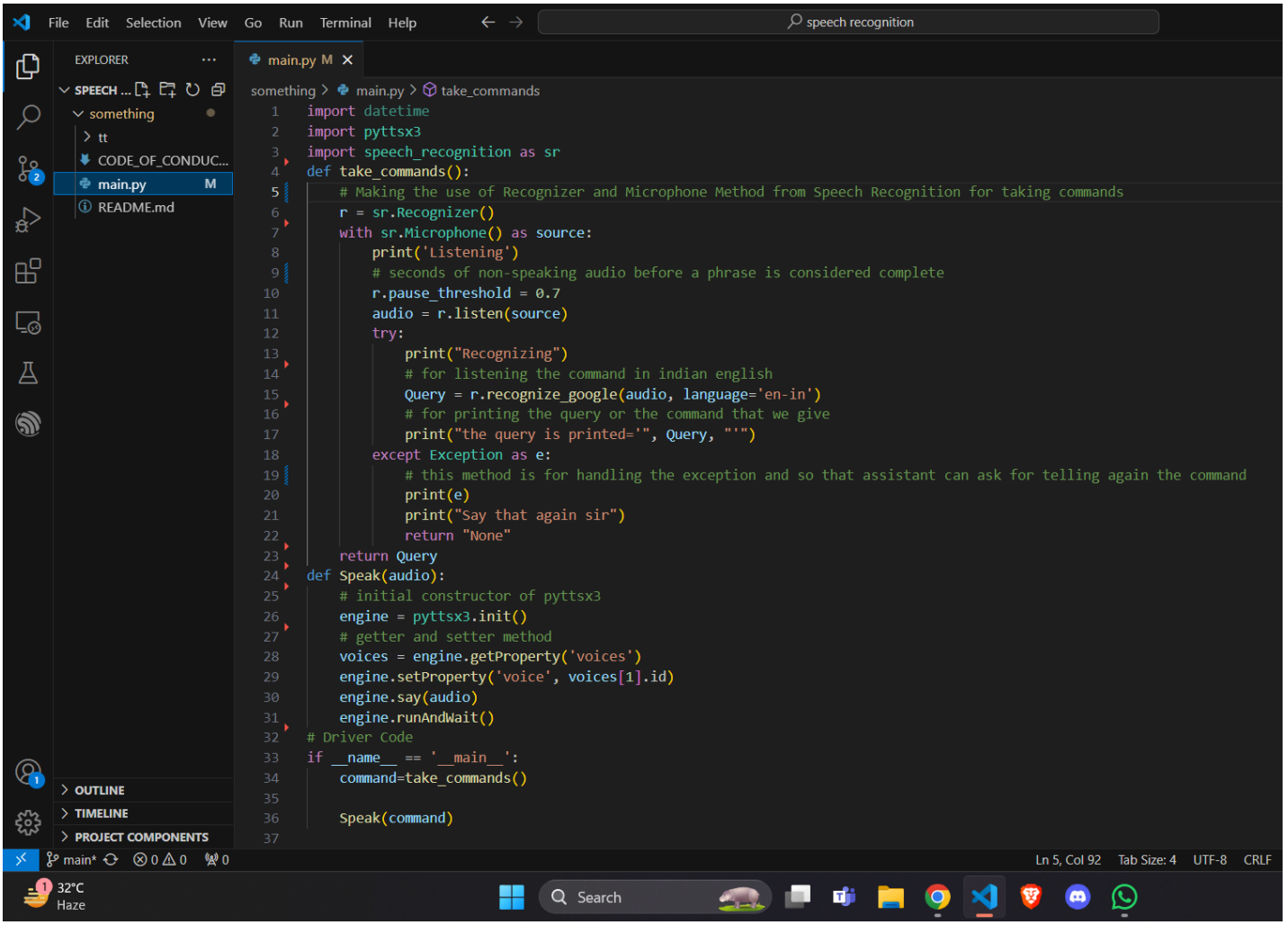
Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition or speech-to-text, is a capability that enables a program to process human speech into a written format.

Speech recognition is an interdisciplinary subfield of computer science and computational linguistics that develops methodologies and technologies that enable the recognition and translation of spoken language into text by computers



Speech Recognition Process

Source Code :



```python
import datetime
import pyttsx3
import speech_recognition as sr
def take_commands():
    # Making the use of Recognizer and Microphone Method from Speech Recognition for taking commands
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print('Listening')
        # seconds of non-speaking audio before a phrase is considered complete
        r.pause_threshold = 0.7
        audio = r.listen(source)
        try:
            print("Recognizing")
            # for listening the command in indian english
            Query = r.recognize_google(audio, language='en-in')
            # for printing the query or the command that we give
            print("the query is printed='", Query, "'")
        except Exception as e:
            # this method is for handling the exception and so that assistant can ask for telling again the command
            print(e)
            print("Say that again sir")
            return "None"
    return Query
def Speak(audio):
    # initial constructor of pyttsx3
    engine = pyttsx3.init()
    # getter and setter method
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id)
    engine.say(audio)
    engine.runAndWait()
# Driver Code
if __name__ == '__main__':
    command=take_commands()

    Speak(command)
```

Output :

```
PS C:User\CIOT\Desktop\speech_recognition>
Listening...
Recognizing...
the query is printed = ' hey this is Computer Graphics experiment 9'

~exit.
```

EXPERIMENT - 10

AIM : Write a program for Video analysis.

System Requirements:

- Operating System: Any operating system compatible with Python and Matplotlib library. (e.g., Windows, Linux, macOS)
- Python (version 3.9 recommended)
- Matplotlib library (Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python)

Description:

Video analysis is the process of extracting useful information from video data by analyzing its content. In computer graphics and multimedia, video analysis plays a crucial role in various applications, including:

1. Object Tracking: Video analysis is used to track the movement of objects within a video sequence. This is essential in surveillance systems, sports analysis, and augmented reality applications.
2. Motion Detection: Video analysis techniques are employed to detect and analyze motion patterns within video frames. This is valuable in security systems, traffic monitoring, and gesture recognition.
3. Activity Recognition: Video analysis algorithms can recognize and classify different activities or behaviors depicted in a video. This is useful in healthcare monitoring, behavior analysis, and video content understanding.
4. Scene Understanding: Video analysis helps in understanding the content and context of video scenes by identifying objects, scenes, and relationships between them. This is important in video summarization, video search, and content-based retrieval.

Source Code :

main.py M | video_analysis U

something > video_analysis > ...

```python
import cv2
def analyze_video(video_path):
    """ Function to analyze a video and perform basic video analysis.
    Parameters:
        video_path (str): Path to the input video file.
    Returns:
        None
    """
    cap = cv2.VideoCapture(video_path)                # Open video capture object
    if not cap.isOpened():                            # Check if video is successfully opened
        print("Error: Unable to open video.")
        return
    frame_count = 0                                   # Initialize frame count
    while True:                                       # Loop through the video frames
        ret, frame = cap.read()                       # Read the next frame
        if not ret:                                   # Break the loop if video ends
            break
        frame_count += 1                              # Perform basic analysis on the frame
    cap.release()                                     # Release video capture object
    print("Video Analysis Complete.")                 # Display analysis results
    print("Total Frames:", frame_count)
def main():
    video_path = 'sample_video.mp4'                   # Path to the input video file
    analyze_video(video_path)                         # Analyze the video
if __name__ == "__main__":
    main()
```

Output :

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
Analyzing "sample_video.mp4"......
Video Analysis is Complete.
Total Frames : 1900
Duration : 987600 seconds
size : 192467 Kb
```