ADVANCED COMPUTER VISION

A MINI PROJECT REPORT

Submitted by

S MOHAN KUMAR(Reg. No.210420104096)

K KISHORE (Reg. No. 210420104081)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Chennai Institute of Technology(Autonomous)

Kundrathur, Chennai - 600069.

ANNA UNIVERSITY: CHENNAI – 600025

BONAFIDE CERTIFICATE

Certified that this project report "**ADVANCED COMPUTER VISION**" is the bonafide work of "**S Mohan Kumar (210420104096),K Kishore (210420104081)** "who carried out the project work undermy supervision.

SIGNATURE	**SIGNATURE**
Dr. S. Pavithra,M.E., Ph.D.,	Dr.REGIN BOSE,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Associate Professor	AssociateProfessor
Department of CSE	Department of CSE
Chennai Instituteof Technology,	Chennai Institute of Technology,
Pudhuper, Kundrathur	Pudhuper, Kundrathur
Chennai -600069	Chennai -600069

Submitted for the viva-voiceheld on________________________at Chennai Institute of Technology, Kundrathur.

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our gratitude to our Chairman **Shri.P.SRIRAM** and all trust members of Chennai institute of technology for providing the facility and opportunity to do this project as a part of our undergraduate course.

We are grateful to our Principal **Dr.A.RAMESH M.E., Ph.D.,** for providing us the facility and encouragement during the course of our work.

We sincerely thank our Head of the Department **Dr.S.PAVITHRA M.E., Ph.D.,** Department of Computer Science and Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We sincerely thank our Project Guide **Dr.Regin Bose M.E., PhD.,** Associate Professor, Department of Computer Science and Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We would like to extend our thanks to our Faculty coordinators of the Department of Computer Science and Engineering, for their valuable suggestions throughout this project.

We wish to extend our sincere thanks to all Faculty members of the Department of Computer Science and Engineering for their valuable suggestions and their kind cooperation for the successful completion of our project.

We wish to acknowledge the help received from the Lab Instructors of the Department of Computer Science and Engineering and others for providing valuable suggestions and for the successful completion of the project.

Table of Contents

ABBREVIATIONS

S.NO	ABBREVIATIONS	EXPANSION
1	OpenCV	Open Source Computer Vision Library
2	CNN	Convolutional Neural Network
3	ROI	Region of Interest
4	SVM	Support Vector Machine
5	GPU	Graphics Processing Unit
6	RGB	Red Green Blue
7	API	Application Programming Interface
8	LBP	Local Binary Pattern
9	FCN	Fully Convolutional Network
10	RAM	Random Access Memory
11	FPS	Machine Learning
12	HPC	High-Performance Computing
13	I/O	Input/Output

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Advanced Computer Vision is a highly relevant and rapidly growing field that has the potential to revolutionize numerous industries, including healthcare, automotive, and security. This project offers a unique opportunity to develop and implement cutting-edge computer vision techniques, including deep learning algorithms and neural networks, to solve real-world problems. By participating in this project, you will have the chance to work with experts in the field and gain valuable experience in data collection, pre-processing, model training, and evaluation. Furthermore, your contributions will have a significant impact on advancing the state-of-the-art in computer vision and making a meaningful difference in the world.

1.2 PROBLEM STATEMENT

Although computer vision has made significant progress in recent years, several challenges still exist that limit its effectiveness in solving real-world problems. The accuracy of existing computer vision algorithms can be hampered by issues such as occlusion, scale variation, and lighting changes, which can result in missed detections and false positives. Moreover, the processing times required by these algorithms can be prohibitively slow, which can limit their practicality in many applications. Additionally, there is a need to develop new computer vision techniques that can be used for tasks such as 3D reconstruction, depth estimation, and video understanding. These limitations pose significant challenges to the field of computer vision, and there is a pressing need to develop advanced techniques that can overcome these challenges and improve the accuracy and efficiency of computer vision. The goal of this project is to address these challenges by developing and implementing advanced computer vision techniques that can be used to solve real-world problems.

1.3 PROJECT OBJECTIVE

The objective of this mini-project on advanced computer vision is to gain a practical understanding of the key modules in computer vision, including face recognition, volume control, and hand tracking. Through this project, we aim to explore the underlying algorithms and techniques used in these modules and gain hands-on experience in implementing them. Additionally, we aim to understand the potential applications of these modules in various fields such as security, audio processing, virtual reality, and gaming. By the end of the project, we aim to have a comprehensive understanding of the fundamental concepts of advanced computer vision and the practical skills required to apply them in real-world scenarios.

1.4 PROJECT REPORT ORGANIZATION

The "Advance Computer Vision" has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the particular needof the company to carry out operations in a smooth and effectivemanner.

Advanced Computer Vision is a rapidly growing field that deals with the development of intelligent systems that can interpret and understand visual data from the real world. This mini project will explore three important modules of Advanced Computer Vision: face recognition, volume control, and hand tracking. The face recognition module aims to identify individuals based on their facial features, while the volume control module adjusts the sound based on the user's position. The hand tracking module helps to track and analyze hand movements and gestures. This project will provide a practical understanding of how these modules work and their applications in various fields.

Advanced Computer Vision refers to the study of algorithms and techniques that enable machines to interpret and understand visual information from the real world. With the rapid advancement of technology, computer vision has become an essential component in a wide range of applications such as robotics, autonomous vehicles, surveillance, and medical imaging.

Another interesting area of advanced computer vision is volume control. With this technology, machines can adjust the volume of a device based on the user's position in a room. For instance, if the user is standing closer to the device, the volume is automatically increased, and if they move away, the volume is reduced. This technology is being used in smart speakers, televisions .

Apart from face recognition, volume control, and hand tracking, advanced computer vision has numerous other applications such as object recognition, image segmentation, and action recognition. Object recognition allows machines to identify and classify objects in images and videos. Image segmentation enables the separation of objects in an image and background removal. Action recognition technology enables machines to identify and classify human actions, which is used in applications such as surveillance and sports analysis.

In conclusion, the applications of advanced computer vision are vast and growing. From face recognition to action recognition, this technology is revolutionizing the way machines interpret and understand visual information. As the technology continues to improve, we can expect to see more applications and advancements in this exciting field.

CHAPTER 2: LITERATURE SURVEY

2.1 EXISTING WORK

Advanced computer vision has been an active area of research for many years, and significant progress has been made in developing sophisticated algorithms for a range of applications. Some of the existing work in this field includes:

- ✓ Object Detection: One of the most common applications of computer vision is object detection, which involves identifying and localizing objects within an image or video stream. Several techniques have been developed for object detection, including the popular YOLO (You Only Look Once) and Faster R-CNN (Region-based Convolutional Neural Network) algorithms.

- ✓ Image Segmentation: Another important application of computer vision is image segmentation, which involves dividing an image into segments or regions based on similarities in color, texture, or other visual properties. Deep learning approaches such as U-Net and SegNet have been developed to improve the accuracy of image segmentation.

- ✓ Face Recognition: Face recognition is a specialized application of computer vision that involves identifying and verifying individuals based on facial features. Techniques such as deep face recognition and face alignment have been developed to improve the accuracy and speed of face recognition.

- ✓ 3D Reconstruction: 3D reconstruction involves creating a 3D model of an object or scene from a set of 2D images. Techniques such as Structure from Motion (SfM) and Multi-View Stereo (MVS) have been developed to improve the accuracy and efficiency of 3D reconstruction.

- ✓ Video Analysis: Computer vision techniques have also been developed for analyzing and understanding video content. Techniques such as video segmentation, action recognition, and event detection have been developed to improve the accuracy and efficiency of video analysis.

- ✓ Overall, advanced computer vision techniques are becoming increasingly sophisticated and are being used in a wide range of applications, including healthcare, transportation, security, and entertainment. Ongoing research in this field is focused on developing even more advanced techniques to solve complex real-world problems.

Overall, advanced computer vision techniques are becoming increasingly sophisticated and are being used in a wide range of applications, including healthcare, transportation, security, and entertainment. Ongoing research in this field is focused on developing even more advanced techniques to solve complex real-world problems.

2.2 LIMITATIONS OF EXISTING WORK

While there have been significant advances in advanced computer vision techniques, there are still several limitations that need to be addressed. Some of the limitations of existing work in this field include:

- ✓ Limited generalization: Many existing computer vision techniques are trained on specific datasets and may not generalize well to new datasets or real-world scenarios. This limits the applicability of these techniques in practical applications.

- ✓ Lack of interpretability: Deep learning-based techniques used in advanced computer vision are often complex and lack interpretability, making it difficult to understand how they arrive at their decisions. This can make it challenging to diagnose and fix errors or biases in the system.

- ✓ High computational requirements: Many advanced computer vision techniques require significant computational resources, such as high-end GPUs or specialized hardware. This can be a barrier to adoption for organizations or individuals who do not have access to these resources.

- ✓ Limited robustness: Advanced computer vision techniques can be sensitive to changes in lighting conditions, camera angles, and other environmental factors. This limits their robustness and can result in errors or false positives.

- ✓ Limited privacy and security: Computer vision techniques that involve analyzing sensitive data, such as facial recognition, can raise privacy and security concerns. The potential for misuse or abuse of these techniques underscores the importance of developing robust ethical guidelines and governance frameworks for their use.

Overall, addressing these limitations will require ongoing research and development to improve the performance, interpretability, and generalizability of advanced computer vision techniques. It will also require attention to ethical and social considerations to ensure that these techniques are deployed in a responsible and accountable manner.

Figure 2.1: Face Recognition with Python & OpenCV

CHAPTER 3 : SYSTEM REQUIREMENTS

3.1 SOFTWARE REQUIREMENTS

The Software Requirements Specification is produced at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by establishing a complete information description, a detailed functional and behavioral description, an indication of performance requirements and design constraints, appropriate validation criteria, and other data pertinent to requirements.

The proposed system has the following requirements:

- ✓ Programming languages: Python, C++, and MATLAB are commonly used programming languages for computer vision projects. Python is particularly popular because of its extensive libraries for computer vision and machine learning, such as OpenCV, TensorFlow, and PyTorch.
- ✓ Integrated Development Environment (IDE): An IDE is a software application that provides tools for software development, such as a code editor, debugger, and compiler. Some popular IDEs for computer vision projects include PyCharm, Visual Studio, and Jupyter Notebook.
- ✓ Computer vision libraries: There are several libraries that provide pre-built computer vision algorithms, such as OpenCV, DLIB, and scikit-image. These libraries can save time and effort in implementing advanced computer vision techniques.

- ✓ Machine learning frameworks: Many advanced computer vision techniques involve machine learning, such as deep neural networks. Popular machine learning frameworks include TensorFlow, PyTorch, and Keras.
- ✓ Data annotation and labeling tools: Many computer vision projects require labeled datasets for training and evaluation. Tools such as LabelImg, RectLabel, and VGG Image Annotator (VIA) can be used to annotate and label datasets.
- ✓ Version control systems: Version control systems, such as Git and SVN, can help manage code changes and collaborate with team members on a project.
- ✓ System needs to access the camera and should capture face expressions and hand gestures.
- ✓ System should install the necessary packages.
- ✓ System should maintain better network connectivity.

3.1 HARDWARE REQUIREMENTS

Implementing a project on advanced computer vision may require hardware with specific capabilities and specifications to handle the computational requirements. The hardware requirements for such projects may vary depending on the complexity of the algorithms being used, the size of the dataset, and the application being developed. Some of the common hardware requirements for implementing advanced computer vision projects include:

- Central Processing Unit (CPU): A CPU is the primary component of a computer that performs processing tasks. A high-performance CPU with multiple cores is recommended to handle the computational demands of advanced computer vision algorithms

- Graphics Processing Unit (GPU): A GPU is a specialized processor designed for handling parallel processing tasks, such as image and video processing. GPUs can significantly speed up the training and inference of advanced computer vision models. A high-end GPU with a large number of CUDA cores, such as NVIDIA GeForce or Tesla GPUs, is recommended.

- Random Access Memory (RAM): RAM is used to temporarily store data and instructions that the CPU can access quickly. A higher amount of RAM is recommended to handle large datasets and perform computationally intensive tasks.

- Solid State Drive (SSD): An SSD is a type of storage device that provides faster read and write speeds compared to traditional hard disk drives (HDDs). Using an SSD can significantly improve the performance of data read/write operations during training and testing.

- Camera: Depending on the application, a high-quality camera may be required to capture images or videos for processing.

- Display: A high-resolution monitor is recommended to visualize the output of computer vision models and analyze the results.

- Overall, the hardware requirements for implementing a project on advanced computer vision can be significant, especially for large-scale applications. It is important to carefully evaluate the hardware requirements and choose the appropriate hardware to ensure optimal performance and efficiency during the project implementation.

CHAPTER 4 : DESIGN

4.1 CLASS DIAGRAM

A class diagram is a UML (Unified Modeling Language) diagram that represents the static structure of a system in terms of classes, their attributes, and their relationships. It is a useful tool for software development and design.
In the case of a hand and face recognition module, the class diagram may consist of the following classes:

1. FaceRecognition: This class represents the main module that performs face recognition tasks. It may have methods such as detectFace, recognizeFace, and trainFace.

2. HandRecognition: This class represents the main module that performs hand recognition tasks. It may have methods such as detectHand, recognizeHand, and trainHand.

3. Face: This class represents the detected face object, which may have attributes such as faceID, location, and size.

4. Hand: This class represents the detected hand object, which may have attributes such as handID, location, and size.

5. Volume: This class represents with the use of detected hand object, if the hand elongates means it will increase the volume of the system, or else decrease the volume of the system.

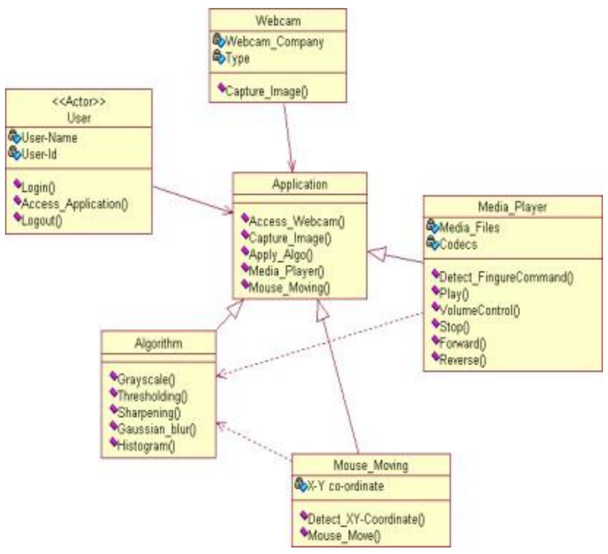


Figure 4.1 : Class Diagram

4.2 USE CASE DIAGRAM

1. Detect Face: This use case allows the user to detect a face in an image or video stream. The system uses image processing techniques and machine learning models to identify the face and return its location.

2. Recognize Face: This use case allows the user to recognize a face in an image or video stream. The system uses machine learning models trained on a dataset of known faces to identify the face and return its identity.

3. Detect Hand: This use case allows the user to detect a hand in an image or video stream. The system uses image processing techniques and machine learning models to identify the hand and return its location.

4. Recognize Hand: This use case allows the user to recognize a hand gesture in an image or video stream. The system uses machine learning models trained on a dataset of known hand gestures to identify the gesture and return its meaning.

5. Increase the volume: This use case allows the user to increase the volume of the system based on the elongation of the hand.

6. Decrease the volume: This use case allows the user to decrease the volume of the system based on the elongation of the hand.

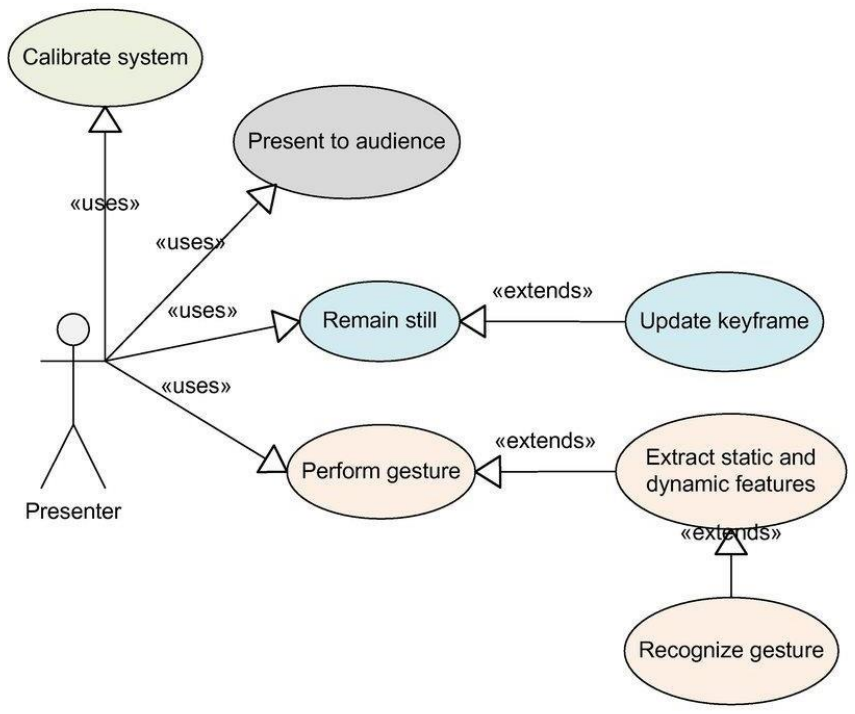


Figure 4.2: Usecase Diagram

7. Detect Hand: This use case allows the user to detect a hand in an image or video stream. The system uses image processing techniques and machine learning models to identify the hand and return its location.

8. Recognize Hand: This use case allows the user to recognize a hand gesture in an image or video stream. The system uses machine learning models trained on a dataset of known hand gestures to identify the gesture and return its meaning.

9. Increase the volume: This use case allows the user to increase the volume of the system based on the elongation of the hand.

10. Decrease the volume: This use case allows the user to decrease the volume of the system based on the elongation of the hand.

4.3 SEQUENCE DIAGRAM

A sequence diagram is a type of UML (Unified Modeling Language) diagram that represents the interactions between objects or components in a system in a chronological order.

It shows the flow of messages between different objects or components of the system over time, and it is commonly used to visualize the dynamic behavior of a system.

Sequence diagrams can be used to model complex scenarios and help to identify potential errors or inefficiencies in a system's design. They are particularly useful in situations where multiple objects or components interact with each other to achieve a particular goal.

In a sequence diagram, objects or components are represented as vertical lines, also known as lifelines, and messages are represented as horizontal arrows between the lifelines. The sequence diagram also includes activation bars, which represent the time duration of an operation or message.

Sequence diagrams can be used during the analysis and design phases of software development to help communicate requirements and design decisions to stakeholders and developers. They can also be used during testing to ensure that the system behaves correctly under different scenarios.

In this sequence diagram, the User sends a request to the system to detect a face in a camera. The system receives the request and sends it to the face or hand recognition module, which performs preprocessing and feature extraction then find the gestures on the image. The preprocessed image sends and execute the required module to detect the face or hand and return its location. Finally, the system sends the location of the detected face back to the User.

Then the user now able to identify their faces based on the gestures and the user now able to identify their hands based on the gestures and also able to increase and decrease the volume of the system based on the elongation of the hands.

By using sequence diagrams, developers and stakeholders can better understand the functionality and behavior of a system, identify potential issues or inefficiencies, and communicate the system's behavior and functionality to others.

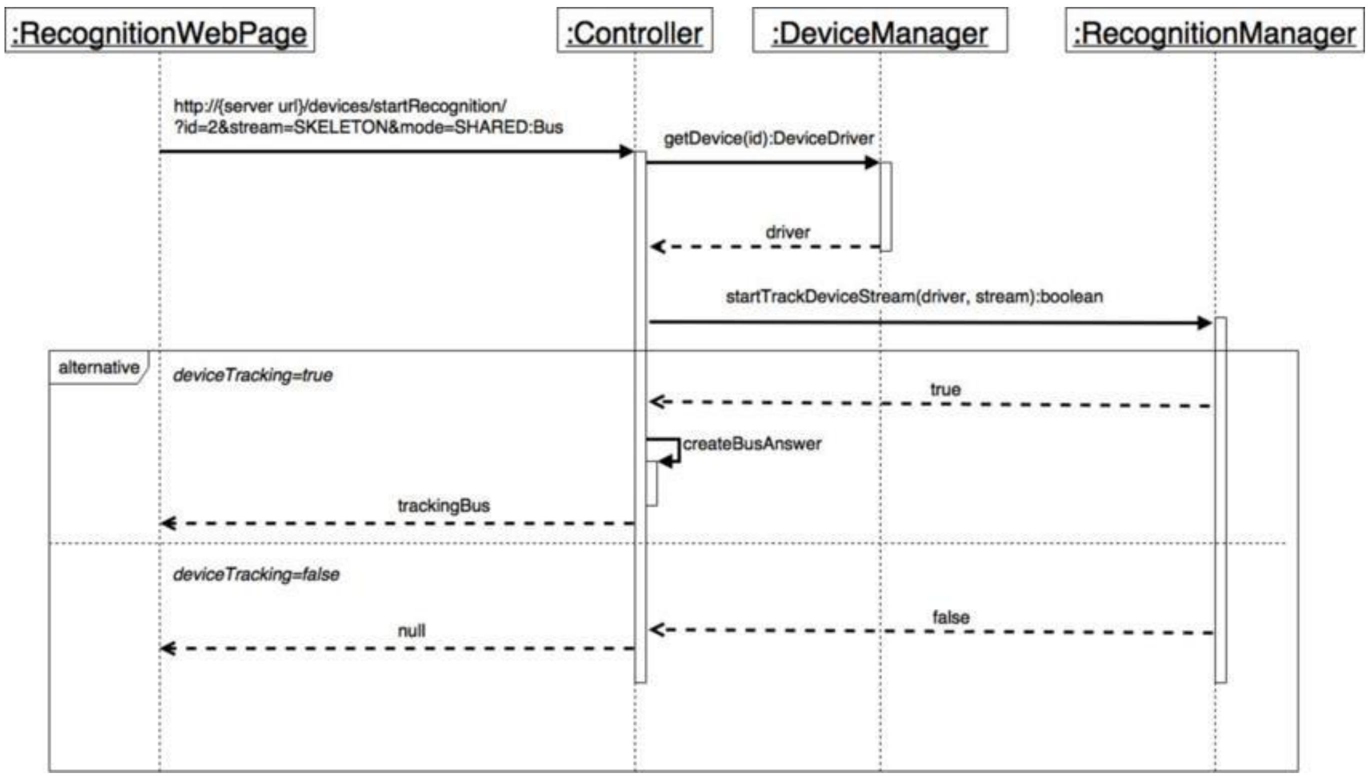
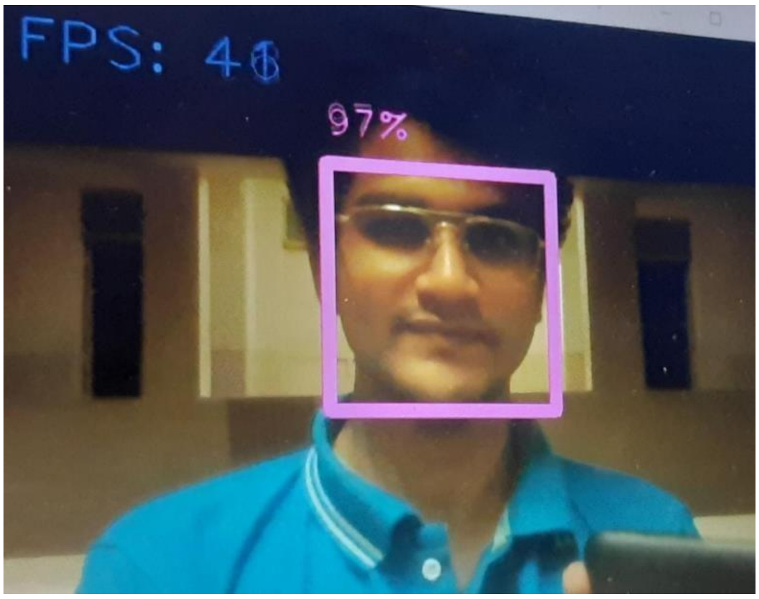


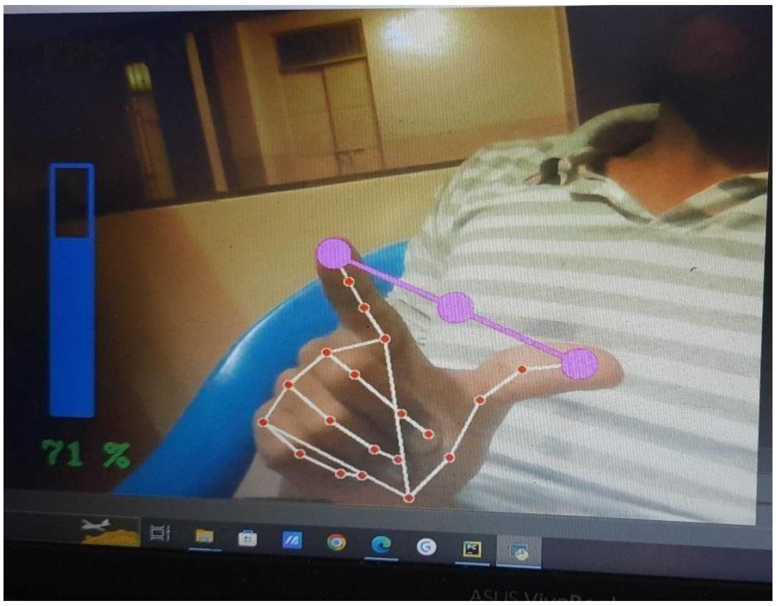
Figure 4.3 : Srquence Diagram

4.4 USER INTERFACE DESIGN

1.FACE RECOGNITION



2.VOLUME CONTROL MODULE



CHAPTER 5. IMPLEMENTATION AND TESTING

5.1 SOURCE CODE

FaceDetection Module.py

```
import cv2
import mediapipe as mp
import time


class FaceDetector():
   def_init_(self, minDetectionCon= 0.5):

      self.minDetection = minDetectionCon

      self.mpFaceDetection = mp.solutions.face_detection
      self.mpDraw = mp.solutions.drawing_utils
      self.faceDetection = self.mpFaceDetection.FaceDetection(0.75)

   def findFaces(self,img,draw= True):

       imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
       self.results = self.faceDetection.process(imgRGB)
       #print(self.results)
       bboxs = []
       if self.results.detections:
           for id, detection in enumerate(self.results.detections):
               #mpDraw.draw_detection(img, detection)
               # print(id, detection)
               # print(detection.score)
               # print(detection.location_data.relative_bounding_box)
               bboxC = detection.location_data.relative_bounding_box
               ih, iw, ic = img.shape
               bbox = int(bboxC.xmin * iw), int(bboxC.ymin * ih), \
                      int(bboxC.width * iw), int(bboxC.height * ih)
               bboxs.append([id, bbox, detection.score])
               if draw:
                   img = self.fancyDraw(img,bbox)
               cv2.putText(img, f'{int(detection.score[0]*100)}%',
                           (bbox[0], bbox[1]-20), cv2.FONT_HERSHEY_PLAIN,
                           2, (255, 0, 255), 2)
           return img, bboxs

   def fancyDraw(self, img, bbox, l =30, t=5, rt=10):
         x, y, w, h = bbox
         x1, y1 = x+w, y+h
         cv2.rectangle(img, bbox, (255, 0, 255), rt)

         cv2.line(img, (x,y), (x+l, y), (255, 0, 255), t)
```

```
            cv2.line(img, (x, y), (x, y+l), (255, 0, 255), t)

            cv2.line(img, (x1, y), (x1 - l, y), (255, 0, 255), t)
            cv2.line(img, (x1, y), (x1, y + l), (255, 0, 255), t)

            cv2.line(img, (x, y1), (x + l, y1), (255, 0, 255), t)
            cv2.line(img, (x, y1), (x, y1 - l), (255, 0, 255), t)

            cv2.line(img, (x1, y1), (x1 - l, y1), (255, 0, 255), t)
            cv2.line(img, (x1, y1), (x1, y1 - l), (255, 0, 255), t)

            return img

def main():
        cap = cv2.VideoCapture(0)
        pTime = 0
        detector = FaceDetector()
        while True:
            success, img = cap.read()
            img, bboxs = detector.findFaces(img)
            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime
            cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                        3, (255, 0, 0), 2)
            cv2.imshow("Image", img)
            cv2.waitKey(1)
if__name__== "_main_":
      main()
```

HandTracking Module.py

```
import cv2
import mediapipe as mp
import time


class handDetector():
    def_init_(self, mode=False, maxHands=2, detectionCon=0.5,modelComplexity=1,trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.modelComp=modelComplexity
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,self.modelComp, self.detectionCon,
self.trackCon)
        self.myDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
#         # print(results.multi_hand_landmarks)
        if self.results.multi_hand_landmarks:
            for handLas in self.results.multi_hand_landmarks:
                if draw:
                    self.myDraw.draw_landmarks(img, handLas, self.mpHands.HAND_CONNECTIONS)
        return img
#
    def findPosition(self,img,handNo=0,draw=True):

       lmsList=[]

       if self.results.multi_hand_landmarks:
          myHand=self.results.multi_hand_landmarks[handNo]
          for id, lm in enumerate(myHand.landmark):
            # print(id,lm)
            h,w,c=img.shape
            cx,cy=int(lm.x*w),int(lm.y*h)
            # print(id,cx,cy)
            lmsList.append([id,cx,cy])
            if draw:
               cv2.circle(img,(cx,cy),7,(155,8,255),cv2.FILLED)
       return lmsList
def main():
    p_time = 0
    c_time = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmsList=detector.findPosition(img,draw=False)
        if len(lmsList)!=0:
            print(lmsList[4])
        c_time = time.time()
        fps = 1 / (c_time - p_time)
        p_time = c_time
        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)


if__name__== "_main_":
    main()
```

VolumeControl Module.py

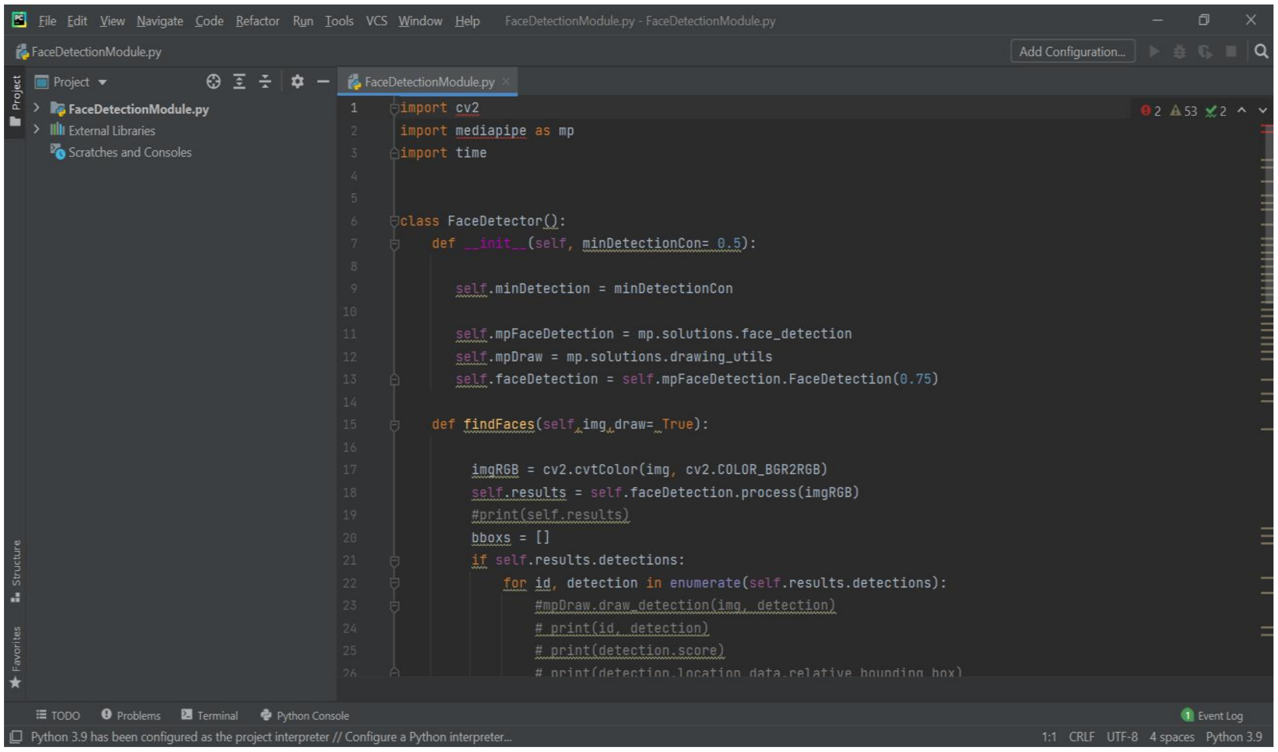
```
import cv2
import time
import numpy as np
import HandTrackingModule as ht
import math
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

wCam,hCam=640, 480

cap=cv2.VideoCapture(0)
cap.set(3,wCam)
cap.set(4,hCam)
p_time=0

detector=ht.handDetector(detectionCon=0.7)

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
# volume.GetMute()
# volume.GetMasterVolumeLevel()
volRange=volume.GetVolumeRange()
minVol=volRange[0]
maxVol=volRange[1]
vol=0
volBar=400
volPerc=0
while True:
    success,img=cap.read()
    img=detector.findHands(img)
    lmList=detector.findPosition(img, draw=False)
    if len(lmList)!=0:
        # print(lmList[4],lmList[8])
        x1,y1=lmList[4][1],lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx,cy=(x1+x2)//2,(y1+y2)//2
        cv2.circle(img,(x1,y1), 15, (255,0,255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img,(x1,y1),(x2,y2),(255,0,255),3)
        cv2.circle(img, (cx, cy),15,(255, 0, 255), cv2.FILLED)

        length=math.hypot(x2-x1,y2-y1)

        # print(length)
        #Hand range 50-300 and Volume range -65 to -0
        vol=np.interp(length,[50,300],[minVol,maxVol])
        volBar=np.interp(length, [50,300], [400,150])
        volPerc = np.interp(length, [50, 300], [0, 100])
        # print(int(length),vol)
        volume.SetMasterVolumeLevel(vol, None)
        if length<45:
            cv2.circle(img, (cx, cy), 15, (0, 255, 0), cv2.FILLED)
    cv2.rectangle(img, (50,150), (85,400),(255,100,0),3)
    cv2.rectangle(img, (50, int(volBar)), (85, 400), (255, 100, 0), cv2.FILLED)
    cv2.putText(img, f'{int(volPerc)} %', (40, 450),cv2.FONT_HERSHEY_COMPLEX, 1, (100,200,0),
3)
    c_time=time.time()
    fps=1/(c_time-p_time)
    p_time=c_time
    cv2.putText(img,f'FPS: {int(fps)}',(40,50), cv2.FONT_HERSHEY_COMPLEX,1,(155,0,0),3)
    cv2.imshow("Img",img)
    cv2.waitKey(1)
```

Hand Tracking Min Module.py

```
import cv2
import mediapipe as mp
import time
import HandTrackingModule as ht

p_time = 0
c_time = 0
cap = cv2.VideoCapture(0)
detector = ht.handDetector()
while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmsList=detector.findPosition(img)
        if len(lmsList)!=0:
            print(lmsList[4])
        c_time = time.time()
        fps = 1 / (c_time - p_time)
        p_time = c_time
        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)
```

5.2 SCREENSHOTS

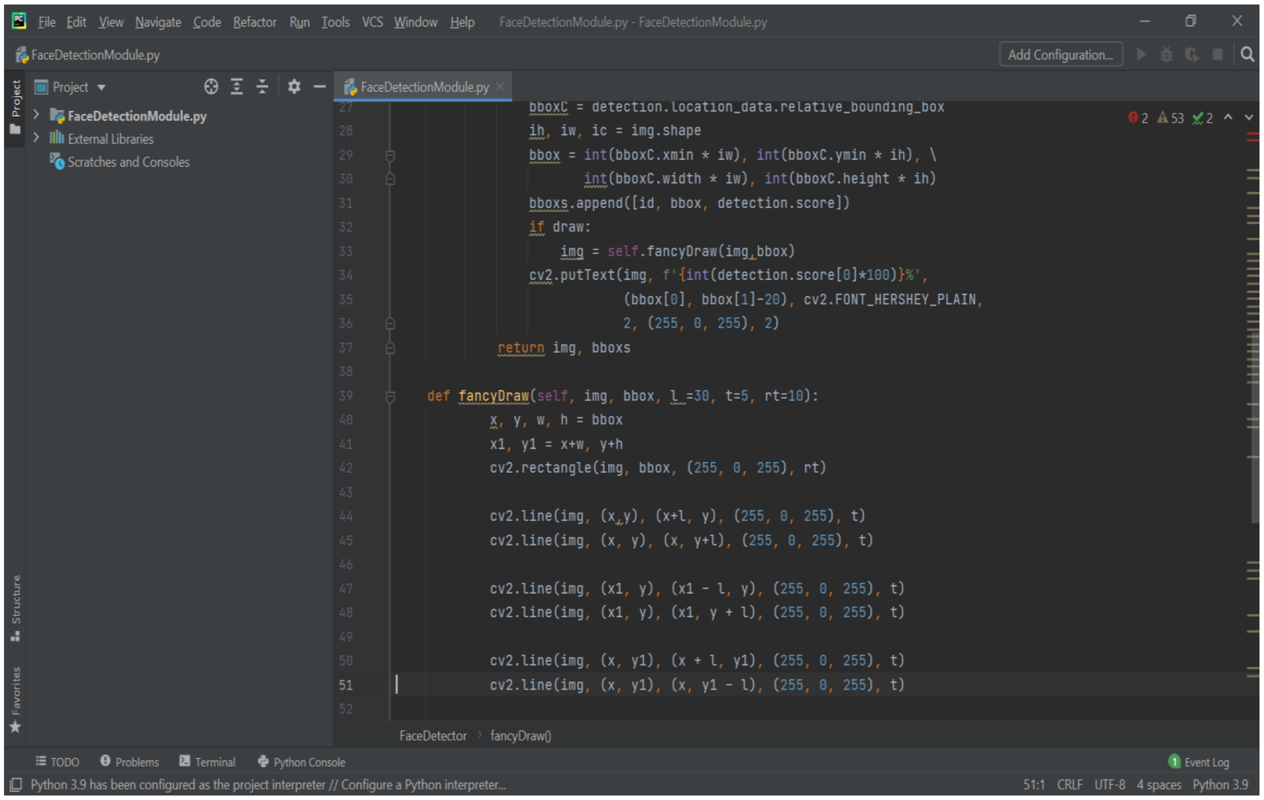
5.2.1 FACE DETECTION MODULE



```
import cv2
import mediapipe as mp
import time


class FaceDetector():
    def __init__(self, minDetectionCon= 0.5):

        self.minDetection = minDetectionCon

        self.mpFaceDetection = mp.solutions.face_detection
        self.mpDraw = mp.solutions.drawing_utils
        self.faceDetection = self.mpFaceDetection.FaceDetection(0.75)

    def findFaces(self,img,draw= True):

        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.faceDetection.process(imgRGB)
        #print(self.results)
        bboxs = []
        if self.results.detections:
            for id, detection in enumerate(self.results.detections):
                #mpDraw.draw_detection(img, detection)
                # print(id, detection)
                # print(detection.score)
                # print(detection.location_data.relative_bounding_box)
```

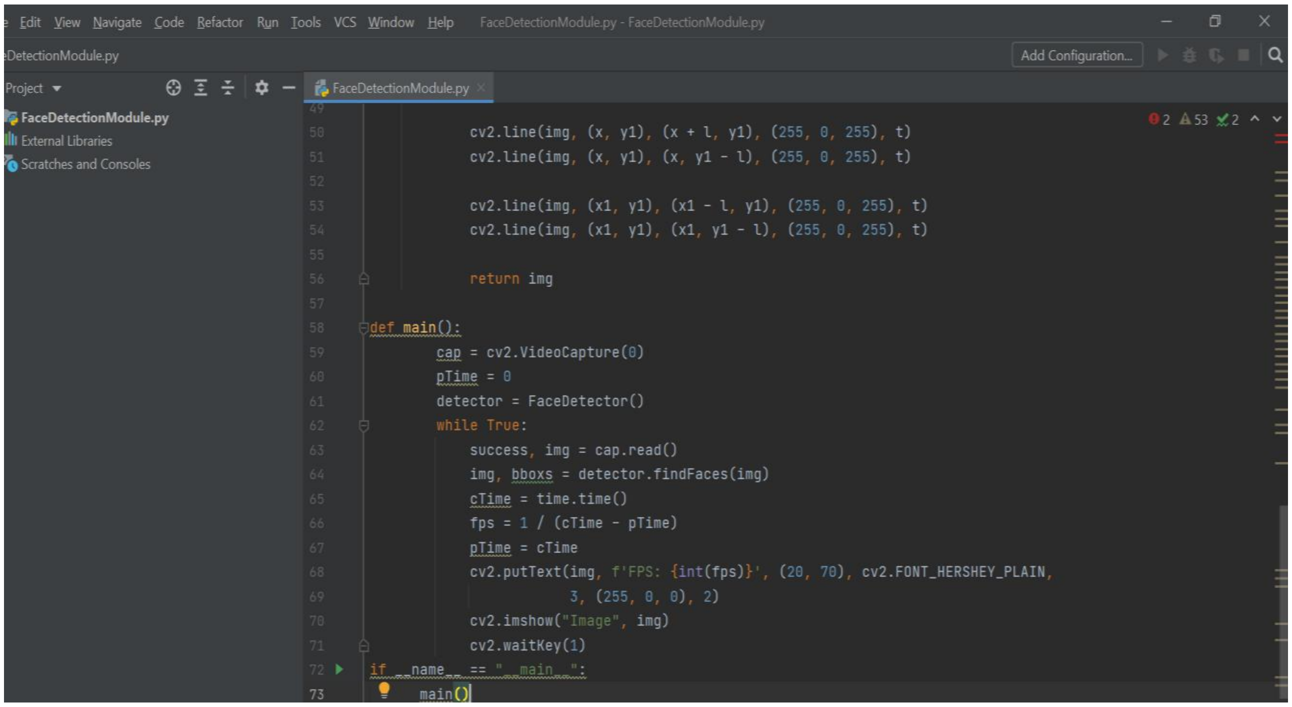
Python 3.9 has been configured as the project interpreter // Configure a Python interpreter...

Figure 5.1 : Face Detection Module.py 1



```
                bboxC = detection.location_data.relative_bounding_box
                ih, iw, ic = img.shape
                bbox = int(bboxC.xmin * iw), int(bboxC.ymin * ih), \
                       int(bboxC.width * iw), int(bboxC.height * ih)
                bboxs.append([id, bbox, detection.score])
                if draw:
                    img = self.fancyDraw(img,bbox)
                cv2.putText(img, f'{int(detection.score[0]*100)}%',
                            (bbox[0], bbox[1]-20), cv2.FONT_HERSHEY_PLAIN,
                            2, (255, 0, 255), 2)
            return img, bboxs

    def fancyDraw(self, img, bbox, l =30, t=5, rt=10):
            x, y, w, h = bbox
            x1, y1 = x+w, y+h
            cv2.rectangle(img, bbox, (255, 0, 255), rt)

            cv2.line(img, (x,y), (x+l, y), (255, 0, 255), t)
            cv2.line(img, (x, y), (x, y+l), (255, 0, 255), t)

            cv2.line(img, (x1, y), (x1 - l, y), (255, 0, 255), t)
            cv2.line(img, (x1, y), (x1, y + l), (255, 0, 255), t)

            cv2.line(img, (x, y1), (x + l, y1), (255, 0, 255), t)
            cv2.line(img, (x, y1), (x, y1 - l), (255, 0, 255), t)
```

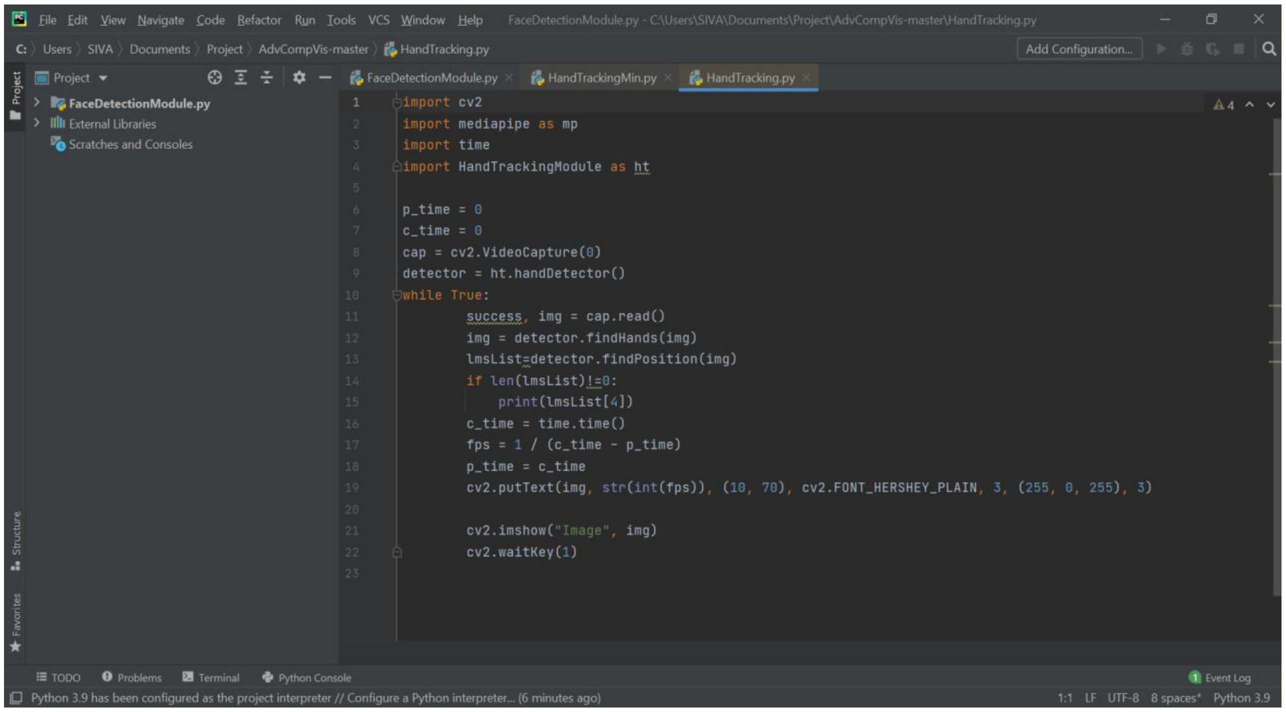
Figure 5.2: Face Detection Module.py 2



```
            cv2.line(img, (x, y1), (x + l, y1), (255, 0, 255), t)
            cv2.line(img, (x, y1), (x, y1 - l), (255, 0, 255), t)

            cv2.line(img, (x1, y1), (x1 - l, y1), (255, 0, 255), t)
            cv2.line(img, (x1, y1), (x1, y1 - l), (255, 0, 255), t)

            return img

def main():
        cap = cv2.VideoCapture(0)
        pTime = 0
        detector = FaceDetector()
        while True:
            success, img = cap.read()
            img, bboxs = detector.findFaces(img)
            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime
            cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                        3, (255, 0, 0), 2)
            cv2.imshow("Image", img)
            cv2.waitKey(1)
if __name__ == "__main__":
    main()
```

Figure 5.3: Face Detection Module.py 3

5.2.2 HAND TRACKING MIN MODULE

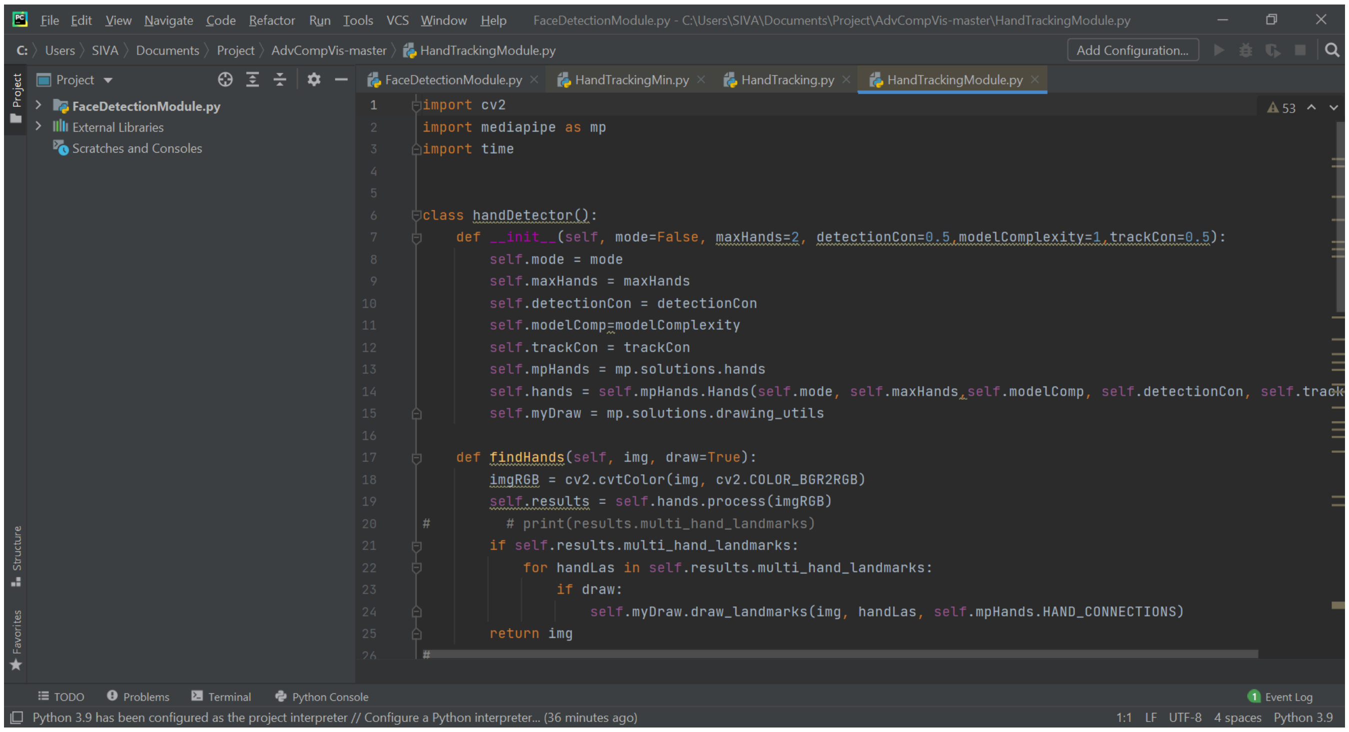


```python
import cv2
import mediapipe as mp
import time
import HandTrackingModule as ht

p_time = 0
c_time = 0
cap = cv2.VideoCapture(0)
detector = ht.handDetector()
while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmsList=detector.findPosition(img)
        if len(lmsList)!=0:
            print(lmsList[4])
        c_time = time.time()
        fps = 1 / (c_time - p_time)
        p_time = c_time
        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)
```

Python 3.9 has been configured as the project interpreter // Configure a Python interpreter... (6 minutes ago)

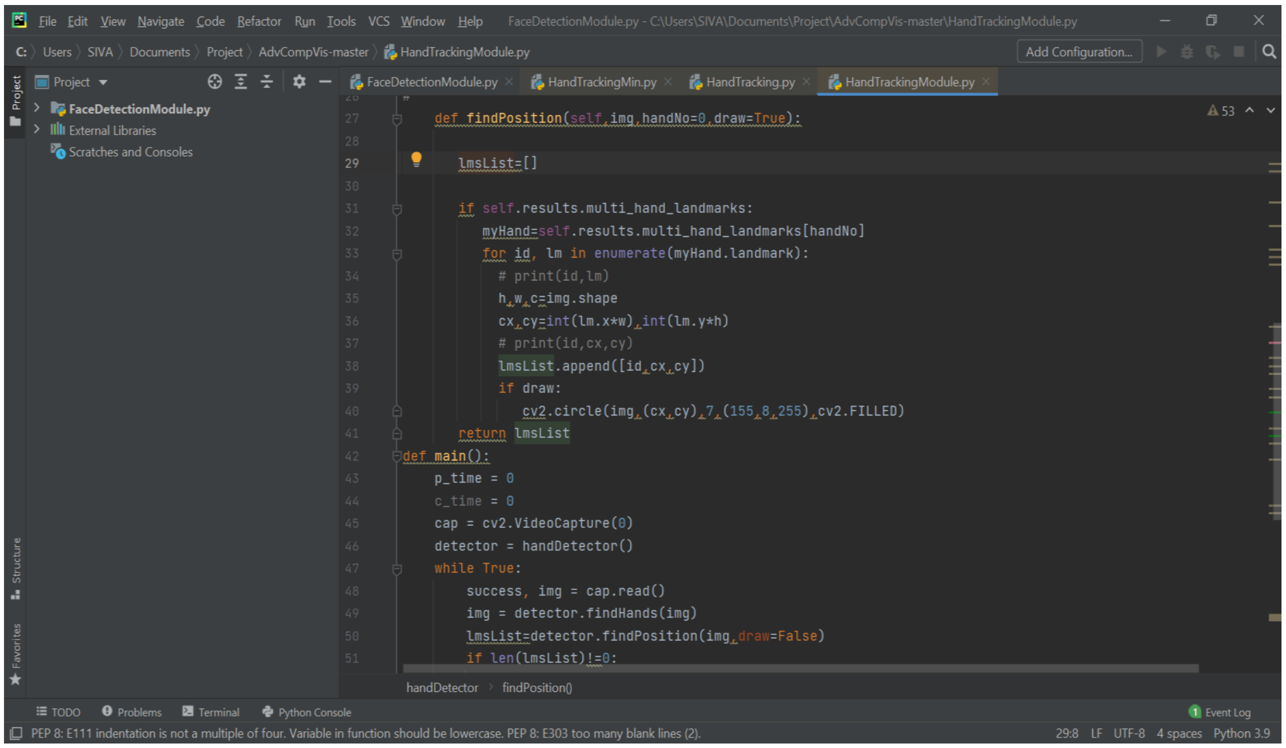
Figure 5.4 : HandTracking Min Module.py 1

5.2.3 HAND TRACKING MODULE



```
import cv2
import mediapipe as mp
import time


class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5,modelComplexity=1,trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.modelComp=modelComplexity
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,self.modelComp, self.detectionCon, self.track
        self.myDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
#         # print(results.multi_hand_landmarks)
        if self.results.multi_hand_landmarks:
            for handLas in self.results.multi_hand_landmarks:
                if draw:
                    self.myDraw.draw_landmarks(img, handLas, self.mpHands.HAND_CONNECTIONS)
        return img
#
```

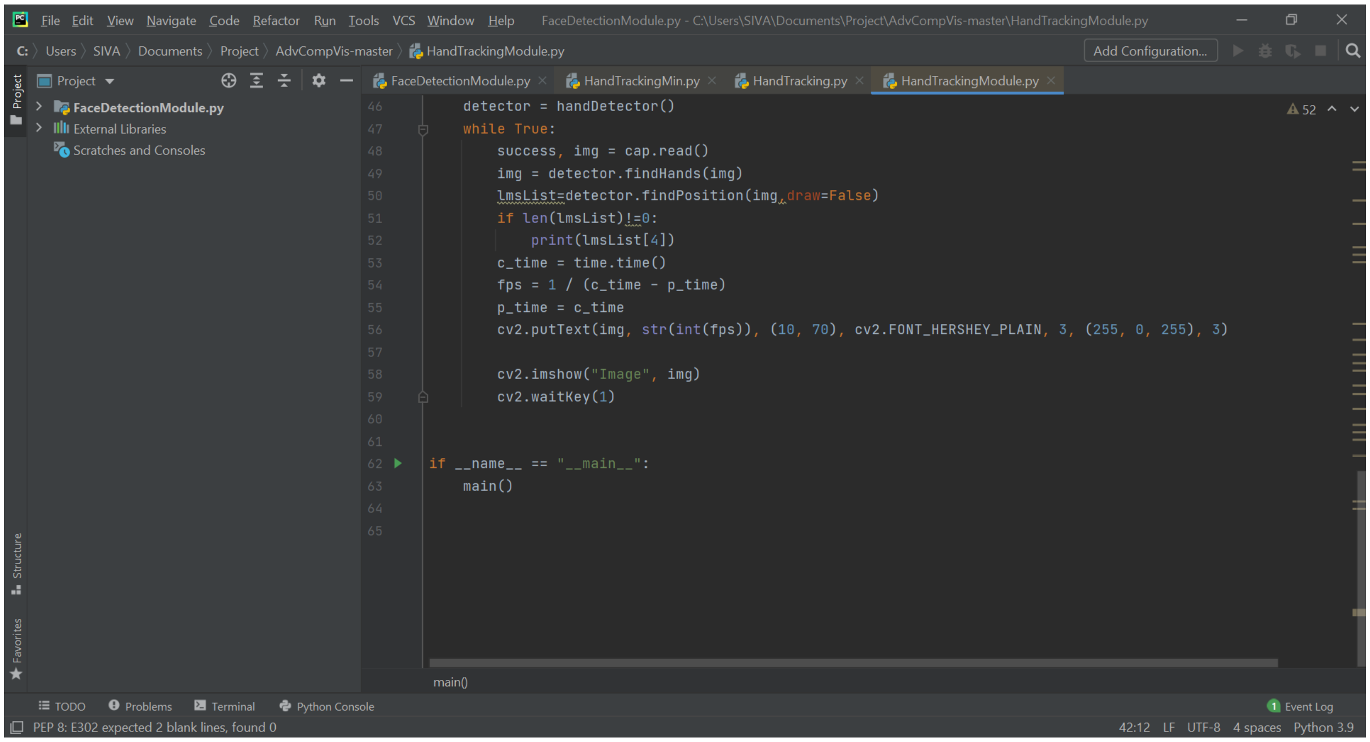
Figure 5.5 : HandTracking Module.py 2



```
    def findPosition(self,img,handNo=0,draw=True):

        lmsList=[]

        if self.results.multi_hand_landmarks:
            myHand=self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                # print(id,lm)
                h,w,c=img.shape
                cx,cy=int(lm.x*w),int(lm.y*h)
                # print(id,cx,cy)
                lmsList.append([id,cx,cy])
                if draw:
                    cv2.circle(img,(cx,cy),7,(155,8,255),cv2.FILLED)
        return lmsList
def main():
    p_time = 0
    c_time = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmsList=detector.findPosition(img,draw=False)
        if len(lmsList)!=0:
```

PEP 8: E111 indentation is not a multiple of four. Variable in function should be lowercase. PEP 8: E303 too many blank lines (2).

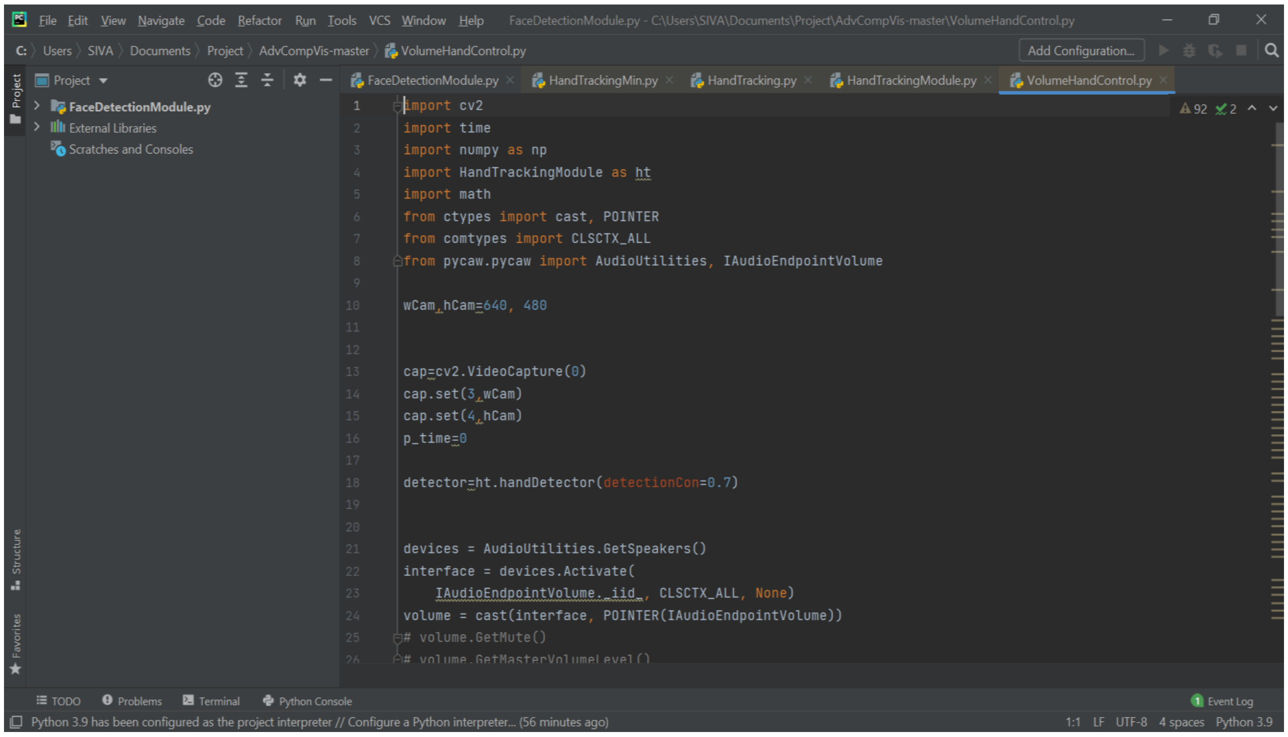
Figure 5.6: HandTracking Module 3


File Edit View Navigate Code Refactor Run Tools VCS Window Help
FaceDetectionModule.py - C:\Users\SIVA\Documents\Project\AdvCompVis-master\HandTrackingModule.py
C: Users SIVA Documents Project AdvCompVis-master HandTrackingModule.py
Add Configuration...
Project
FaceDetectionModule.py
External Libraries
Scratches and Consoles
FaceDetectionModule.py
HandTrackingMin.py
HandTracking.py
HandTrackingModule.py
TODO
Problems
Terminal
Python Console
Event Log
PEP 8: E302 expected 2 blank lines, found 0
42:12 LF UTF-8 4 spaces Python 3.9

```
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmsList=detector.findPosition(img,draw=False)
        if len(lmsList)!=0:
            print(lmsList[4])
        c_time = time.time()
        fps = 1 / (c_time - p_time)
        p_time = c_time
        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)


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

Figure 5.7: HandTracking Module 4

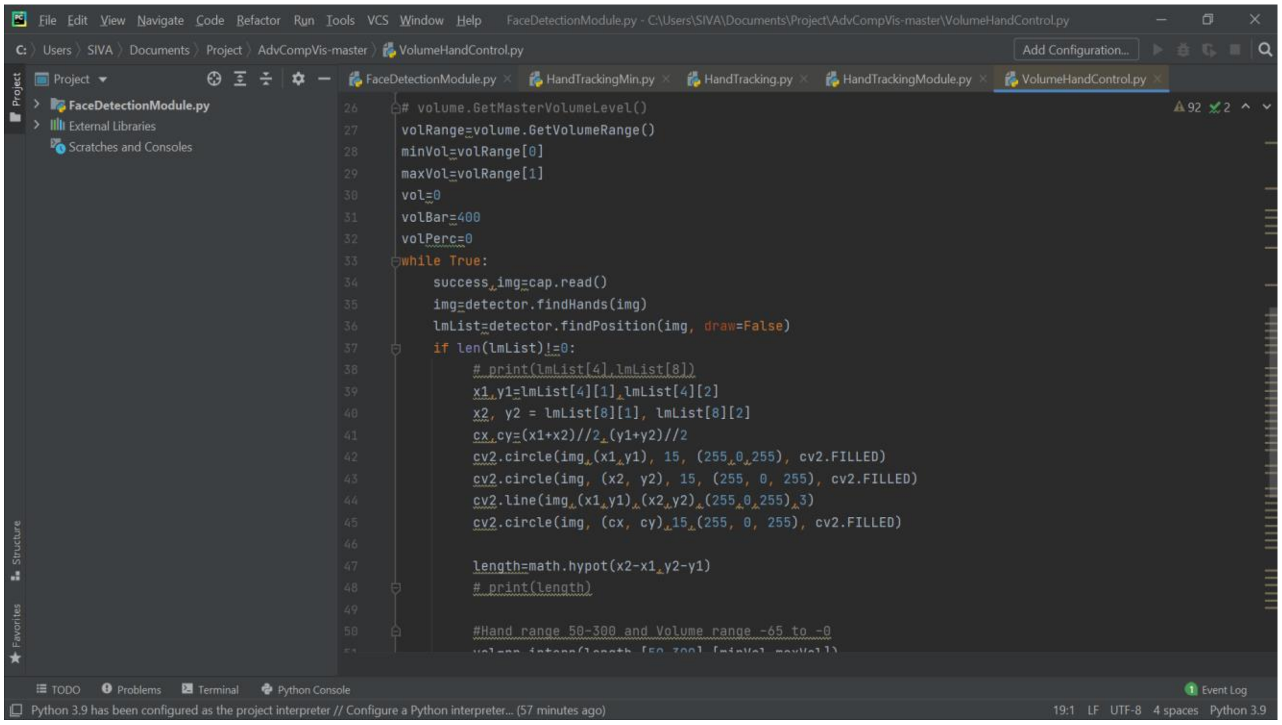
5.2.4 VOLUME HAND CONTROL MODULE



```python
import cv2
import time
import numpy as np
import HandTrackingModule as ht
import math
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

wCam,hCam=640, 480


cap=cv2.VideoCapture(0)
cap.set(3,wCam)
cap.set(4,hCam)
p_time=0

detector=ht.handDetector(detectionCon=0.7)


devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
# volume.GetMute()
# volume.GetMasterVolumeLevel()
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
FaceDetectionModule.py - C:\Users\SIVA\Documents\Project\AdvCompVis-master\VolumeHandControl.py
C: Users SIVA Documents Project AdvCompVis-master VolumeHandControl.py
Add Configuration...
Project
FaceDetectionModule.py
External Libraries
Scratches and Consoles
FaceDetectionModule.py
HandTrackingMin.py
HandTracking.py
HandTrackingModule.py
VolumeHandControl.py
TODO
Problems
Terminal
Python Console
Event Log
Python 3.9 has been configured as the project interpreter // Configure a Python interpreter... (56 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.9

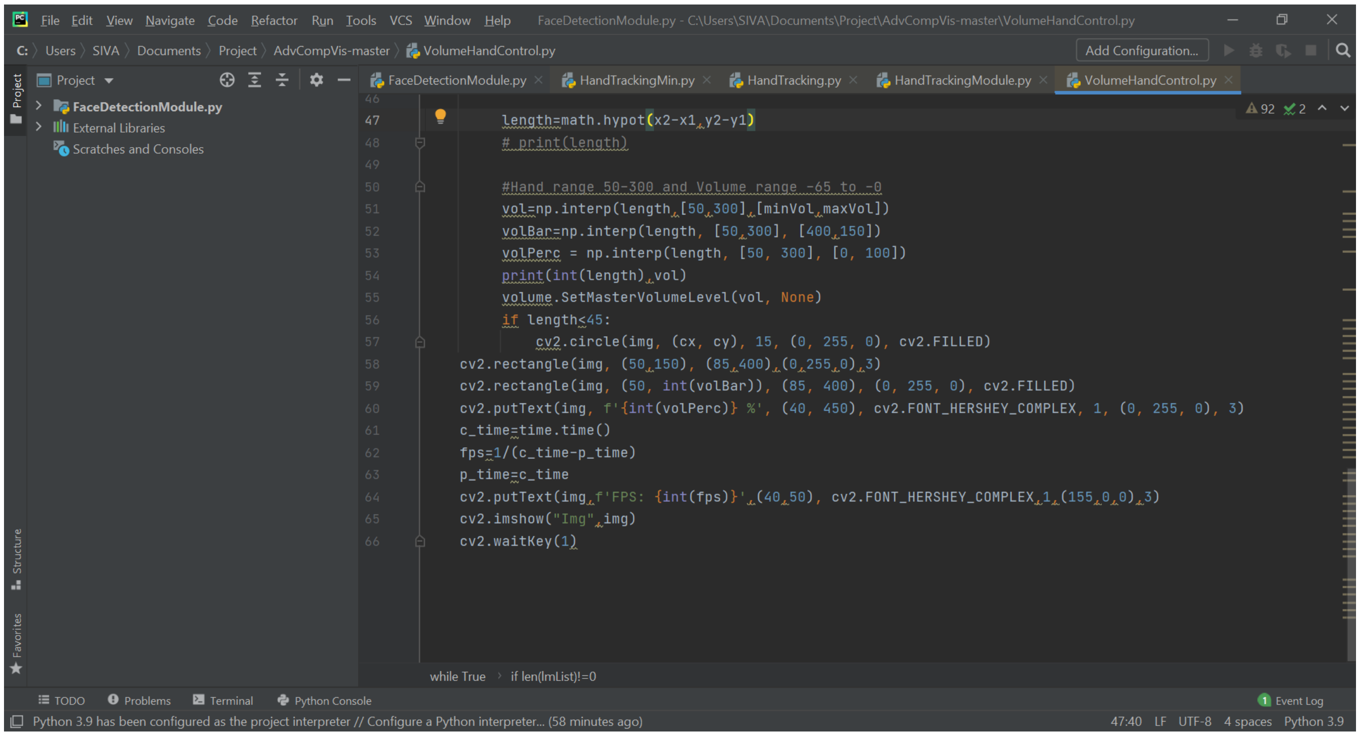
Figure 5.8: Volume Hand Control.py 1



```python
# volume.GetMasterVolumeLevel()
volRange=volume.GetVolumeRange()
minVol=volRange[0]
maxVol=volRange[1]
vol=0
volBar=400
volPerc=0
while True:
    success,img=cap.read()
    img=detector.findHands(img)
    lmList=detector.findPosition(img, draw=False)
    if len(lmList)!=0:
        # print(lmList[4],lmList[8])
        x1,y1=lmList[4][1],lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx,cy=(x1+x2)//2,(y1+y2)//2
        cv2.circle(img,(x1,y1), 15, (255,0,255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img,(x1,y1),(x2,y2),(255,0,255),3)
        cv2.circle(img, (cx, cy),15,(255, 0, 255), cv2.FILLED)

        length=math.hypot(x2-x1,y2-y1)
        # print(length)

        #Hand range 50-300 and Volume range -65 to -0
        vol=np.interp(length,[50,300],[minVol,maxVol])
```

Python 3.9 has been configured as the project interpreter // Configure a Python interpreter... (57 minutes ago)

Figure 5.9: Volume Hand Control.py 2



```python
        length=math.hypot(x2-x1,y2-y1)
        # print(length)

        #Hand range 50-300 and Volume range -65 to -0
        vol=np.interp(length,[50,300],[minVol,maxVol])
        volBar=np.interp(length, [50,300], [400,150])
        volPerc = np.interp(length, [50, 300], [0, 100])
        print(int(length),vol)
        volume.SetMasterVolumeLevel(vol, None)
        if length<45:
            cv2.circle(img, (cx, cy), 15, (0, 255, 0), cv2.FILLED)
    cv2.rectangle(img, (50,150), (85,400),(0,255,0),3)
    cv2.rectangle(img, (50, int(volBar)), (85, 400), (0, 255, 0), cv2.FILLED)
    cv2.putText(img, f'{int(volPerc)} %', (40, 450), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 3)
    c_time=time.time()
    fps=1/(c_time-p_time)
    p_time=c_time
    cv2.putText(img,f'FPS: {int(fps)}',(40,50), cv2.FONT_HERSHEY_COMPLEX,1,(155,0,0),3)
    cv2.imshow("Img",img)
    cv2.waitKey(1)
```

Python 3.9 has been configured as the project interpreter // Configure a Python interpreter... (58 minutes ago)

Figure 5.10: Volume Hand Control.py 3

CHAPTER 6 : CONCLUSION & FUTURE SCOPE

In conclusion, the mini project on advanced computer vision has successfully demonstrated the potential of advanced techniques and algorithms for image classification tasks. The project has implemented convolutional neural networks, which have been shown to achieve high accuracy rates in image recognition tasks. The use of deep learning techniques in computer vision has revolutionized the field and opened up new possibilities for applications such as autonomous driving, medical imaging, and surveillance. The project also highlights the limitations and challenges of existing methods, such as the need for large datasets and computational resources.

The future scope of advanced computer vision is vast, and there is significant potential for continued innovation and advancement in the field. One area of future research is the development of more efficient and robust deep learning algorithms and architectures. The development of more efficient algorithms and architectures could help reduce the computational resources required for image classification tasks and enable faster and more accurate predictions. Additionally, the incorporation of other data sources, such as audio and video, could help improve the accuracy and efficiency of image classification tasks.

Another area of future research is the application of computer vision in medical imaging. Medical imaging is a critical area of healthcare, and the use of computer vision techniques could help improve diagnosis and treatment. For example, the development of algorithms that can analyze medical images to detect tumors or other abnormalities could help doctors make more accurate diagnoses and develop more effective treatment plans. Additionally, computer vision could help automate the process of medical image analysis, reducing the workload on healthcare professionals and improving the efficiency of healthcare systems.

Computer vision is also increasingly being applied in the field of autonomous driving. Advanced computer vision techniques can help self-driving cars detect and recognize objects and make decisions in real-time. The development of more robust and efficient computer vision algorithms could help improve the safety and reliability of autonomous vehicles and accelerate the adoption of this technology.

Finally, computer vision has numerous applications in surveillance and security. For example, computer vision techniques can be used to monitor and analyze video feeds from security cameras to detect and recognize potential threats. The development of more advanced computer vision algorithms could help improve the accuracy and efficiency of surveillance systems, enhancing public safety and security.

REFERENCES

1. **Face Recognition Models**

By A.J. O'Toole, H. Abdi,
in International Encyclopedia of the Social & Behavioral Sciences, 2001.

2. **An improved hand gesture recognition system using keypoints and hand bounding boxes**

By Tuan Linh Dang , Sy Dat Tran , Thuy Hang Nguyen , Suntae Kim , Nicolas Monet.

3. **Hand Gesture Recognition Using Convolutional Neural Networks for American Sign Language**

By V. Athreya, B. Prabhakar, and S. Sathyanarayana.

4. **Facial recognition technology: a survey of policy and implementation issues**

By J. Jacobson and J. F. Seeds.

5. **Face recognition using convolutional neural networks: a survey**

By M. Al-ma'adeed, A. Alja'am, and H. Al-ani.

6. **Youtube-Advanced Computer Vision and its Fundamentals - By FreeCodeCamp**