

ABSTRACT

While in taking online classes or in day-to-day activities writing is more important task, as we are living in digital age which means most of our tasks are been shifted to digital trends such as like paying bills, booking tickets, emotion tracking etc., among this writing is at most important task weather you share ideas plan for a particular task taking notes, in this data age stats show humans more often prefer to write and use about 100 pages per average. It is quite our instinct to understand and represent any information which is written in handwriting, but when it comes to digital world though they are gesture based, touch based methods they couldn't match the natural hand-write representation. So, we come with solution for this problem for some extent by using hand tracking and hand landmark position representation which is offered by media pipe library form google. This technique uses image processing (OpenCV) to capture video from webcam which will be given to our python application which uses media pipe library to recognize hand and represents it as 0-20 landmark positions. Which will be used to find index finger for writing and use two fingers for removing or erasing pre-written information. This type of application is very useful in a day to day basic activates for students, teachers and makers in order to easily plan and organize projects, notes etc., Though this technique neither match a real writing experience nor the writing tablet for computers and smartphones which can translate our writing into digital notebook, it does not require any additional equipment and can be easily used to if practiced. This type of application can improvise the understanding by allowing users to explain in a whiteboard like environment in online education.

Table of			
			Page No.
		List of Tables	VII
		List of Figures	VIII
		Abbreviations	IX
1		Introduction	1
	1.1	Motivation	1
	1.2	Problem statement	2
	1.3	Project Objectives	2
	1.4	Project report Organization	3
2		Literature Survey	4
	2.1	Existing work	4
	2.2	Limitations of Existing work	6
3		Software & Hardware specifications	7
	3.1	Software requirements	7
	3.1.1	Functional Requirements	8
	3.1.2	Non-Functional Requirements	9
	3.2	Hardware requirements	9
4		Proposed System Design	10
	4.1	Proposed methods	11
	4.1.1	Palm detector model architecture	14
	4.1.2	Hand landmark model	15
	4.1.3	Dataset and Annotation	16
	4.2	Class Diagram	17
	4.3	Use case Diagram	18
	4.4	Activity Diagram	19
	4.5	Sequence Diagram	20
	4.6	System Architecture	21
	4.7	Technology Description	22

5		Implementation & Testing	25
	5.1	Screenshot	25
	5.1.1	Detection Model	25
	5.1.2	Draw rectangle	26
	5.1.3	Draw circle	26
	5.2	Testing	27
	5.3	Result	27
	5.4	Advantages	28
	5.5	Application	28
6		Conclusion & Future scope	29
	6.1	Conclusion	29
	6.2	Future scope	29
		References:	30
		Appendix: (If any like Published paper / source code)	

LIST OF TABLES

TABLE	TITLE	PAGE NO.
1.	Functional requirements of Pose Tracker.....	8
2	Functional requirements table.....	8
3	Test Cases.....	27

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
Fig 1:	Comparision of existing works.....	5
Fig 2:	Bottom Rendered synthetic hand images with ground truth annotation.....	11
Fig 3:	hand landmarks.....	12
Fig 4:	Hand perception pipeline overview.....	13
Fig 5:	Palm detector model architecture.....	14
Fig 6:	Hand landmark model	15
Fig 7:	Class Diagram.....	17
Fig 8:	Use Case Diagram.....	18
Fig 9:	Activity Diagram.....	19
Fig 10:	Sequence Diagram	20
Fig 11:	System Architecture	21
Fig 12:	Hand detection.....	24
Fig 13:	Free-hand writing.....	24
Fig 14:	Drawing rectangle.....	25
Fig 15:	Drawing circle.....	25

ABBREVIATIONS

CNN	Convolutional neural network
API	Application Program Interface
AI	Artificial Intelligence
Open-CV	Open-Source Computer Vision Library
GPU	Graphical Processing Unit
SQL	Structured Query Language
JSON	Java Script Object Notation
PDF	Portable Document Format
OS	Operating System
RAM	Random Access Memory
CPU	Central Processing Unit
UML	Unified Modelling Language

CHAPTER-1

INTRODUCTION

1. MOTIVATION

Hand tracking and gesture tracking have recently become commonplace, presenting a wide range of opportunities and constraints. As a result of the growing interest in computer vision, accessible technology that can enable new advancements in AI is rapidly developing and improving. We discuss, implement, and review the problems and future prospects in the realm of human user interaction and virtual reality in this paper. In light of the COVID-19 outbreak, the objective for this research is to limit human interaction and reliance on gadgets to operate computers. These findings could lead to more research and, in the long run, help people use virtual worlds more effectively.

Bluetooth and wireless technologies are becoming increasingly accessible, thanks to recent breakthroughs in virtual reality and their implementation in our daily lives. This research offers a visual AI system that uses hand motions and hand tip acquisition to perform mouse, keyboard, and stylus activities using computer vision. Instead of using typical headsets or external devices, the proposed system uses a web camera or built-in camera to track finger and hand movements to process the computer. This solution may be removed indefinitely due to its simplicity and effectiveness. the utilisation of additional hardware, battery durability, and, in the end, user convenience

2. PROBLEM STATEMENT

Build a Digital Handwriting Recognition using Hand Tacking by using media pipe and OpenCV libraries. The hand tracking is done by hand landmark position representation which is offered by media pipe library form google. We use image processing techniques which are offered by OpenCv to capture video from webcam which will be given to our python application which uses media pipe library to recognize hand and represents it as 0-30 landmark positions. Which will be used to find index finger for writing and use two fingers for removing or erasing pre-written information.

3. PROJECT OBJECTIVES

Using hand detection module we need to develop a program by which following are satisfied

- a. By using single, one should be able to draw or write
- b. by using two fingers, one should be able to erase the content on the screen

4. PROJECT REPORT ORGANIZATION

This book contains six chapters. The first chapter contains motivation, problem statement and project objectives. The second chapter includes the Literature survey which includes existing work and limitations of existing work. The third chapter includes specifications, software and hardware requirements needed for the project. The fourth chapter contains UML diagrams, Technology Description and Proposed methods. The fifth chapter includes Implementation which contains the technologies used for developing the application and code snippets. The fifth chapter also contains test cases and screenshots of the applications. The sixth chapter investigates the future enhancements and conclusion of the project.

CHAPTER-2

LITERATURE SURVEY

1. EXISTING WORK

This method uses touch data from the camera to generate photos. The vision-based method places a strong emphasis on touch-captured images and draws attention to the most prominent and recognised feature. At the outset of the vision-based approach, colour belts were utilised. The standard colour that had to be applied to the fingers was the method's biggest drawback. Then, instead of using colourful ribbons, use your hands. This is a difficult difficulty because real-time performance necessitates a background, continuous lighting, personal frames, and a camera. Furthermore, such systems must be designed to meet certain criteria, such as accuracy and resilience.

Theoretical analysis is based on how humans perceive information about their surroundings, yet it is perhaps the most hardest to master. So far, several different strategies have been tried. The first step is to create a three-dimensional model of a human hand. The model is compared to hand images captured by one or more cameras, and the parameters related to the palm form and combined angles are calculated. The touch phase is then created using these parameters. The second method is to use the camera to snap a picture and extract particular traits, which are then used as input in the partition to divide the data.

ClayAIR: Its hand tracking solutions aim at higher performance, quicker implementation time and higher accuracy. It can predict 22 3D key point coordinates. Using regression algorithms trained on 1.4 million images including real and synthetic images. It is being used by some leading tech giants like Enovo, Nreal, Qualcomm to bring virtual reality to the digital world.

SOTA Hardware : Data Gloves: Data gloves are pure VR devices in the sense that it can detect activity of the joints and on the other hand the feedback enables the user to feel the virtual targets in a pseudo-physical sense. They are especially

famous in the VR field since they are highly accurate and the inference time is less. Additionally, they are a great way to collect data of hand-landmarks for machine learning models. But since photoelectric sensors and position trackers are costly, the production and maintenance of these gloves is also high.

Inertial Sensors : The Nintendo Wii was the commercial release of inertial sensors. They are composed of an actuator and a sensor which help to collect and obtain information about gestures. Built with an accelerometer and an infrared sensor, they can capture the user's wrist and arm gestures.

KCF : KCF algorithm or Kernelized correlation filter algorithm is mainly focused around creating large number of training examples by shifting the target area in a circular manner. It was widely used for object tracking and is the base of many recent tracking algorithms. Unfortunately the algorithm doesn't perform well in case of scale variations i.e changes in the size of target objects. Additionally it is not easy to train it for detecting multiple landmarks.

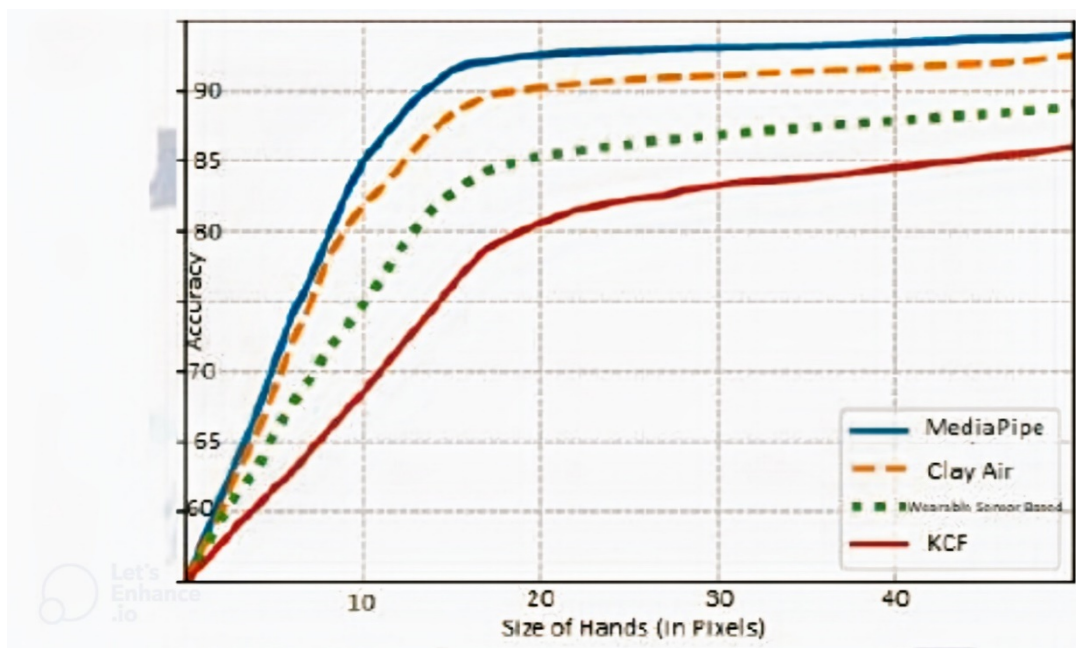


Figure 1 comparison of existing works

2. LIMITATIONS OF EXISTING WORK

Bluetooth and wireless technologies are becoming increasingly accessible, thanks to recent breakthroughs in virtual reality and their implementation in our daily lives. This research offers a visual AI system that uses hand motions and hand tip acquisition to perform mouse, keyboard, and stylus activities using computer vision. Instead of using typical headsets or external devices, the proposed system uses a web camera or built-in camera to track finger and hand movements to process the computer. This solution may be removed indefinitely due to its simplicity and effectiveness. the utilisation of additional hardware, battery durability, and, in the end, user convenience

The Python programming language and the OpenCV computer library were used to create the AI mouse programme. The proposed visual AI mouse system makes use of the Media Pipe package to track hands and titles, as well as the Pynput, Autopy, PyGames, and PyAutoGUI packages to browse the computer screen and perform actions like left-click, right-click, and scroll. The suggested model's findings demonstrate a very high level of accuracy, and the proposed model can operate extremely well in real-world applications with only a CPU and no GPU.

CHAPTER-3

SOFTWARE AND HARDWARE SPECIFICATIONS

1. SOFTWARE REQUIREMENTS






The software requirements we have used for the maor-project are:

- Python
- PyCharm IDE

The python modules used in the projects are:

- Open-CV
- NumPy
- Pandas
- Media pipe
- Time

1. Functional Requirements

-  Hand detector has the ability to detect the hands and fingers.
-  To draw various shapes using our fingers
-  To draw those shapes in different colors.
-  A canvas to display the drawings which can further be shared to others.
-  Only detect one hand to avoid any confusion.

Functional requirements for hand writing detection:

Purpose	This screen provides the canvas to draw various shapes and letters.
Inputs	People must have put their hands in front of the camera to be detected by the program.
Processing	If hand movements detection is successful and two fingers are lifted it is in selection mode or if only the index finger is lifted the drawing mode is triggered.
Outputs	On successful we get to draw on the canvas while guiding the finger.

Table-1: Functional requirements of hand Tracker.

Functional Requirements table:

ID - Functional Requirement	Description of Functional Requirements
FR-1	Hand movement detection
FR-2	Capturing hand movements and training model
FR-3	Storing images in database
FR-4	Gesture Recognition

FR-5	Drawing and selecting options
------	-------------------------------

Table-2: Functional requirements table.

2. Non-Functional Requirements

- Database will be handled by the Software.
- For normal conditions, 95% of the image processing should be processed in less than 2seconds.
- This application is very easy to run.
- It supports all types of cameras.

3.2 HARDWARE REQUIREMENTS

- 4 GB RAM (Minimum)
- 80 GB HDD
- Dual Core processor
- CDROM (installation only). VGA resolution monitor
- Microsoft Windows 98/2000/NT with service pack 6 / XP with service pack 2/Windows 11

CHAPTER-4

PROPOSED SYSTEM DESIGN

1. PROPOSED SYSTEM

The capacity to recognise the shape and motion of hands can help improve the user experience across a wide range of technological domains and platforms. It can, for example, provide as the foundation for comprehending sign language and controlling hand gestures, as well as enabling the overlay of digital content and information on top of the physical world in augmented reality. Because hands frequently occlude themselves or one other (e.g. finger/palm occlusions and hand shaking) and lack high contrast patterns, robust real-time hand perception is a difficult computer vision problem.

Artificial Intelligence (AI, a wide term describing a collection of advanced methodologies, tools, and algorithms for automating the execution of diverse tasks) has infiltrated virtually every corporate activity over the years. Hand tracking is one of the most prominent AI solutions; it is used to estimate the position and orientation of a person's hand given an image of them.

Your brain is programmed to accomplish all of this naturally and instantaneously as a human being. In fact, humans are overly adept at recognising faces, resulting in the appearance of faces in common items. Because computers are incapable of such high-level generalisation, we must teach them each step of the process separately. We need to create a pipeline in which each phase of face recognition is solved separately and the result of the current step is passed on to the next. To put it another way, we'll chain together a number of machine learning algorithms.

Because most solutions rely on key points and heat maps, we must first collect pose alignment data for each position. We can investigate several test cases in which the entire hand is displayed and key spots for the various hand portions can be detected. We can employ occlusion mimicking augmentation to ensure that the

hand tracker can perform in high occlusions, which are distinct test cases than typical ones. There are 30000 real-world photos in the training data set, each with 21 3D coordinates.

This may appear to be a random choice, but there's a good rationale for using gradients instead of pixels. If we look at pixels directly, we may see that very dark and very light photos of the same individual have completely distinct pixel values. However, if you simply evaluate the direction in which brightness varies, both extremely black and extremely bright images will have the same precise representation. This makes it a lot easier to fix the problem. However, keeping the gradient for each and every pixel gives us far too much information. We lose sight of the forest for the trees. It would be preferable if we could simply view the basic flow of lightness/darkness at a higher level in order to see the image's underlying pattern.

The histogram of oriented gradients (HOG), a feature descriptor commonly used in computer vision, is used to detect objects. It works by measuring the number of times a gradient orientation appears in a specific area of an image. Edge orientation histogram, scale invariant feature transform descriptor, and shape contexts are all methods that are similar.

In our photograph, we isolated the hand. But now we have to cope with the fact that a computer sees a hand turned in different directions in a completely different way. To compensate for this, we'll try to warp each image so that the fingers are always in the same location.

Following palm detection over the entire image, our next hand landmark model uses regression to accomplish exact keypoint localization of 21 3D hand-knuckle coordinates within the detected hand regions, i.e. direct coordinate prediction. Even with partially visible hands and self-occlusions, the model develops a consistent internal hand posture representation.



Graph 2 Top: Ground truth annotation on aligned hand crops provided to the tracking network. Bottom: Ground truth annotation on rendered synthetic hand images.

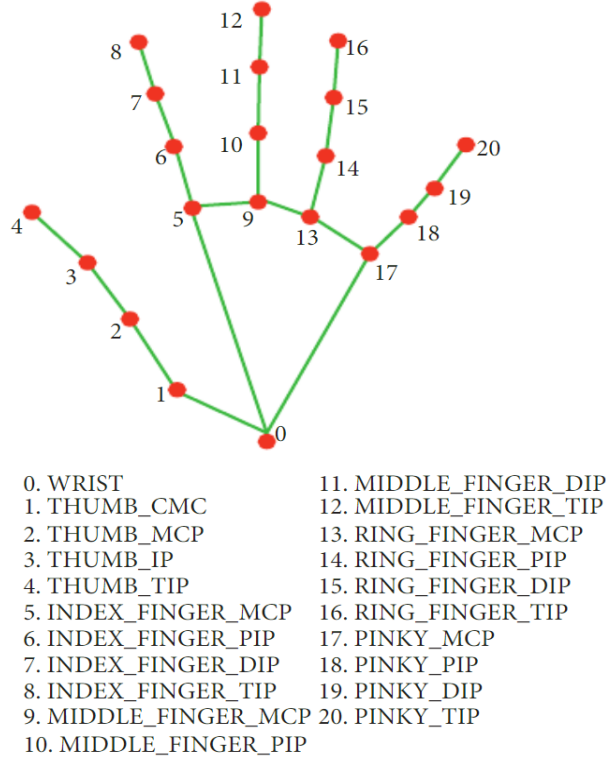


Figure 3 hand landmarks.

We used Open CV and Media Pipe, which is a library that uses machine learning techniques as well as other numerical and algorithmic tools.

The Media Pipe posture estimation tool employs a 21-point technique, in which it finds the important points and then guesses the pose based on the data collection. It uses the flame pose tool, which uses a Machine Learning technique to stance identification, to track the pose from a real-time camera frame or RGB video.

We use a single-shot detector model called BlazePalm, which is tailored for mobile real-time application in a similar way to BlazeFace, which is also accessible in MediaPipe, to detect initial hand placements. Hand detection is a difficult task: our model must detect occluded and self-occluded hands while working across a wide range of hand sizes with a significant scale span ($>20x$) relative to the image frame. Whereas faces contain strong contrast patterns, such as around the eyes and mouth, hands lack similar traits, making it more difficult to accurately distinguish them based on their visual features alone. Providing additional context, such as arm, body, or human traits, instead helps with precise

hand localisation.

Our approach employs a variety of ways to overcome the aforementioned issues. First, instead of training a hand detector, we train a palm detector because estimating bounding boxes of rigid objects like palms and fists is much easier than recognising hands with articulated fingers. Furthermore, because palms are smaller objects, the non-maximum suppression method performs effectively even in two-hand self-occlusion situations such as handshakes. Furthermore, palms can be simulated using square bounding boxes (anchors in ML language) that ignore other aspect ratios, resulting in a reduction of 3-5 anchors. Second, even for little objects, an encoder-decoder feature extractor is used for larger picture context awareness (similar to the RetinaNet approach). Finally, because to the high scale variance, we limit focus loss during training to support a large number of anchors.

We attain an average precision of 95.7 percent in palm detection using the strategies described above. With no decoder and a regular cross entropy loss, the baseline is just 86.22 percent.

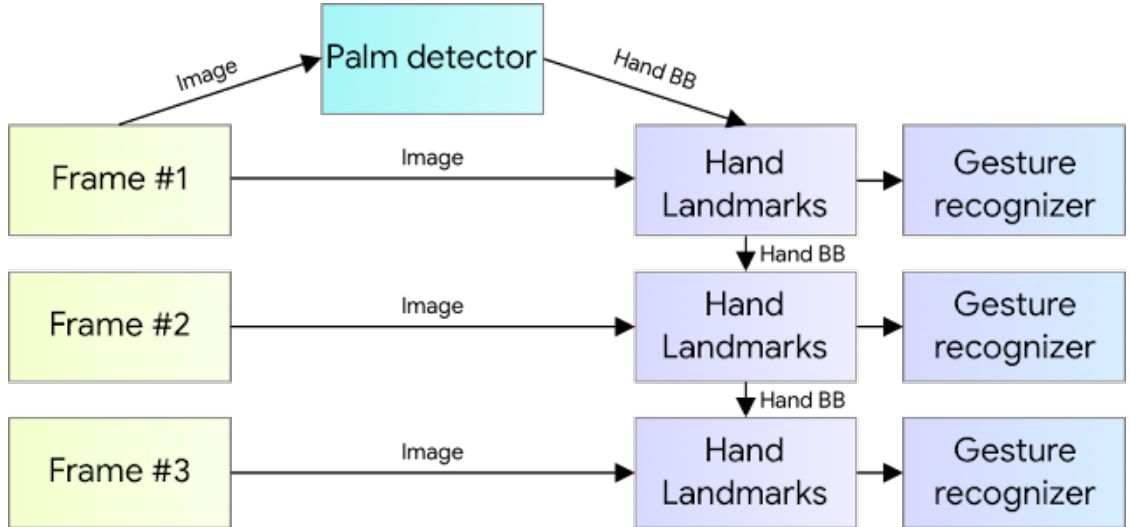


Figure 4 Hand perception pipeline overview.

1.1. PALM DETECTOR MODEL ARCHITECTURE:

Our hand tracking method makes use of a machine learning pipeline that consists of two models that work together: • A palm detector that uses an aligned hand bounding box to locate palms on a whole input image. • A hand landmark model that uses the palm detector's clipped hand bounding box to generate high-fidelity 2.5D landmarks. Providing the hand landmark model with a correctly cropped palm image greatly minimises the need for data augmentation. (e.g. rotations, translations, and scale) and allows the network to focus most of its resources on pinpointing landmarks with high accuracy. In a real-time tracking scenario, we use a bounding box derived from the previous frame's landmark prediction as input for the current frame, avoiding the need to apply the detector on every frame. Instead, the detector is only applied on the first frame or when the hand prediction indicates that the hand is lost.

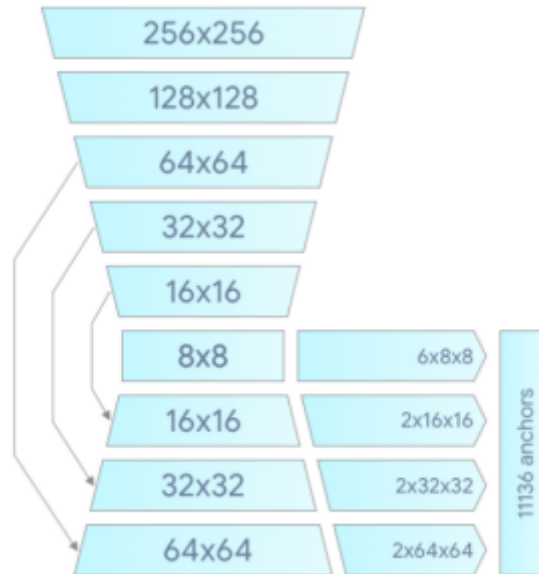


Figure 5 Palm detector model architecture

4.1.2 HAND LANDMARK MODEL

Following palm detection over the entire image, our hand landmark model uses regression to accomplish precise landmark localization of 21 2.5D coordinates within the detected hand regions. Even with partially visible hands and self-occlusions, the model develops a consistent internal hand posture representation. There are three outputs from the model (see Figure 3): 1. 21 hand landmarks with x , y , and relative depth.

2. A hand flag that indicates the likelihood of a hand appearing in the input image.

3. A classification of handedness that is either left or right.

For the 21 landmarks, we apply the same topology as [14]. As stated below, the 2D coordinates are learned from both real-world and synthetic datasets, with the relative depth w.r.t. the wrist point learned only from synthetic images. To recover from tracking failure, we created a new model output, similar to [8,] that generates the chance that a reasonably aligned hand is present in the provided crop. The detector is triggered to reset tracking if the score falls below a threshold. Another crucial element for good hand interaction in AR/VR is handedness. This is especially beneficial in some applications where each hand has its own set of functions. As a result, we created a binary classification head to determine whether the input hand is left or right. Our system is optimised for real-time mobile GPU inference, but we've also created lighter and heavier variants of the model to accommodate CPU inference on mobile devices without GPU support and higher accuracy needs for desktop use, respectively.

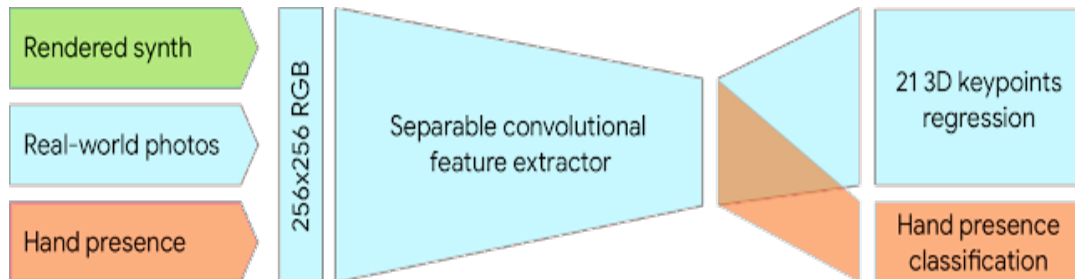


Figure 6 Hand landmark model

4.1.3 Dataset and Annotation:

We generated the following datasets to collect ground truth data, each addressing a distinct part of the problem:

- **The in-the-wild collection:** contains 6K photos with a wide range of characteristics, such as geographical diversity, lighting circumstances, and hand look. This dataset's drawback is that it lacks complicated hand articulation.
- **In-house gesture dataset:** This dataset contains 10K photos that cover all physically conceivable hand gestures from various angles. The dataset's shortcoming is that it was compiled from only 30 persons with little variety in their backgrounds. The in-the-wild and in-house datasets are excellent complements for improving robustness.
- **Synthetic dataset:** We render a high-quality synthetic hand model over diverse backgrounds and map it to the relevant 3D coordinates to better cover all potential hand poses and give additional depth supervision. We use a commercial 3D hand model with 24 bones and 36 blendshapes to regulate the thickness of the fingers and palm. The figure also comes with five different skin tones and textures. We made video sequences of hand positions transitioning and sampled 100K photos from the videos. Each position was rendered using three distinct cameras and a random high-dynamic-range lighting scenario.

We only utilise the in-the-wild dataset for the palm detector because it is sufficient for hand localization and has the most variability in appearance. All datasets, however, are used to train the hand landmark model. For synthetic images, we use projected groundtruth 3D joints and annotate realworld images with 21 landmarks. We sample on the region excluding annotated hand regions as negative instances and select a subset of real-world images as positive examples for hand presence. To offer such data for handedness, we annotate a subset of real-world photographs with handedness annotations.

2. CLASS DIAGRAM

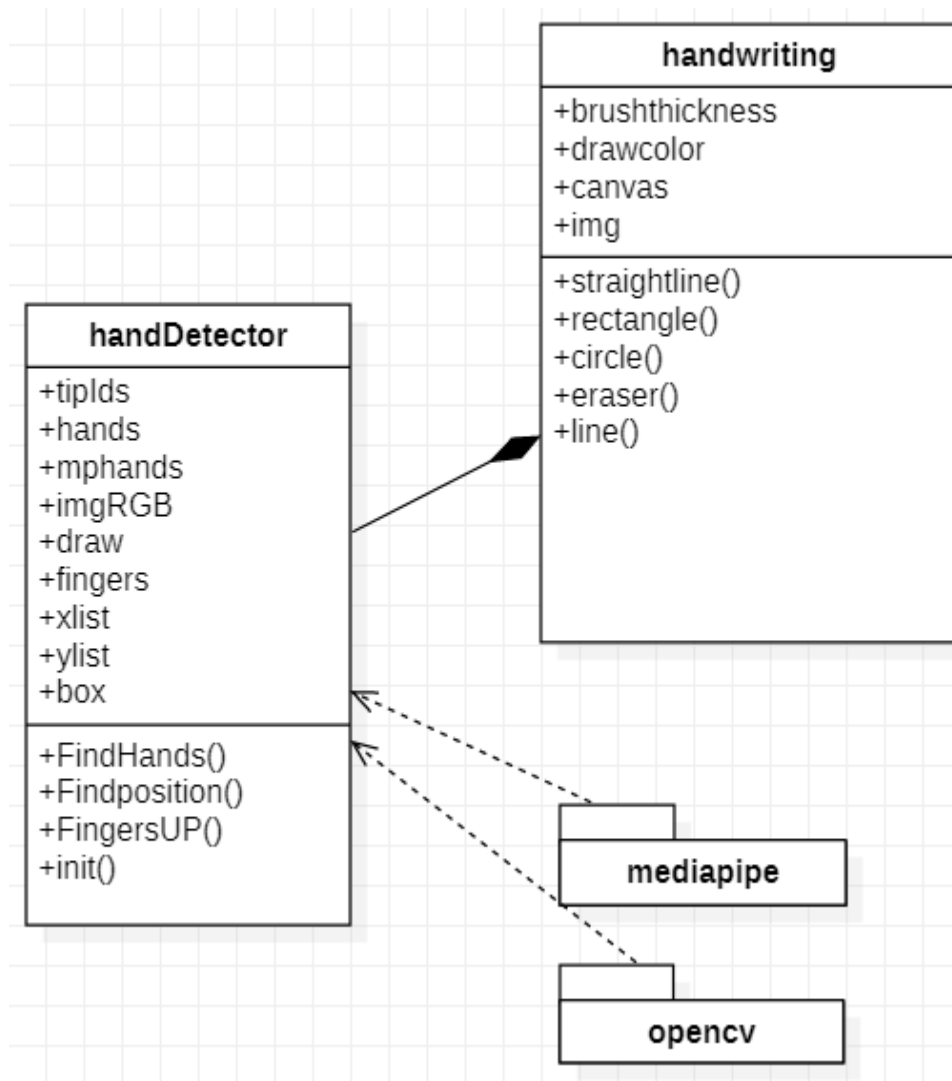


Figure 7 Class diagram

This class diagram shows two classes that correlate to **handDetector** and **handwriting**, respectively. There are a few characteristics and operations in each of these classes.

3. USE CASE DIAGRAM

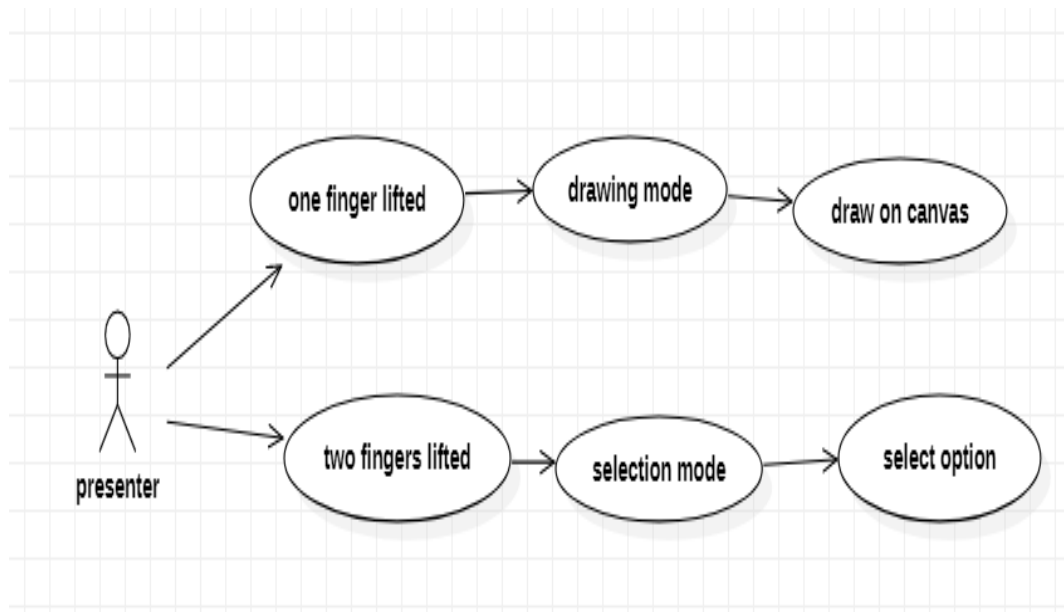


Figure 8 Use case diagram

The developed system's numerous operations are depicted in a UML use case diagram. Users can choose from a variety of settings and draw on the canvas.

4. ACTIVITY DIAGRAM

Activity Diagrams are used to depict the control flow in a system and to relate to the stages involved in executing a use case. Using activity diagrams, we model sequential and concurrent activities. As a result, we use an activity diagram to clearly depict workflows. The focus of an activity diagram is on the state of flow and the order in which it occurs. An activity diagram is used to describe or represent what causes a specific occurrence. Structure diagrams, interaction diagrams, and behaviour diagrams are the three types of diagrams that UML models. A behavioural diagram is an activity diagram. i.e. it depicts a system's behaviour. The control flow from a start point to a finish point is depicted in an activity diagram, which shows the numerous decision routes that exist while the activity is being performed. An activity diagram can be used to show both sequential and concurrent processing of activities. They are commonly used in business and process modelling to represent the dynamic features of a system. A flowchart and an activity diagram are extremely similar.

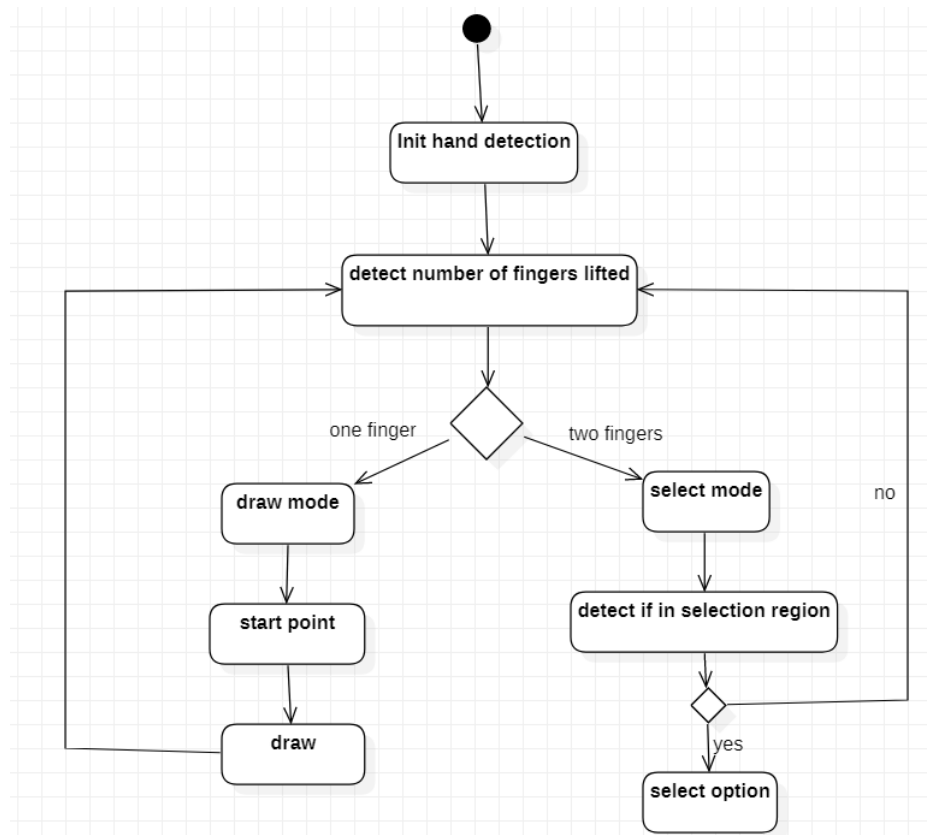


Figure 9 Activity diagram

5. SEQUENCE DIAGRAM

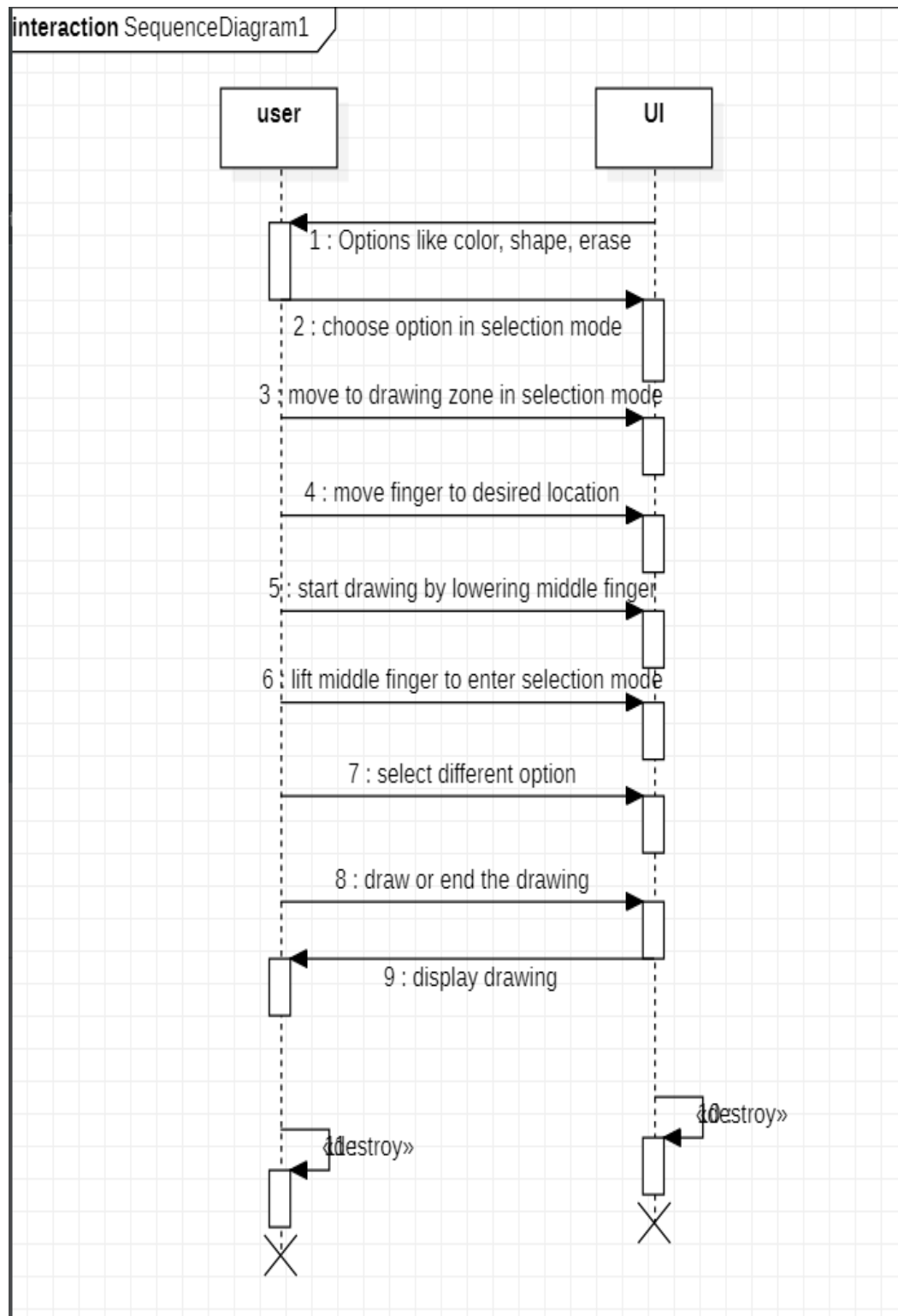


Figure 10 Sequence Diagram.

6. SYSTEM ARCHITECTURE

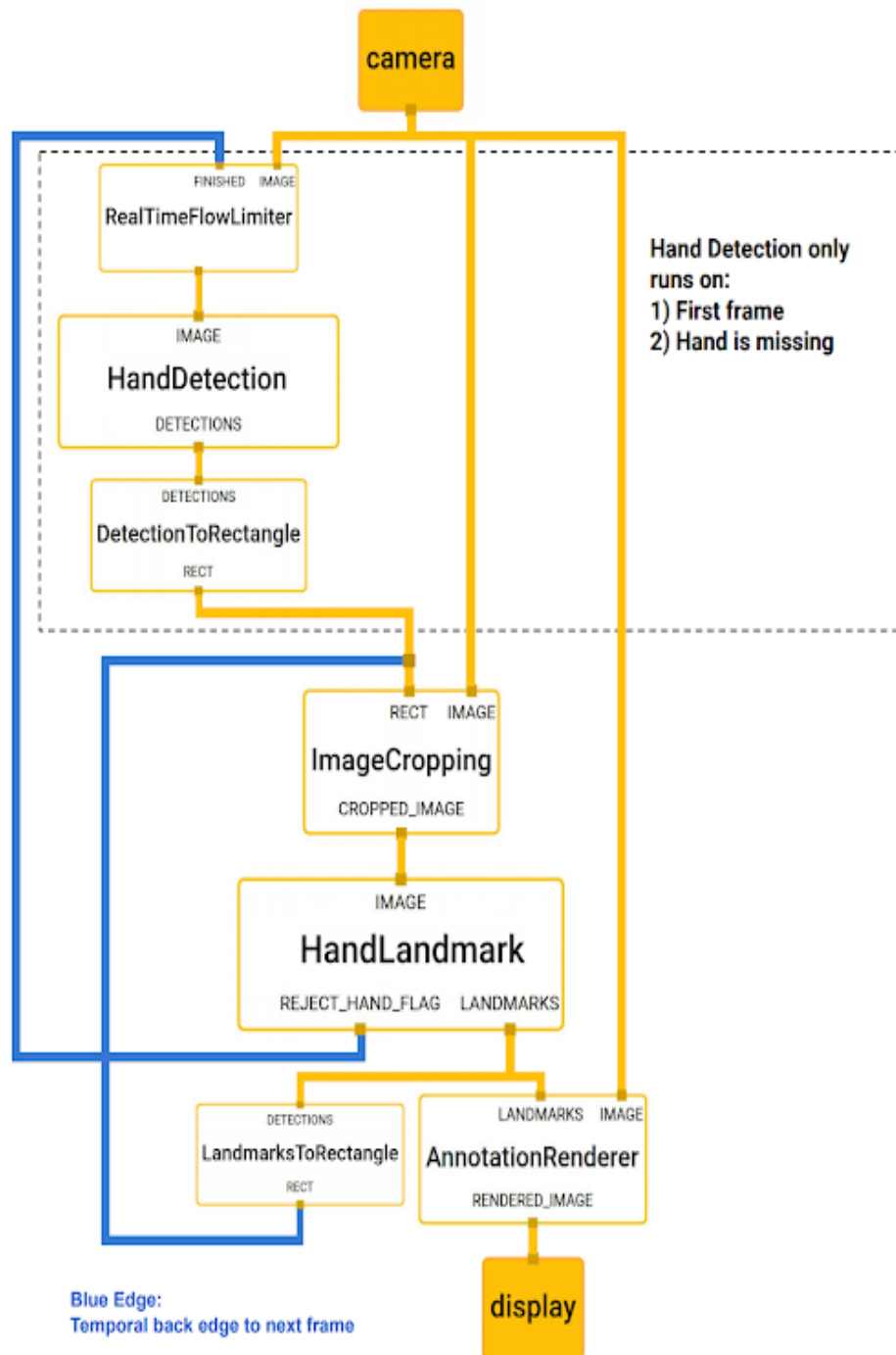


Figure 11 System Architecture

7. TECHNOLOGY DESCRIPTION

The software technologies we have used for the mini-project include:

Python:

Python is our project's artificial language, as well as one of the most powerful and well-known programming languages, well-known for its widespread application in machine learning and computers. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics and a simple, easy-to-learn syntax that prioritises readability and hence lowers programme maintenance costs. Modules and packages are supported by Python, which facilitates programme modularity and code reuse. The Python interpreter and its substantial standard library are free to download and distribute in source or binary form for all major platforms. Despite the fact that scripting and automation account for the majority of Python's use cases (more on that later), Python is also used to create professional-grade software, both as standalone apps and as web services. Python isn't the quickest programming language, but it makes up for it in variety.

NumPy:

Numpy is a Python library that adds support for multi-dimensional arrays and matrices, as well as a large number of high-level mathematical functions to operate on them. NumPy's "ndarray" data structure, which stands for "n-dimensional array," lies at the heart of its capabilities. These arrays are memory strided views. These arrays are homogeneously typed, unlike Python's built-in list data structure, which requires that all items of a single array be of the same type. Without the need to copy data around, such arrays can be put into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter, allowing for interoperability with current numerical libraries.

PyCharm:

PyCharm is a Python Integrated Development Environment (IDE) that includes a variety of key tools for Python developers that are tightly integrated to create a pleasant environment for effective Python, web, and data science development.

Pandas:

Pandas is mostly used to analyse data. Pandas supports data import from a variety of file formats, including comma-separated values, JSON, SQL, and Microsoft Excel. Pandas supports a wide range of data manipulation operations, including merging, reshaping, and selecting, as well as data cleaning and wrangling. It's an open-source data analysis and manipulation tool built on top of the Python programming language that's quick, powerful, versatile, and simple to use.

Datetime:

It's a mix of date and time, as well as the year, month, day, hour, minute, second, microsecond, and information attributes. Depending on whether or not they include time zone information, date and time objects can be classified as "aware" or "naive." An aware object can locate itself relative to other aware objects if it has sufficient knowledge of appropriate algorithmic and political time modifications, such as time zone and daylight-saving time information. An aware object denotes a specific point in time that cannot be interpreted in any way.

A naive object lacks sufficient information to locate itself unambiguously in relation to other date/time objects. It is entirely up to the computer whether a naive object represents Coordinated Universal Time (UTC), local time, or time in another time zone, just as it is entirely up to the programme whether a certain integer represents metres, miles, or mass. While naive objects are simple to comprehend and deal with, they do so at the expense of neglecting some features of reality.

OpenCV:

OpenCV is a programming package focused mostly toward real-time computer vision. It was created by Intel and then sponsored by Willow Garage and Itseez (which was later acquired by Intel). Under the open-source Apache 2 License, the library is cross-platform and free to use. OpenCV now has GPU acceleration for real-time operations since 2011.

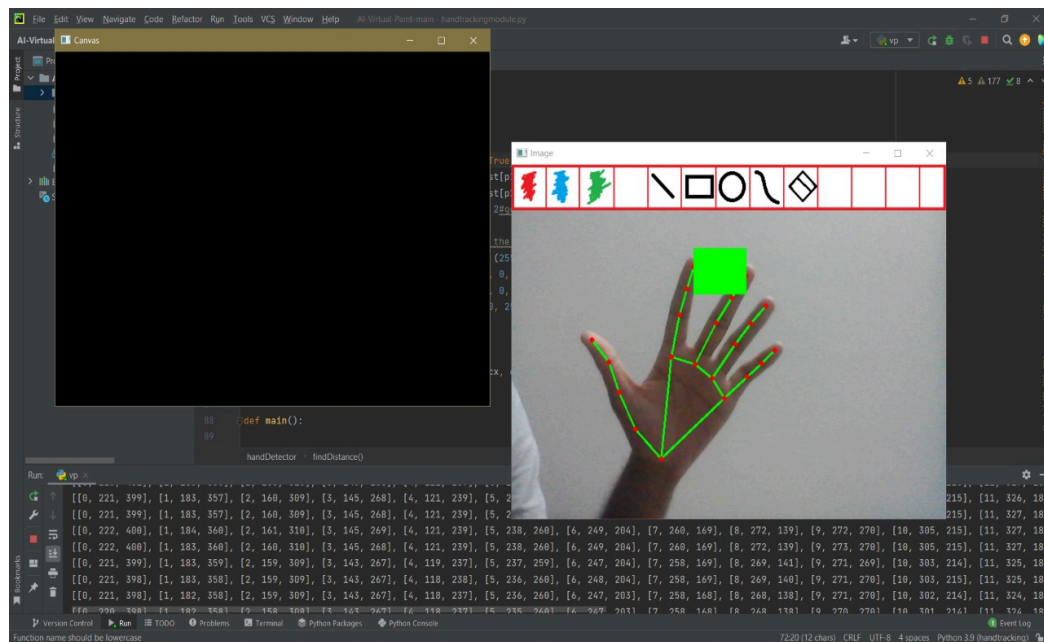
Media-pipe:

Media pipe is a framework that is primarily used to create multimodal audio, video, and time series data. The Media Pipe framework may be used to create an outstanding ML pipeline for inference models such as Tensor Flow and TFLite, as well as media processing routines.

CHAPTER-5

IMPLEMENTATION AND TESTING

1. SCREENSHOTS



1.2. Draw rectangle

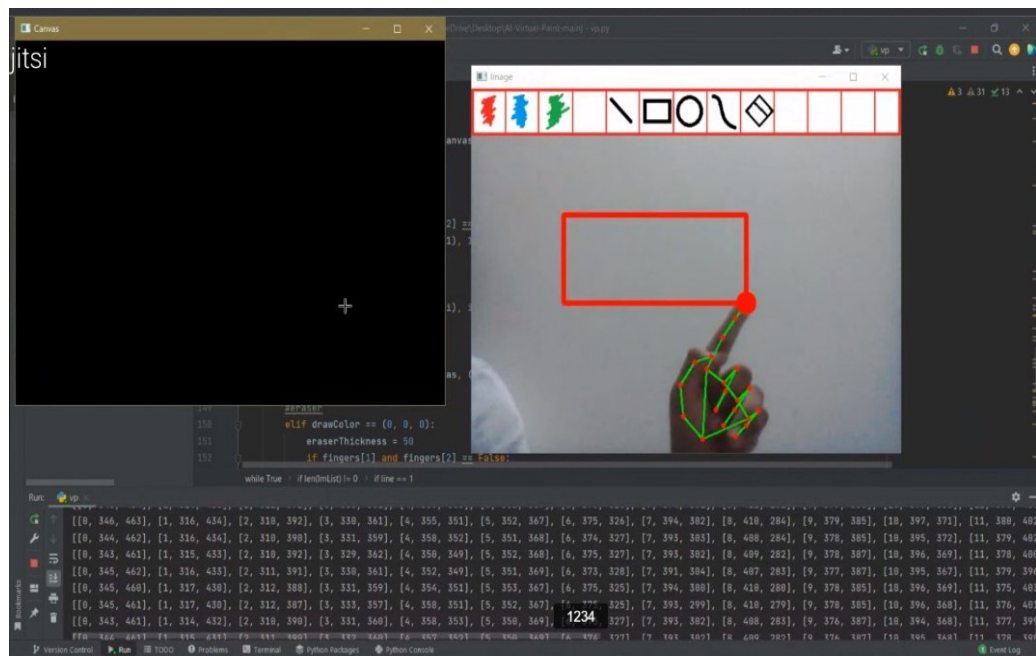


Figure 13 Drawing rectangle

1.3. Draw circle

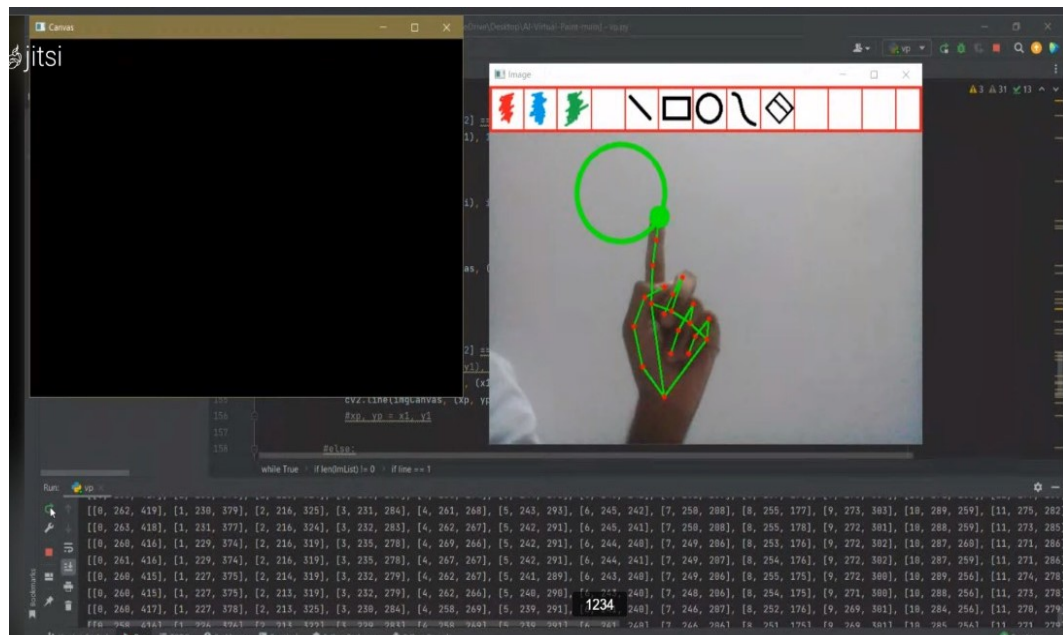


Figure 14 Drawing circle

2. TESTING

Testing the functional requirements defined in software requirements specification

Test Case-ID	Test Case Description
TC-1	Web camera detecting hand live
TC-2	Selecting different modes
TC-3	Drawing in different colors
TC-4	Drawing different shapes
TC-5	Erasing the existing drawings
TC-6	Introducing another hand while drawing with one

Table-3: Test Cases.

3. RESULT

Human hand detection from video is important in a variety of applications, including quantitative sign language recognition and detecting handwriting. It can be used to support sign language translation, gesture recognition, and gesture control, for example. In augmented reality, it can also enable the overlay of digital content and information on top of the physical world. Media Pipe Pose is a machine learning solution for high-fidelity hand pose tracking that uses our Blazepalm research, which also drives the ML Kit hand Detection API, to infer 21 3D landmarks on the entire hand from RGB video frames. The Open-CV library has a built-in solution for interacting with a streaming device, capturing a video stream, and generating video frames. The Open-CV Video Capture library can be used to accomplish this. This library is capable of reading video frames and displaying them in a window. BGR format frames were retrieved from Open-CV. As a result,

we first convert it to RGB. We can use Media Pipe's hand on video frames to track hand posture once we get our video frames in RGB.

ADVANTAGES:

1. It is recommended that an artificial intelligence virtual system be used to run the PC mouse functions without using physical devices during the COVID-19 condition since it is not safe to use the devices by touching them owing to the danger of the virus spreading by contacting the devices.
2. The proposed model has a 95 percent accuracy, which is significantly greater than other proposed models for virtual systems, and it has a wide range of applications in a variety of sectors.
3. The technology can be used to control robots and automation systems directly without the use of external devices.
4. Hand motions can be used to draw 2D and 3D drawings utilising the AI virtual system.
5. This device allows you to play virtual reality and augmented reality games without the use of a wireless or cable mouse.

5. APPLICATION:

1. People who suffer from hand illnesses can utilise this technology to manage the mouse functionalities of a computer with ease.
2. The proposed system, such as HCI, could be used to control robots and other machines in the field of robotics.
3. In the fields of design and architecture, the suggested technology may be utilised for virtual prototyping and design in general.

CHAPTER-6

CONCLUSION AND FUTURE SCOPE

1. CONCLUSION

The visual AI mouse system's main purpose is to allow users to manage mouse cursor functions with a hand touch rather than by manipulating objects. Hand gestures and hand tips are detected and processed by the suggested system, which can be accessed via a webcam or a built-in camera.

We may deduce from the model's findings that the proposed artificial intelligence system has performed very well and is incredibly accurate when compared to current models, and that the model addresses the majority of the existing system's constraints.

Because it is so exact, the proposed model can also be used in real-world applications. It can, for example, be utilised to reduce the spread of COVID-19 while also obviating the need for portable equipment.

The model has some flaws, such as a slight loss of precision when using the right-click feature and the difficulty of selecting text by clicking and dragging. As a result, we'll work to overcome these limitations by developing a fingerprint acquisition process that will produce more accurate results in the future.

2. FUTURE ENHANCEMENTS

This project can be improvised by adding Neural Network for high accuracy.

REFERENCES

1. <https://arxiv.org/pdf/2006.10214.pdf>
2. https://www.researchgate.net/publication/342302340_MediaPipe_Hands_On-device_Real-time_Hand_Tracking
3. <https://kulinpatel.com/real-time-writing-with-fingers-on-web-camera-screen-opencv/>
4. https://www.researchgate.net/publication/357622313_Virtual_Control_Using_Hand_Tracking
5. Rao, A.K., Gordon, A.M., 2001. Contribution of tactile information to accuracy in pointing movements. *Exp. Brain Res.* 138, 438–445. <https://doi.org/10.1007/s002210100717>
6. Masurovsky, A., Chojecki, P., Runde, D., Lafci, M., Przewozny, D., Gaebler, M., 2020. Controller-Free Hand Tracking for Grab-and Place Tasks in Immersive Virtual Reality: Design Elements and Their Empirical Study. *Multimodal Technol. Interact.* 4, 91. <https://doi.org/10.3390/mti4040091>.
7. Lira, M., Egito, J.H., Dall'Agnol, P.A., Amodio, D.M., Gonçalves, Ó.F., Boggio, P.S., 2017. The influence of skin colour on the experience of ownership in the rubber hand illusion. *Sci. Rep.* 7, 15745. <https://doi.org/10.1038/s41598-017-16137-3>.
8. Danckert, J., Goodale, M.A., 2001. Superior performance for visually guided pointing in the lower visual field. *Exp. Brain Res.* 137, 303–308. <https://doi.org/10.1007/s002210000653>.
9. Carlton, B., 2021. HaptX Launches True-Contact Haptic Gloves For VR And Robotics. VRScout. URL <https://vrscout.com/news/haptx-truecontact-haptic-gloves-vr/> (accessed 3.10.21).
10. Brenton, H., Gillies, M., Ballin, D., Chatting, D., 2005. D.: The uncanny valley: does it exist, in: In: 19th British HCI Group Annual Conference: Workshop on Human-Animated Character Interaction.

APPENDIX 1

Source Code

Handtrackingmodule.py

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

class handDetector():
    def
__init__(self,mode=False,maxHands=2,detectionCon=0.5,trackCon=0.5):
#constructor
    self.mode=mode
    self.maxHands=maxHands
    self.detectionCon=detectionCon
    self.trackCon=trackCon
    self.mpHands=mp.solutions.hands#initializing hands module
for the instance

    self.hands=self.mpHands.Hands(self.mode,self.maxHands,self.detectionCon,self.trackCon) #object for Hands for a particular instance
    self.mpDraw=mp.solutions.drawing_utils#object for Drawing
    self.tipIds = [4, 8, 12, 16, 20]

    def findHands(self,img,draw=True):
        imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)#converting to
RGB bcoz hand recognition works only on RGB image
        self.results=self.hands.process(imgRGB)#processing the RGB
image
        if self.results.multi_hand_landmarks:# gives x,y,z of every
landmark or if no hand than NONE
            for handLms in self.results.multi_hand_landmarks:#each
hand landmarks in results
                if draw:

self.mpDraw.draw_landmarks(img,handLms,self.mpHands.HAND_CONNECTIONS)#joining points on our hand

            return img

    def findPosition(self,img,handNo=0,draw=True):
        xList=[]
        yList=[]
        bbox=[]
        self.lmlist=[]
        if self.results.multi_hand_landmarks:# gives x,y,z of every
landmark
            myHand=self.results.multi_hand_landmarks[handNo]#Gives
result for particular hand
            for id,lm in enumerate(myHand.landmark):#gives id and
lm(x,y,z)
                h,w,c=img.shape#getting h,w for converting decimals
x,y into pixels
                cx,cy=int(lm.x*w),int(lm.y*h)# pixels coordinates
for landmarks
                # print(id, cx, cy)
```



```

        xList.append(cx)
        yList.append(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):#checking which finger is open
        fingers = []#storing final result
        # Thumb < sign only when we use flip function to avoid
        mirror inversion else > sign
        if self.lmlist[self.tipIds[0]][1] >
self.lmlist[self.tipIds[0] - 1][1]:#checking x position of 4 is in
right to x position of 3
            fingers.append(1)
        else:
            fingers.append(0)

        # Fingers
        for id in range(1, 5):#checking tip point is below
        tippoint-2 (only in Y direction)
            if self.lmlist[self.tipIds[id]][2] <
self.lmlist[self.tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)

        # totalFingers = fingers.count(1)

        return fingers

def main():

    PTime=0# previous time
    CTime=0# current time
    cap=cv2.VideoCapture(0)
    detector=handDetector()

    while True:
        success,img=cap.read()#T or F,frame
        img =detector.findHands(img)
        lmlist,bbox= detector.findPosition(img)
        if len(lmlist)!=0:
            print(lmlist[4])

        CTime=time.time()#current time
        fps=1/(CTime-PTime)#FPS
        PTime=CTime#previous time is replaced by current time

        cv2.putText(img,str(int(fps)),
(10,70),cv2.FONT_HERSHEY_COMPLEX,3,(255,0,255),3)# showing Fps on
screen

```

```

        cv2.imshow("Image",img)#showing img not imgRGB
        cv2.waitKey(1)

if __name__=="__main__":
    main()

```

handwritingmodule.py

```

import cv2
import time
import handtrackingmodule as htm
import numpy as np
import os

overlayList = [] # list to store all the images

brushThickness = 5
#eraserThickness = 50
drawColor = (0, 0, 255) # setting purple color
line = 0
xp, yp = 0, 0
imgCanvas = np.zeros((480, 640, 3), np.uint8) # defining canvas

var_inits = False
mask = np.ones((480, 640))*255
mask = mask.astype('uint8')

# images in header folder
folderPath = "Header"
myList = os.listdir(folderPath) # getting all the images used in code
# print(myList)
for imPath in myList: # reading all the images from the folder
    image = cv2.imread(f'{folderPath}/{imPath}')
    overlayList.append(image) # inserting images one by one in the overlayList
header = overlayList[0] # storing 1st image
cap = cv2.VideoCapture(0)
cap.set(3, 640) # width
cap.set(4, 480) # height

detector = htm.handDetector(detectionCon=0.85, maxHands=1) # making object

while True:

    # 1. Import image
    success, img = cap.read()
    img = cv2.flip(img, 1) # for neglecting mirror inversion

    # 2. Find Hand Landmarks
    img = detector.findHands(img) # using functions fo connecting landmarks
    lmList, bbox = detector.findPosition(img, draw=False) # using function to find specific landmark position, draw false means no

```

```

circles on landmarks

    if len(lmList) != 0:
        print(lmList)
        x1, y1 = lmList[8][1], lmList[8][2] # tip of index finger
        x2, y2 = lmList[12][1], lmList[12][2] # tip of middle
finger

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

    # 4. If Selection Mode - Two finger are up
    if fingers[1] and fingers[2]:
        xp, yp = 0, 0
        # print("Selection Mode")
        # checking for click

        if y1 < 62:
            if 0 < x1 < 50:
                drawColor = (0, 0, 255)
            elif 50 < x1 < 100:
                drawColor = (255, 0, 200)
            elif 100 < x1 < 150:
                drawColor = (0, 255, 0)
            #elif 150 < x1 < 200:
            #line = 4
            elif 200 < x1 < 250: # straight line
                # header = overlayList[0]
                line = 2
            elif 250 < x1 < 300: # rectangle
                # header = overlayList[1]
                line = 3
            elif 300 < x1 < 350: # circle
                # header = overlayList[2]
                line = 4
            elif 350 < x1 < 400: # line
                # header = overlayList[3]
                line = 1
            elif 400 < x1 < 450: # eraser
                # header = overlayList[3]
                drawColor = (0, 0, 0)
            cv2.rectangle(img, (x1, y1 - 25), (x2, y2 + 25),
drawColor,
                                cv2.FILLED) # selection mode is
represented as rectangle

        # 5. If Drawing Mode - Index finger is up
        #line
        if line == 1:

            if fingers[1] and fingers[2] == False:
                cv2.circle(img, (x1, y1), 15, drawColor,
cv2.FILLED)
                cv2.line(mask, (xp, yp), (x1, y1), drawColor,
brushThickness)
                cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor,
brushThickness)
                xp, yp = x1, y1

            else:
                xp = x1

```

```

        yp = y1

        #straight line
        elif line == 2:

            if fingers[1] and fingers[2] == False:
                cv2.circle(img, (x1, y1), 15, drawColor,
cv2.FILLED)
                if not(var_inits):
                    xi, yi = x1, y1
                    var_inits = True
                cv2.line(img, (xi, yi), (x1, y1), drawColor,
brushThickness)

            else:
                if var_inits:
                    cv2.line(imgCanvas, (xi, yi), (x1, y1),
drawColor, brushThickness)
                    var_inits = False

        #rectangle
        elif line == 3:

            if fingers[1] and fingers[2] == False:
                cv2.circle(img, (x1, y1), 15, drawColor,
cv2.FILLED)
                if not(var_inits):
                    xi, yi = x1, y1
                    var_inits = True
                cv2.rectangle(img, (xi, yi), (x1, y1), drawColor,
brushThickness)

            else:
                if var_inits:
                    cv2.rectangle(imgCanvas, (xi, yi), (x1, y1),
drawColor, brushThickness)
                    var_inits = False

        #circle
        elif line == 4:
            if fingers[1] and fingers[2] == False:
                cv2.circle(img, (x1, y1), 15, drawColor,
cv2.FILLED)
                if not(var_inits):
                    xi, yi = x1, y1
                    var_inits = True
                cv2.circle(img, (xi, yi), int(((xi-x1)**2 + (yi-
y1)**2)**0.5), drawColor, brushThickness)

            else:
                if var_inits:
                    cv2.circle(imgCanvas, (xi, yi), int(((xi-x1)**2
+ (yi-y1)**2)**0.5), drawColor, brushThickness)
                    var_inits = False

        #eraser
        elif drawColor == (0, 0, 0):
            eraserThickness = 50
            if fingers[1] and fingers[2] == False:
                #cv2.circle(img, (x1, y1), 15, drawColor,
cv2.FILLED)
                cv2.line(img, (xp, yp), (x1, y1), drawColor,

```

```

eraserThickness)
        cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor,
eraserThickness)
        #xp, yp = x1, y1

    #else:
        #xp = x1
        #yp = y1

    # merging two windows into one imgcanvas and img

    # 1 converting img to gray
    imgGray = cv2.cvtColor(imgCanvas, cv2.COLOR_BGR2GRAY)

    # 2 converting into binary image and thn inverting
    _, imgInv = cv2.threshold(imgGray, 50, 255,
                                cv2.THRESH_BINARY_INV) # on canvas
all the region in which we drew is black and where it is black it
is cosidered as white,it will create a mask

    imgInv = cv2.cvtColor(imgInv,
                                cv2.COLOR_GRAY2BGR) # converting again
to gray bcoz we have to add in a RGB image i.e img

    # add original img with imgInv ,by doing this we get our
drawing only in black color
    img = cv2.bitwise_and(img, imgInv)

    # add img and imgcanvas,by doing this we get colors on img
    img = cv2.bitwise_or(img, imgCanvas)

    # setting the header image
    img[0:62, 0:640] = header # on our frame we are setting our
JPG image acc to H,W of jpg images

    cv2.imshow("Image", img)
    cv2.imshow("Canvas", imgCanvas)
    # cv2.imshow("Inv", imgInv)
    cv2.waitKey(

```