CV - MEDIAPIPE

A PROJECT REPORT SUBMITTED TO

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

AWARD OF THE DEGREE OF

BACHELOR OF COMPUTER APPLICATIONS

BY

ABHIJITH U

REG NO. RA1931241010017

UNDER THE GUIDANCE OF

Dr. S. ALBERT ANTONY RAJ M.Sc., M.Phil., Ph.D.

DEPARTMENT OF COMPUTER APPLICATIONS

COLLEGE OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Kattankulathur – 603 203

Chennai, Tamil Nadu

May - 2022

BONAFIDE CERTIFICATE

This is to certify that the project report titled “CV - Mediapipe” is a bonafide work carried out by ABHIJITH U (RA1931241010017) under my supervision for the award of the Degree of Bachelor of Computer Applications. To my knowledge, the work reported herein is the original work done by this student.

Dr. S. Albert Antony Raj
Deputy Dean & Head
Department of Computer Application
(GUIDE)

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

With profound gratitude to the **ALMIGHTY,** I take this chance to thank the people who helped me to complete this project.

I take this as a right opportunity to say THANKS to my parents who are there to stand with me always with the words “YOU CAN”.

I am thankful to **Dr.T.R. Paarivendhar**, Chancellor, SRM Institute of Science & Technology who gave us the platform to establish me to reach greater heights.

I earnestly thank **Dr.A. Duraisamy**, Dean, Faculty of Science and Humanities, SRM Institute of Science & Technology who always encourage us to do novel things.

I express my sincere thanks to **Dr. S. Albert Antony Raj M.Sc., M.Phil., Ph.D.,** Deputy Dean, Department of Computer Applications, for his valuable guidance and support to execute all incline in learning and also for his help, support, encouragement, suggestions, and guidance throughout the development phases of the project.

I convey my gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to friends and people who are known and unknown to me who helped in carrying out this project work a successful one.

ABHIJITH U

ABSTRACT

Computer vision is a field that shows how computers can achieve great understanding from digital images and videos. It enables systems to provide meaningful information from these images, videos and other visual inputs and do actions based on that information. Live capturing of human pose, face and hand tracking in real time has many modern life applications like sport-fitness analysis, gesture control, AR try-on and effects. The project enables this technology combined with whiteboard software features (Microsoft Whiteboard can be taken as an example) creates a VIRTUAL WHITEBOARD that can be controlled using hand gestures. This project allows users to draw images and write notes in thin air using their fingers. Therefore, no stylus or pointing-writing-drawing devices required anymore. All you need is a webcam and your hands. This can be done using Google MediaPipe that offers cross-platform, customizable ML solutions for live and streaming media. The proposed system is using the MediaPipe Holistic pipeline as it integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. This project aims in creating an easy and hectic-free environment for effective meetings and to engage learning by helping users to visualize ideas and to work creatively with notes, shapes and more. The project also facilitates distance learning by running collaborative lessons. I hope that through this project the aim of maximizing learning outcomes for all the knowledge seekers out there is full filled.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 OBJECTIVES

- ✓ Using OpenCV to track live webcam during online class/presentation.
- ✓ This live video will be sent to an application which uses MediaPipe holistic pipeline to create face, pose and hand landmarks to record gestures.
- ✓ Certain gestures are already prebuilt into the application. Like to go into the drawing mode or to explain a certain concept etc.
- ✓ Based on the user's gestures, respective functions will take place. Therefore, no need of any external hardware.
- ✓ With the same gesture control, the user can even create new gestures for new actions.

1.2 FEATURES

1. Using real-time video from webcam to read and interpret hand movements as commands.
2. Virtual Whiteboard/Paint where the user can draw or present.
3. Using images/videos and explaining concepts like we have seen in science fiction movies.
4. This project can be used in online meetings or presentations and also by teachers for explaining topics and much more.

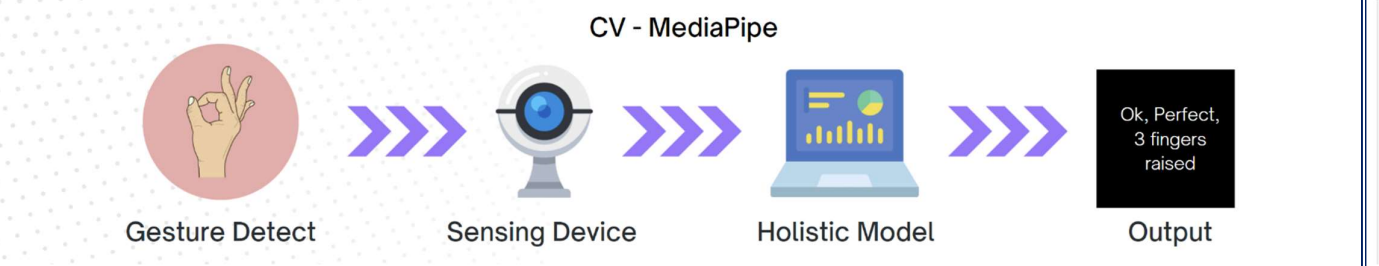


Fig 1: Working of the project

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 SOFTWARE SPECIFICATIONS

A Software Requirements Specification is a document or set of documentation that describes the features and behaviour of a system or software application. It is useful in estimating the cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity

Operation System	:	Windows 8,10 or 11
Programming Language	:	PYTHON 3.6 and higher
IDE	:	PyCharm 2021.2 and higher

2.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the systems and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design.

Processor	:	Intel(R) Core(TM) i7 or Higher AMD Ryzen 5 or Higher
RAM	:	32.00 GB or Higher
System type	:	64-bit operating system, x64-based processor
Keyboard	:	Normal or Multimedia
Mouse	:	Compatible mouse
Webcam	:	USB 2.0 HD UVC Webcam or any external camera

3. ABOUT THE SOFTWARE

PyCharm 2021.2.1 (Community Edition) Python IDE

PyCharm is an integrated development environment used in computer programming, specifically for the Python programming language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda. PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also an educational version, as well as a Professional Edition with extra features (released under a subscription-funded proprietary license).

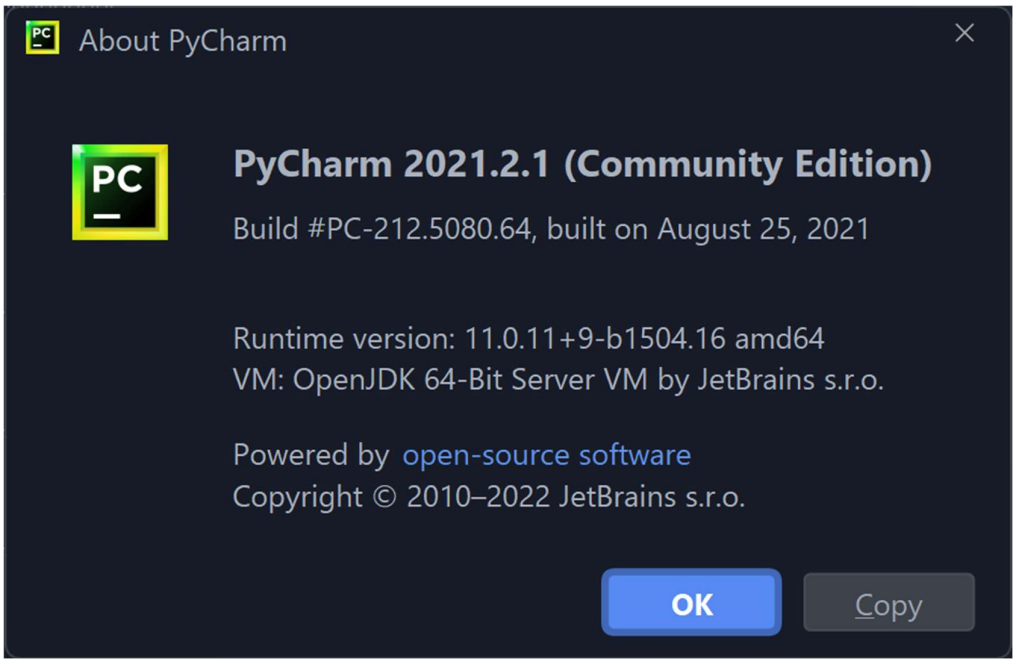


Fig 2: PyCharm 2021.2.1 (Community Edition)

4. CORE CONCEPTS

4.1 Artificial Intelligence

Artificial intelligence (AI) is the ability of a machine to imitate intelligent human behaviour. It enables machines to process information and make decisions based on logic and reasoning. This includes Computer Vision, Machine Learning, Autonomous Systems, Natural Language Processing, Pattern Recognition, Neuromorphic Computing, Cognitive Cyber-Security, Robotic Personal Assistants, Autonomous Surgical Robotics, Next-Gen Cloud Robotics and so on.

4.2 Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. Machine learning is the study of computer algorithms that can improve automatically through experience and by the use of data.

4.2.1 Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output. In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher. Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y). In the real-world, supervised learning can be used for Risk Assessment, Image classification, Fraud Detection, spam filtering, etc.

4.2.2 Unsupervised Machine Learning

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format. In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

4.2.3 Reinforcement Machine Learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

4.3 Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

4.4 Computer Vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs and take actions or make recommendations based on that information. It is used to detect and classify objects (e.g., road signs or traffic lights), create 3D maps or motion estimation, and played a key role in making autonomous vehicles a reality.

4.5 Google AI MediaPipe

MediaPipe is a cross-platform framework for building multimodal applied machine learning pipelines created by Google AI.

- ✓ **End-to-End acceleration**: Built-in fast ML inference and processing accelerated even on common hardware
- ✓ **Build once, deploy anywhere**: Unified solution works across Android, iOS, desktop/cloud, web and IoT
- ✓ **Ready-to-use solutions**: Cutting-edge ML solutions demonstrating full power of the framework
- ✓ **Free and open source**: Framework and solutions both under Apache 2.0, fully extensible and customizable

The ML Solutions in MediaPipe are Face Detection, Face Mesh, Iris, Hands, Pose, Holistic, Hair Segmentation, Object Detection, Box Tracking, Instant Motion Tracking, Objectron, KNIFT.

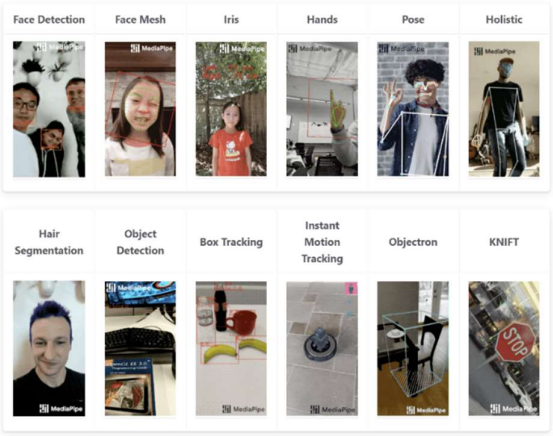


Fig 3: ML Solutions in MediaPipe

4.5.1 MediaPipe Holistic Pipeline

Live perception of simultaneous human pose, face landmarks, and hand tracking in real-time on mobile devices can enable various modern life applications: fitness and sport analysis, gesture control and sign language recognition, augmented reality try-on and effects. MediaPipe already offers fast and accurate, yet separate, solutions for these tasks. Combining them all in real-time into a semantically consistent end-to-end solution is a uniquely difficult problem requiring simultaneous inference of multiple, dependent neural networks.

Fig 4: MediaPipe Holistic Pipeline

4.5.2 ML Pipeline

The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. However, because of their different specializations, the input to one component is not well-suited for the others. The pose estimation model, for example, takes a lower, fixed resolution video frame (256x256) as input. But if one were to crop the hand and face regions from that image to pass to their respective models, the image resolution would be too low for accurate articulation. Therefore, we took MediaPipe Holistic as

a multi-stage pipeline, which treats the different regions using a region appropriate image resolution.

First, we estimate the human pose with pose detector and subsequent landmark model. Then, using the inferred pose landmarks we derive three regions of interest (ROI) crops for each hand (2x) and the face, and employ a re-crop model to improve the ROI. We then crop the full-resolution input frame to these ROIs and apply task-specific face and hand models to estimate their corresponding landmarks. Finally, we merge all landmarks with those of the pose model to yield the full 540+ landmarks.

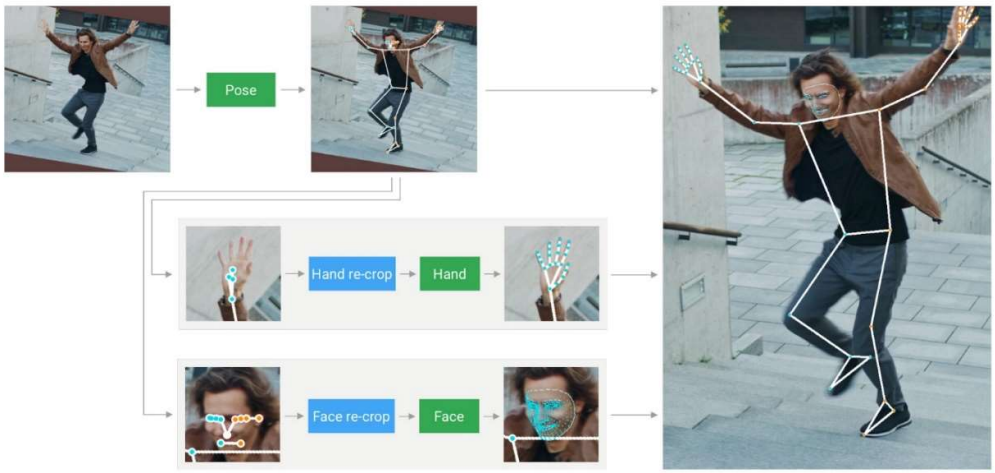


Fig 5: Pose, Face and Hand Detection

To streamline the identification of ROIs for face and hands, we utilize a tracking approach similar to the one we use for standalone face and hand pipelines. It assumes that the object doesn't move significantly between frames and uses estimation from the previous frame as a guide to the object region on the current one. However, during fast movements, the tracker can lose the target, which requires the detector to re-localize it in the image. MediaPipe Holistic uses pose prediction (on every frame) as an additional ROI prior to reduce the response time of the pipeline when reacting to fast movements. This also enables the model to retain semantic consistency across the body and its parts by preventing a mix-up between left and right hands or body parts of one person in the frame with another.

In addition, the resolution of the input frame to the pose model is low enough that the resulting ROIs for face and hands are still too inaccurate to guide the re-cropping of those regions, which require a precise input crop to remain lightweight. To close this accuracy gap, we use lightweight face and hand re-crop models that play the role of spatial transformers and cost only ~10% of corresponding model's inference time.

The pipeline is implemented as a MediaPipe graph that uses a holistic landmark subgraph from the holistic landmark module and renders using a dedicated holistic renderer subgraph. The holistic landmark subgraph internally uses a pose landmark module, hand landmark module and face landmark module. Please check them for implementation details.

4.5.3 Pose Estimation Quality

To evaluate the quality of our models against other well-performing publicly available solutions, we use three different validation datasets, representing different verticals: Yoga, Dance and HIIT. Each image contains only a single person located 2-4 meters from the camera. To be consistent with other solutions, we perform evaluation only for 17 key points from COCO topology.

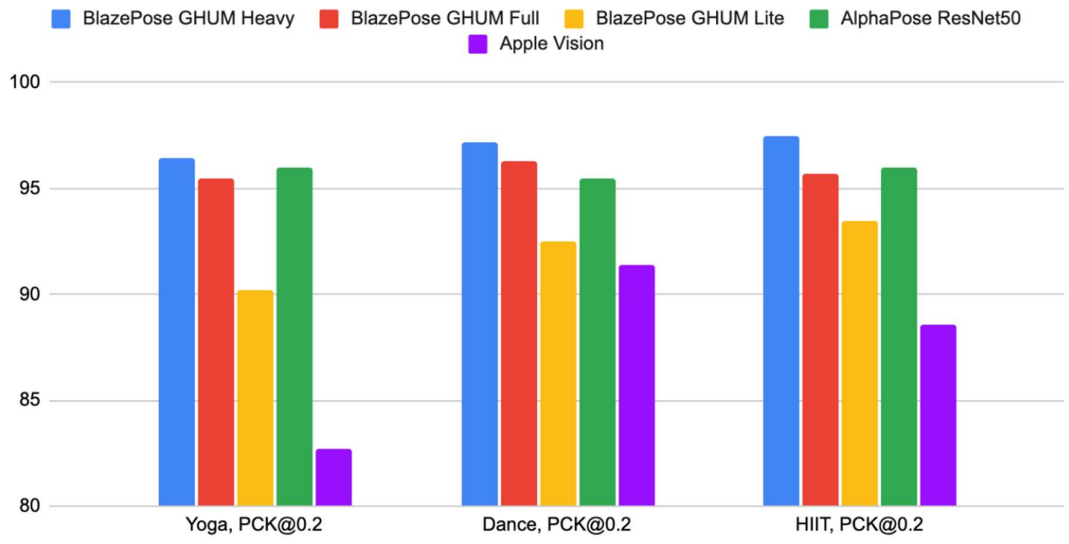
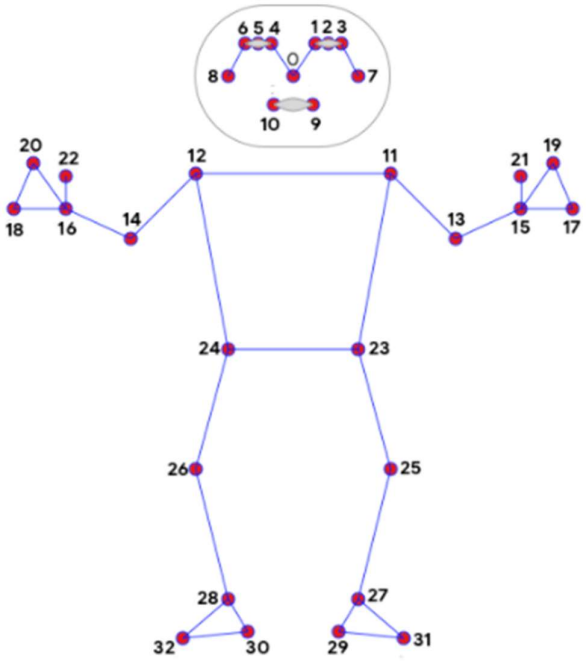


Fig 6: Graphical representation for Yoga, Dance, HIIT.

4.5.4 Pose Landmark Model

The landmark model in MediaPipe Pose predicts the location of 33 pose landmarks.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Fig 7: Pose Landmark model

4.6 Introduction to Python 3.9.10

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

4.6.1 Python Variables

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value. In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type. Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

4.6.2 Python Data Types

Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type. For example, a = 5. The variable **a** holds integer value five and we did not define its type. Python interpreter will automatically interpret variables **a** as an integer type. Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.

4.6.3 Python Decision Making

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Sl.No	Statement & Description
1.	An **if statement** consists of a boolean expression followed by one or more statements.
2.	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
3.	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

4.6.4 Python Loops

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times.

Sl.No	Loop Type & Description
1.	**While Loop :** Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2.	**For Loop :** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3.	**Nested Loop** : You can use one or more loop inside any another while, for or do..while loop.

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Sl.No	Control Statements and Description
1.	**Break :** Terminates the loop statement and transfers execution to the statement immediately following the loop.
2.	**Continue :** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3.	**Pass** : The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

4.6.5 Python Lists, Tuples, Dictionary

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

```
list1 = [‘physics’, ‘chemistry’, 1997, 2000]

list2 = [1, 2, 3, 4, 5]

list3 = [“a”, “b”, “c”, “d”]
```

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

```
tup1 = (‘physics’, ‘chemistry’, 1997, 2000)

tup2 = (1, 2, 3, 4, 5)

tup3 = (“a”, “b”, “c”, “d”)
```

In dictionary, each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}. Keys are unique within a dictionary while values may not be.

```
dict1 = {‘Name’ : ‘Neha’, ‘Age’ :  22, ‘Class’ : ’First’}

dict1[‘Gender’] = “Female”

print(dict1[‘Name’]) # Neha
```

5. SYSTEM ANALYSIS

The purpose of the designing phase is to plan a solution for the problem specified by the requirement document. System analysis is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components. It specifies what the system should do.

5.1 EXISTING SYSTEM

There are some existing systems which does some of these features but not in an efficient manner. But there is no such system that has this exact set of features.

5.2 PROPOSED SYSTEM

After understanding the existing system and the need to create something better and helpful for the society, we developed a new system. The proposed system is using the MediaPipe Holistic pipeline as it integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. The following are the objectives considered in the proposed system

- ✓ Using OpenCV to track live webcam during online class/presentation.
- ✓ This live video will be sent to an application which uses MediaPipe holistic pipeline to create face, pose and hand landmarks to record gestures.
- ✓ Certain gestures are already prebuilt into the application. Like to go into the drawing mode or to explain a certain concept etc.
- ✓ Based on the user's gestures, respective functions will take place. Therefore, no need of any external hardware.
- ✓ With the same gesture control, the user can even create new gestures for new actions.

5.3 FEASIBILITY STUDY

A feasibility study is an analysis of how successfully a project can be completed, accounting for that affect it such as economic, technological, legal and scheduling factors. Feasibility study tests the visibility of an idea, project or even a new business. The goal of a Feasibility study is to place emphasis on potential problems that could occur if a project is pursued and determine if, after all significant factors are considered, and how it will operate, potential obstacles, competition and the finding needed to get the business up and running.

Steps involved in feasible study:

1. Analyse the problem

2. Do your research

3. Make a plan

4. Select the best - proposed system.

5. Check your data

6. Prepare and report final project directive to management.

5.4 TECHNICAL FEASIBILITY

A study of resource availability that may affect the ability to achieve an acceptable system. This evaluation determines whether the technology needed for the proposed system is available or not.

- ✓ Can the work for the project be done with current equipment existing software technology & available personal?
- ✓ Can the system be upgraded if developed?
- ✓ If new technology is needed then what can be developed?

This study is carried out to check the technical feasibility that is the technical requirements of the system. Any system developed must not have a high demand on the

available technical resources. This will lead to high demands on the available technical resources. The application is developed using OpenCV and PyCharm and Google AI.

5.5 ECONOMICAL FEASIBILITY

This is based on in making recommendations a study of the proposed system should be made. In the project all software's are open source. Economic justification is generally the "Bottom Line" consideration for most systems. Economic justification includes a broad range of concerns that includes cost benefit analysis. In this we weight the cost and the benefits associated with the candidate system and if it suits the basic purpose of the organization i.e. profit making, the project is making to the analysis and design phase.

The financial and the economic questions during the preliminary investigation are verified to estimate the following:

- The cost to conduct a full system investigation.
- The cost of hardware and software for the class of application being considered.
- The proposed system will give the minute information, as a result the Performance is improved.

6. SYSTEM DESIGN

6.1 Introduction

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. The Process of design involves concerning and planning out in the mind and making a pattern or sketch.

6.2 UML Diagrams

A UML diagram is a diagram based on the UML (Unified Modelling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

- Class
- Use-case
- Sequence
- Collaboration
- State Chart

6.3 Diagrams

6.3.1 Class Diagram

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

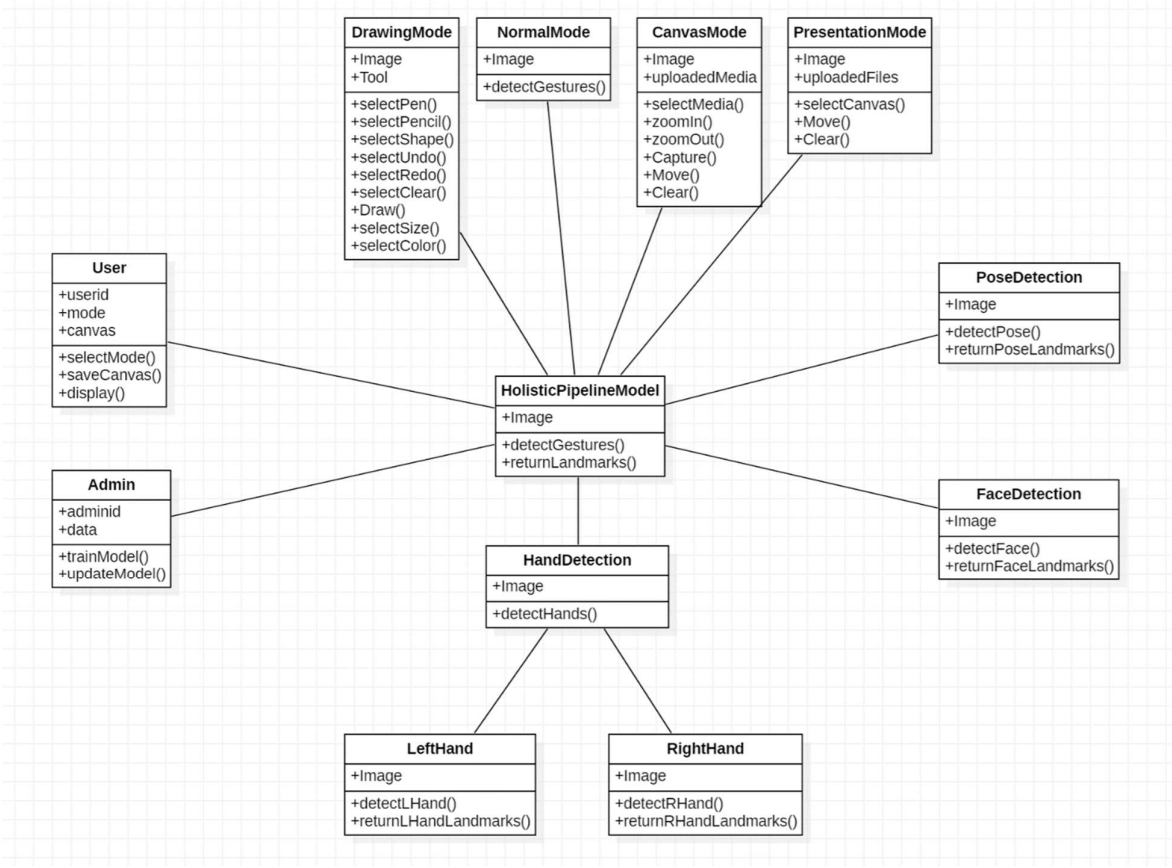


Fig 8: Class Diagram

6.3.2 Use-Case Diagram

Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

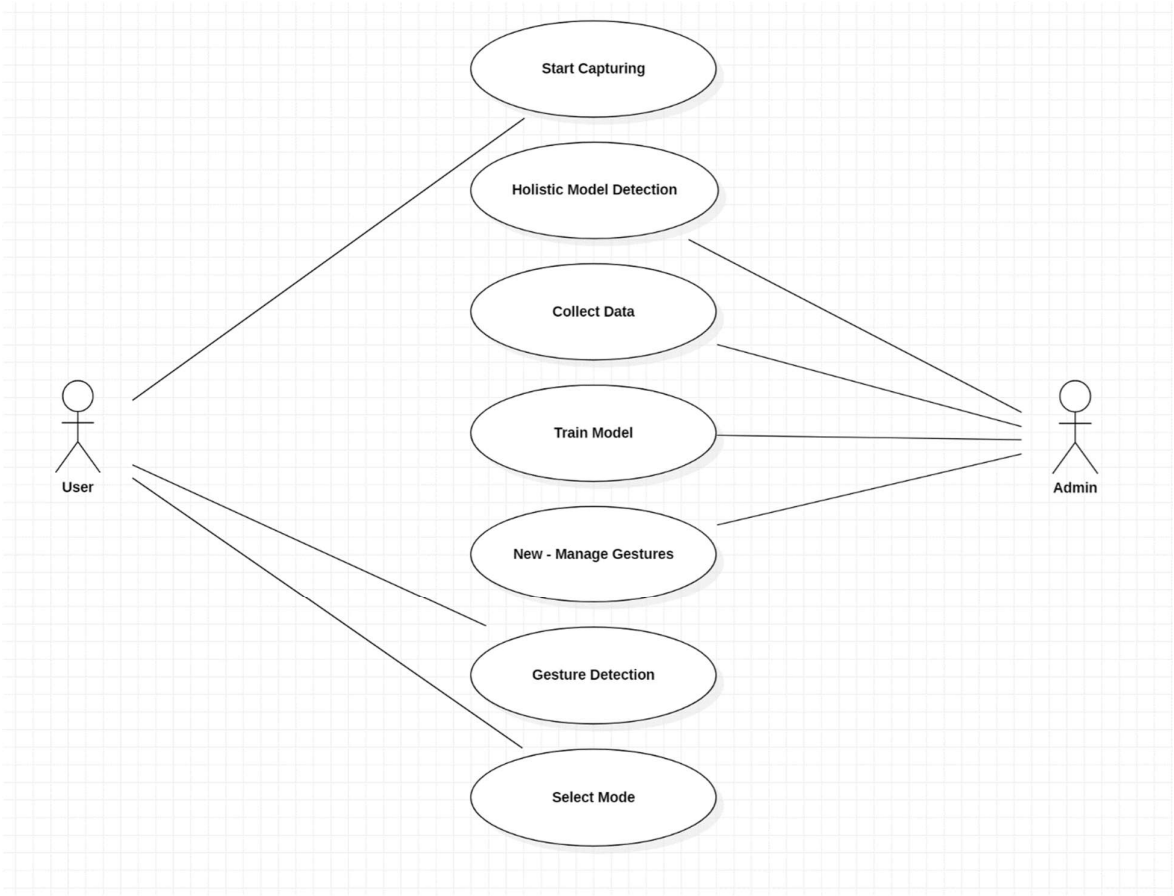


Fig 9: Use-Case Diagram

6.3.3 Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system.

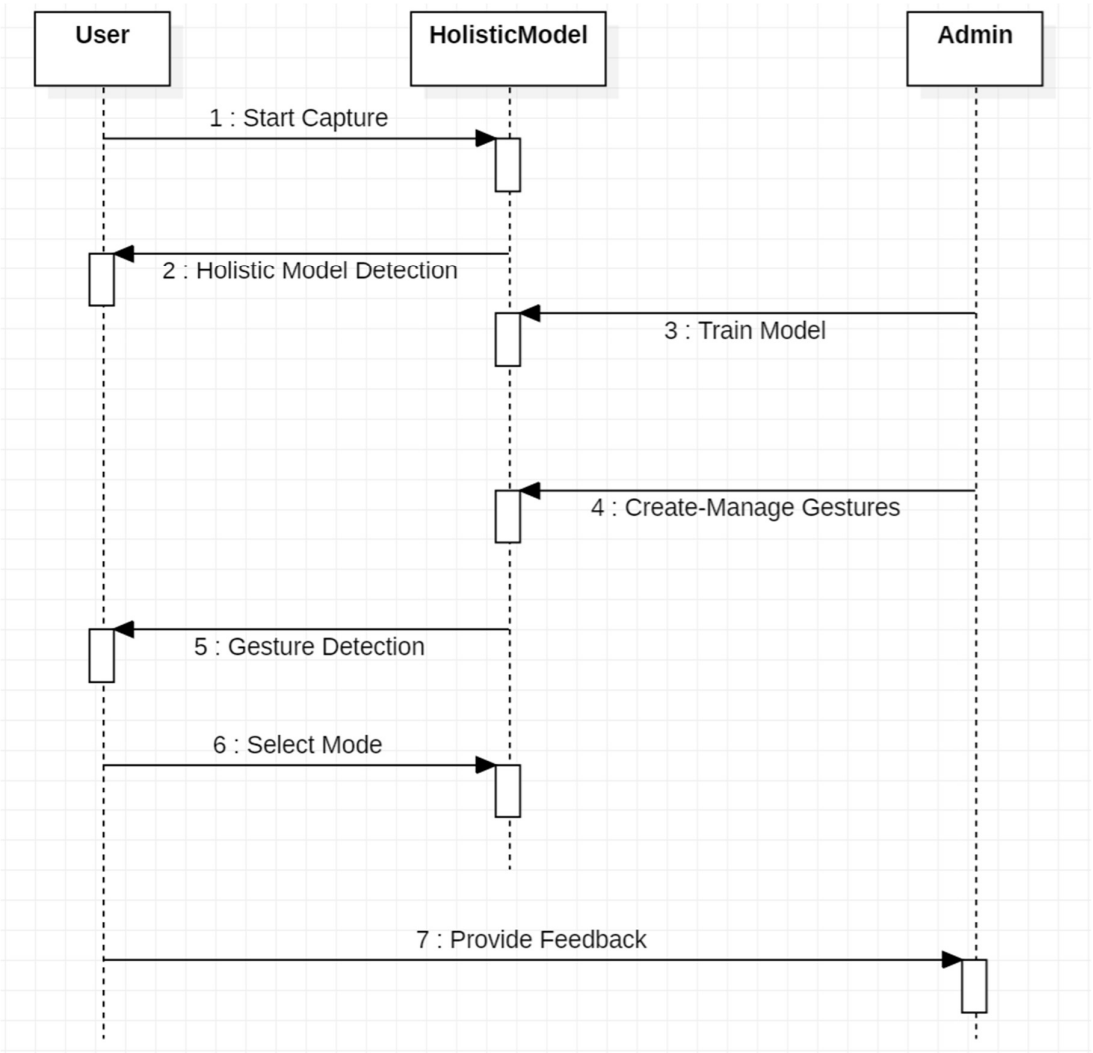


Fig 10: Sequence Diagram

6.3.4 Collaboration Diagram

The collaboration diagram is used to show the relationship between the objects in a system. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

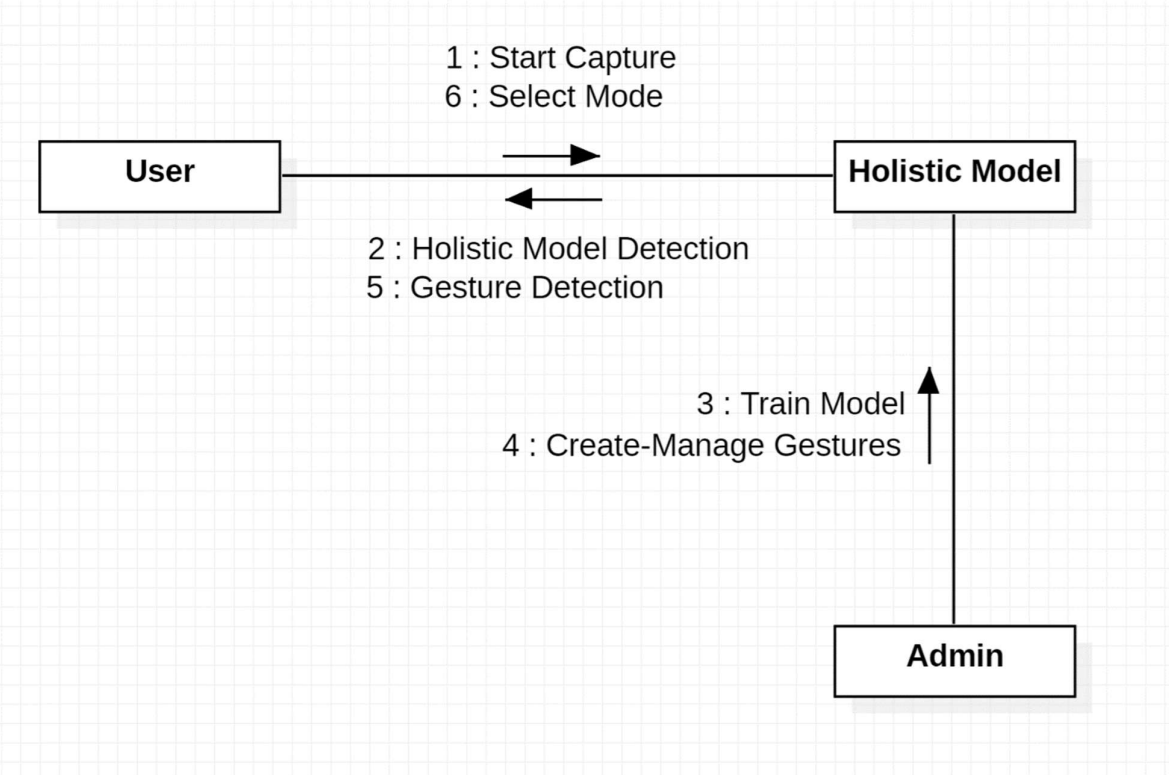


Fig 11: Collaboration Diagram

6.3.5 State Chart Diagram

State chart captures the software system's behaviour. It models the behaviour of a class, a subsystem, a package, and a complete system.

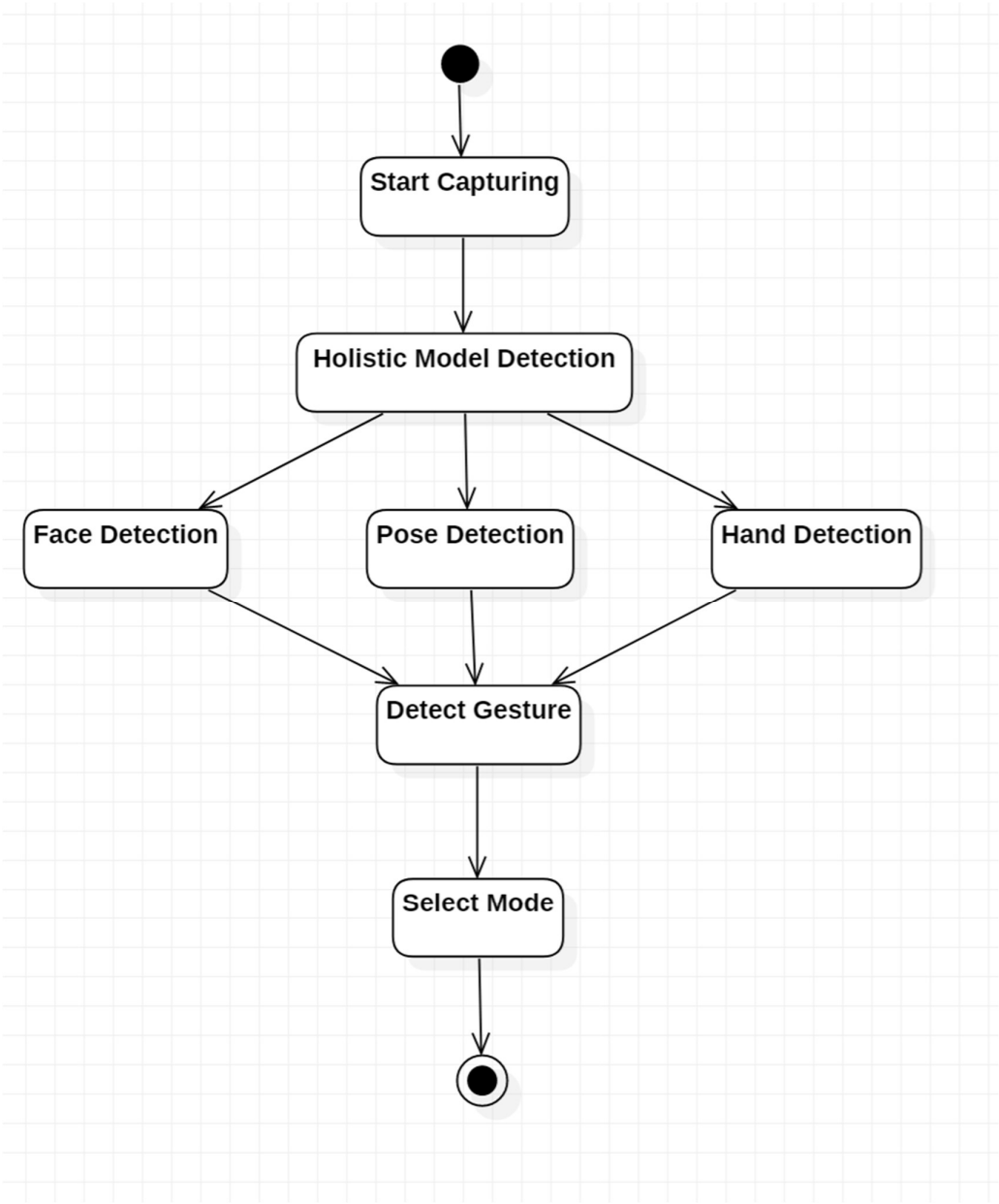


Fig 12: State Chart Diagram

7. CODING

7.1 File Structure

CV - MediaPipe

Sl. No.	File Name	File Description
1.	bg_asset - white.png	Background image used for drawing mode.
2.	menu - 1.png - 2.png - 3.png - 4.png - 5.png - 6.png - 7.png	Menu images for default view, pencil, brush, shape, undo, redo and clear all options.
3.	AI virtual Painter.py	Module for Virtual Drawing
4.	body_segmentation.py	Module for body segmentation using TensorFlow.
5.	fingercount-bothhands.py	Module for counting the number of fingers raised – first step towards gesture control.
6.	handdetectionmodule.py	Module that returns the landmark points in hand detection using OpenCV
7.	handwheelholistic.py	Creating wheel menu bar in holistic detection.
8.	project.py	Final Project Module

9.	selfie_segmentation.py	Module for selfie segmentation – can be used as a feature for removing or blurring the background.
10.	slider_control_holistic.py	Creating a slider menu bar in holistic detection.
11.	requirements.txt	A file with all the versions of packages required for this specific project.
12.	gesturecontrol.png	List of gestures and instructions.

7.2 SOURCE CODE

AI virtual Painter.py

```
import cv2
import numpy as np
import time
import os
import handdetectionmodule as htm

folderPath = "menu"
myList = os.listdir(folderPath)
print(myList)
overlayList = []
for imPath in myList:
    image = cv2.imread(f'{folderPath}/{imPath}')
    overlayList.append(image)

header = overlayList[0]

cap = cv2.VideoCapture(0)

################################
wCam, hCam = 1280, 720
################################

cap.set(3, wCam)
cap.set(4, hCam)

detector = htm.handDetector(detectionCon=0.85)
```

```
drawColor = (255, 0, 255)
brushThickness = 15
xp, yp = 0, 0
imgCanvas = np.zeros((hCam, wCam, 3), np.uint8)
while True:
    # 1. Import image
    success, image = cap.read()
    image = cv2.flip(image, 1)

    # 2. Find Hand Landmarks
    image = detector.findHands(image)
    lmlist = detector.findPosition(image, draw=False)

    if len(lmlist) != 0:
        # tip of index and middle fingers
        x1, y1 = lmlist[8][1:]
        x2, y2 = lmlist[12][1:]

        # 3. Check which fingers are up
        fingers = detector.fingersUp()
        # print(fingers)

        # 4. If Selection mode - two fingers are up
        if fingers[1] and fingers[2]:
            xp, yp = 0, 0
            print(x1, y1)
            cv2.rectangle(image, (x1, y1 - 25), (x2, y2 + 25), drawColor, cv2.FILLED)
            print("Selection Mode")
            # Check for click
            if y1 < 125:
                if 100 < x1 < 200:
                    drawColor = (255, 0, 255)
                    header = overlayList[1]  # pencil tool
                elif 250 < x1 < 350:
                    drawColor = (255, 0, 255)
                    header = overlayList[2]  # Brush tool
                elif 400 < x1 < 510:
                    drawColor = (0, 0, 0)
                    header = overlayList[3]  # Eraser tool
                elif 550 < x1 < 700:
                    drawColor = (255, 0, 255)
                    header = overlayList[4]  # Shapes tool
                elif 750 < x1 < 850:
                    drawColor = (255, 0, 255)
                    header = overlayList[5]  # Undo tool
                elif 900 < x1 < 1000:
                    drawColor = (255, 0, 255)
                    header = overlayList[6]  # Redo tool
                elif 1100 < x1 < 1200:
                    drawColor = (255, 0, 255)
                    imgCanvas = np.zeros((hCam, wCam, 3), np.uint8)
                    header = overlayList[7]  # Delete tool
                    # header = overlayList[0]
                # else:
                #     header = overlayList[0]

        # 5. If Drawing mode - Index finger is up
        if fingers[1] and fingers[2] == False:
            cv2.circle(image, (x1, y1), 15, drawColor, cv2.FILLED)
            print("Drawing Mode")
            if xp == 0 and yp == 0:
                xp, yp = x1, y1
            if drawColor == (0, 0, 0):
                brushThickness = 50
            else:
                brushThickness = 15
```

```
            cv2.line(image, (xp, yp), (x1, y1), drawColor, brushThickness)
            cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, brushThickness)

            xp, yp = x1, y1

    # Combining with image
    imgGray = cv2.cvtColor(imgCanvas, cv2.COLOR_BGR2GRAY)
    _, imgInv = cv2.threshold(imgGray, 50, 255, cv2.THRESH_BINARY_INV)
    imgInv = cv2.cvtColor(imgInv, cv2.COLOR_GRAY2BGR)
    image = cv2.bitwise_and(image, imgInv)
    image = cv2.bitwise_or(image, imgCanvas)

    # Setting the header image
    image[0:125, 0:1280] = header
    # image = cv2.addWeighted(image, 0.5, imgCanvas, 0.5, 0)  # blend with the canvas
    cv2.imshow("image", image)
    # cv2.imshow("Canvas", imgInv)
    cv2.waitKey(1)
```

body_segmentation.py

```
import time
import cv2
import mediapipe as mp


mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic


pTime = 0
# cap = cv2.VideoCapture('http://192.168.29.201:4747/video')
# cap = cv2.VideoCapture('other_assets/Pexels Videos 2675513.mp4')
cap = cv2.VideoCapture(0)
currmin, currmax  = 60000, 0

##############################
wCam, hCam = 1280, 960
##############################

cap.set(3, wCam)
cap.set(4, hCam)


def set_color(color, t, r): # color - BGR, thickness, circle_radius
    return mp_drawing.DrawingSpec(color=color, thickness=t, circle_radius=r)


def add_fps(image):
    global pTime, currmin, currmax
    # FPS CODE
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    # find the max fps

    if fps > currmax:
        currmax = fps

    # find the min fps
    if fps < currmin:
        currmin = fps

    print("Max = ", currmax, " --- Min = ", currmin, " --- FPS = ", fps)
```

```
    # cv2.putText(image, f'FPS: {int(fps)}', (40, 250), cv2.FONT_HERSHEY_COMPLEX,
    #             1, (255, 0, 0), 3)
    # return image

with mp_holistic.Holistic(min_detection_confidence=0.7, min_tracking_confidence=0.7)
as holistic:
    while True:
        ret, image = cap.read()
        add_fps(image)

        # image.flags.writeable = False
        # image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = holistic.process(image)
        # image.flags.writeable = True
        # image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                                  set_color((80, 110, 10), 1, 1),
                                  set_color((80, 256, 121), 1, 1))
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                  set_color((121, 22, 76), 2, 4),
                                  set_color((121, 44, 250), 2, 2))
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                  set_color((121, 22, 76), 2, 4),
                                  set_color((121, 44, 250), 2, 2))
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                  set_color((245, 117, 66), 2, 4),
                                  set_color((245, 66, 230), 2, 2))


        cv2.imshow("Canvas", image)
        cv2.waitKey(1)
    cap.release()
```

fingercount-bothhands.py

```
import mediapipe as mp
import cv2
import time


pTime = 0
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

cap = cv2.VideoCapture(0)

##################################
wCam, hCam = 1280, 960
##################################

cap.set(3, wCam)
cap.set(4, hCam)

tipIds = [4, 8, 12, 16, 20]

def set_color(color, t, r): # color - BGR, thickness, circle_radius
```

```
    return mp_drawing.DrawingSpec(color=color, thickness=t, circle_radius=r)


def fingersUp(lmList, hand):
        fingers = []

        if hand == 0:  # left hand
            fingers.append(0)  # denoting left hand
            # Thumb
            if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
                fingers.append(1)
            else:
                fingers.append(0)
        elif hand == 1:  # right hand
            fingers.append(1)  # denoting right hand
            # Thumb
            if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
                fingers.append(0)
            else:
                fingers.append(1)

        # 4 Fingers
        for id in range(1, 5):
            if lmList[tipIds[id]].y < lmList[tipIds[id] - 2].y:
                fingers.append(1)
            else:
                fingers.append(0)

        return fingers


def detection_start():
    global pTime
    # Initiate holistic model
    with mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)

            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = holistic.process(image)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                                      set_color((80, 110, 10), 1, 1),
                                      set_color((80, 256, 121), 1, 1))
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                      set_color((245, 117, 66), 2, 4),
                                      set_color((245, 66, 230), 2, 2))

            Fcount = 0
            # Start counting
            if results.left_hand_landmarks is not None:
                raised_lfingers = fingersUp(results.left_hand_landmarks.landmark, 1)
                Fcount = Fcount + raised_lfingers.count(1) - 1

            if results.right_hand_landmarks is not None:
                raised_rfingers = fingersUp(results.right_hand_landmarks.landmark, 0)
```

```
                Fcount = Fcount + raised_rfingers.count(1)

            cv2.rectangle(image, (20, 225), (170, 425), (0, 255, 0), cv2.FILLED)

            if Fcount < 10:
                size = 5
            else:
                size = 3

            cv2.putText(image, str(Fcount), (40, 375), cv2.FONT_HERSHEY_COMPLEX,
                    size, (255, 0, 0), 25)

            # FPS CODE
            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime
            cv2.putText(image, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX,
                        1, (255, 0, 0), 3)

            cv2.imshow("Img", image)
            cv2.waitKey(1)


if __name__ == '__main__':
    detection_start()
```

handdetectionmodule.py

```
import cv2
import mediapipe as mp
import time
import math


class handDetector():
    def __init__(self, mode=False, maxHands=2, modelComplexity=1, detectionCon=0.5,
trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.modelComplex = modelComplexity
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.modelComplex,
self.detectionCon, self.trackCon)

        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]


    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 handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
self.mpHands.HAND_CONNECTIONS)
        return img
```

```
    def findPosition(self, img, handNo=0, draw=True):
        xList = []
        yList = []
        bbox = []
        self.lmList = []
        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)
                xList.append(cx)
                yList.append(cy)
                # print(id, cx, cy)
                self.lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            bbox = xmin, ymin, xmax, ymax

        if draw:
            cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                          (bbox[2] + 20, bbox[3] + 20), (0, 255, 0), 2)

        return self.lmList#, bbox


    def fingersUp(self):
        fingers = []
        # Thumb
        if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)
        # 4 Fingers
        for id in range(1, 5):
            if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)
        return fingers


    def findDistance(self, p1, p2, img, draw=True):
        x1, y1 = self.lmList[p1][1], self.lmList[p1][2]
        x2, y2 = self.lmList[p2][1], self.lmList[p2][2]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        if draw:
            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)
        return length, img, [x1, y1, x2, y2, cx, cy]


def main():
    pTime = 0
    cap = cv2.VideoCapture(1)
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
```

```
        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[4])

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime

    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()
```

handwheelholistic.py

```
import math

import mediapipe as mp
import cv2
import time


pTime = 0
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

cap = cv2.VideoCapture(0)

##############################
wCam, hCam = 1280, 720
##############################

cap.set(3, wCam)
cap.set(4, hCam)

tipIds = [4, 8, 12, 16, 20]
drawColor = (255, 0, 255)

def set_color(color, t, r):  # color - BGR, thickness, circle_radius
    return mp_drawing.DrawingSpec(color=color, thickness=t, circle_radius=r)


def fingersUp(lmList, hand):
    fingers = []

    if hand == 0:  # left hand
        fingers.append(0)  # denoting left hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(1)
        else:
            fingers.append(0)
    elif hand == 1:  # right hand
        fingers.append(1)  # denoting right hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(0)
```

```
        else:
            fingers.append(1)

    # 4 Fingers
    for id in range(1, 5):
        if lmList[tipIds[id]].y < lmList[tipIds[id] - 2].y:
            fingers.append(1)
        else:
            fingers.append(0)

    return fingers


def identify_pose(fingers, positions):
    global xp, yp
    # [0,0,1,1,0,0]
    temp = [0] * 6
    temp[0] = fingers[0]

    for i in range(6):
        for j in positions:
            if i == j:
                temp[i] = 1

    # print(temp, " and ", fingers)

    if fingers == temp:
        return True
    else:
        reset_gesture = [1] * 6
        reset_gesture[0] = fingers[0]

        if fingers == reset_gesture:
            xp, yp = 0, 0
        return False


def detection_start():
    global pTime, drawColor
    col1, col2, col3, col4 = (243, 179, 147), (134, 144, 228), (117, 215, 247), (153,
199, 145)
    tc1, tc2, tc3, tc4 = col1, col2, col3, col4
    selectionCol = (0, 0, 255)
    # Initiate holistic model
    with mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)

            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = holistic.process(image)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                                      set_color((80, 110, 10), 1, 1),
                                      set_color((80, 256, 121), 1, 1))
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                      set_color((245, 117, 66), 2, 4),
```

```
                                    set_color((245, 66, 230), 2, 2))

            # Start Choosing
            ## Right Hand
            if results.left_hand_landmarks is not None:

                rfingers = fingersUp(results.left_hand_landmarks.landmark, 1)

                if results.right_hand_landmarks is not None:
                    lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)

                    if rfingers[1:] == [0, 1, 1, 1, 1] and lfingers[2:] == [0, 0, 0,
0]:
                        rx1, ry1 = int(results.left_hand_landmarks.landmark[0].x *
1280), int(
                            results.left_hand_landmarks.landmark[0].y * 700)
                        rx2, ry2 = int(results.left_hand_landmarks.landmark[8].x *
1280), int(
                            results.left_hand_landmarks.landmark[8].y * 700)
                        rx3, ry3 = int(results.left_hand_landmarks.landmark[12].x *
1280), int(
                            results.left_hand_landmarks.landmark[12].y * 700)

                        d1 = int(math.hypot(rx2 - rx1, ry2 - ry1))+50

                        cv2.circle(image, (rx1-50, (ry1-d1)+20), 12, col1, cv2.FILLED)
                        cv2.circle(image, (rx1, (ry1-d1)+10), 12,  col2, cv2.FILLED)
                        cv2.circle(image, (rx1+50, (ry1-d1)+10), 12,  col3,
cv2.FILLED)
                        cv2.circle(image, (rx1 + 100, (ry1 - d1) + 30), 12,  col4,
cv2.FILLED)

                        cv2.circle(image, (rx3, ry3-25), 17,  selectionCol)

                        # print(rx3, rx1-50)

                        if abs(rx3-(rx1-50)) < 20:
                            selectionCol = (255, 0, 0)
                            col1, col2, col3, col4 = selectionCol, tc2, tc3, tc4

                            cv2.putText(image, 'Normal Mode', (800, 50),
cv2.FONT_HERSHEY_COMPLEX, 1, selectionCol, 2)
                            cv2.putText(image, 'A mode in which no special', (800,
100), cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'gestures are used apart from', (800,
140), cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'the menu gesture.', (800, 180),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)

                        elif abs(rx3-rx1) < 20:
                            selectionCol = (0, 0, 255)
                            col1, col2, col3, col4 = tc1, selectionCol, tc3, tc4

                            cv2.putText(image, 'Drawing Mode', (800, 50),
cv2.FONT_HERSHEY_COMPLEX, 1, selectionCol, 2)
                            cv2.putText(image, 'A mode for drawing and', (800, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'writing, creating shapes and', (800,
140), cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'much more.', (800, 180),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
```

```
                        elif abs(rx3-(rx1+50)) < 20:
                            selectionCol = (0, 255, 255)
                            col1, col2, col3, col4 = tc1, tc2, selectionCol, tc4

                            cv2.putText(image, 'Canvas Mode', (800, 50),
cv2.FONT_HERSHEY_COMPLEX, 1, selectionCol, 2)
                            cv2.putText(image, 'A mode for playing around', (800,
100), cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'with elements, creating', (800, 140),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'new screens.', (800, 180),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                        elif abs(rx3-(rx1+100)) < 20:
                            selectionCol = (0, 255, 0)
                            col1, col2, col3, col4 = tc1, tc2, tc3, selectionCol

                            cv2.putText(image, 'Demo Mode', (800, 50),
cv2.FONT_HERSHEY_COMPLEX, 1, selectionCol, 2)
                            cv2.putText(image, 'A mode for working with', (800, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                            cv2.putText(image, 'images and videos.', (800, 140),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                                        selectionCol, 1)
                        else:
                            col1, col2, col3, col4 = tc1, tc2, tc3, tc4

                        # print(rx1, (ry1-d1)+3)

                        ## Keep a point constant
                        # try:
                        #     cv2.circle(image, (crx1, cry1-25), 15, drawColor,
cv2.FILLED)
                        # except UnboundLocalError:
                        #     crx1, cry1 = tuple((rx1, ry1))


            # FPS CODE
            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime
            cv2.putText(image, f'FPS: {int(fps)}', (40, 250),
cv2.FONT_HERSHEY_COMPLEX,
                        1, (255, 0, 0), 3)

            cv2.imshow("Img", image)
            cv2.waitKey(1)


if __name__ == '__main__':
    detection_start()
```

project.py

```
import math
import os

import mediapipe as mp
import cv2
import time
import numpy as np
```

```
pTime = 0
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

cap = cv2.VideoCapture(0)

##################################
# Cam Variables
wCam, hCam = 1280, 720
cap.set(3, wCam)
cap.set(4, hCam)
##################################


##################################
# opencv variables
tipIds = [4, 8, 12, 16, 20]
##################################


##################################
# Color Variables
drawColor = (255, 0, 255)
col1, col2, col3, col4 = (243, 179, 147), (134, 144, 228), (117, 215, 247), (153, 199,
145)
tc1, tc2, tc3, tc4 = col1, col2, col3, col4
selectionCol = (0, 0, 255)
##################################


##################################
# Drawing Variables
brushThickness = 15
xp, yp = 0, 0
imgCanvas = np.zeros((hCam, wCam, 3), np.uint8)
##################################


##################################
# Slider Vatiables
percent = 20


##################################


def set_color(color, t, r):  # color - BGR, thickness, circle_radius
    return mp_drawing.DrawingSpec(color=color, thickness=t, circle_radius=r)


def fingersUp(lmList, hand):
    fingers = []

    if hand == 0:  # left hand
        fingers.append(0)  # denoting left hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(1)
        else:
            fingers.append(0)
    elif hand == 1:  # right hand
        fingers.append(1)  # denoting right hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(0)
        else:
            fingers.append(1)
```

```
    # 4 Fingers
    for id in range(1, 5):
        if lmList[tipIds[id]].y < lmList[tipIds[id] - 2].y:
            fingers.append(1)
        else:
            fingers.append(0)

    return fingers


def identify_pose(fingers, positions):
    global xp, yp
    # [0,0,1,1,0,0]
    temp = [0] * 6
    temp[0] = fingers[0]

    for i in range(6):
        for j in positions:
            if i == j:
                temp[i] = 1

    # print(temp, " and ", fingers)

    if fingers == temp:
        return True
    else:
        reset_gesture = [1] * 6
        reset_gesture[0] = fingers[0]

        if fingers == reset_gesture:
            xp, yp = 0, 0
        return False


def add_landmarks(frame, holistic):
    global mp_drawing, mp_holistic

    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = holistic.process(image)
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                              set_color((80, 110, 10), 1, 1),
                              set_color((80, 256, 121), 1, 1))
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              set_color((121, 22, 76), 2, 4),
                              set_color((121, 44, 250), 2, 2))
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              set_color((121, 22, 76), 2, 4),
                              set_color((121, 44, 250), 2, 2))
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              set_color((245, 117, 66), 2, 4),
                              set_color((245, 66, 230), 2, 2))

    return image, results


def add_fps(image):
    global pTime
    # FPS CODE
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(image, f'FPS: {int(fps)}', (40, 250), cv2.FONT_HERSHEY_COMPLEX,
```

```
                1, (255, 0, 0), 3)
    return image


def add_desc(image, selectionCol, heading, s1, s2, s3=''):
    cv2.putText(image, heading, (800, 200), cv2.FONT_HERSHEY_COMPLEX, 1, selectionCol,
2)
    cv2.putText(image, s1, (800, 250), cv2.FONT_HERSHEY_SIMPLEX, 1,
                selectionCol, 1)
    cv2.putText(image, s2, (800, 280), cv2.FONT_HERSHEY_SIMPLEX, 1,
                selectionCol, 1)
    if s3 != '':
        cv2.putText(image, s3, (800, 320), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    selectionCol, 1)

    return image


def get_coordinates(landmark, index):
    return int(landmark[index].x * 1280), int(landmark[index].y * 700)


def get_distance(x1, y1, x2, y2):
    return int(math.hypot(x2 - x1, y2 - y1))


def create_menuWheel(image, landmark):
    rx1, ry1 = get_coordinates(landmark, 0)
    rx2, ry2 = get_coordinates(landmark, 8)
    rx3, ry3 = get_coordinates(landmark, 12)

    d1 = get_distance(rx1, ry1, rx2, ry2) + 50

    cv2.circle(image, (rx1 - 50, (ry1 - d1) + 20), 12, col1, cv2.FILLED)
    cv2.circle(image, (rx1, (ry1 - d1) + 10), 12, col2, cv2.FILLED)
    cv2.circle(image, (rx1 + 50, (ry1 - d1) + 10), 12, col3, cv2.FILLED)
    cv2.circle(image, (rx1 + 100, (ry1 - d1) + 30), 12, col4, cv2.FILLED)

    cv2.circle(image, (rx3, ry3 - 25), 17, selectionCol)

    return image


def detection_start():
    global pTime, drawColor, col1, col2, col3, col4, tc1, tc2, tc3, tc4, selectionCol,
mp_holistic
    # Initiate holistic model
    with mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)
            # To improve performance, optionally mark the image as not writeable to
            # pass by reference.
            frame.flags.writeable = False

            image, results = add_landmarks(frame, holistic)

            image = add_fps(image)

            # Start Choosing
            ## Right Hand
            if results.left_hand_landmarks is not None:

                rfingers = fingersUp(results.left_hand_landmarks.landmark, 1)

                if results.right_hand_landmarks is not None:
                    lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)
```

```
                    if rfingers[1:] == [0, 1, 1, 1, 1] and lfingers[2:] == [0, 0, 0,
0]:

                        rx1, ry1 =
get_coordinates(results.left_hand_landmarks.landmark, 0)
                        rx2, ry2 =
get_coordinates(results.left_hand_landmarks.landmark, 8)
                        rx3, ry3 =
get_coordinates(results.left_hand_landmarks.landmark, 12)

                        d2 = get_distance(rx2, ry2, rx3, ry3)
                        d3 = get_distance(rx1, ry1, rx3, ry3)

                        if d3 < 350:

                            image = create_menuWheel(image,
results.left_hand_landmarks.landmark)
                            # print(d2, d3)

                            # todo: Creating a hand based proper wheel alignment

                            if abs(rx3 - (rx1 - 50)) < 20:
                                selectionCol = (255, 0, 0)
                                col1, col2, col3, col4 = selectionCol, tc2, tc3, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Normal Mode',
                                                 s1='A mode in which no special',
                                                 s2='gestures are used apart from',
                                                 s3='the menu gesture.')

                            elif abs(rx3 - rx1) < 20:
                                selectionCol = (0, 0, 255)
                                col1, col2, col3, col4 = tc1, selectionCol, tc3, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Drawing Mode',
                                                 s1='A mode for drawing and',
                                                 s2='writing, creating shapes and',
                                                 s3='much more.')

                                if d2 < 30:
                                    print("Drawing Mode")
                                    # countdown of 5
                                    while True:
                                        status = drawing_mode()
                                        if status:
                                            break

                            elif abs(rx3 - (rx1 + 50)) < 20:
                                selectionCol = (0, 255, 255)
                                col1, col2, col3, col4 = tc1, tc2, selectionCol, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Canvas Mode',
                                                 s1='A mode for playing around',
                                                 s2='with elements, creating',
                                                 s3='new screens.')

                            elif abs(rx3 - (rx1 + 100)) < 20:
                                selectionCol = (0, 255, 0)
                                col1, col2, col3, col4 = tc1, tc2, tc3, selectionCol

                                image = add_desc(image, selectionCol,
                                                 heading='Demo Mode',
                                                 s1='A mode for working with',
                                                 s2='images and videos.')
```

```
                            else:
                                col1, col2, col3, col4 = tc1, tc2, tc3, tc4

                            ## Keep a point constant
                            # try:
                            #     cv2.circle(image, (crx1, cry1-25), 15, drawColor,
cv2.FILLED)
                            # except UnboundLocalError:
                            #     crx1, cry1 = tuple((rx1, ry1))

            cv2.imshow("Canvas", image)
            cv2.waitKey(1)


def get_menu():
    folderPath = "menu"
    myList = os.listdir(folderPath)
    overlayList = [cv2.imread(f'{folderPath}/{imPath}') for imPath in myList]
    header = overlayList[0]
    return header, overlayList


def checkHandexists(results, hand):
    if hand == 0:  # left hand
        try:
            found = results.right_hand_landmarks.landmark
            return True
        except AttributeError:
            return False
    elif hand == 1:  # right hand
        try:
            found = results.left_hand_landmarks.landmark
            return True
        except AttributeError:
            return False


def create_slider(image, pose, control, control_pose):
    global percent
    circleSize = 10
    xs = int(1110 - (310 - ((percent * 310) / 100))) + 3

    if pose:
        cv2.putText(image, "A", (800, 240), cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 0), 3)
        cv2.putText(image, "A", (1070, 240), cv2.FONT_HERSHEY_PLAIN, 4, (255, 0, 0),
3)

        cv2.rectangle(image, (800, 250), (1100, 260), (208, 157, 170), cv2.FILLED)
        cv2.rectangle(image, (800, 250), (xs, 260), (243, 163, 208), cv2.FILLED)
        cv2.circle(image, (xs, 255), 20, (69, 69, 69), cv2.FILLED)
        cv2.circle(image, (xs, 255), circleSize, (243, 163, 208), cv2.FILLED)

        if control is not None:
            rfingers = fingersUp(control.landmark, 1)
            if identify_pose(rfingers, control_pose):
                rx1, ry1 = get_coordinates(control.landmark, 8)
                if 830 < rx1 < 1100 and 230 < ry1 < 270:
                    xs = rx1 + 10
                    circleSize = 15

                    cv2.rectangle(image, (910, 170), (990, 230), (0, 0, 0),
cv2.FILLED)
                    cv2.putText(image, str(percent) + "%", (920, 210),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 3)
                else:
                    circleSize = 10
```

```
            percent = int(((310 - (1110 - xs)) / 310) * 100)

    return percent


def drawing_mode():
    global pTime, drawColor, col1, col2, col3, col4, tc1, tc2, tc3, tc4, selectionCol,
mp_holistic
    global xp, yp, brushThickness, imgCanvas

    header, overlayList = get_menu()

    with mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)
            # To improve performance, optionally mark the image as not writeable to
            # pass by reference.
            frame.flags.writeable = False

            image, results = add_landmarks(frame, holistic)

            image = add_fps(image)

            # Start Choosing

            # Reset
            if (header == overlayList[6]).all():
                header = overlayList[0]

            # if checkHandexists(results, 0):
            #     lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)
            #     if lfingers[2:] == [0, 0, 0, 0]:
            #         header = overlayList[0]

            ## Left Hand
            # Selection Using Left
            if results.right_hand_landmarks is not None:
                xp, yp = 0, 0
                rx6, ry6 = get_coordinates(results.right_hand_landmarks.landmark, 8)
                rx7, ry7 = get_coordinates(results.right_hand_landmarks.landmark, 12)

                lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)

                brush_percent = create_slider(image, identify_pose(lfingers, [1, 2]),
results.left_hand_landmarks, [2, 3])

                if identify_pose(lfingers, [2, 3]):
                    cv2.rectangle(image, (rx6, ry6 - 25), (rx7, ry7 + 25), drawColor,
cv2.FILLED)
                    print("Left Selection Mode")
                    # Check for click
                    if ry6 < 125:
                        if 100 < rx6 < 200:
                            drawColor = (255, 0, 255)
                            header = overlayList[1]  # pencil tool
                        elif 250 < rx6 < 350:
                            drawColor = (255, 0, 255)
                            header = overlayList[2]  # Brush tool
                        elif 400 < rx6 < 510:
                            drawColor = (0, 0, 0)
                            header = overlayList[3]  # Shapes tool
                        elif 750 < rx6 < 850:
                            drawColor = (255, 0, 255)
                            header = overlayList[4]  # Undo tool
                        elif 900 < rx6 < 1000:
                            drawColor = (255, 0, 255)
```

```
                            header = overlayList[5]  # Redo tool
                        elif 1100 < rx6 < 1200:
                            drawColor = (255, 0, 255)
                            imgCanvas = np.zeros((hCam, wCam, 3), np.uint8)
                            header = overlayList[6]  # Delete tool

            ## Right Hand
            if results.left_hand_landmarks is not None:

                rfingers = fingersUp(results.left_hand_landmarks.landmark, 1)

                if results.right_hand_landmarks is not None:
                    lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)

                    if rfingers[1:] == [0, 1, 1, 1, 1] and lfingers[2:] == [0, 0, 0,
0]:
                        rx1, ry1 =
get_coordinates(results.left_hand_landmarks.landmark, 0)
                        rx2, ry2 =
get_coordinates(results.left_hand_landmarks.landmark, 8)
                        rx3, ry3 =
get_coordinates(results.left_hand_landmarks.landmark, 12)

                        d2 = get_distance(rx2, ry2, rx3, ry3)
                        d3 = get_distance(rx1, ry1, rx3, ry3)

                        if d3 < 350:
                            image = create_menuWheel(image,
results.left_hand_landmarks.landmark)

                            if abs(rx3 - (rx1 - 50)) < 20:
                                selectionCol = (255, 0, 0)
                                col1, col2, col3, col4 = selectionCol, tc2, tc3, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Normal Mode',
                                                 s1='A mode in which no special',
                                                 s2='gestures are used apart from',
                                                 s3='the menu gesture.')

                                if d2 < 30:
                                    print("Normal Mode")
                                    # countdown of 5
                                    return True

                            elif abs(rx3 - rx1) < 20:
                                selectionCol = (0, 0, 255)
                                col1, col2, col3, col4 = tc1, selectionCol, tc3, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Drawing Mode',
                                                 s1='A mode for drawing and',
                                                 s2='writing, creating shapes and',
                                                 s3='much more.')

                            elif abs(rx3 - (rx1 + 50)) < 20:
                                selectionCol = (0, 255, 255)
                                col1, col2, col3, col4 = tc1, tc2, selectionCol, tc4

                                image = add_desc(image, selectionCol,
                                                 heading='Canvas Mode',
                                                 s1='A mode for playing around',
                                                 s2='with elements, creating',
                                                 s3='new screens.')

                            elif abs(rx3 - (rx1 + 100)) < 20:
                                selectionCol = (0, 255, 0)
                                col1, col2, col3, col4 = tc1, tc2, tc3, selectionCol
```

```
                                image = add_desc(image, selectionCol,
                                                 heading='Demo Mode',
                                                 s1='A mode for working with',
                                                 s2='images and videos.')
                            else:
                                col1, col2, col3, col4 = tc1, tc2, tc3, tc4

                    rx4, ry4 = get_coordinates(results.left_hand_landmarks.landmark, 8)
                    rx5, ry5 = get_coordinates(results.left_hand_landmarks.landmark, 12)

                    # Selection Using Right
                    if identify_pose(rfingers, [2, 3]):
                        xp, yp = 0, 0
                        cv2.rectangle(image, (rx4, ry4 - 25), (rx5, ry5 + 25), drawColor,
cv2.FILLED)
                        print("Right Selection Mode")
                        # Check for click
                        if ry4 < 125:
                            if 100 < rx4 < 200:
                                drawColor = (255, 0, 255)
                                header = overlayList[1]  # pencil tool
                            elif 250 < rx4 < 350:
                                drawColor = (255, 0, 255)
                                header = overlayList[2]  # Brush tool
                            elif 400 < rx4 < 510:
                                drawColor = (0, 0, 0)
                                header = overlayList[3]  # Shapes tool
                            # elif 550 < rx1 < 700:
                            #     drawColor = (255, 0, 255)
                            #     header = overlayList[4]  # Logo tool
                            elif 750 < rx4 < 850:
                                drawColor = (255, 0, 255)
                                header = overlayList[4]  # Undo tool
                            elif 900 < rx4 < 1000:
                                drawColor = (255, 0, 255)
                                header = overlayList[5]  # Redo tool
                            elif 1100 < rx4 < 1200:
                                drawColor = (255, 0, 255)
                                imgCanvas = np.zeros((hCam, wCam, 3), np.uint8)
                                header = overlayList[6]  # Delete tool

                    # Pen Mode
                    if (header == overlayList[1]).all() and identify_pose(rfingers, [2])
and checkHandexists(results,

0) == False:
                        cv2.circle(image, (rx4, ry4), 15, drawColor, cv2.FILLED)
                        print("Drawing Mode")
                        if xp == 0 and yp == 0:
                            xp, yp = rx4, ry4
                        drawColor = (255, 0, 255)
                        brushThickness = int((brush_percent*30)/100)

                        cv2.line(image, (xp, yp), (rx4, ry4), drawColor, brushThickness)
                        cv2.line(imgCanvas, (xp, yp), (rx4, ry4), drawColor,
brushThickness)
                        xp, yp = rx4, ry4

                    # Eraser Mode
                    if identify_pose(rfingers, [2, 3, 4]) and checkHandexists(results, 0)
== False:
                        drawColor = (0, 0, 0)

                        if xp == 0 and yp == 0:
                            xp, yp = rx4, ry4

                        brushThickness = int((brush_percent*50)/100) + 50
```

```
                    cv2.line(image, (xp, yp), (rx4, ry4), drawColor, brushThickness)
                    cv2.line(imgCanvas, (xp, yp), (rx4, ry4), drawColor,
brushThickness)

                    xp, yp = rx4, ry4

            # Combining with image
            imgGray = cv2.cvtColor(imgCanvas, cv2.COLOR_BGR2GRAY)
            _, imgInv = cv2.threshold(imgGray, 50, 255, cv2.THRESH_BINARY_INV)
            imgInv = cv2.cvtColor(imgInv, cv2.COLOR_GRAY2BGR)
            image = cv2.bitwise_and(image, imgInv)
            image = cv2.bitwise_or(image, imgCanvas)

            # Setting the header image
            image[0:125, 0:1280] = header

            cv2.imshow("Canvas", image)
            # cv2.imshow("Canvas2", imgCanvas)

            cv2.waitKey(1)

    return False


if __name__ == '__main__':
    detection_start()


# todo: 1. bring all control elements infront of imgCanvas.
# todo: 2. Find an option to select Color for pen and brush
# todo: 3. Able to move the elements across
# todo: 4. Check if diagram is a closed diagram, usage of brush tool to fill colors.
# todo: 5. If not a closed diagram, ???
# todo: 6. Shape Wheel. How to choose and how to implement.
# todo: 7. Undo-Redo, how to implement.
```

selfie_segmentation.py

```
import cv2
import mediapipe as mp
import numpy as np
import time

mp_drawing = mp.solutions.drawing_utils
mp_selfie_segmentation = mp.solutions.selfie_segmentation
last_time = 0
BG_COLOR = (255, 255, 255)
# cap = cv2.VideoCapture('other_assets/Pexels Videos 2675513.mp4')
cap = cv2.VideoCapture(0)
# For webcam input
with mp_selfie_segmentation.SelfieSegmentation(model_selection=0) as
selfie_segmentation:
    bg_image = None
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            print("Can't receive frame (stream end?). Exiting ...")
            continue

        frame = cv2.flip(frame, 1)
```

```
        # segmentation code
        frame.flags.writeable = False
        results = selfie_segmentation.process(frame)
        frame.flags.writeable = True

        # apply joint bilateral filter
        condition = np.stack((results.segmentation_mask,) * 3, axis=-1) > 0.1

        # Apply background magic
        # bg_image = cv2.imread('bg_asset/white.png')  # create a virtual bg
        # bg_image = cv2.GaussianBlur(frame, (55, 55), 0)  # blur current bg

        if bg_image is None:
            bg_image = np.zeros(frame.shape, dtype=np.uint8)
            bg_image[:] = BG_COLOR

        out_image = np.where(condition, frame, bg_image)  # return elements chosen
from x or y depending on condition

        # Get frame rate
        fps = cap.get(cv2.CAP_PROP_FPS)
        print("FPS: ", fps)

        # Get current time
        current_time = time.time()
        fps = 1 / (current_time - last_time)
        last_time = current_time
        cv2.putText(out_image, "FPS: {:.2f}".format(fps), (0, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

        # Display results
        cv2.imshow('frame', out_image)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
```

slider_control_holistic.py

```
import mediapipe as mp
import cv2
import time

pTime = 0
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic

cap = cv2.VideoCapture(0)

##################################
wCam, hCam = 1280, 960
##################################

cap.set(3, wCam)
cap.set(4, hCam)

tipIds = [4, 8, 12, 16, 20]


def set_color(color, t, r):  # color - BGR, thickness, circle_radius
    return mp_drawing.DrawingSpec(color=color, thickness=t, circle_radius=r)
```

```
def fingersUp(lmList, hand):
    fingers = []

    if hand == 0:  # left hand
        fingers.append(0)  # denoting left hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(1)
        else:
            fingers.append(0)
    elif hand == 1:  # right hand
        fingers.append(1)  # denoting right hand
        # Thumb
        if lmList[tipIds[0]].x > lmList[tipIds[0] - 1].x:
            fingers.append(0)
        else:
            fingers.append(1)

    # 4 Fingers
    for id in range(1, 5):
        if lmList[tipIds[id]].y < lmList[tipIds[id] - 2].y:
            fingers.append(1)
        else:
            fingers.append(0)

    return fingers


def get_coordinates(landmark, index):
    return int(landmark[index].x * 1280), int(landmark[index].y * 700)


percent = 20


def create_slider(image, results):
    global percent
    circleSize = 10
    # Finding the number
    xs = int(1110 - (310 - ((percent * 310) / 100))) + 3

    # print(xs)

    # Slider Control

    if results.right_hand_landmarks is not None:
        lfingers = fingersUp(results.right_hand_landmarks.landmark, 0)
        if identify_pose(lfingers, [1, 2]):
            # cv2.putText(image, 'A', (800, 240), cv2.FONT_HERSHEY_COMPLEX, 1, (255,
0, 0), 3)
            cv2.putText(image, "A", (800, 240), cv2.FONT_HERSHEY_PLAIN, 2, (255, 0,
0), 3)
            cv2.putText(image, "A", (1070, 240), cv2.FONT_HERSHEY_PLAIN, 4, (255, 0,
0), 3)

            cv2.rectangle(image, (800, 250), (1100, 260), (208, 157, 170), cv2.FILLED)
            cv2.rectangle(image, (800, 250), (xs, 260), (243, 163, 208), cv2.FILLED)
            cv2.circle(image, (xs, 255), 20, (69, 69, 69), cv2.FILLED)
            cv2.circle(image, (xs, 255), circleSize, (243, 163, 208), cv2.FILLED)

    if results.left_hand_landmarks is not None:
        rx1, ry1 = get_coordinates(results.left_hand_landmarks.landmark, 8)
        if 830 < rx1 < 1100 and 230 < ry1 < 270:
            xs = rx1 + 10
            circleSize = 15
```

```
            cv2.rectangle(image, (910, 170), (990, 230), (0, 0, 0), cv2.FILLED)
            cv2.putText(image, str(percent) + "%", (920, 210), cv2.FONT_HERSHEY_PLAIN,
2, (255, 255, 255), 3)

        else:
            circleSize = 10

    percent = int(((310 - (1110 - xs)) / 310) * 100)

    return percent


def identify_pose(fingers, positions):
    global xp, yp
    # [0,0,1,1,0,0]
    temp = [0] * 6
    temp[0] = fingers[0]

    for i in range(6):
        for j in positions:
            if i == j:
                temp[i] = 1

    # print(temp, " and ", fingers)

    if fingers == temp:
        return True
    else:
        reset_gesture = [1] * 6
        reset_gesture[0] = fingers[0]

        if fingers == reset_gesture:
            xp, yp = 0, 0
        return False


def detection_start():
    global pTime
    global percent
    circleSize = 10
    # Initiate holistic model
    with mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)

            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = holistic.process(image)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                                      set_color((80, 110, 10), 1, 1),
                                      set_color((80, 256, 121), 1, 1))
            mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                      set_color((121, 22, 76), 2, 4),
                                      set_color((121, 44, 250), 2, 2))
            mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                      set_color((245, 117, 66), 2, 4),
                                      set_color((245, 66, 230), 2, 2))
```

```
            print(create_slider(image, results))


            # FPS CODE
            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime
            cv2.putText(image, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX,
                        1, (255, 0, 0), 3)

            cv2.imshow("Img", image)
            cv2.waitKey(1)


if __name__ == '__main__':
    detection_start()
```

8. SYSTEM TESTING

Introduction Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation. Once the source code has been generated, software must be tested to uncover as many errors as possible before delivery to the customer. In order to find the highest possible number of errors, tests must be conducted systematically and test cases must be designed using disciplined techniques.

8.1 Test Case

- We have to install first the python pip modules that we are using to do the detection (mediapipe, opencv-python, numpy). These packages with their exact versions are added in requirements.txt file.
- Install the required all pip module using the **pip** command inside the Python39 folder where it is installed.

```
pip install requirements.txt
```

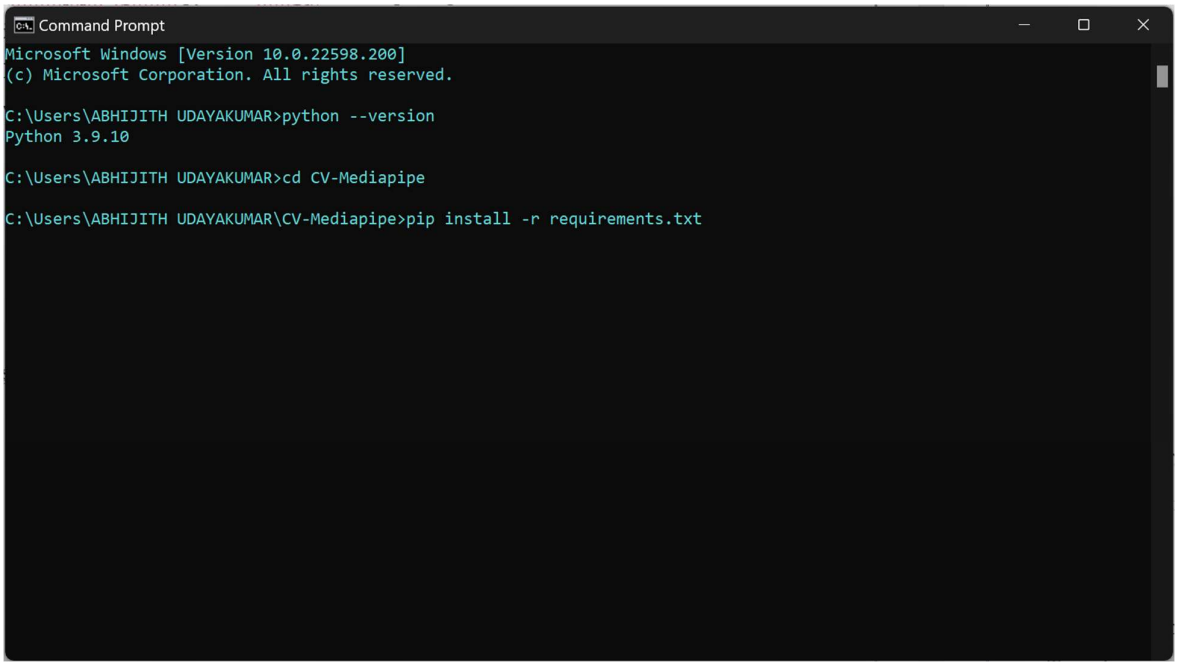


Fig 13: Installing requirements

- Run project.py and use the gestures given in gesturecontrol.png to control your actions in the screen.

```
python project.py
```

8.2 Logs of the results

Inputs: Set of gestures to draw/write a HI on the screen.

Output : Yes, I was able to successfully draw a HI with the brush tool.

8.3 Screenshots

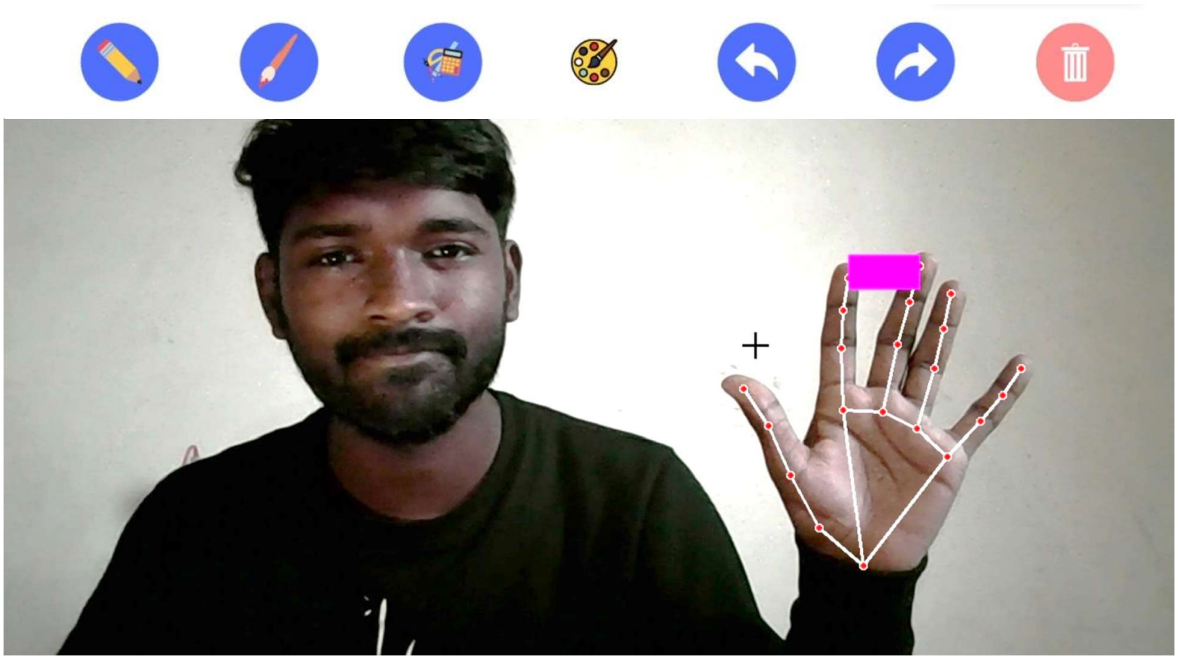


Fig 14: Project Screenshot 1

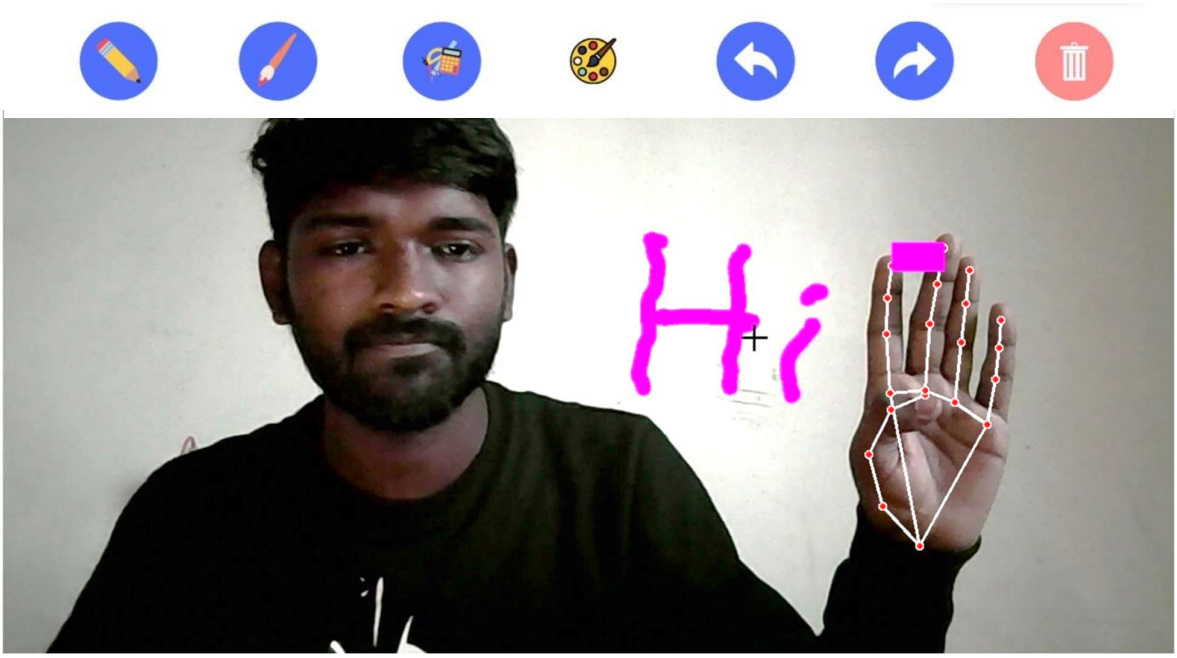


Fig 15: Project Screenshot 2

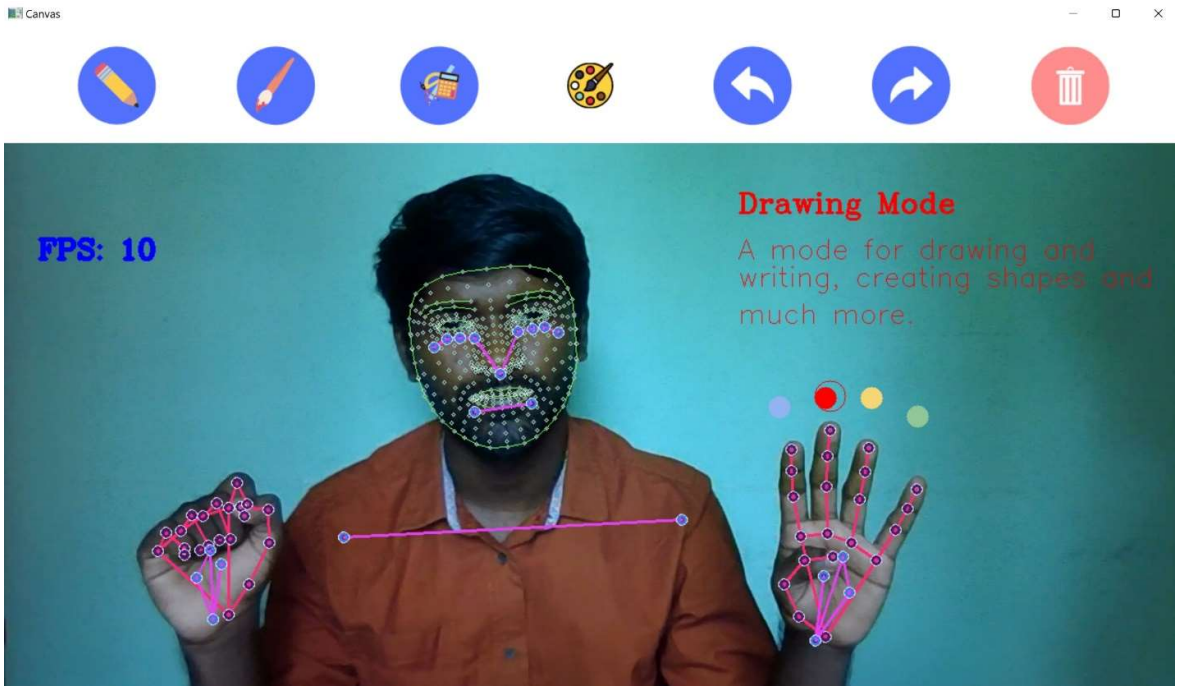


Fig 16: Project Screenshot 3

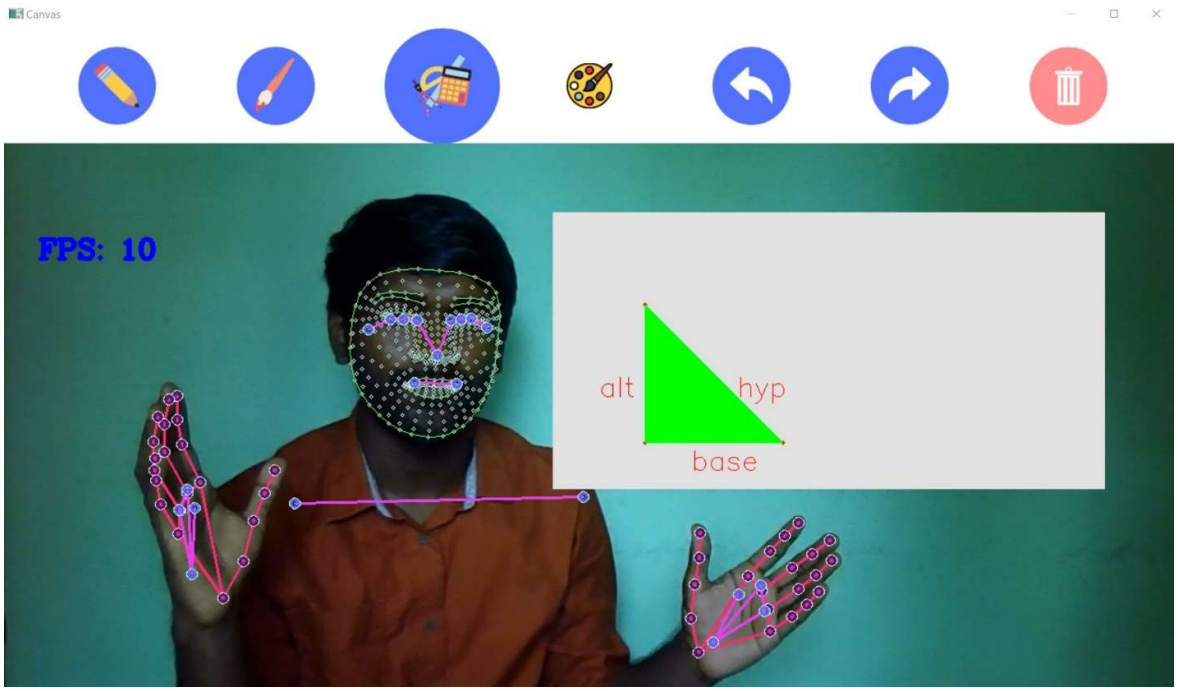


Fig 17: Project Screenshot 4

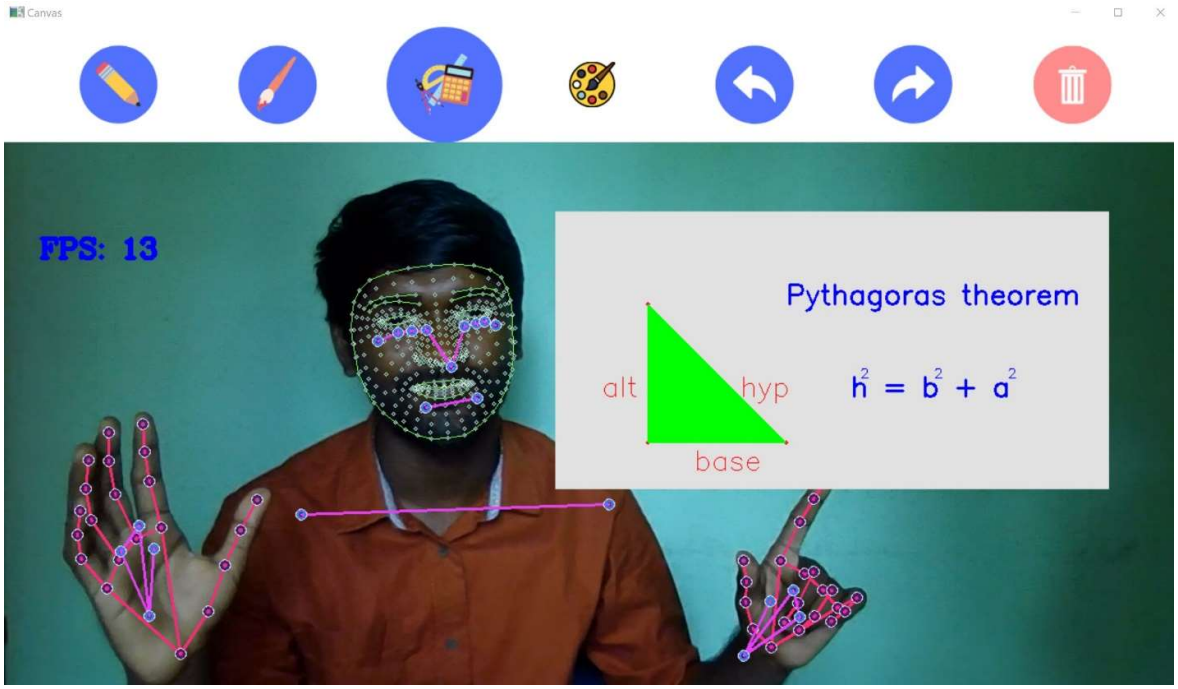


Fig 18: Project Screenshot 5

9. CONCLUSION

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image.

Computer vision trains machines to perform these functions, but it must do it in much less time with cameras, data, and algorithms rather than retinas, optic nerves, and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

This technology along with the ability of Machine Learning, we will be able to automate literally everything in our day-to-day life. OpenCV uses Machine Learning with Convolutional Neural Networks (CNN) to give us different detection modules.

Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will "look" at the data and teach itself to tell one image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.

A CNN helps a machine learning or deep learning model "look" by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is "seeing." The neural network runs convolutions and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

This enables a wide range of applications and for Google AI to create the Holistic MediaPipe Pipeline. Working with these technologies, I aim in creating an easy and hectic-free environment for effective meetings and to engage learning by helping users to visualize ideas and to work creatively with notes, shapes and more.

The project also facilitates distance learning by running collaborative lessons. I hope that through this project the aim of maximizing learning outcomes for all the knowledge seekers out there is full filled.

10. FUTURE ENHANCEMENT

Currently, this project has a lot of room for improvement.

First, include gesture addition and management as a custom action done at the user end. Just like Google assistant/ Apple Siri learns different from each user, I hope to make this project learn different custom gestures specific for different users. And then, I will be able to include the actions, the project is most used for.

Second, Less like a whiteboard software and More like a personalised utility tool. At present, even though the project is feature-rich and better than a whiteboard. Still, it has that whiteboard feeling. I would like to change that by giving it a much better implementation of different actions.

Third, Smooth UI/UX. Currently the project runs in an OpenCV image capture window. This should be changed to a better looking and faster software tool. However, python does not directly allow us to create a software; and implementing Computer Vision and Machine Learning in other languages can be something of concern. I think, an API of this project can be released in python and then use this API for creating the software UI, would work.

Finally, the ability to create anything - anytime – anywhere in just thin air. It sounds science-fiction, but I feel this is possible. For this, the model should be trained with lots and lots of data and should be able to recognise and implement gestures rapidly without any delay.

11. REFERENCES

1) Python Doc : https://www.python.org/doc/
2) Python : https://www.w3schools.com/python/
3) Python : https://www.tutorialspoint.com/python/index.htm
4) Computer Vision : https://www.ibm.com/in-en/topics/computer-vision
5) Machine Learning : https://www.python-engineer.com/courses/mlfromscratch/
6) TensorFlow : https://www.python-engineer.com/courses/tensorflowbeginner/
7) OpenCV : https://www.youtube.com/watch?v=qCR2Weh64h4&list=PLzMcBGfZo4-lUA8uGjeXhBUUzPYc6vZRn
8) Computer Vision with ML : https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e
9) Google AI : https://ai.google
10) MediaPipe : https://google.github.io/mediapipe/

12. README

11.1 Installation Instructions

- ✓ Install Python version 3.6 or higher.
  - Python 3.9
  - Link : https://www.python.org/downloads/release/python-390/
- ✓ Install a Python IDE (I will be using PyCharm as it is my default Python IDE).
  - PyCharm Community 2022.1
  - Link : https://www.jetbrains.com/pycharm/download/#section=windows
- ✓ After installing Python and PyCharm, we need to install the necessary packages for the project.
- ✓ Note : It is recommended to create a virtual environment and install your packages inside the virtual environment.

```
# Installing virtualenv
pip install virtualenv

# Creating virtual environment folder:
virtualenv venv


# Activating environment

# Windows
venv\Scripts\activate

# Finally installing packages inside environment
pip install -r requirements.txt
```

- ✓ If you are getting an error stating that pip is not recognised as an internal or external command, that means you don't have pip installed on your system.
- ✓ Pip is a package management system used to install and manage software packages written in Python.
  - Pip 22.0.3
  - Link : https://www.geeksforgeeks.org/how-to-install-pip-on-windows/

- ✓ These are the packages used in this project :

```
absl-py==1.0.0
astunparse==1.6.3
attrs==21.4.0
cachetools==5.0.0
certifi==2021.10.8
charset-normalizer==2.0.12
cycler==0.11.0
flatbuffers==2.0
fonttools==4.29.1
gast==0.5.3
google-auth==2.6.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.44.0
h5py==3.6.0
idna==3.3
importlib-metadata==4.11.1
keras==2.8.0
Keras-Preprocessing==1.1.2
kiwisolver==1.3.2
libclang==13.0.0
Markdown==3.3.6
matplotlib==3.5.1
mediapipe==0.8.9.1
numpy==1.22.2
oauthlib==3.2.0
opencv-contrib-python==4.5.5.62
opencv-python==4.5.5.62
opt-einsum==3.3.0
packaging==21.3
Pillow==9.0.1
protobuf==3.19.4
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==3.0.7
python-dateutil==2.8.2
requests==2.27.1
requests-oauthlib==1.3.1
rsa==4.8
six==1.16.0
tensorboard==2.8.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.8.0
tensorflow-gpu==2.8.0
tensorflow-hub==0.12.0
tensorflow-io-gcs-filesystem==0.24.0
tensorflowjs==3.13.0
termcolor==1.1.0
tf-bodypix==0.4.0
tf-estimator-nightly==2.8.0.dev2021122109
tfjs-graph-converter==1.5.0
typing_extensions==4.1.1
urllib3==1.26.8
Werkzeug==2.0.3
wincertstore==0.2
wrapt==1.13.3
zipp==3.7.0
```

- ✓ Now that you have activated your virtual environment and installed all necessary packages, you can run the modules and check their functionalities. Make sure that your webcam is active and accessible for the program.

13. PLAGIARISM REPORT

ABHIJITH U

RA1931241010017

Submission date: 21-Apr-2022 01:59PM (UTC+0530)
Submission ID: 1816201820
File name: project_abstract.pdf (813.14K)
Word count: 453
Character count: 2542

A COMPUTER VISION PROJECT USING GOOGLE MEDIAPIPE

ABHIJITH U
RA1931241010017
III BCA A

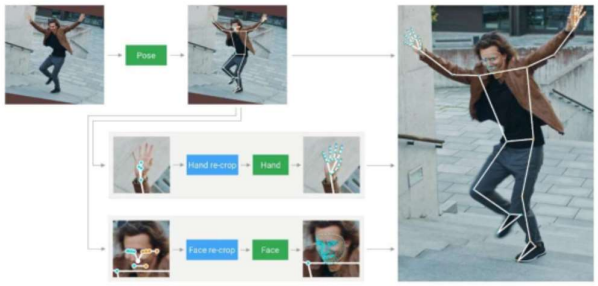
What is Computer Vision?

Computer vision is a field that shows how computers can achieve great understanding from digital images and videos. It enables systems to provide meaningful information from these images, videos and other visual inputs and do actions based on that information.

Abstract

Live capturing of human pose, face and hand tracking in real time has many modern life applications like sport-fitness analysis, gesture control, AR try-on and effects. The project enables this technology combined with whiteboard software features (Microsoft Whiteboard can be taken as an example) creates a VIRTUAL WHITEBOARD that can be controlled using hand gestures. This project allows users to draw images and write notes in thin air using their fingers. Therefore, no stylus or pointing-writing-drawing devices required anymore. All you need is a webcam and your hands.

This can be done using Google MediaPipe that offers cross-platform, customizable ML solutions for live and streaming media. The proposed system is using the MediaPipe Holistic pipeline as it integrates separate models for pose, face and hand components, each of which are optimized for their particular domain.



Hardware Requirements:

- A working webcam

Software Requirements:

- Python 3.6 or higher
- Any web browser (Google Chrome, Mozilla Firefox, Microsoft Edge) or video conference software (Zoom, Microsoft Teams, Skype) for online meetings.

WHAT WE ARE UP TO

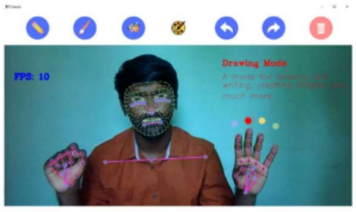
- Using OpenCV to track live webcam during online class/presentation.
- This live video will be sent to an application which uses MediaPipe holistic pipeline to create face, pose and hand landmarks to record gestures.
- Certain gestures are already prebuilt into the application. Like to go into the drawing mode or to explain a certain concept etc.
- Based on the user's gestures, respective functions will take place. Therefore, no need of any external hardware.
- With the same gesture control, the user can even create new gestures for new actions.

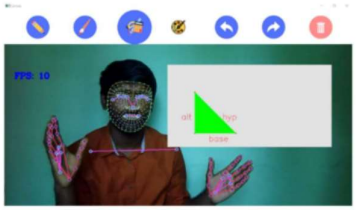
FEATURES

1. Using real-time video from webcam to read and interpret hand movements as commands.
2. Virtual Whiteboard/Paint where the user can draw or present.
3. Using images/videos and explaining concepts like we have seen in science fiction movies.
4. This project can be used in online meetings or presentations and also by teachers for explaining topics and much more.

Conclusion

This project aims in creating an easy and hectic-free environment for effective meetings and to engage learning by helping users to visualize ideas and to work creatively with notes, shapes and more. The project also facilitates distance learning by running collaborative lessons. I hope that through this project the aim of maximizing learning outcomes for all the knowledge seekers out there is full filled.





ABHIJITH U

RA1931241010017

III BCA A

FSH SRMIST

CV-MediaPipe

ORIGINALITY REPORT

10%	10%	0%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	google.github.io Internet Source	5%
2	Submitted to Hong Kong Baptist University Student Paper	3%
3	Submitted to National College of Ireland Student Paper	3%

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	On		