

A VISION BASE APPLICATION FOR VIRTUAL MOUSE INTERFACE USING HAND GESTURE

Project Submitted in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Technology in the field of Computer Science and Engineering

BY

Sourav Sahoo (123180703089)

Sayan Ghosh(123180703078)

Indrani Naskar(123180703037)

Sankha Sarkar (380117011033)

Under the supervision
of
Sumanta Chatterjee



Department of Computer Science and Engineering
JIS College of Engineering

Block-A, Phase-III, Kalyani, Nadia, Pin-741235
West Bengal, India
May, 2022



JIS College of Engineering

Block 'A', Phase-III, Kalyani, Nadia, 741235
Phone: +91 33 2582 2137, Telefax: +91 33 2582 2138
Website: www.jiscollege.ac.in, Email: info@jiscollege.ac.in

CERTIFICATE

This is to certify that **Sourav Sahoo(123180703089),Sayan Ghosh(123180703078),Indrani Naskar(123180703037) ,Sankha sarkar(380117011033)** has completed his/her project entitled A VISION BASE APPLICATION FOR VIRTUAL MOUSE INTERFACE USING HAND GESTURE, under the guidance of **Sumanta Chatterjee** in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Science and Engineering** from JIS college of Engineering (An Autonomous Institute) is an authentic record of their own work carried out during the academic year 2021-22 and to the best of our knowledge, this work has not been submitted elsewhere as part of the process of obtaining a degree, diploma, fellowship or any other similar title.

Signature of the Supervisor

Signature of the HOD

Signature of the Principal

Signature of the External Expert

Place:

Date:

ACKNOWLEDGEMENTS

The analysis of the project work wishes to express our gratitude to Sumanta Chatterjee for allowing the degree attitude and providing effective guidance in development of this project work. His conscription of the topic and all the helpful hints, he provided, contributed greatly to successful development of this work, without being pedagogic and overbearing influence.

We also express our sincere gratitude to Dr. Dharmpal Singh, Head of the Department of Computer Science and Engineering of JIS College of Engineering and all the respected faculty members of Department of CSE for giving the scope of successfully carrying out the project work.

Finally, we take this opportunity to thank to Prof. **(Dr.) Partha Sarkar**, Principal of JIS College of Engineering for giving us the scope of carrying out the project work.

.....
Sayan Ghosh
B.TECH in Computer Science and Engineering
4thYEAR/8th SEMESTER
Univ Roll--123180703078

.....
Sourav Shao
B.TECH in Computer Science and Engineering
4thYEAR/8th SEMESTER
Univ Roll--123180703089

.....
Indrani Naskar
B.TECH in Computer Science and Engineering
4thYEAR/8th SEMESTER
Univ Roll--123180703033

.....
Sankha Sarkar
B.TECH in Computer Science and Engineering
4thYEAR/8th SEMESTER
Univ Roll--38011701103

TABLE OF CONTENT

Title Page	1
Certificate	2
Acknowledgement	3
List of Figures & Tables	4
Abstract	7
1.Introduction	8
2.literature Survey	10
3.Methodology	25
4.Result and Discussion	35
5.Conclusion	41
6.Limitation	42
7.Futurework	42
8. Application	43
9.Reference	44
Publication from the work	45

LIST OF FIGURES

Fig. No	Description
1	Mechanical mouse, with top cover removed
2	Optical Mouse, with top cover removed
3	MediaPipe hand recognition graph
4	Co-ordinates or land marks in the hand
5	The Flow Chart of Portable Vision-Based Human Computer Interaction
6	Agile method overview
7	Gesture for the computer to perform left button click.
8	Gesture for the computer to perform right button click.
9	Gesture for the computer to perform scroll up function.
10	Virtual mouse stabilization modulated by placing the actual cursor plane
11	Graph of Accuracy for different performed actions
12	Accuracy graph for different available mouse system

LIST OF TABLES

Fig. No	Description
1	Advantage and disadvantage of the Mechanical Mouse
2	Advantage and disadvantage of the Optical and Laser Mouse
3	Specification of pycharm
4	Gesture for specific mouse functions
5	Verification Plan P1
6	Different brightness testing results
7	Verification Plan P2
8	Verification Plan P3
9	Distance testing results
10	Experimental results & accuracy

ABSTRACT

This project promotes an approach for the Human Computer Interaction (HCI) where cursor movement can be controlled using a real-time camera, it is an alternative to the current methods including manual input of buttons or changing the positions of a physical computer mouse. Instead, it utilizes a camera and computer vision technology to control various mouse events and is capable of performing every task that the physical computer mouse can.

The Virtual Mouse colour recognition program will constantly acquiring real-time images where the images will undergone a series of filtration and conversion. Whenever the process is complete, the program will apply the image processing technique to obtain the coordinates of the targeted colours position from the converted frames. After that, it will proceed to compare the existing colours within the frames with a list of colour combinations, where different combinations consists of different mouse functions. If the current colours combination found a match, the program will execute the mouse function, which will be translated into an actual mouse function to the users' machine.

1. INTRODUCTION

A mouse, in computing terms is a pointing device that detects two-dimensional movements relative to a surface. This movement is converted into the movement of a pointer on a display that allows to control the Graphical User Interface (GUI) on a computer platform. There are a lot of different types of mouse that have already existed in the modern days technology, there's the mechanical mouse that determines the movements by a hard rubber ball that rolls around as the mouse is moved. Years later, the optical mouse was introduced that replace the hard rubber ball to a LED sensor to detects table top movement and then sends off the information to the computer for processing. On the year 2004, the laser mouse was then introduced to improve the accuracy movement with the slightest hand movement, it overcome the limitations of the optical mouse which is the difficulties to track high-gloss surfaces. However, no matter how accurate can it be, there are still limitations exist within the mouse itself in both physical and technical terms. For example, a computer mouse is a consumable hardware device as it requires replacement in the long run, either the mouse buttons were degraded that causes inappropriate clicks, or the whole mouse was no longer detected by the computer itself.

Despite the limitations, the computer technology still continues to grow, so does the importance of the human computer interactions. Ever since the introduction of a mobile device that can be interact with touch screen technology, the world is starting to demand the same technology to be applied on every technological devices, this includes the desktop system. However, even though the touch screen technology for the desktop system is already exists, the price can be very steep. Therefore, a virtual human computer interaction device that replaces the physical mouse or keyboard by using a webcam or any other image capturing devices can be an alternative way for the touch screen. This device which is the webcam will be constantly utilized by a software that monitors the gestures given by the user in order to process it and translate to motion of a pointes, as similar to a physical mouse.

I.1 Review of the Physical Mouse

It is known that there are various types of physical computer mouse in the modern technology, the following will discuss about the types and differences about the physical mouse.

1.2 Mechanical Mouse

Known as the trackball mouse that is commonly used in the 1990s, the ball within the mouse are supported by two rotating rollers in order to detect the movement made by the ball itself.

One roller detects the forward/backward motion while the other detects the left/right motion. The ball within the mouse are steel made that was covered with a layer of hard rubber, so that the detection are more precise. The common functions included are the left/right buttons and a scroll-wheel. However, due to the constant friction made between the mouse ball and the rollers itself, the mouse are prone to degradation, as overtime usage may cause the rollers to degrade, thus causing it to unable to detect the motion properly, rendering it useless. Furthermore, the switches in the mouse buttons are no different as well, as long term usage may cause the mechanics within to be loosed and will no longer detect any mouse clicks till it was disassembled and repaired.

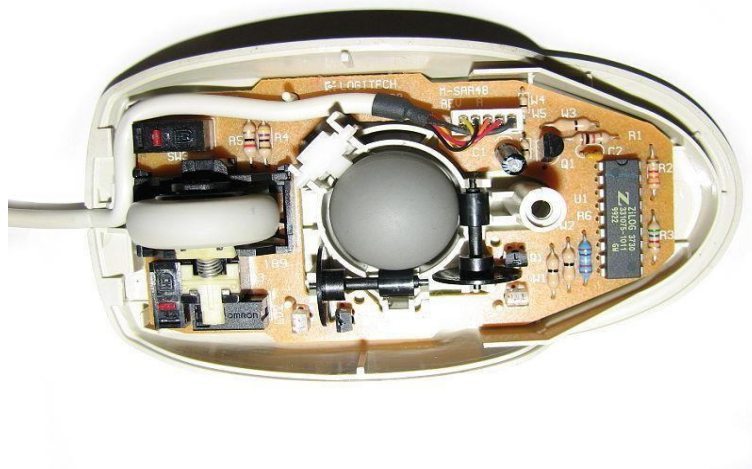


Figure 1: Mechanical mouse, with top cover removed

The following table describes the advantages and disadvantages of the Mechanical Mouse.

Advantage	Disadvantage
<ul style="list-style-type: none"> • Allows the users to control the computer system by moving the mouse. • Provides precise mouse tracking movements 	<ul style="list-style-type: none"> • Prone to degradation of the mouse rollers and button switches, causing to be faulty. • Requires a flat surface to operate.

Table 1: Advantage and disadvantage of the Mechanical Mouse

1.3 Optical And Laser Mouse

A mouse that commonly used in these days, the motions of optical mouse rely on the Light Emitting Diodes (LEDs) to detect movements relative to the underlying surface, while the laser mouse is an optical mouse that uses coherent laser lights. Comparing to its predecessor, which

is the mechanical mouse, the optical mouse no longer rely on the rollers to determine its movement, instead it uses an imaging array of photodiodes. The purpose of implementing this is to eliminate the limitations of degradation that plagues the current predecessor, giving it more durability while offers better resolution and precision. However, there's still some downside, even-though the optical mouse are functional on most opaque diffuse surface, it's unable to detect motions on the polished surface. Furthermore, long term usage without a proper cleaning or maintenance may leads to dust particles trap between the LEDs, which will cause both optical and laser mouse having surface detection difficulties. Other than that, it's still prone to degradation of the button switches, which again will cause the mouse to function improperly unless it was disassembled and repaired.



Figure 2: Optical Mouse, with top cover removed

The following table describes the advantages and disadvantages of the Optical and Laser Mouse.

Table 2: Advantage and disadvantage of the Optical and Laser Mouse

Advantages	Disadvantages
<p>Allows better precision with lesser hand movements.</p> <ul style="list-style-type: none"> • Longer life-span. 	<p>Prone to button switches degradation.</p> <p>Does not function properly while on a polished surface.</p>

2. LITERATURE SURVEY

As modern technology of human computer interactions become important in our everyday lives, varieties of mouse with all kind of shapes and sizes were invented, from a casual office mouse to a hard-core gaming mouse. However, there are some limitations to these hardware as they are not as environmental friendly as it seems. For example, the physical mouse requires a flat surface to operate, not to mention that it requires a certain area to fully utilize the functions offered. Furthermore, some of these hardware are completely useless when it comes to interact with the computers remotely due to the cable lengths limitations, rendering it inaccessible.

2.1 Problem Statement

It's no surprised that every technological devices have its own limitations, especially when it comes to computer devices. After the review of various type of the physical mouse, the problems are identified and generalized. The following describes the general problem that the current physical mouse suffers:

- Physical mouse is subjected to mechanical wear and tear.
- Physical mouse requires special hardware and surface to operate.
- Physical mouse is not easily adaptable to different environments and its performance varies depending on the environment.
- Mouse has limited functions even in present operational environments.
- All wired mouse and wireless mouse have its own lifespan.

2.2 Motivation of Virtual Mouse

It is fair to say that the Virtual Mouse will soon to be substituting the traditional physical mouse in the near future, as people are aiming towards the lifestyle where that every technological devices can be controlled and interacted remotely without using any peripheral devices such as the remote, keyboards, etc. it doesn't just provides convenience, but it's cost effective as well.

2.3 Convenient

It is known in order to interact with the computer system, users are required to use an actual physical mouse, which also requires a certain area of surface to operate, not to mention that it

suffers from cable length limitations. Virtual Mouse requires none of it, as it only a webcam to allow image capturing of user's hand position in order to determine the position of the pointers that the user want it to be. For example, the user will be able to remotely control and interact the computer system by just facing the webcam or any other image capturing devices and moving your fingers, thus eliminating the need to manually move the physical mouse, while able to interact with the computer system from few feet away.

2.4 Cost Effective

A quality physical mouse is normally cost from the range of 30 ringgit to a hefty 400 ringgit, depending on their functionality and features. Since the Virtual Mouse requires only a webcam, a physical mouse are no longer required, thus eliminating the need to purchase one, as a single webcam is sufficient enough to allow users to interact with the computer system through it, while some other portable computer system such as the laptop, are already supplied with a built-in webcam, could simply utilize the Virtual Mouse software without having any concerns about purchasing any external peripheral devices.

2.5 Project Scope

Virtual Mouse that will soon to be introduced to replace the physical computer mouse to promote convenience while still able to accurately interact and control the computer system. To do that, the software requires to be fast enough to capture and process every image, in order to successfully track the user's gesture. Therefore, this project will develop a software application with the aid of the latest software coding technique and the open-source computer vision library also known as the OpenCV. The scope of the project is as below:

- Real time application.
- User friendly application.
- Removes the requirement of having a physical mouse.

The process of the application can be started when the user's gesture was captured in real time by the webcam, which the captured image will be processed for segmentation to identify which pixels values equals to the values of the defined colour. After the segmentation is completed, the overall image will be converted to Binary Image where the identified pixels will show as white, while the rest are black. The position of the white segment in the image will be recorded and set as the position of the mouse pointer, thus resulting in simulating the mouse pointer

without using a physical computer mouse. The software application is compatible with the Windows platform. The functionality of the software will be coded with C++ programming language code with the integration of an external library that does the image processing known as the OpenCV.

2.6 Project Objective

The purpose of this project is to develop a Virtual Mouse application that targets a few aspects of significant development. For starters, this project aims to eliminate the needs of having a physical mouse while able to interact with the computer system through webcam by using various image processing techniques. Other than that, this project aims to develop a Virtual Mouse application that can be operational on all kind of surfaces and environment.

The following describes the overall objectives of this project:

- To design to operate with the help of a webcam.

The Virtual Mouse application will be operational with the help of a webcam, as the webcam are responsible to capture the images in real time. The application would not work if there are no webcam detected.

- To design a virtual input that can operate on all surface.
The Virtual Mouse application will be operational on all surface and indoor environment, as long the users are facing the webcam while doing the motion gesture.
- To program the camera to continuously capturing the images, which the images will be analyzed, by using various image processing techniques.

As stated above, the Virtual Mouse application will be continuously capturing the images in real time, where the images will be undergo a series of process, this includes HSV conversion, Binary Image conversion, salt and pepper noise filtering, and more.

- To convert hand gesture/motion into mouse input that will be set to a particular screen position.
The Virtual Mouse application will be programmed to detect the position of the defined colours where it will be set as the position of the mouse pointers. Furthermore, a combination of different colors may result in triggering different types of mouse events, such as the right/left clicks, scroll up/down, and more.

2.7 Impact, Significance and Contribution

The Virtual Mouse application is expected to replace the current methods of utilizing a physical computer mouse where the mouse inputs and positions are done manually. This application offers a more effortless way to interact with the computer system, where every task can be done by gestures. Furthermore, the Virtual Mouse application could assist the motor-impaired users where he/she could interact with the computer system by just showing the correct combination of colors to the webcam.

2.8 Portable Vision-Based Human Computer Interaction (HCI)

Another "Ubiquitous Computing" approach proposed by Chu-Feng Lien (2015), requires only finger-tips to control the mouse cursor and click events. The proposed system doesn't require hand-gestures nor colour tracking in order to interact with the system, instead it utilizes a feature name Motion History Images (MHI), a method that is used to identify movements with a row of images in time. The proposed system is not capable to detect fast moving movements as the frame-rates are not able to keep up, thus leading to an increase of error rate. Furthermore, due to the mouse click events occurring when the finger holds on a certain position, this may lead to user constant finger movements to prevent false alarm, which may result in inconvenience. To detect the colours, they have utilized MATLAB's built-in "*subtract*" function, with the combination of noise filtering by using median filter, which is effective in filtering out or at least reduce the "salt and pepper" noise. The captured image will be converted to Binary Scale Image by using MATLAB's built-in "*im2bw*" function to differentiate the possible values for each pixel. When the conversion is done, the captured image will undergo another filtering process by using "*bwareaopen*" to remove the small areas in order to get an accurate number of the object detected in the image.

Even though the proposed system possesses good accuracy in a well-controlled environment, it does have its own limitations. The proposed system is not capable to detect fast moving movements as the frame-rates are not able to keep up, thus leading to an increase of error rate. Furthermore, due to the mouse click events occurring when the finger holds on a certain position, this may lead to user constant finger movements to prevent false alarm, which may result in inconvenience.

2.9 Hardware Requirement

The following describes the hardware needed in order to execute and develop the Virtual Mouse application:

2.9.1 Computer Desktop or Laptop

The computer desktop or a laptop will be utilized to run the visual software in order to display what webcam had captured. A notebook which is a small, lightweight and inexpensive laptop computer is proposed to increase mobility.

System will be using

Processor	:	Core i5
Main Memory	:	4GB RAM
Hard Disk	:	500GB
Display	:	14" Monitor

2.9.2 Webcam

Webcam is utilized for image processing, the webcam will continuously taking image in order for the program to process the image and find pixel position.

2.10 Software Requirement

The following describes the software needed in-order to develop the Virtual Mouse application:

2.10.1 Python Language

The coding technique on developing the Virtual Mouse application will be the Python with the aid of the integrated development environment (IDE) that are used for developing computer programs, known as the Pycharm. A python library provides more than 35 operators, covering basic arithmetic, bit manipulation, indirection, comparisons, logical operations and others.

2.10.2 Pycharm IDE

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.

PyCharm is available in three editions:

- *Community* (free and open-sourced): for smart and intelligent Python development, including code assistance, refactorings, visual debugging, and version control integration.
- *Professional* (paid) : for professional Python, web, and data science development, including code assistance, refactorings, visual debugging, version control integration, remote configurations, deployment, support for popular web frameworks, such as Django and Flask, database support, scientific tools (including Jupyter notebook support), big data tools.
- *Edu* (free and open-sourced): for learning programming languages and related technologies with integrated educational tools.

-

➤ Supported languages

To start developing in Python with PyCharm you need to download and install Python from python.org depending on your platform.

PyCharm supports the following versions of Python:

- **Python 2:** version 2.7
- **Python 3:** from the version 3.6 up to the version 3.11

Besides, in the *Professional* edition, one can develop Django , Flask, and Pyramid applications. Also, it fully supports HTML (including HTML5), CSS, JavaScript, and XML: these languages are bundled in the IDE via plugins and are switched on for you by default. Support for the other languages and frameworks can also be added via plugins (go to Settings | Plugins or PyCharm | Preferences | Plugins for macOS users, to find out more or set them up during the first IDE launch).

➤ Supported platforms

PyCharm is a cross-platform IDE that works on Windows, macOS, and Linux. Check the system requirements:

Table 3: pycharm Specification

Requirement	Minimum	Recommended
RAM	4 GB of free RAM	8 GB of total system RAM
CPU	Any modern CPU	Multi-core CPU. PyCharm supports multithreading for different operations and

		processes making it faster the more CPU cores it can use.
Disk space	2.5 GB and another 1 GB for caches	SSD drive with at least 5 GB of free space
Monitor resolution	1024×768	1920×1080
Operating system	Microsoft Windows 8 or later macOS 10.14 or later Any Linux distribution that supports Gnome, KDE , or Unity DE. PyCharm is not available for some Linux distributions,	Latest 64-bit version of Windows, macOS, or Linux (for example, Debian, Ubuntu, or RHEL)

2.10.3 Open CV Library

OpenCV are also included in the making of this program. **OpenCV (Open Source Computer Vision)** is a library of programming functions for real time computer vision. OpenCV have the utility that can read image pixels value, it also have the ability to create real time eye tracking and blink detection.

Software will be using:

OS	:	Window 10 64-bit
Language	:	Python
Tool Used	:	Open CV and Autopy

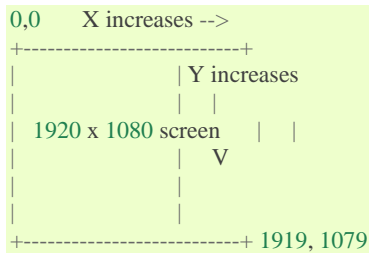
2.10.4 PyAutoGUI

PyAutoGUI is a cross-platform GUI automation Python module for human beings. Used to programmatically control the mouse & keyboard.

Mouse Control function

The Screen and Mouse Position

Locations on your screen are referred to by X and Y Cartesian coordinates. The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down.



The pixel at the top-left corner is at coordinates 0, 0. If your screen's resolution is 1920 x 1080, the pixel in the lower right corner will be 1919, 1079 (since the coordinates begin at 0, not 1).

The screen resolution size is returned by the `size()` function as a tuple of two integers. The current X and Y coordinates of the mouse cursor are returned by the `position()` function.

For example:

```
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.position()
(187, 567)
```

Here is a short Python 3 program that will constantly print out the position of the mouse cursor:

```
#!/ python3
import pyautogui, sys
print('Press Ctrl-C to quit.')
try:
    while True:
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        print(positionStr, end='')
        print('\b' * len(positionStr), end='', flush=True)
except KeyboardInterrupt:
    print('\n')
```

To check if XY coordinates are on the screen, pass them (either as two integer arguments or a single tuple/list arguments with two integers) to the `onScreen()` function, which will return `True` if they are within the screen's boundaries and `False` if not. For example:

```
>>> pyautogui.onScreen(0, 0)
True
>>> pyautogui.onScreen(0, -1)
False
>>> pyautogui.onScreen(0, 99999999)
False
>>> pyautogui.size()
```

```
(1920, 1080)
```

```
>>> pyautogui.onScreen(1920, 1080)
```

```
False
```

```
>>> pyautogui.onScreen(1919, 1079)
```

```
True
```

- **Mouse Movement**

The `moveTo()` function will move the mouse cursor to the X and Y integer coordinates you pass it. The `None` value can be passed for a coordinate to mean “the current mouse cursor position”. For example:

```
>>> pyautogui.moveTo(100, 200) # moves mouse to X of 100, Y of 200.
```

```
>>> pyautogui.moveTo(None, 500) # moves mouse to X of 100, Y of 500.
```

```
>>> pyautogui.moveTo(600, None) # moves mouse to X of 600, Y of 500.
```

Normally the mouse cursor will instantly move to the new coordinates. If you want the mouse to gradually move to the new location, pass a third argument for the duration (in seconds) the movement should take. For example:

If you want to move the mouse cursor over a few pixels *relative* to its current position, use the `move()` function. This function has similar parameters as `moveTo()`. For example:

```
>>> pyautogui.moveTo(100, 200) # moves mouse to X of 100, Y of 200.
```

```
>>> pyautogui.move(0, 50) # move the mouse down 50 pixels.
```

```
>>> pyautogui.move(-30, 0) # move the mouse left 30 pixels.
```

```
>>> pyautogui.move(-30, None) # move the mouse left 30 pixels.
```

- **Mouse Drags**

PyAutoGUI's `dragTo()` and `drag()` functions have similar parameters as the `moveTo()` and `move()` functions. In addition, they have a `button` keyword which can be set to `'left'`, `'middle'`, and `'right'` for which mouse button to hold down while dragging. For example:

```
>>> pyautogui.dragTo(100, 200, button='left') # drag mouse to X of 100, Y of 200 while holding down left mouse button
```

```
>>> pyautogui.dragTo(300, 400, 2, button='left') # drag mouse to X of 300, Y of 400 over 2 seconds while holding down left mouse button
```

```
>>> pyautogui.drag(30, 0, 2, button='right')  
# drag the mouse left 30 pixels over 2 seconds while holding down the right mouse button
```

Tween / Easing Functions

Tweening is an extra feature to make the mouse movements fancy. You can probably skip this section if you don't care about this.

A tween or easing function dictates the progress of the mouse as it moves to its destination. Normally when moving the mouse over a duration of time, the mouse moves directly towards the destination in a straight line at a constant speed. This is known as a *linear tween* or *linear easing* function.

PyAutoGUI has other tweening functions available in the `pyautogui` module. The `pyautogui.easeInQuad` function can be passed for the 4th argument to `moveTo()`, `move()`, `dragTo()`, and `drag()` functions to have the mouse cursor start off moving slowly and then speeding up towards the destination. The total duration is still the same as the argument passed to the function. The `pyautogui.easeOutQuad` is the reverse: the mouse cursor starts moving fast but slows down as it approaches the destination. The `pyautogui.easeOutElastic` will overshoot the destination and “rubber band” back and forth until it settles at the destination.

For example:

```
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInQuad) # start slow, end fast
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeOutQuad) # start fast, end slow
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInOutQuad) # start and end fast, slow in middle
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInBounce) # bounce at the end
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInElastic) # rubber band at the end
```

If you want to create your own tweening function, define a function that takes a single float argument between `0.0` (representing the start of the mouse travelling) and `1.0` (representing the end of the mouse travelling) and returns a float value between `0.0` and `1.0`.

- **Mouse Clicks**

The `click()` function simulates a single, left-button mouse click at the mouse’s current position. A “click” is defined as pushing the button down and then releasing it up. For example:

```
>>> pyautogui.click() # click the mouse
```

To combine a `moveTo()` call before the click, pass integers for the `x` and `y` keyword argument:

```
>>> pyautogui.click(x=100, y=200) # move to 100, 200, then click the left mouse button.
```

To specify a different mouse button to click, pass `'left'`, `'middle'`, or `'right'` for the `button` keyword argument:

```
>>> pyautogui.click(button='right') # right-click the mouse
```

To do multiple clicks, pass an integer to the `clicks` keyword argument. Optionally, you can pass a float or integer to the `interval` keyword argument to specify the amount of pause between the clicks in seconds. For example:

```
>>> pyautogui.click(clicks=2) # double-click the left mouse button
>>> pyautogui.click(clicks=2, interval=0.25) # double-click the left mouse button, but with a quarter second
pause in between clicks
>>> pyautogui.click(button='right', clicks=3, interval=0.25) ## triple-click the right mouse button with a quarter
second pause in between clicks
```

As a convenient shortcut, the `doubleClick()` function will perform a double click of the left mouse button. It also has the optional `x`, `y`, `interval`, and `button` keyword arguments. For example:

```
>>> pyautogui.doubleClick() # perform a left-button double click
There is also a tripleClick() function with similar optional keyword arguments.
```

The `rightClick()` function has optional `x` and `y` keyword arguments.

• The `mouseDown()` and `mouseUp()` Functions

Mouse clicks and drags are composed of both pressing the mouse button down and releasing it back up. If you want to perform these actions separately, call the `mouseDown()` and `mouseUp()` functions. They have the same `x`, `y`, and `button`. For example:

```
>> pyautogui.mouseDown(); pyautogui.mouseUp() # does the same thing as a left-button mouse click
>>> pyautogui.mouseDown(button='right') # press the right button down
>>> pyautogui.mouseUp(button='right', x=100, y=200) # move the mouse to 100, 200, then release the right
button up.
```

• Mouse Scrolling

The mouse scroll wheel can be simulated by calling the `scroll()` function and passing an integer number of “clicks” to scroll. The amount of scrolling in a “click” varies between platforms. Optionally, integers can be passed for the `x` and `y` keyword arguments to move the mouse cursor before performing the scroll. For example:

```
>> pyautogui.scroll(10) # scroll up 10 "clicks"
>>> pyautogui.scroll(-10) # scroll down 10 "clicks"
>>> pyautogui.scroll(10, x=100, y=100) # move mouse cursor to 100, 200, then scroll up 10 "clicks"
On OS X and Linux platforms, PyAutoGUI can also perform horizontal scrolling by calling
the hscroll() function. For example:
```

```
>>> pyautogui.hscroll(10) # scroll right 10 "clicks"
>>> pyautogui.hscroll(-10) # scroll left 10 "clicks"
The scroll() function is a wrapper for vscroll(), which performs vertical scrolling.
```

2.10.5. Mediapipe

MediaPipe is a framework which is used for applying in a machine learning pipeline, and it is an opensource framework of Google. The MediaPipe framework is useful for cross platform development since the framework is built using the time series data. The MediaPipe framework is multimodal, where this framework can be applied to various audios and videos . The MediaPipe framework is used by the developer for building and analyzing the systems through graphs, and it also been used for developing the systems for the application purpose.

The steps involved in the system that uses MediaPipe are carried out in the pipeline configuration. The pipeline created can run in various platforms allowing scalability in mobile and desktops. The MediaPipe framework is based on three fundamental parts; they are performance evaluation, framework for retrieving sensor data, and a collection of components which are called calculators, and they are reusable.

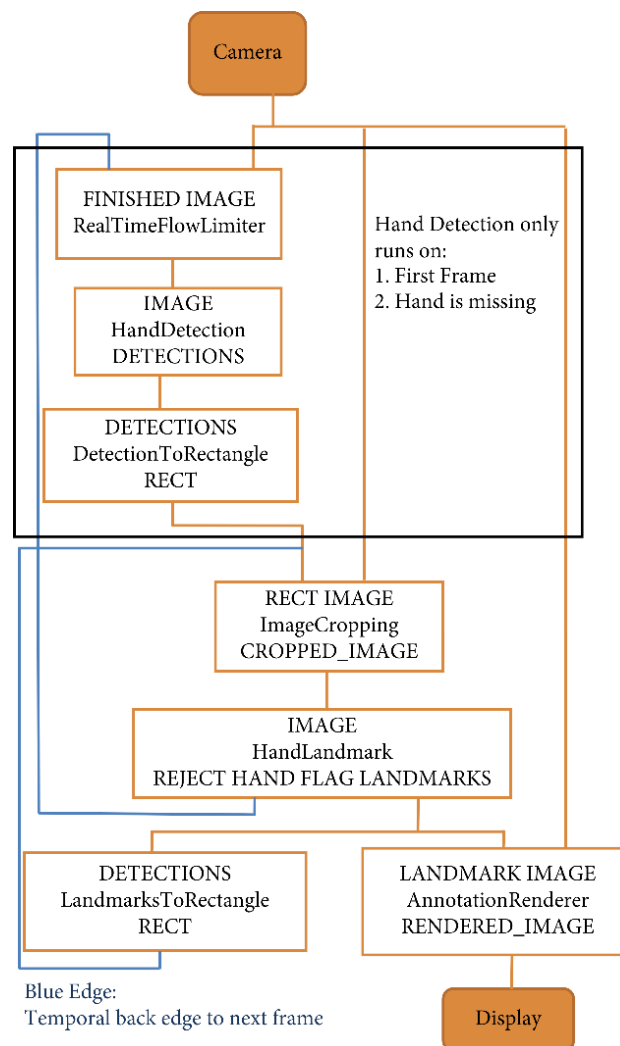


Figure 3: MediaPipe hand recognition graph

A pipeline is a graph which consists of components called calculators, where each calculator is connected by streams in which the packets of data flow through. Developers are able to replace or define custom calculators anywhere in the graph creating their own application. The calculators and streams combined create a data-flow diagram; the graph (Figure 1) is created with MediaPipe where each node is a calculator and the nodes are connected by stream

Single-shot detector model is used for detecting and recognizing a hand or palm in real time. The single-shot detector model is used by the MediaPipe. First, in the hand detection module, it is first trained for a palm detection model because it is easier to train palms. Furthermore, the non maximum suppression works significantly better on small objects such as palms or fists . A model of hand landmark consists of locating 21 joint or knuckle co-ordinates in the hand region, as shown in Figure .

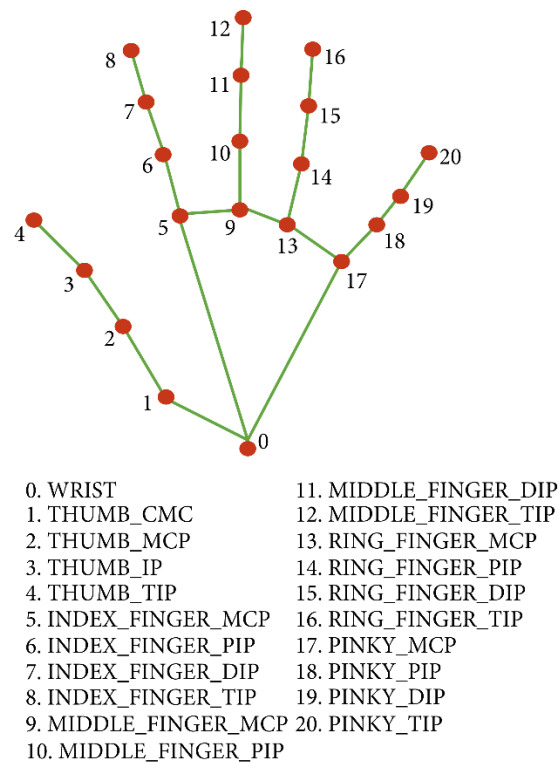


Figure 4: Co-ordinates or land marks in the hand

2.10.6. Autopy

It is cross platform GUI automation module for python. That module keeps tracks of finger in this proposed system. Autopy track the fingertip and tell us which finger is up and which one is down This process is happing by giving system an input in the form of 0 and 1. From this module the mediapipe module takes output and done the process and give the proper output. From this

output opencv visualize everything and create the proper frame from image.

Autopy Moudule for working with mouse

This module contains functions for getting the current state of and controlling the mouse cursor.

Unless otherwise stated, coordinates are those of a screen coordinate system, where the origin is at the top left.

- **Functions**

`autopy.mouse.location()` -> *(float, float)*

Returns a tuple (x, y) of the current mouse position.

`autopy.mouse.toggle(button: Button=None, down: bool)`

Holds down or releases the given mouse button in the current position. Button can be LEFT, RIGHT, MIDDLE, or None to default to the left button.

`autopy.mouse.click(button: Button=None, delay: float=None)`

Convenience wrapper around toggle() that holds down and then releases the given mouse button. By default, the left button is pressed.

`autopy.mouse.move(x: float, y: float)`

Moves the mouse to the given (x, y) coordinate.

Exceptions:

ValueError is thrown if the point is out of index.

`autopy.mouse.smooth_move(x: float, y: float)`

Smoothly moves the mouse to the given (x, y) coordinate in a straight line.

Exceptions:

ValueError is thrown if the point is out of index.

`class autopy.mouse.Button`

- **constants**

`class autopy.mouse.Button`

Button: left

Button: Right

Button: Middle

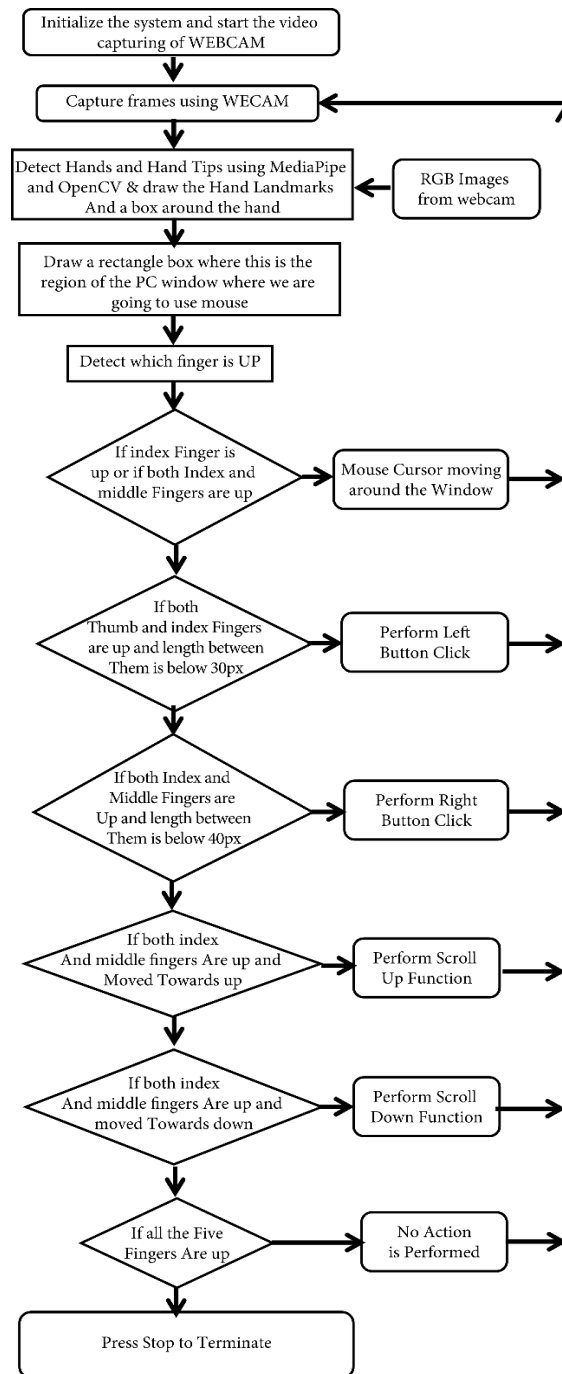


Figure 5: The Flow Chart of Portable Vision-Based Human Computer Interaction

There are abundance of methods for computer interaction besides the traditional physical mouse interaction. With the evolutions of modern technology and programming, so does the Human Computer Interaction (HCI) methods, as it allows unlimited ways to access the computers. This approach allows the developers to design specific/unique system that suit the needs of the users, from gesture movement tracking to coloured tracking, it's no surprise that in near future, physical mouse will no longer be needed and be replaced by video cameras that tracks gestures.

3. METHODOLOGY

Visualization of mouse actions for a given hand gesture is a challenging problem in the deep learning domain. In this article, we will use different techniques of computer vision and Deep learning to recognize the hand gestures and perform the mouse operations. We will build a Ai Virtual mouse by using Autopy and Mediapipe.

Ai Virtual Mouse is a challenging HCI problem where a hand gesture work as a mouse. It requires both methods from computer vision to understand the content of the image and a language model from the field of Deep learning to turn the understanding of the image into gesture in the right order. Recently, deep learning methods have achieved state-of-the-art results on examples of this problem.

Deep learning methods have demonstrated state-of-the-art results on gesture generation problems. What is most impressive about these methods is a single end-to-end model can be defined to predict a gesture, given a photo, instead of requiring sophisticated data preparation or a pipeline of specifically designed models.

The system can be broken down in three main modules.

Image Acquisition: We need a sensor for the system to detect the user's hand movements. As a sensor, the computer's camera is used. The webcam captures real-time video at a fixed frame rate and resolution determined by the camera's hardware. If necessary, the system allows you to change the frame rate and resolution. When the camera takes a picture, it inverts it. The pointer image moves to the right, and versa if we move the color pointer to the left. It is similar to the picture we get when we stand before a mirror, but it is used to avoid the picture flickering.

Image processing and hand detection: The computational complexity of a grey picture is less than a colorful one. This transforms the image into a grey image. All necessary operations were performed once the image was converted to greyscale. Then you have to subject a noise filter, smoothing, and threshold.

Hand Gesture: After completing the previous modules, the mouse movement, left-click, right-click, drag/select, scroll up and scroll down will be carried out with color caps or tapes on our fingers. For this project we'll be using the Agile Software Development methodology approach in developing the application. The stated approach is an alternative to the traditional waterfall model that helps the project team respond to unpredictability through incremental and iterative work. It promotes adaptive planning, evolutionary development, early delivery, continuous

improvement, and encourages rapid and flexible response to change. The following describes the principles of the Agile Software Development methodology.

- Satisfy the customer by early and continuous delivery of workable software.
- Encourage changes of requirement.
- Workable software is delivered frequently.
- Continuous collaboration between the stakeholders and the developers.
- Projects are developed around motivated individuals.
- Encourage informal meetings.
- Operational software is the principle measure of progress.
- Sustainable development, able to maintain a constant pace.
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

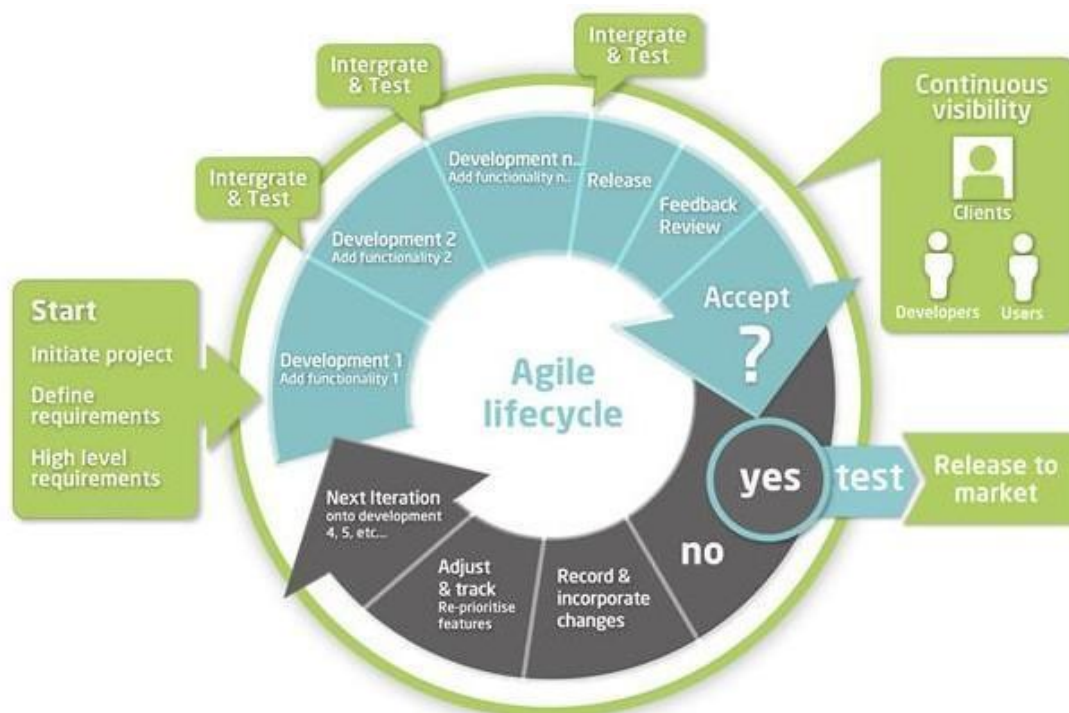


Figure 6: Agile method overview

The reason for choosing this methodology is due to the fact that the Virtual Mouse are still considered to be at the introduction stage, which means it still requires a great deal of extensive research and development before it could actually make it into the market. Therefore, this project requires a thorough yet iterative planning and requirements gathering where the

20

lifecycle will be continually revisited to re-evaluate the direction of the project and to eliminate the ambiguities in the process of the development, and at the same time welcome changes of requirements, which promotes adaptability and flexibility. Furthermore, due to the Virtual Mouse application are more towards serving the users, this project requires continuous customer collaboration, as they're essential for gathering the proper requirements in all aspects. This is why that the agile methodology is the ideal approach for developing the project.

The following describes the phases within the agile methodology approach:

- Planning

A thorough planning will be conducted in this phase where the existing systems/product, for this case, physical computer mouse will be reviewed and studied to identify the problems existed, a comparison of problems will be made to compare which problems are more crucial and requires improvement. An outline objective and the scope will be identified in order to provide an alternative solution to the problem.

- Requirement Analysis

The phase that gathers and interpreting the facts, diagnosing problems and recommending improvements to the system. In this phase, the collected problem statements will be extensively studied in order to find a proper solution or at least an improvements to the proposed system. All proposed solutions will be converted into requirements where it will be documented in a requirement specification.

- Designing

The requirement specification from the previous phase will be studied and prioritize to determine which requirement are more important where the requirement with the highest priority will be delivered first. After the study, the system design will be prepared as it helps in defining the overall system architecture and specifying the hardware and the software requirements.

- Building

The phase where the actual coding implementation takes place. By referring to the inputs from the system design, the system will be developed based on the prioritize requirements. However,

due to we're using the agile methodology approach, the developed system will be considered as a prototype system where it will be integrated and tested by the users.

- Testing

The phase where the prototype system going through a series of test. The prototype system will first undergo integration where the features from the previous iteration cycle are added to the latest cycle. After the integration, the prototype system will be thoroughly tested by the users to determine whether they are satisfied with the latest deliverables, the completion of the project depends on whether they've accepted it or otherwise. If the users requires additional features or modification, feedback gathering will be conducted, which resulted in further modification of the requirements and features where it will recorded and documented for the requirement analysis phase on the next iteration

3.1 Import the required libraries

```
import math
from ctypes import cast, POINTER
from enum import IntEnum

import cv2
import mediapipe as mp
import pyautogui
from comtypes import CLSCTX_ALL
from google.protobuf.json_format import MessageToDict
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
```

3.2 Capturing the Image and Processing

The AI virtual mouse system uses the webcam where each frame is captured till the termination of the program. The video frames are processed from BGR to RGB color space to find the hands in the video frame by frame as shown in the following code:

```
def findHands(self, img, draw = True):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)
```

3.3 Preparation of Gesture Encoding

The proposed AI virtual mouse system is based on the frames that have been captured by the webcam in a laptop or PC. By using the Python computer vision library OpenCV, the video capture object is created and the web camera will start capturing video, as shown in Figure 4. The web camera captures and passes the frames to the AI virtual system.

The AI virtual mouse system makes use of the transformational algorithm, and it converts the co-ordinates of fingertip from the webcam screen to the computer window full screen for controlling the mouse. When the hands are detected and when we find which finger is up for performing the specific mouse function, a rectangular box is drawn with respect to the computer window in the webcam region where we move throughout the window using the mouse cursor.

```
# import screen_brightness_control as sbcontrol
```

```
pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
```

```
# Gesture Encodings
```

```
class Gest(IntEnum):
```

```
    # Binary Encoded
```

```
    FIST = 0
```

```
    PINKY = 1
```

```
    RING = 2
```

```
    MID = 4
```

```
    LAST3 = 7
```

```
    INDEX = 8
```

```
    FIRST2 = 12
```

```
    LAST4 = 15
```

```
    THUMB = 16
```

```
    PALM = 31
```

```
# Extra Mappings
```

```
V_GEST = 33
```

```
TWO_FINGER_CLOSED = 34
```

```
PINCH_MAJOR = 35
```

```
PINCH_MINOR = 36
```

```
# Multi-handedness Labels
```

```
class HLabel(IntEnum):
```

```
    MINOR = 0
```

```
    MAJOR = 1
```

3.4 Converting Mediapipe Landmarks To Recognizable Gestures

we will use *mediapipe* python library to detect face and hand landmarks. We will be using a Holistic model from *mediapipe* solutions to detect all the face and hand landmarks. We will be also seeing how we can access different landmarks of the face and hands which can be used for different computer vision applications such as sign language detection, drowsiness detection, etc.

3.4.1 Static image mode: It is used to specify whether the input images must be treated as static images or as a video stream. The default value is False.

3.4.2 Model complexity: It is used to specify the complexity of the pose landmark model: 0, 1, or 2. As the model complexity of the model increases the landmark accuracy and latency increase. The default value is 1.

3.4.3 Smooth landmarks: This parameter is used to reduce the jitter in the prediction by filtering pose landmarks across different input images. The default value is True.

3.4.4 Min detection confidence: It is used to specify the minimum confidence value with which the detection from the person-detection model needs to be considered as successful. Can specify a value in [0.0,1.0]. The default value is 0.5.

3.4.5 Min tracking confidence: It is used to specify the minimum confidence value with which the detection from the landmark-tracking model must be considered as successful. Can specify a value in [0.0,1.0]. The default value is 0.5.

Detecting Face and Hand landmarks from the image. Holistic model processes the image and produces landmarks for Face, Left Hand, Right Hand and also detects the Pose of the

1. Capture the frames continuously from the camera using OpenCV.
2. Convert the BGR image to an RGB image and make predictions using initialized holistic model.
3. The predictions made by the holistic model are saved in the results variable from which we can access the landmarks using results.face_landmarks, results.right_hand_landmarks, results.left_hand_landmarks respectively.
4. Draw the detected landmarks on the image using the draw_landmarks function from drawing utils.
5. Display the resulting Image.

The holistic model produces 21 Left-Hand landmarks, and 21 Right-Hand landmarks. The individual landmarks can be accessed by specifying the index of the required landmark. Example: results.left_hand_landmarks.landmark[0]

```
# Convert Mediapipe Landmarks to recognizable Gestures
```

```
class HandRecog:
```

```
    def __init__(self, hand_label):  
        self.finger = 0
```

```

self.ori_gesture = Gest.PALM
self.prev_gesture = Gest.PALM
self.frame_count = 0
self.hand_result = None
self.hand_label = hand_label

def update_hand_result(self, hand_result):
    self.hand_result = hand_result

def get_signed_dist(self, point):
    sign = -1
    if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
        sign = 1
    dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
    dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
    dist = math.sqrt(dist)
    return dist * sign

def get_dist(self, point):
    dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x) ** 2
    dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y) ** 2
    dist = math.sqrt(dist)
    return dist

def get_dz(self, point):
    return abs(self.hand_result.landmark[point[0]].z - self.hand_result.landmark[point[1]].z)

```

3.5 Finding Gesture Using Current finger State

In this stage, we are detecting which finger is up using the tip Id of the respective finger that we found using the MediaPipe and the respective co-ordinates of the fingers that are up, as shown in Figure and according to that, the particular mouse function is performed

```

# Function to find Gesture Encoding using current finger state.
# Finger state: 1 if finger is open, else 0
def set_finger_state(self):
    if self.hand_result == None:
        return

    points = [[8, 5, 0], [12, 9, 0], [16, 13, 0], [20, 17, 0]]
    self.finger = 0
    self.finger = self.finger | 0 # thumb
    for idx, point in enumerate(points):

        dist = self.get_signed_dist(point[:2])
        dist2 = self.get_signed_dist(point[1:])

        try:
            ratio = round(dist / dist2, 1)
        except:
            ratio = round(dist1 / 0.01, 1)

        self.finger = self.finger << 1
        if ratio > 0.5:
            self.finger = self.finger | 1

```


3.6 Handling Fluctuations Due To Noise

In this stage we are eliminating the noise that means if we are performing the work gesture, according to the previous discussed gesture then the gesture will be eliminated and no action will be performed.

```
# Handling Fluctuations due to noise
def get_gesture(self):
    if self.hand_result == None:
        return Gest.PALM

    current_gesture = Gest.PALM
    if self.finger in [Gest.LAST3, Gest.LAST4] and self.get_dist([8, 4]) < 0.05:
        if self.hand_label == HLabel.MINOR:
            current_gesture = Gest.PINCH_MINOR
        else:
            current_gesture = Gest.PINCH_MAJOR

    elif Gest.FIRST2 == self.finger:
        point = [[8, 12], [5, 9]]
        dist1 = self.get_dist(point[0])
        dist2 = self.get_dist(point[1])
        ratio = dist1 / dist2
        if ratio > 1.7:
            current_gesture = Gest.V_GEST
        else:
            if self.get_dz([8, 12]) < 0.1:
                current_gesture = Gest.TWO_FINGER_CLOSED
            else:
                current_gesture = Gest.MID

    else:
        current_gesture = self.finger

    if current_gesture == self.prev_gesture:
        self.frame_count += 1
    else:
        self.frame_count = 0

    self.prev_gesture = current_gesture

    if self.frame_count > 4:
        self.ori_gesture = current_gesture
    return self.ori_gesture
```

3.7 Execute Command According to detected Gestures

The proposed AI virtual mouse system is based on the frames that have been captured by the webcam in a laptop or PC. By using the Python computer vision library OpenCV, the video capture object is created and the web camera will start capturing video, as shown in Figure 4. The web camera captures and passes the frames to the AI virtual system.

3.7.1 For the Mouse to Perform Left Button Click

If both the index finger with tip Id = 1 and the thumb finger with tip Id = 0 are up and the distance between the two fingers is lesser than 30px, the computer is made to perform the left mouse button click using the pynput Python package, as shown in Figures 24.

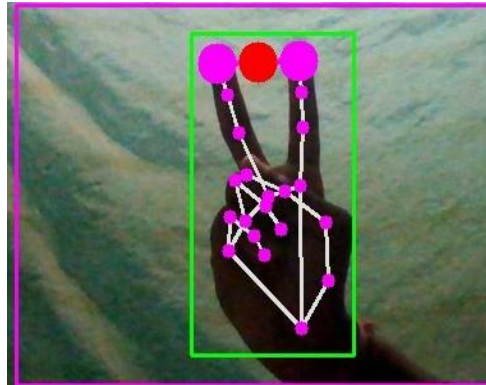


Figure 7: Gesture for the computer to perform left button click.

3.7.2 For the Mouse to Perform Right Button Click

If both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is lesser than 40 px, the computer is made to perform the right mouse button click using the pynput Python package, as shown in Figure 25.

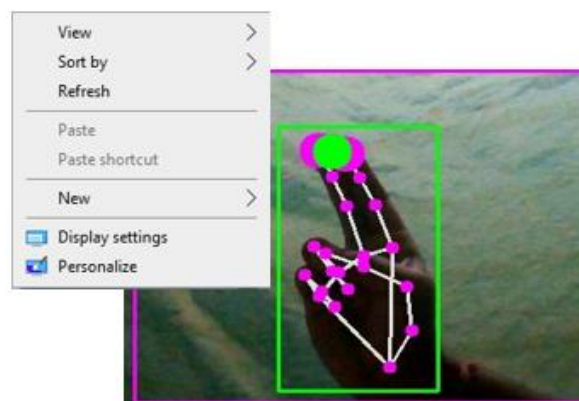


Figure 8: Gesture for the computer to perform right button click.

3.7.3 For the Mouse to Perform Scroll up Function

If both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is greater than 40 px and if the two fingers are moved up the page, the computer is made to perform the scroll up mouse function using the PyAutoGUI Python package, as shown in Figure 26.

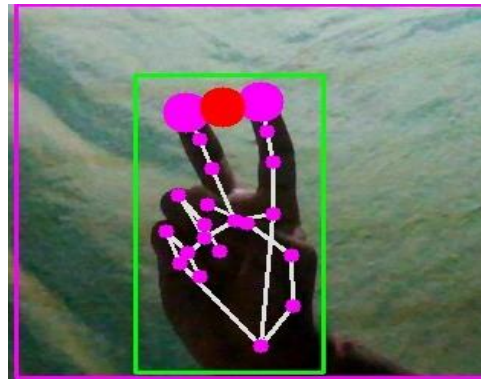


Figure 9: _Gesture for the computer to perform scroll up function.

```
# Executes commands according to detected gestures
class Controller:
    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        dist = round(((Controller.pinchstartycoord - hand_result.landmark[8].y) * 10, 1)
        return dist

    def getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord) * 10, 1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = sbcontrol.get_brightness() / 100.0
        currentBrightnessLv += Controller.pinchlv / 50.0
        if currentBrightnessLv > 1.0:
```

```

    currentBrightnessLv = 1.0
elif currentBrightnessLv < 0.0:
    currentBrightnessLv = 0.0
sbcontrol.fade_brightness(int(100 * currentBrightnessLv), start=sbcontrol.get_brightness())

def changesystemvolume():
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
    volume = cast(interface, POINTER(IAudioEndpointVolume))
    currentVolumeLv = volume.GetMasterVolumeLevelScalar()
    currentVolumeLv += Controller.pinchlv / 50.0
    if currentVolumeLv > 1.0:
        currentVolumeLv = 1.0
    elif currentVolumeLv < 0.0:
        currentVolumeLv = 0.0
    volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

def scrollVertical():
    pyautogui.scroll(120 if Controller.pinchlv > 0.0 else -120)

def scrollHorizontal():
    pyautogui.keyDown('shift')
    pyautogui.keyDown('ctrl')
    pyautogui.scroll(-120 if Controller.pinchlv > 0.0 else 120)
    pyautogui.keyUp('ctrl')
    pyautogui.keyUp('shift')

```

3.8 Stabilize The Cursor

The typical desktop Windows, Icons, Menus, and Pointer (WIMP) interface is a special case of image plane selection. Although not a true three dimensional representation of a desktop, common operating systems such as Microsoft Windows and the Windows system typically display windows in a depth ordered stack with the currently active window on top. The mouse cursor in these systems almost always remains visible in front of the desktop content. If we consider the mouse as residing somewhere in between the user viewpoint and the desktop windows, then we can see that selecting the first visible object with a ray cast from the viewpoint through the tip of the cursor emulates the familiar behavior of the mouse. The desktop situation is a simplified case of image-plane selection where the display is monoscopic, the user viewpoint remains fixed and the virtual cursor is restricted to moving in a plane a fixed distance between the viewpoint and desktop content.

Virtual mouse stabilization is modulated by placing the actual cursor plane a fraction of the distance d_c between effective cursor plane and content underneath the virtual cursor.

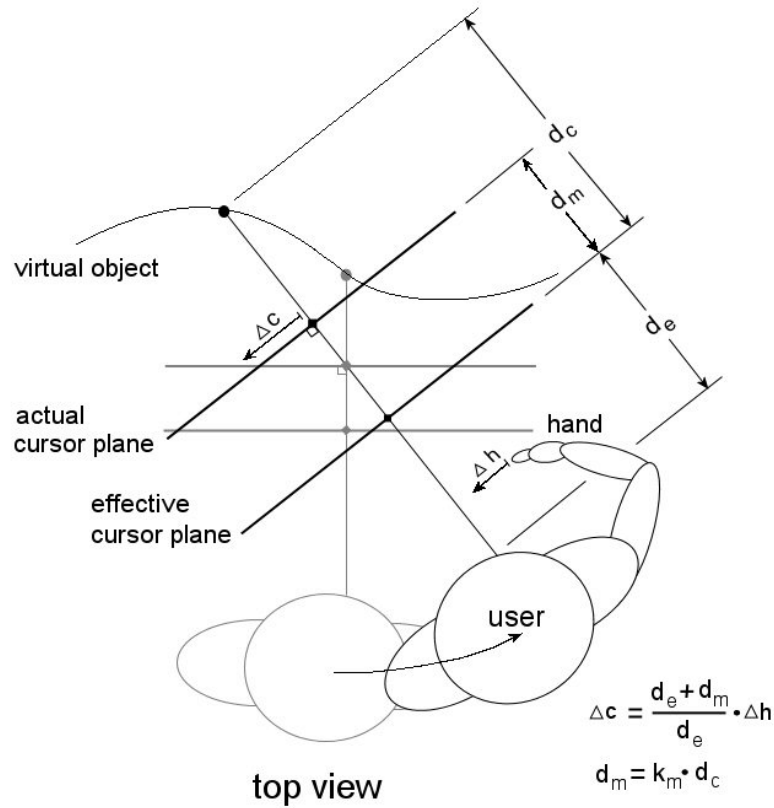


Figure 10: Virtual mouse stabilization modulated by placing the actual cursor plane

```

# Locate Hand to get Cursor Position
# Stabilize cursor by Dampening
def get_position(hand_result):
    point = 9
    position = [hand_result.landmark[point].x, hand_result.landmark[point].y]
    sx, sy = pyautogui.size()
    x_old, y_old = pyautogui.position()
    x = int(position[0] * sx)
    y = int(position[1] * sy)
    if Controller.prev_hand is None:
        Controller.prev_hand = x, y
    delta_x = x - Controller.prev_hand[0]
    delta_y = y - Controller.prev_hand[1]

    distsq = delta_x ** 2 + delta_y ** 2
    ratio = 1
    Controller.prev_hand = [x, y]

    if distsq <= 25:
        ratio = 0
    elif distsq <= 900:
        ratio = 0.07 * (distsq ** (1 / 2))
    else:
        ratio = 2.1
    x, y = x_old + delta_x * ratio, y_old + delta_y * ratio
    return (x, y)

```

```

def pinch_control_init(hand_result):
    Controller.pinchstartxcoord = hand_result.landmark[8].x
    Controller.pinchstartycoord = hand_result.landmark[8].y
    Controller.pinchlv = 0
    Controller.prevpinchlv = 0
    Controller.framecount = 0

```

3.9 Hold The Final Position To Change Status

At this stage system will wait for 5 frames and they perform the operations. If the program execute the gesture immediately many actions will not be recognizable by the system and the mouse does not work properly.

```

# Hold final position for 5 frames to change status
def pinch_control(hand_result, controlHorizontal, controlVertical):
    if Controller.framecount == 5:
        Controller.framecount = 0
        Controller.pinchlv = Controller.prevpinchlv

        if Controller.pinchdirectionflag == True:
            controlHorizontal() # x

        elif Controller.pinchdirectionflag == False:
            controlVertical() # y

    lvx = Controller.getpinchxlv(hand_result)
    lvy = Controller.getpinchylv(hand_result)

    if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
        Controller.pinchdirectionflag = False
        if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
            Controller.framecount += 1
        else:
            Controller.prevpinchlv = lvy
            Controller.framecount = 0

    elif abs(lvx) > Controller.pinch_threshold:
        Controller.pinchdirectionflag = True
        if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
            Controller.framecount += 1
        else:
            Controller.prevpinchlv = lvx
            Controller.framecount = 0

def handle_controls(gesture, hand_result):
    x, y = None, None
    if gesture != Gest.PALM:
        x, y = Controller.get_position(hand_result)

    # flag reset
    if gesture != Gest.FIST and Controller.grabflag:
        Controller.grabflag = False
        pyautogui.mouseUp(button="left")

    if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:

```

```

    Controller.pinchmajorflag = False

    if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
        Controller.pinchminorflag = False

    # implementation
    if gesture == Gest.V_GEST:
        Controller.flag = True
        pyautogui.moveTo(x, y, duration=0.1)

    elif gesture == Gest.FIST:
        if not Controller.grabflag:
            Controller.grabflag = True
            pyautogui.mouseDown(button="left")
            pyautogui.moveTo(x, y, duration=0.1)

    elif gesture == Gest.MID and Controller.flag:
        pyautogui.click()
        Controller.flag = False

    elif gesture == Gest.INDEX and Controller.flag:
        pyautogui.click(button='right')
        Controller.flag = False

    elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
        pyautogui.doubleClick()
        Controller.flag = False

    elif gesture == Gest.PINCH_MINOR:
        if Controller.pinchminorflag == False:
            Controller.pinch_control_init(hand_result)
            Controller.pinchminorflag = True
            Controller.pinch_control(hand_result, Controller.scrollHorizontal, Controller.scrollVertical)

    elif gesture == Gest.PINCH_MAJOR:
        if Controller.pinchmajorflag == False:
            Controller.pinch_control_init(hand_result)
            Controller.pinchmajorflag = True
            Controller.pinch_control(hand_result, Controller.changesystembrightness,
            Controller.changesystemvolume)

class GestureController:
    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None # Right Hand by default
    hr_minor = None # Left hand by default
    dom_hand = True

    def __init__(self):
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT = GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH = GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)

```

```

def classify_hands(results):
    left, right = None, None
    try:
        handedness_dict = MessageToDict(results.multi_handedness[0])
        if handedness_dict['classification'][0]['label'] == 'Right':
            right = results.multi_hand_landmarks[0]
        else:
            left = results.multi_hand_landmarks[0]
    except:
        pass

    try:
        handedness_dict = MessageToDict(results.multi_handedness[1])
        if handedness_dict['classification'][0]['label'] == 'Right':
            right = results.multi_hand_landmarks[1]
        else:
            left = results.multi_hand_landmarks[1]
    except:
        pass

    if GestureController.dom_hand == True:
        GestureController.hr_major = right
        GestureController.hr_minor = left
    else:
        GestureController.hr_major = left
        GestureController.hr_minor = right

def start(self):

    handmajor = HandRecog(HLabel.MAJOR)
    handminor = HandRecog(HLabel.MINOR)

    with mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
        while GestureController.cap.isOpened() and GestureController.gc_mode:
            success, image = GestureController.cap.read()

            if not success:
                print("Ignoring empty camera frame.")
                continue

            image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            results = hands.process(image)

            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            if results.multi_hand_landmarks:
                GestureController.classify_hands(results)
                handmajor.update_hand_result(GestureController.hr_major)
                handminor.update_hand_result(GestureController.hr_minor)

                handmajor.set_finger_state()
                handminor.set_finger_state()
                gest_name = handminor.get_gesture()

```



```

    if gest_name == Gest.PINCH_MINOR:
        Controller.handle_controls(gest_name, handminor.hand_result)
    else:
        gest_name = handmajor.get_gesture()
        Controller.handle_controls(gest_name, handmajor.hand_result)

    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
    else:
        Controller.prev_hand = None
        cv2.imshow('Gesture Controller', image)
        if cv2.waitKey(5) & 0xFF == 13:
            break
    GestureController.cap.release()
    cv2.destroyAllWindows()

# uncomment to run directly
gc1 = GestureController()
gc1.start()

```

4. RESULT AND DISCUSSION

4.1 overview

The Virtual Mouse Colour Recognition requires being able to recognize most of the colours provided by the users with high accuracy, consistency, and minimal performance impact on other processes. However, the recognition results may vary whenever the qualities of the captured frames have changed, as it may be affected by different situation in terms of environment, brightness, and the weather. The following describes the situations which may result in false detection and/or any other problem that may occur during recognition phase:

- a) The real-time images are taken under dark or bright environment condition.
- b) The real-time images are taken in a colour conflicts background.
- c) The users interact with the program in near or far distance.
- d) The real-time images are rotated in a clockwise or anti-clockwise rotation. In order to achieve greater accuracy and consistency throughout the whole recognition cycle, verification plan is required to be implemented in order for the program to perform flawlessly. The verification plans is as follows:

In order to achieve accuracy, and consistency of the Virtual Mouse colour recognition, testing phase have been conducted on various scenarios.

4.2 Performance in Various Environments

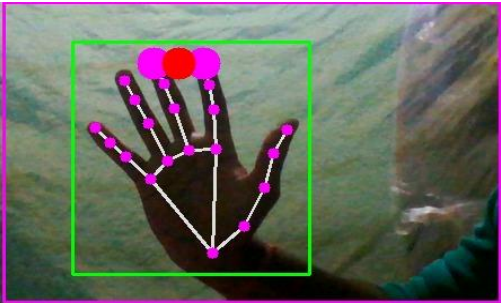
The purpose of testing phase is to ensure that the final deliverable is able to perform flawlessly in terms of accuracy, consistency, and performance. To achieve that, the program has to be able to recognize the colours input provided by the users with minimal adjustment, provide that the colours are thoroughly calibrated at first hand. Furthermore, the program is required to be able to execute the mouse functions efficiently and accurately as well.

Table 5: Verification Plan P1

Procedure Number	P1
Method	Testing
Applicable Requirements	Recognize finger gesture input in a dark environment
Purpose / Scope	To recognize gesture input in a dark environment
Item Under Test	Real-time image captured from webcam
Precautions	The targeted gesture must be calibrated first
Special Conditions/Limitations	If the Gesture are not calibrated, the program will not be able to detect the targeted gesture accurately.
Equipment/Facilities	Laptop
Data Recording	None
Acceptance Criteria	The gesture are detected successfully
Procedure	Run the program in the dark room. Calibrate the desired gesture. Performs gesture recognition by attempting to execute mouse functions by interacting with the programe.
Troubleshooting	Modify the HSV track-bars
Post-Test Activities	None

The following describes the outcome of the program testing in various environments:

Table 6: Different brightness testing results

Brightness	Results
Normal Environment	
	All Gestures are successfully recognized. The three highlighted circles indicate that the targeted finger are identified, compared, and execute a cordingly.
Brighter Environment	

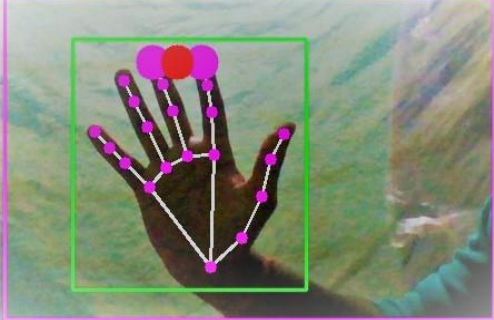
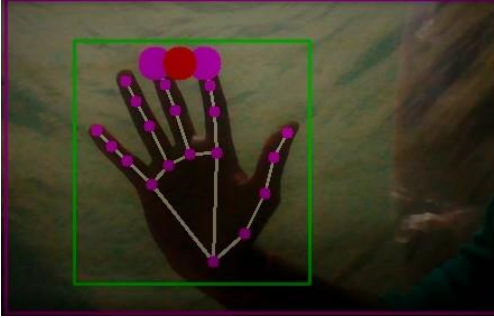
	All Gestures are successfully recognized. The three highlighted circles indicate that the targeted finger are identified, compared, and execute accordingly.
Dark Environment	
	All Gestures are successfully recognized. The three highlighted circles indicate that the targeted finger are identified, compared, and execute accordingly.

Table 7: Verification Plan P2

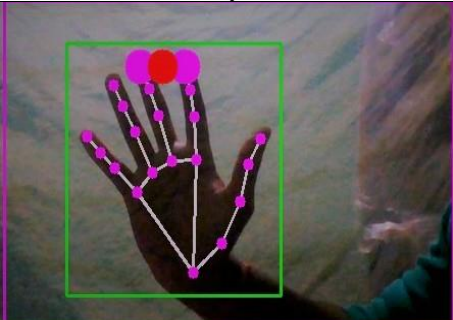
Procedure Number	P3
Method	Testing
Applicable Requirements	Recognize gesture input in near or far distance.
Purpose / Scope	To recognize gesture input in near or far distance.
Item Under Test	Real-time image captured from webcam
Precautions	The targeted gesture must be calibrated first
Special Conditions/Limitations	If the gesture are not calibrated, the program will not be able to detect the targeted gesture accurately.
Equipment/Facilities	Laptop
Data Recording	None
Acceptance Criteria	The gesture are detected successfully regardless of distance.
Procedure	Run the program. Calibrate the desired gestures. Performs gesture recognition by attempting to execute mouse functions by interacting with the program in near or far distance.

Rotation

Table 8: Verification Plan P3

Procedure Number	P4
Method	Testing
Applicable Requirements	Recognize colours input in tilted or rotated angle.
Purpose / Scope	To recognize colours input in tilted or rotated angle.
Item Under Test	Real-time image captured from webcam
Precautions	The targeted colours must be calibrated first
Special Conditions/Limitations	If the colours are not calibrated, the program will not be able to detect the targeted colours accurately.
Equipment/Facilities	Laptop
Data Recording	None
Acceptance Criteria	The colours are detected successfully
Procedure	<p>Flip or tilt the webcam</p> <p>Run the program.</p> <p>Calibrate the desired colours.</p> <p>Performs colour recognition by attempting to execute mouse functions by interacting with the.</p>
Troubleshooting	Modify the HSV track-bars
Post-Test Activities	None

Table 9: Distance testing results

Distance	
Near (15cm away from webcam)	
	<p>Results:</p> <p>All colours are successfully recognized. The three highlighted squares indicate that the targeted colours are identified, compared, and execute accordingly</p>

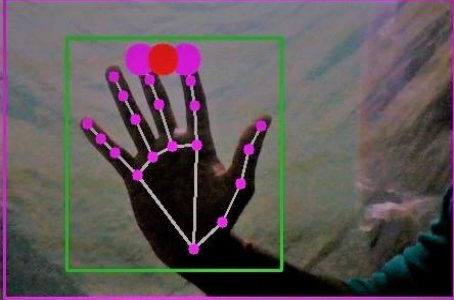
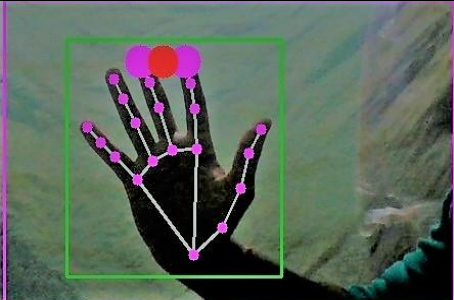
Far (25cm away from webcam)	
	Results: All colours are successfully recognized. The three highlighted squares indicate that the targeted colours are identified, compared, and execute accordingly.
Farther (35cm away from webcam)	
	Results: All colours cannot be recognized. Reason: The area of targeted colours were too small and was filtered off.

Table10: Experimental results & accuracy

Finger Tip Gesture	Mouse Function Performed	Accuracy (%)
Tip Id 1 or both tip Id's 1 and 2 are up	Curser movement	100
Tip Id 0 and 1 are up and the distance between the fingers is <15	Left click	95
Tip Id 0 and 1 are up and the distance between the fingers is <20	Right click	98
Tip Id 0 and 1 are up and the distance between the fingers is >20 and both fingers are moved down	Scroll up	100
Tip Id 0 and 1 are up and the distance between the figures is <20 and both fingers are moved up	Scroll down	100
All five tip Id's 0,1,2,3,4 are up	No action	100
Result		100

The results shows that there is no significant difference between physical mouse and virtual mouse operation. In our proposed system we can perform mouse operations with 100 percent accuracy .The results doesn't show any difference between fainted light and normal light. We shall first discuss the fingertip tracking preference in different lighting condition, noisy background and also in distant tracking conditions. In this analysis we record rapid gestures performance with their percentage of accuracy. In figure 9 the line graph shows the accuracy

level of different performed actions in our virtual mouse system. In Figure11 our experimental results compared to another approaches using gestures for virtual mouse systems. But in our proposed system we can perform mouse operation with high accuracy.

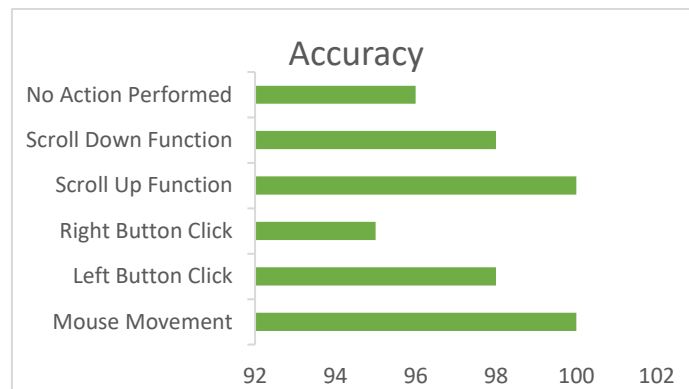


Fig 11: Graph of Accuracy for different performed actions

In Figure12 our experimental results compared to another approaches using gestures for virtual mouse systems. But in our proposed system we can perform mouse operation with high accuracy.

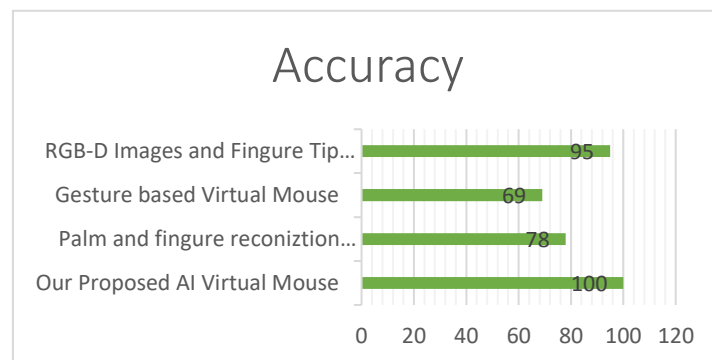


Fig 12: Accuracy graph for different available mouse system

5. CONCLUSION

In conclusion, it's no surprise that the physical mouse will be replaced by a virtual non-physical mouse in the Human-Computer Interactions (HCI), where every mouse movement can be executed with a swift of your fingers everywhere and anytime without any environmental restrictions. This project had developed a colour recognition program with the purpose of replacing the generic physical mouse without sacrificing the accuracy and efficiency, it is able to recognize colour movements, combinations, and translate them into actual mouse functions. Due to accuracy and efficiency playing an important role in making the program as useful as an actual physical mouse, a few techniques had to be implemented.

First and foremost, the coordinates of the colours that are in charge of handling the cursor movements are averaged based on a collection of coordinates, the purpose of this technique is to reduce and stabilize the sensitivity of cursor movements, as slight movement might lead to unwanted cursor movements. Other than that, several colour combinations were implemented with the addition of distance calculations between two colours within the combination, as different distance triggers different mouse functions. The purpose of this implementation is to promote convenience in controlling the program without much of a hassle. Therefore, actual mouse functions can be triggered accurately with minimum trial and errors. Furthermore, to promote efficient and flexible tracking of colours, a calibration phase was implemented, this allows the users to choose their choices of colours on different mouse functions, as long as the selected colours don't fall within the same/similar RGB values (e.g. blue and sky-blue). Other than that, adaptive calibrations were also implemented as well, it basically allows the program to save different sets of HSV values from different angles where it will be used during the recognition phase.

In Overall, the modern technologies have come a long way in making the society life better in terms of productivity and lifestyle, not the other way around. Therefore, societies must not mingle on the past technologies while reluctant on relating work.

6. LIMITATION

In this project, there are several existing problems that may hinder the results of colour recognitions. One of the problems is the environmental factor during the recognition phase takes place. The recognition process are highly sensitive on the intensity of brightness, as immense brightness or darkness may cause the targeted colours to be undetected within the captured frames. Besides that, distance is also the one of the problem that may affect the colour recognition results, as the current detection region can support up to 25cm radius, any display of colours exceed the mentioned distance will be considered as a noise and be filtered off.

Furthermore, the performance of the program are highly dependent on the users' hardware, as processor speed and/or resolutions taken from the webcam could have an effect on performance load. Therefore, the slower the processing speed and/or the higher the resolutions, the longer time are required to process a single frame.

7. FUTUTRE WORKS

There are several features and improvements needed in order for the program to be more user friendly, accurate, and flexible in various environments. The following describes the improvements and the features required:

a) Smart Recognition Algorithm

Due to the current recognition process are limited within 25cm radius, an adaptive zoom-in/out functions are required to improve the covered distance, where it can automatically adjust the focus rate based on the distance between the users and the webcam.

b) Better Performance

The response time are heavily rely on the hardware of the machine, this includes the processing speed of the processor, the size of the available RAM, and the available features of webcam. Therefore, the program may have better performance when it's running on a decent machines with a webcam that performs better in different types of lightings.

8. APPLICATIONS

The AI virtual mouse system is useful for many applications; it can be used to reduce the space for using the physical mouse, and it can be used in situations where we cannot use the physical mouse. The system eliminates the usage of devices, and it improves the human-computer interaction.

Major applications:

- (i) The proposed model has a greater accuracy of 99% which is far greater than the that of other proposed models for virtual mouse, and it has many applications
- (ii) Amidst the COVID-19 situation, it is not safe to use the devices by touching them because it may result in a possible situation of spread of the virus by touching the devices, so the proposed AI virtual mouse can be used to control the PC mouse functions without using the physical mouse(iii)The system can be used to control robots and automation systems without the usage of devices(iv)2D and 3D images can be drawn using the AI virtual system using the hand gestures(v)AI virtual mouse can be used to play virtual reality- and augmented reality-based games without the wireless or wired mouse devices(vi)Persons with problems in their hands can use this system to control the mouse functions in the computer(vii)In the field of robotics, the proposed system like HCI can be used for controlling robots(viii)In designing and architecture, the proposed system can be used for designing virtually for prototyping

9. REFERENCES

1. Banerjee, A., Ghosh, A., Bharadwaj, K., & Saikia, H. (2014). Mouse control using a web camera based on colour detection. *arXiv preprint arXiv:1403.4722*.
2. Chu-Feng,L. (2008).Portable Vision-Based HCI. [online] Available at: http://www.csie.ntu.edu.tw/~p93007/projects/vision/vision_hci_p93922007.pdf [Accessed 25 Aug. 2015].
3. Park, H. (2008). A method for controlling mouse movement using a real-time camera. *Brown University, Providence, RI, USA, Department of computer science*.
4. Kumar N, M. (2011). *Manual Testing: Agile software development*. [online] Manojforqa.blogspot.com. Available at: <http://manojforqa.blogspot.com/2011/09/agile-software-development.html> [Accessed 27 Aug. 2015].
5. Niyazi, K. (2012). Mouse Simulation Using Two Coloured Tapes. *IJIST*, 2(2), pp.57- 63.
6. Sekeroglu, K. (2010). Virtual Mouse Using a Webcam. [online] Available at: http://www.ece.lsu.edu/ipi/SampleStudentProjects/ProjectKazim/Virtual%20Mouse%20Using%20a%20Webcam_Kazim_Sekeroglu.pdf [Accessed 29 Aug. 2015].
7. Tutorialspoint.com, (n.d.). *SDLC - Agile Model*. [online] Available at: http://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm [Accessed 27 Aug. 2015].
8. Tabernae.com, (n.d.). *Software Life Cycle / Web Development Outsourcing/ IT Offshore Outsourcing*. [online] Available at: <http://www.tabernae.com/process.aspx> [Accessed 28 Aug. 2015].
9. Zhengyou, Z., Ying, W. and Shafer, S. (2001). Visual Panel: Virtual Mouse, Keyboard and 3D Controller with an Ordinary Piece of Paper. [online] Available at [http://research.microsoft.com/en-us/um/people/zhang/Papers/PUI2001- VisualPanel.pdf](http://research.microsoft.com/en-us/um/people/zhang/Papers/PUI2001-VisualPanel.pdf) [Accessed 25 Aug. 2015].

A Vision Base Application For Virtual Mouse Interface Using Hand Gesture

Sankha Sarkar

Department of Computer Science and Engineering
JIS College of Engineering
Kalyani, West Bengal, India

Sourav Sahoo

Department of Computer Science and Engineering
JIS College of Engineering
Kalyani, West Bengal, India

Indrani Naskar

Department of Computer Science and Engineering
JIS College of Engineering
Kalyani, West Bengal, India

Sayan Ghosh

Department of Computer Science and Engineering
JIS College of Engineering
Kalyani, West Bengal, India

Sumanta Chatterjee

Department of Computer Science and Engineering
JIS College of Engineering
Kalyani, West Bengal, India

Abstract– This paper proposes a way of controlling the position of the mouse cursor with the help of a fingertip without using any electronic device. We can be performing the operations like clicking and dragging objects easily with help of different hand gestures. The proposed system will require only a webcam as an input device. Python and OpenCV will be required to implement this software. The output of the webcam will be displayed on the system's screen so that it can be further calibrated by the user. The python dependencies that will be used for implementing this system are NumPy, Autopy and Mediapipe.

Keywords :- OpenCv, Autopy, Mediapipe Numpy; Calibrated, Gesture.

I. INTRODUCTION

With the developing technologies within the twenty-one century, the areas of virtual reality devices that we are using in our daily lifestyle, these devices are become more compact with wireless technologies like Bluetooth. This paper shows an AI virtual mouse system that take input of hand gestures and detection for fingertip movement to perform mouse operations in computer by using computer vision. The most important objective of the proposed system is to perform mouse operation like single click, double click, drag and scroll by employing an in-built camera or a web-camera within the computer rather than using a traditional mouse. Hand gesture and fingertip detection by using computer vision is employed as an individual's and Computer Interaction, simply referred to as HCI. Once we are using the AI virtual mouse system, we will track hand gestures and fingertips by using an in-built camera or web camera and perform the mouse cursor operations like scrolling, single click, double click. While employing a wireless or a Bluetooth mouse, that's not purely wireless some devices like the mouse, little dongle we've to attach to the PC, and also a battery cell to power the mouse. But in this system, the user uses his/her hand to manage the pc mouse operations. During this time, the camera captures

images and store temporary and processes the frames and then recognizes the predefined hand gesture operations. Python programming language is mostly used for developing this kind virtual mouse system, and OpenCV library is additionally used for contract computer vision. The system makes use of the well-known python libraries like MediaPipe and Autopy for the tracking of the hands and fingertip in real-time and also, Pypot, and PyAutoGUI packages were used for tracking the finger traveling in the screen for performing operations like left-click, right-click, drag and scrolling functions. The output results and accuracy level of the proposed model showed very high, and also the proposed model can work in the dark envirmnt as well as 60cm far from the camera in real-world applications with the employment of a CPU without the employment of a GPU.

II. ALGORITHM USED FOR HAND TRACKING

A. Mediapipe

It is a cross-platform framework that's mostly used for building multimodal pipeline in machine learning, and it's an open-source framework by Google. MediaPipe framework is most useful for cross-platform development work since this framework is constructed by statistic data. The MediaPipe framework works as multimodal, which implies this framework is applied to face detection, facemesh, iris scanner pose detection, handdetection, hair segmentation object detection, motion tracking and objection. The MediaPipe framework is that the best option to the developer for building, analyzing and designing the systems performance in the form of graphs, and it's also been used for developing various application and systems within the cross-platform (Android, IOS, web, edges devices). The involving steps in our proposed system uses MediaPipe framework as pipeline structure configuration. This pipeline structure create and run in various platforms which allowing scalability in mobile and desktop system also. The MediaPipe package gives us three reliability, they're like performance, evaluation, and creation which isretrievingby the sensor data, and using a set of components. Mediapipe processing inside a graph which is defines flowing of packet between nodes.

Apipelinesstructures could be a graph that consists of components called calculators, where each calculator is connected by specific streams during which the information packets flow. Developers also are ready to replace or define custom calculators anywhere within the graph for creating their custom applications. The calculators and streams are combined to form a data-flow diagram. The graph (Figure 1) is made by using the MediaPipe module. Each calculator modules runs according to the timestamp become available.it used real-time graph system to define when the timestamp become available. A model of hand landmark which is consists of 21 locating points, as shown in Figure 1.

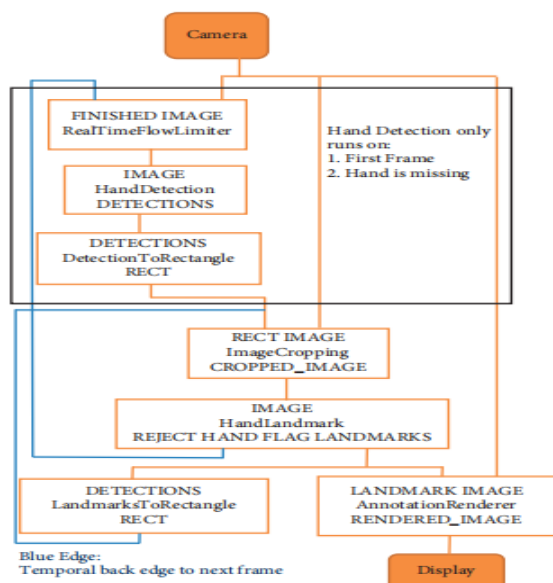


Fig. 1: MediaPipe hand recognition flowchart

B. Autopy

It is cross platform GUI automation module for python. That module keeps tracks of finger in this proposed system. Autopy track the fingertip and tell us which finger is up and which one is down .This process is happing by giving system an input in the form of 0 and 1.From this module the mediapipe module takes output and done the process and give the proper output. From this output opencv visualize everything and create the proper frame from image.

C. OpenCv

Open source computer vision library is a programming functions written in C++ mainly aimed at computer vision. It was licensed by Apache and introduced by Intel. This library is cross platform and free to use.it provides a features of Gpu acceleration for real time operation.opencv are used in wide areas like as 2D & 3D feature toolkits, Facial and gesture recognition system, mobile robotics.

III. METHODOLOGY

A. Importing the necessary library:

At very first install& import all the necessary library in preferred IDE.We are supposed to install onencv, numpy,mediapipe and autopy framework.After successfully install all the library in pycharm, our next is to import all that library into the code section so that we can make use this

library in our system.

B. Initializing the capturing device:

Our next task istoinitialize the connected image capture device. We are make use of system primary camera as `cap = cv2.VideoCapture(0)`.if any secondary image capturing device connected that is not used by this program.

C. Capturing the frames and processing:

The proposed AI virtual mouse system uses primary camera for capturing the frames. The video frames are processed and converted BGR to RGB format to find the hand in video frame. This color conversion is done by the following code:

```
check. img = cap.read()
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
lmlist = handLandmarks(imgRGB)
if len(lmlist) !=0:
    finger = fingers(lmlist)
    x1, y1 = lmlist[0][1:]
    x2, y2 = lmlist[12][1:]
    x3 = numpy.interp(x1, (75, 640 - 75), (0, wScr))
    y3 = numpy.interp(y1, (75, 480 - 75), (0, hScr))
```

D. Initializing mediapipe and capture window:

After that we are Initializing mediapipe which is responsible for tracking the hand gesture and other actions. Then we are setting the minimum confidence for hand detection and minimum confidence for hand tracking by `mainHand=initHand.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.8)`.

That means it gives a success rate of 80% according to the performed action by the main hand. After that mediapipe draw the axis line across the palm and middle of fingers. That keeps track about finger movement and create connection with neighbor finger. Then we define the axis of the hand in the 2D format by defining the x-axis and y-axis value. By the help of this axis we are keep track about the finger that which one is up and which one is down. We also keep a record of the previous frame of x-axis and y-axis values in the form of 0 and 1.

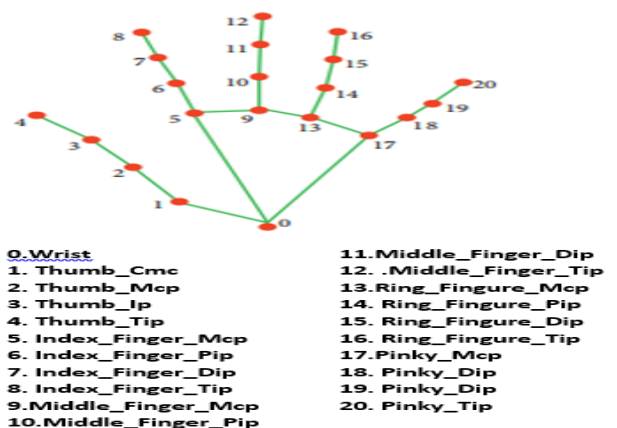


Fig. 2 : Co-ordinates or landmarks in the hand using Mediapipe

E. Defining hand landmarks:

After confirming the hand axis our next task is to define the full hand land mark capture. We are capture the hand land mark by the predefined mediapipe 21 locating point (Figure 2). These 21 joint or locating point are to be tracked by the mediapipe. If there is no landmark found then we print the default value that is empty. That's means no action need to be performed. Otherwise it returns the coordinate value of X,Y and Z axis. The thumb finger is not included in the hand landmark region. We are creating a loop in between 21 point and return the tip point value by drawing graph line in between finger. When we fold the finger or down the finger the lines are also down and the corresponding graph line are down. The gap n between the finger is not important in this point. The gap n between finger can become 5cm. Otherwise it get back to the initial position of finding. After locating height, width and center of the hand landmark we converts the decimal coordinate relative to the index according to the each image. After capturing the total hand landmarks a green rectangle box appeared around the hand where we can perform the mouse operation.

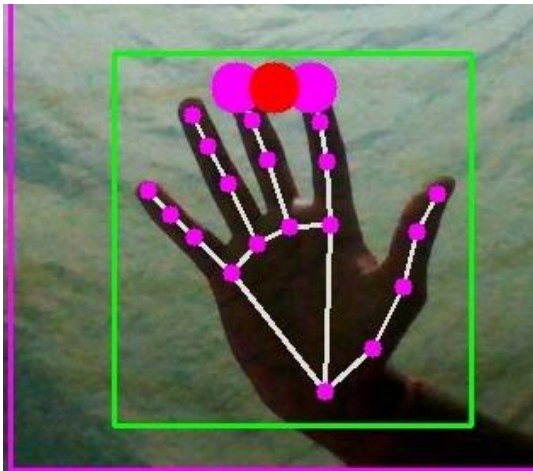


Fig3: capturing the hand landmarks

F. Defining the finger landmarks:

After initializing the axis point are defining the hand landmark. That is occupied the green dialog box and the finger height width. The height and width are measured by the captured image frame and counting the finger up down number by their movement. We are make use only the three finger. The thumb finger is not included in the hand landmark region. We are creating a loop in between 21 point and return the tip point value by drawing graph line in between finger. Otherwise it get back to the initial position of finding. After locating height, width and center of the hand landmark we converts the decimal coordinate relative to the index according to the each image. Here we are creating an array of size 4 which is storing the four index value of the different finger. The Tip_ID only considered the 4, 8,12,16,20 no index point. Then we are checking for the two condition (1) If thumb is up and (2) If finger are up except the thumb. In condition (1) we are checking `if landmarks[tipIds[0]][1] > lmList[tipIds[0] - 1][1]:`

Then append the fingerTip and returned to the finger landmark section. If condition (2) is satisfied then the

program checked for which finger is up by their corresponding index ID's

```
. for id in range(1, 5):
if landmarks[tipIds[id]][2] < landmarks[tipIds[id] - 3][2]:
fingerTips.append(1)
else:
finger Tips.append(0)
```

Otherwise it returned to the Fingertip function.

Color convert: Hand contour are the curve of the hand segmented image extracted points. In this section contours are detected using Moore neighbor algorithm. After extracting the contour from the binary image the palm and finger region area is selected. If second condition is satisfied that means if other finger is up excluding thumb finger then we proceeds the code to change the color. First we read the frame from the image and change the format of the frames from BGR to RGB.

```
if len(lmList) != 0:
x1, y1 = lmList[8][1:]
x2, y2 = lmList[12][1:]
```

It gets the index 8s and 12s for x and y axis values and that is also skip the index values because it starts with value 1. Then check for pointing finger is up and thumb finger is down .If this condition is true then we performs two operations (1) converts width of the window relative to the screen width (2)converts height of the window relative to the screen height. And keep an eye on the pervious values of X and Y .`autopy.mouse.move(wScr-cX, cY)` It moves the mouse cursor across the x3 and y3 values by inverting the direction. `if finger[1] == 0 and finger[0] == 1:` It checks to see if the pointer finger is down and thumb finger is up then it performs left click. After end of this process the obtained pixels are stored in an ordered array. These values are used for gesture detection. The following code is help to detect the color region:

```
check. img = cap.read()
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
lmList = handLandmarks(imgRGB)
```

```
if len(lmList) != 0:
x1, y1 = lmList[0][1:]
x2, y2 = lmList[12][1:]
x3 = numpy.interp(x1, (75, 640 - 75), (0, wScr))
y3 = numpy.interp(y1, (75, 480 - 75), (0, hScr))
```

if results.multi_hand_landmarks:

```
for num, hand in enumerate(results.multi_hand_landmarks):
mp_drawing.draw_landmarks(image, hand,
mp_hands.HAND_CONNECTIONS, mp_drawing.DrawingSpec(
color=(121, 22, 76), thickness=2, circle_radius=4),
mp_drawing.DrawingSpec(color=(250, 44, 250),
thickness=2, circle_radius=2),
```

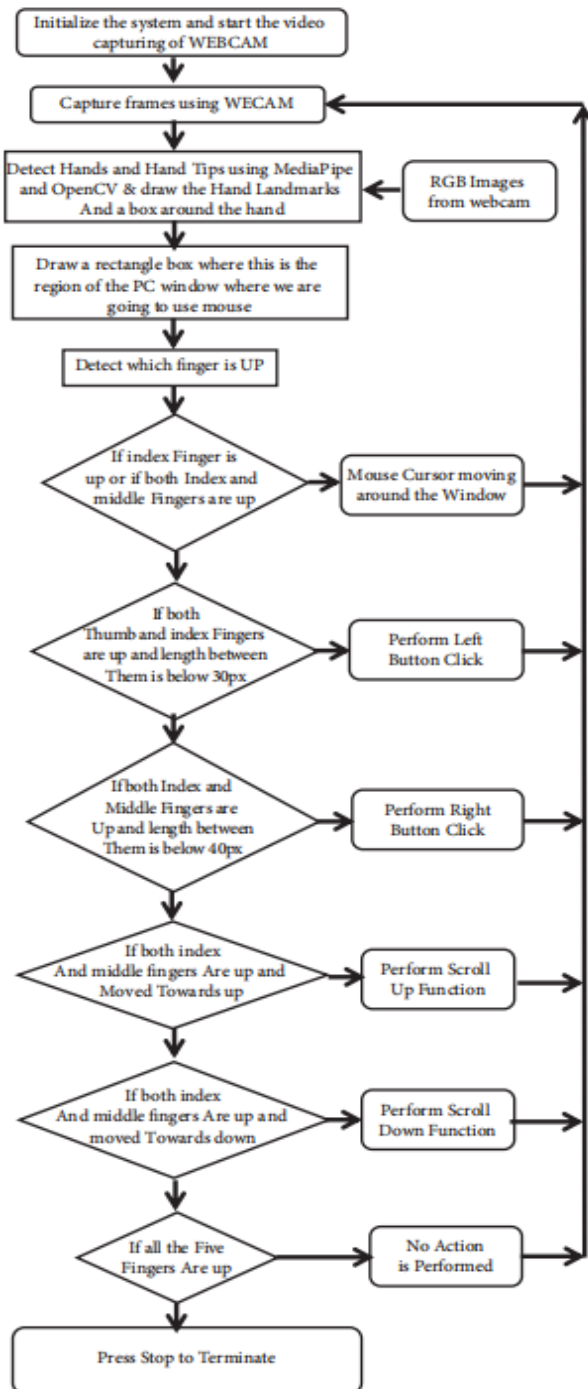



Fig. 4 : Flowchart of the proposed AI virtual mouse system

IV. EXPERIMENTAL RESULTS

In this paper, we are proposed a finger gesture based mouse system. The hand region of image frame and the center of the palm are first extracted from 3D structured image by the mediapipe framework. Then hand contours are extracted and show a graph line in 21 region point in hand. The autopsy algorithm keeps tracks the finger movement and count the up finger by FingertipID python variable. Finally, to control the mouse cursor based on the finger location and their gesture. In our research we are considering the four function: cursor movement, left click, right click and scroll up-down. The fully explained working flow diagram of our

proposed system is shown Figure 4. To explore working condition of gestures with real-time tracking, we investigate this system in some complicated cases, e.g., maximum distant from the camera, low light condition, dark or busy background during the tracking. The proposed can detect one person hand at a time. Unlike existing systems, this study uses only single CPU and does not require any external GPU system or special devices.

For performing cursor moving operations if the index finger with Tipid=1 is up and middle perform no action, as shows in Figure 5.

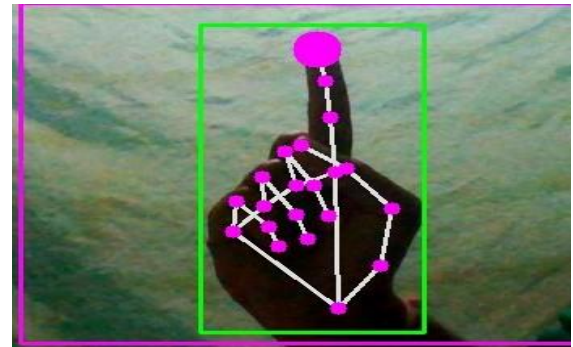


Fig. 5 : Gesture for cursor movement

For performing left click index finger with tipid=1 and middle finger with tipId=2 both are up and difference between two figure is 40pixels, as shows in Figure 6.

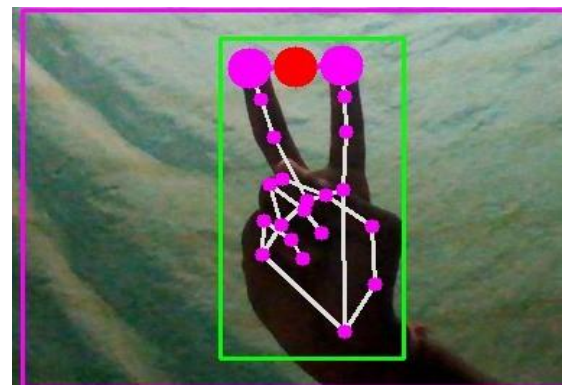


Fig. 6 : Gesture for left click

For performing right click if both the index finger tipid=1 and middle finger index id=2 are up and the distance between the two fingers is 40pixels, as shows in figure 7.

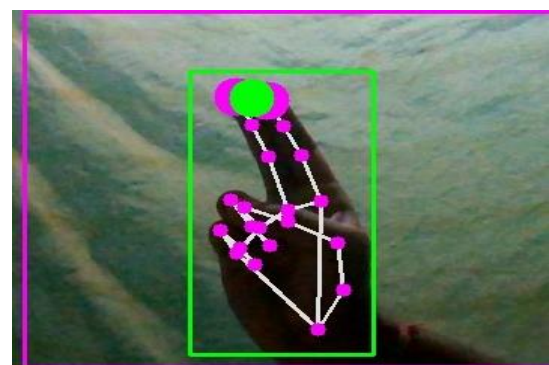


Fig. 7: Gesture for right click

For performing scroll up and down if the index finger with tipId=1 and middle finger with tipId=2 and distant between two finger is greater than 40Pixels, as shows in Figure 9.

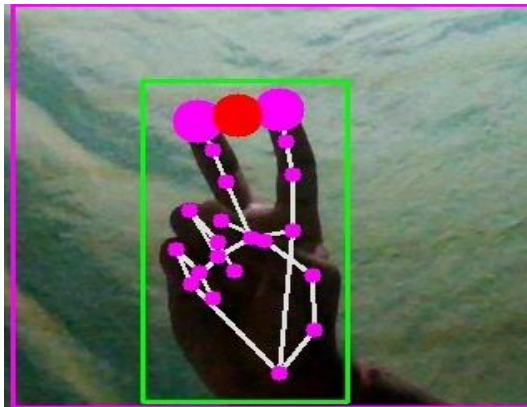


Fig. 8 : Gesture for scroll operation

We suggest the scroll operation with two finger for smooth operation but in this system we can scroll by three finger also. With three finger it is very hard to differentiate between two finger and three finger if the gestures are fast.

Finger Tip Gesture	Mouse Function Performed	Accuracy (%)
Tip Id 1 or both tip Id's 1 and 2 are up	Cursor movement	100
Tip Id 0 and 1 are up and the distance between the fingers is <15	Left click	95
Tip Id 0 and 1 are up and the distance between the fingers is <20	Right click	98
Tip Id 0 and 1 are up and the distance between the fingers is >20 and both fingers are moved down	Scroll up	100
Tip Id 0 and 1 are up and the distance between the figures is <20 and both fingers are moved up	Scroll down	100
All five tip Id's 0,1,2,3,4 are up	No action	100
Result		100

Table1: Experimental results

Table 1 shows the experimental test results of our AI virtual mouse system. The results shows that there is no significant difference between physical mouse and virtual mouse operation. In our proposed system we can perform mouse operations with 100 percent accuracy .The results doesn't show any difference between fainted light and normal light. We shall first discuss the fingertip tracking preference in different lighting condition, noisy background and also in distant tracking conditions. In this analysis we record rapid gestures performance with their percentage of accuracy. In figure 9 the line graph shows the accuracy level of different performed actions in our virtual mouse system. In Figure10 our experimental results compared to another

approaches using gestures for virtual mouse systems. But in our proposed system we can perform mouse operation with high accuracy.

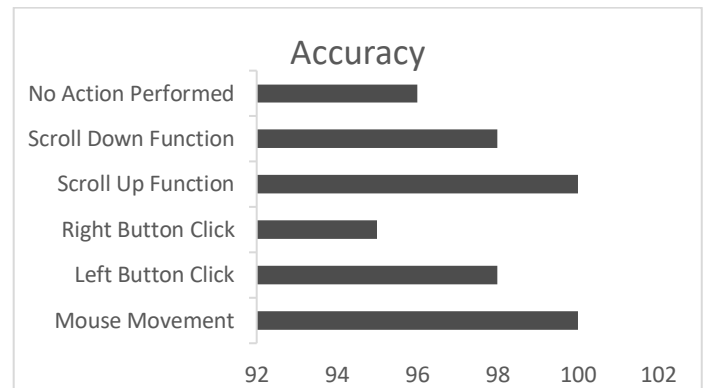


Fig. 9 :Graph of Accuracy for different performed actions

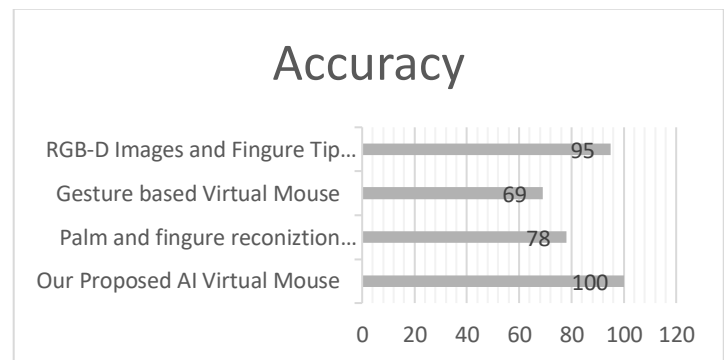


Fig. 10 : Accuracy graph for different available mouse systems

V. APPLICATIONS

The AI virtual mouse system is beneficial for several applications like, it is utilized in situations where we cannot use the physical mouse and it will be accustomed reduce the space and value for using the physical mouse, and it improves the human-computer interaction. Major applications:

- The proposed model encompasses a accuracy of 99% which is way better than the that of other proposed models for virtual mouse.
- In COVID-19 situation, it's not so safe to use the physical devices by touching them because it should possible to make situation of spread the virus, so this proposed AI virtual mouse system may be wont to control the PC mouse functions without using the physical mouse.
- The system may be modified and wont to control automation and robots systems without usage of devices.
- AI virtual mouse will be accustomed play augmented reality and computer game based games without the wired or wireless mouse devices
- within the field of robotics, the proposed system like HCI are often used for controlling robots
- In the field of designing and architecture, this system will be used for designing virtually.

VI. CONCLUSION

This paper presented a brandnew AI virtual mouse method using OpenCv, autopsy and mediapipe by the help of fingertip movement interacted with the computer in front of camera without using any physical device. The approach demonstrated high accuracy and highly accurate gesture eliminates in practical applications. The proposed method overcomes the limitations of already existing virtual mouse systems. It has many advantages, e.g., working well in low light as well as changing light condition with complex background, and fingertip tracking of different shape and size of finger with same accuracy. The experimental results shows that the approaching system can perform very well in real-time applications. We also intend to add new gesture on it to handle the system more easily and interact with other smart systems. It is possible to enrich the tracking system by using machine learning algorithm like Open pose. It is also possible to including body, hand and facial key points for different gestures.

REFERENCE

- [1.]J. Katona, "A review of human-computer interaction and video game research fields in cognitive Info Communications," Applied Sciences, vol. 11, no. 6, p. 2646,2021.
- [2.]D. L. Quam, "Gesture recognition with an information Glove," IEEE Conference on Aerospace and Electronics, vol. 2, pp. 755–760, 1990.
- [3.]D.-H. Liou, D. Lee, and C.-C. Hsieh, "A real time hand gesture recognition system using motion history image," in Proceedings of the 2010 2nd International Conference on Signal Processing Systems, July 2010.
- [4.]S. U. Dudhane, "Cursor system using hand gesture recognition," IJARCCCE, vol. 2, no. 5, 2013.