A

Project Report on

**Digital Handwriting Recognition using Hand Tracking by using Media pipe and OpenCV libraries**

Submitted to

The Department of Information Technology

In partial fulfilment of the academic requirements of

Jawaharlal Nehru Technological University

For

The award of the degree of

Bachelor of Technology

in

Information Technology

(2018-2022)

By

B Rahul Teja – 18311A12J8

Md Sohail – 18311A12M6

N S Siddharth – 18311A12N0

Under the Guidance of

Mr. M. Nagaraju

Associate Professor



Sreenidhi Institute of Science and Technology

Yamnampet, Ghatkesar, R.R. District, Hyderabad - 501301

Affiliated to

Jawaharlal Nehru Technology University

Hyderabad - 500085

Department of Information Technology

Sreenidhi Institute of Science and Technology



**CERTIFICATE**

This is to certify that this Project-1 report on “**Digital Handwriting Recognition using Hand Tracking by using Media pipe and OpenCV libraries**”, submitted by B Rahul Teja (18311A12J8), Md Sohail(18311A12M6), N S Siddharth (18311A12N0) in the year 2022 in partial fulfilment of the academic requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in InformationTechnology, is a bonafide work that has been carried out by them as part of their **Project during Fourth Year Second Semister**, under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Mr. M Nagaraju,

Associate Professor,

Department of IT,

Internal Guide

Dr. Sunil Bhutada

Prof & Head, Department of IT

Dr.M. Srinivas,

Assistant Professor,

Department of IT,

Project Coordinator

External Examiner

**DECLARATION**

We, **B Rahul Teja (18311A12J8), Md Sohail(18311A12M6), N S Siddharth (18311A12N0)** students of **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY, YAMNAMPET, GHATKESAR,** studying IV-year II-sem, **INFORMATION TECHNOLOGY** solemnly declare that the Project-1 work, titled **“Digital Handwriting Recognition using Hand Tracking by using Media pipe and OpenCV libraries”** is submitted to **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY** for partial fulfilment for the award of degree of Bachelor of technology in **INFORMATION TECHNOLOGY**.

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for award of any degree.

**B RAHUL TEJA 18311A12J8**

**MD SOHAIL 18311A12M6**

**N S SIDDHARTH 18311A12N0**

**ACKNOWLEDGEMENT**

I would like to express my gratitude to all the people behind the screen who helped me to transform an idea into a real application.

I would like to express my heart-felt gratitude to my parents without whom I would not have been privileged to achieve and fulfill my dreams. I am grateful to our principal, **Dr. T. Ch. Siva Redd**y, who most ably runs the institution and has had the major hand in enabling me to do my project.

I profoundly thank **Dr. Sunil Bhutada**, Head of the Department of Information Technology who has been an excellent guide and also a great source of inspiration to my work.

I would like to thank our Coordinator **Dr. M. Srinivas** & my internal guide **Mr. M Nagaraju** for Project-1 for their technical guidance, constant guidelines, encouragement and support in carrying out my project on time at college.

The satisfaction and euphoria that accompany the successful completion of the task would be great but incomplete without the mention of the people who made it possible with their constant guidance and encouragement crowns all the efforts with success. In this context, I would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

**B RAHUL TEJA 18311A12J8**

**MD SOHAIL 18311A12M6**

**N S SIDDHARTH 18311A12N0**

**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 Contents** | | | |
|  | | | 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 TITLE PAGE NO.

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

#### MOTIVATION

In recent times hand tracking and gesture tracking has become an integral part and it has opened up a variety of possibilities and challenges. There is rising interest in computer vision and therefore the fast development and improvement of accessible hardware which can support new developments in AI. In this article we outline, implement a handful of these possibilities and further summarize the challenges and future prospects in the spectrum of human user interaction and virtual reality. The motivation behind this paper is to reduce human interaction and dependency of devices to control the computer in view of COVID-19 spread. These results can drive further research and within the long run contribute to assist the use of virtual environments.

With the latest advances in virtual reality and its application in our daily lives, Bluetooth and wireless technology are becoming increasingly accessible. This paper proposes a visual AI system that uses computer vision to perform mouse, keyboard, and stylus functions using hand gestures and hand tip acquisition. Instead of using standard headsets or external devices, the suggested system traces the finger and hand movements to process the computer using a web camera or built-in camera. Because it is simple and effective, this solution may be continuously removed. the use of additional hardware, battery durability, and ultimately bring ease to the user.

#### 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.

#### PROJECT OBJECTIVES

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

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

#### 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**

#### EXISTING WORK

This method takes pictures on camera as touch data. The vision-based approach focuses heavily on touch-captured images and brings out the main and recognizable feature. Colour belts were used at the beginning of the vision-based approach. The main disadvantage of this method was the standard colour to be applied to the fingers. Then use bare hands instead of coloured ribbons. This creates a challenging problem as these systems require background, uninterrupted lighting, personal frames and a camera to achieve real-time performance. In addition, such systems must be developed to meet the requirements, including accuracy and robustness.

Theoretical analysis is based on how people perceive information about their environment, yet it is probably the most difficult to use effectively. Several different methods have been tested so far. The first is to build a three dimensional human hand model. The model is compared to hand images with one or more cameras, and the parameters corresponding to the shape of the palm and the combined angles are estimated. These parameters are then used to create the touch phase. The second is to take a picture using the camera and extract certain features and those features are used as input in the partition to separate

**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 or 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.

Chart

Description automatically generated

Figure 1 comparision of existing works

#### LIMITATIONS OF EXISTING WORK

With the latest advances in virtual reality and its application in our daily lives, Bluetooth and wireless technology are becoming increasingly accessible. This paper proposes a visual AI system that uses computer vision to perform mouse, keyboard, and stylus functions using hand gestures and hand tip acquisition. Instead of using standard headsets or external devices, the suggested system traces the finger and hand movements to process the computer using a web camera or built-in camera. Because it is simple and effective, this solution may be continuously removed. the use of additional hardware, battery durability, and ultimately bring ease to the user.

The AI mouse program is developed using the Python programming language, as well as OpenCV, a computer library. Models in the proposed visual AI mouse system use the Media Pipe package to track hands and title, as well as packages of Pynput, Autopy, PyGames, and PyAutoGUI to navigate the computer screen and perform tasks such as left-click, right-click, and then scroll. The findings of the proposed model show a very high level of accuracy, and the proposed model can work very well in real-world applications using only a CPU and no GPU.

CHAPTER-3

# SOFTWARE AND HARDWARE SPECIFICATIONS

### 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

#### 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 infront 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:

|  |  |
| --- | --- |
| Functional Requirement - ID | Functional Requirements Description |
| 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.

#### Non-Functional Requirements

* + - * Database will be handled by the Software.
      * For normal conditions, 95% of the image processing should be processed in less than 2 seconds.
      * 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

### PROPOSED SYSTEM

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and hand shakes) and lack high contrast patterns.

Artificial Intelligence (Al, a broad name for a group of advanced methods, tools, and algorithms for automatic execution of various tasks) has invaded practically all functional areas of business over the years. Hand tracking is among the most popular solutions that AI has to offer; it is used to determine the position and orientation of the hand given an image containing a person.

As a human, your brain is wired to do all of this automatically and instantly. In fact, humans are too good at recognizing faces and end up seeing faces in everyday objects. Computers are not capable of this kind of high-level generalization so we have to teach them how to do each step in this process separately. We need to build a pipeline where we solve each step of face recognition separately and pass the result of the current step to the next step. In other words, we will chain together several machine learning algorithms.

As most of the solutions use the key points and the heat maps, first we require to pose alignment data for each position. We can take into consideration the different test cases where if the complete hand is visible and there are detectable key points for the hand parts. To make sure that the hand tracker can perform in heavy occlusions which are some different test cases than normal ones, we can make use of occlusion simulating augmentation. The training data set has 30000 real-world images with 21 3D coordinates .

This might seem like a random thing to do, but there’s a really good reason for replacing the pixels with gradients. If we analyse pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve. But saving the gradient for every single pixel gives us way too much detail. We end up [missing the forest for the trees.](https://en.wiktionary.org/wiki/see_the_forest_for_the_trees) It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

Object detection is accomplished using histogram of oriented gradients (HOG) which is a feature descriptor widely used in computer vision. It is based on counting the occurrences of gradient orientation in localized portions of an image. This method has similarity with edge orientation histogram, scale invariant feature transform descriptor, shape contexts.

We isolated the hand in our image. But now we have to deal with the problem that hand turned different directions look totally different to a computer. To account for this, we will try to war p each picture so that the fingers are always in the sample place in the image.

After the palm detection over the whole image our subsequent hand landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

A picture containing text, indoor, different, plant

Description automatically generated

Figure 2 Top: Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation.

Chart

Description automatically generated

Figure 3 hand landmarks.

We have used different libraries such as Open CV and Media Pipe which is a library using ML algorithms along with different numerical and algorithms.

The Media Pipe pose estimation tool uses a 21 key point’s approach wherein it detects the key points and accordingly uses and studying the data set estimates the pose. It tracks the pose from the real-time camera frame or RGB video by using the blaze pose tool that has a Machine Learning approach in pose detection.

To detect initial hand locations, we employ a single-shot detector model called BlazePalm, optimized for mobile real-time uses in a manner similar to BlazeFace, which is also available in MediaPipe. Detecting hands is a decidedly complex task: our model has to work across a variety of hand sizes with a large scale span (~20x) relative to the image frame and be able to detect occluded and self-occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

Our solution addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance.

With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 4 Hand perception pipeline overview.

#### PALM DETECTOR MODEL ARCHITECTURE:

Our hand tracking solution utilizes an ML pipeline consisting of two models working together: • A palm detector that operates on a full input image and locates palms via an oriented hand bounding box. • A hand landmark model that operates on the cropped hand bounding box provided by the palm detector and returns high-fidelity 2.5D landmarks. Providing the accurately cropped palm image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and allows the network to dedicate most of its capacity towards landmark localization accuracy. In a real-time tracking scenario, we derive a bounding box from the landmark prediction of the previous frame as input for the current frame, thus avoiding applying 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.

`Chart

Description automatically generated

Figure 5 Palm detector model architecture

#### 4.1.2 HAND LANDMARK MODEL

After running palm detection over the whole image, our subsequent hand landmark model performs precise landmark localization of 21 2.5D coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The model has three outputs (see Figure 3): 1. 21 hand landmarks consisting of x, y, and relative depth.

2. A hand flag indicating the probability of hand presence in the input image.

3. A binary classification of handedness, eg. left or right hand.

We use the same topology as [14] for the 21 landmarks. The 2D coordinates are learned from both real-world images as well as synthetic datasets as discussed below, with the relative depth w.r.t. the wrist point being learned only from synthetic images. To recover from tracking failure, we developed another output of the model similar to [8] for producing the probability of the event that a reasonably aligned hand is indeed present in the provided crop. If the score is lower than a threshold then the detector is triggered to reset tracking. Handedness is another important attribute for effective interaction using hands in AR/VR. This is especially useful for some applications where each hand is associated with a unique functionality. Thus we developed a binary classification head to predict whether the input hand is the left or right hand. Our setup targets real-time mobile GPU inference, but we have also designed lighter and heavier versions of the model to address CPU inference on the mobile devices lacking proper GPU support and higher accuracy requirements of accuracy to run on desktop, respectively.

Text, letter

Description automatically generated

Figure 6 Hand landmark model

#### 4.1.3 Dataset and Annotation:

To obtain ground truth data, we created the following datasets addressing different aspects of the problem:

• **In-the-wild dataset**: This dataset contains 6K images of large variety, e.g. geographical diversity, various lighting conditions and hand appearance. The limitation of this dataset is that it doesn’t contain complex articulation of hands.

• **In-house collected gesture dataset**: This dataset contains 10K images that cover various angles of all physically possible hand gestures. The limitation of this dataset is that it’s collected from only 30 people with limited variation in background. The in-the-wild and in-house dataset are great complements to each other to improve robustness.

• **Synthetic dataset**: To even better cover the possible hand poses and provide additional supervision for depth, we render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates. We use a commercial 3D hand model that is rigged with 24 bones and includes 36 blendshapes, which control fingers and palm thickness. The model also provides 5 textures with different skin tones. We created video sequences of transformation between hand poses and sampled 100K images from the videos. We rendered each pose with a random high-dynamic-range lighting environment and three different cameras.

For the palm detector, we only use in-the-wild dataset, which is sufficient for localizing hands and offers the highest variety in appearance. However, all datasets are used for training the hand landmark model. We annotate the realworld images with 21 landmarks and use projected groundtruth 3D joints for synthetic images. For hand presence, we select a subset of real-world images as positive examples and sample on the region excluding annotated hand regions as negative examples. For handedness, we annotate a subset of real-world images with handedness to provide such data.

* 1. **CLASS DIAGRAM**

Diagram

Description automatically generated

Figure 7 Class diagram

This class diagram contains two classes which corresponds to handDetector, handwriting. All these classes contain a few attributes and operations.

#### USE CASE DIAGRAM

Diagram

Description automatically generated

Figure 8 Use case diagram

UML use case diagram outlining the various operations allowed by the developed system. Users are allowed to select various options and draw on the canvas.

#### ACTIVITY DIAGRAM

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram. UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behaviour diagrams. An activity diagram is a **behavioural diagram** i.e. it depicts the behaviour of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system. An activity diagram is very **similar to a flowchart**.

Diagram

Description automatically generated

Figure 9 Activity diagram

#### SEQUENCE DIAGRAM

A picture containing diagram

Description automatically generated

Figure 10 Sequence Diagram.

#### SYSTEM ARCHITECTURE

Diagram

Description automatically generated

Figure 11 System Architecture

#### TECHNOLOGY DESCRIPTION

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

#### Python:

Python is the artificial language for our project, and additionally one in every of the extremely powerful and renowned programming languages better-known for its large use in machine learning and computing. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics and simple, easy to learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed. Even though scripting and automation cover a large chunk of Python’s use cases (more on that later), Python is also used to build professionalquality software, both as standalone applications and as web services. Python may not be the fastest language, but what it lacks in speed, it makes up for in versatility.

#### NumPy:

Numpy is a library for Python, adding support for multi-dimensional arrays and matrices, in conjunction with an enormous assortment of high-level mathematical functions to operate on these arrays. The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure, these arrays are homogeneously typed: all elements of a single array must be of the same type. Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries.

**PyCharm:**

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

**Pandas:**

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft

Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. It is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

**Datetime:**

It’s a combination of date and time along with the attribute’s year, month, day, hour, minute, second, microsecond, and info. Date and time objects may be categorized as “aware” or “naive” depending on whether or not they include time zone information. With sufficient knowledge of applicable algorithmic and political time adjustments, such as time zone and daylight-saving time information, an aware object can locate itself relative to other aware objects. An aware object represents a specific moment in time that is not open to interpretation.

A naive object does not contain enough information to unambiguously locate itself relative to other date/time objects. Whether a naive object represents Coordinated Universal Time (UTC), local time, or time in some other time zone is purely up to the program, just like it is up to the program whether a particular number represents meter’s, miles, or mass. Naive objects are easy to understand and to work with, at the cost of ignoring some aspects of reality.

**OpenCV:**

OpenCV is a library of programming functions primarily geared toward real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source Apache 2 License. Starting with 2011, OpenCV features GPU acceleration for real-time operations.

#### Media-pipe:

Media pipe is a framework mainly used for building multimodal audio, video, or any time series data. With the help of the Media Pipe framework, an impressive ML pipeline can be built for instance of inference models like Tensor Flow, TFLite, and also for media processing functions.

CHAPTER-5

**IMPLEMENTATION AND TESTING**

* 1. **SCREENSHOTS**

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12 Hand detection

Fig 10: Detection of hand

#### Draw rectangle

![A screenshot of a computer

Description automatically generated with medium confidence]()

Figure 13 Drawing rectangle

#### Draw circle

![A screenshot of a computer

Description automatically generated with medium confidence]()

Figure 14 Drawing circle

#### 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.

#### RESULT

Human hand detection from video plays a critical role in various applications such as quantifying sign language recognition, and hand writing detection. For example, it can form the basis for sign language translation, gesture recognition, and gesture control. It can also enable the overlay of digital content and information on top of the physical world in augmented reality. Media Pipe Pose is a ML solution for high-fidelity hand pose tracking, inferring 21 3D landmarks on the whole hand from RGB video frames utilizing our Blazepalm research that also powers the ML Kit hand Detection API. Open-CV library provides a built-in solution to engage a streaming device, capture a video stream, and provide video frames. We can use this by calling the Open-CV Video Capture library. This library can read frames of video and display them in a window. The frames extracted from Open-CV are BGR format. So, we first convert it to RGB format. Once we have our video frames in RGB, we can apply Media Pipe’s hand on video frames to track hand posture.

#### ADVANTAGES:

1. Since it is not safe to use the devices by touching them during the COVID-19 condition due to the possibility of the virus spreading by touching the devices, it is recommended that an artificial intelligence virtual system be used to operate the PC mouse functions without utilising physical devices.
2. The suggested model has a better accuracy of 95 percent, which is much higher than the accuracy of other proposed models for virtual systems, and it has a wide range of applications in many fields.
3. With no need for external devices, the system may be utilised to control robots and automation systems directly.
4. Drawing 2D and 3D pictures using the AI virtual system is possible via the use of hand motions
5. Playing virtual reality and augmented reality games without the usage of a wireless or wired mouse device is possible with this gadget.

#### APPLICATION:

1. People who suffer from hand illnesses can utilise this technology to manage the mouse functionalities of a computer with ease.
2. In the realm of robotics, the suggested system, such as HCI, may be utilised to control robots and other machinery.
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

#### CONCLUSION

The primary goal of the visual AI mouse system is to allow users to control mouse cursor functions with a hand touch rather than by manipulating things. Hand gestures and hand tips are detected and processed by the proposed system, which may be accessible by a webcam or a built-in camera that can be used to access the system.

Based on the findings of the model, we can infer that the suggested artificial intelligence system has done extremely well and is extremely accurate when compared to current models, and that the model addresses the majority of the constraints of the existing system.

Being very precise, the suggested model may be utilised for real-world applications as well. For example, it can be used to decrease the spread of COVID-19 and can eliminate the need for portable equipment.

Some drawbacks of the model include a modest loss in the precision of the right-click function and the difficulty of selecting text by clicking and dragging. As a result, we will seek to address these restrictions by designing a fingerprint acquisition method that will yield more accurate findings in the future.

#### FUTURE ENHANCEMENTS

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

## REFERENCES

* + 1. [https://arxiv.org/pdf/2006.10214.pdf](https://arxiv.org/pdf/2006.10214.pdf%2520)
    2. [https://www.researchgate.net/publication/342302340\_MediaPipe\_Hands\_On-device\_Real-time\_Hand\_Tracking](https://www.researchgate.net/publication/342302340_MediaPipe_Hands_On-device_Real-time_Hand_Tracking%2520)
    3. [https://kulinpatel.com/real-time-writing-with-fingers-on-web-camera-screen-opencv/](https://kulinpatel.com/real-time-writing-with-fingers-on-web-camera-screen-opencv/%2520)
    4. [https://www.researchgate.net/publication/357622313\_Virtual\_Control\_Using\_Hand\_-Tracking](https://www.researchgate.net/publication/357622313_Virtual_Control_Using_Hand_-Tracking%2520)
    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](https://doi.org/10.1038/s41598-017-%252016137-3).
    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).](https://vrscout.com/news/haptx-truecontact-haptic-gloves-vr/%2520(accessed%25203.10.21).)
    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(