

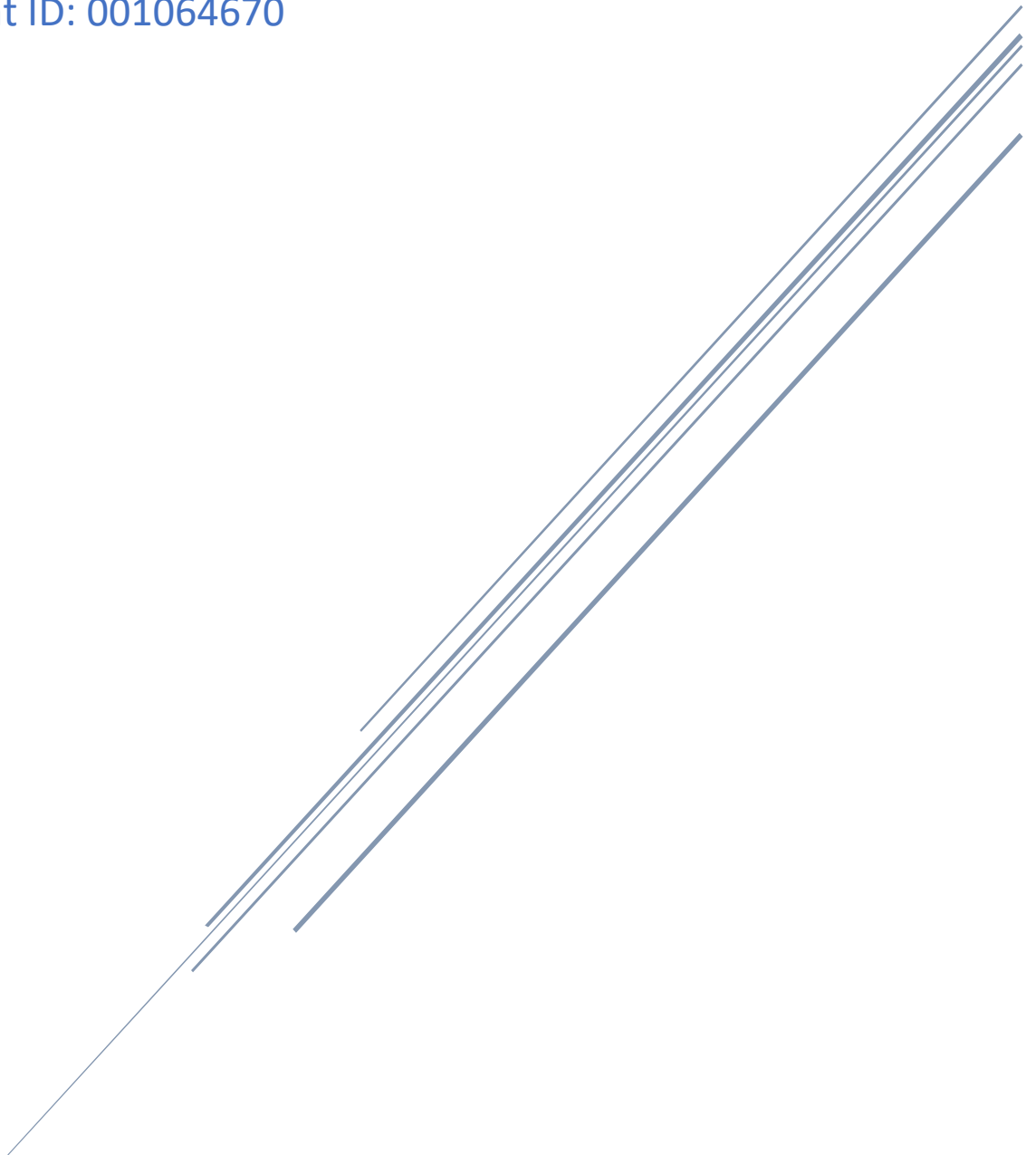
REAL-TIME HAND TRACKING AS AN INPUT METHOD

FINAL YEAR PROJECT REPORT

Author: Lakshman Thillainathan

Supervisor: Dr. Markus Wolf

Student ID: 001064670



University Of Greenwich
BEng (Hons) Software Engineering

ABSTRACT

In the last couple of years, hand recognition algorithms and systems have become more and more advanced. Many phones even have an element of hand recognition within them, such as some current Samsung phones are able to take a quick snapshot when a fully-opened hand is presented in front of them. The ability to track hands and make applications based around them is the future of computer vision products.

This paper will go into the several standards of current input methods for a computer and reasoning as to why hand tracking input can replace everyday peripherals. Then the creation of a software tool that will allow for users to use a standard webcam to mimic input onto the computer.

ACKNOWLEDGEMENTS

I would like to thank all the researchers that I was able to learn computer vision methodologies from as well as thank my supervisor Dr Markus Wolf for encouraging me through the project progress. I would also like to thank my family for being supportive during the project cycle. Also, I would like to thank my university for its numerous resources that I was able to make use of.

CONTENTS

Abstract	1
Acknowledgements	1
1 Introduction.....	5
1.1 Introduction & Problem Domain	5
1.2 Scope of the project.....	7
1.3 Report Structure	7
2 Background & Literature Review.....	8
2.1 Overview	8
2.2 Standard Methods of Input	8
2.2.1 Mouse	8
2.2.2 Keyboard.....	9
2.3 Non-Standard Methods of Input.....	9
2.3.1 Touch	9
2.3.2 Pens	10
2.3.3 Gestures.....	11
2.3.4 Microphones.....	12
2.3.5 Other.....	12
2.4 Hand Tracking.....	12
2.4 Existing Work.....	14
2.5 Fitts Law	18
3 Specification requirements	19
3.1 Functional Requirements.....	19
3.2 Non-Functional Requirements.....	20
4 Design	21
4.1 Assumptions	21
4.2 Tools & Technologies Used.....	21
4.2.1 Python.....	21
4.2.2 opencv	21
4.2.3 mediapipe framework	22
4.2.4 time.....	23
4.2.5 Math	23

4.2.6	numpy.....	23
4.2.7	os.....	23
4.2.8	sys	23
4.2.9	mouse (module).....	24
4.2.10	Keyboard (module).....	24
4.2.11	win32api and win32con.....	24
4.2.12	threading	24
4.2.13	pycaw.....	24
4.2.14	pyqt5 and qt designer.....	24
4.2.15	speechrecognition	25
4.2.16	camera & microphone(hardware).....	25
4.3	Design Principles.....	25
4.4	Design Decisions	26
4.5	System Flow Diagram	27
4.6	UML Design.....	28
4.7	Use Case Diagram	29
5	Legal, social, ethical and professional issues.....	30
6	Implementation.....	31
6.1	Initialise Webcam	31
6.2	Hand Landmarks Recognition	31
6.3	Mouse	35
6.2.1	cursor control	35
6.2.2	left click.....	36
6.2.3	Right click.....	36
6.2.4	Scroll	37
6.2.5	Double Click	37
6.2.6	Left click drag.....	37
6.3	Text Input.....	37
6.3.1	On-screen keyboard	37
6.3.2	speech recognition	38
6.4	Media Keys.....	38
6.4.1	Pause/play	38

6.4.2	Skip/Previous	39
6.4.3	Volume.....	39
6.5	Shortcuts	40
6.5.1	keyboard shortcuts	40
6.5.2	Windows ShortcuTS.....	40
6.6	Settings Storage	41
6.7	Graphical User Interface	41
6.7.1	The real-time hand input interface.....	41
6.7.1	The settings gui	42
7	Results & Evaluation	45
7.1	Results Evaluation	45
7.2	Comparison Against Existing Work	46
7.3	Evaluation of the software.....	46
7.3.1	Requirements	47
7.3.2	software design	49
7.4	Personal Evaluation	50
8	Conclusion & Future work	51
8.1	Conclusion.....	51
8.2	Future Work.....	51
	References	52
	Appendix.....	57

1 INTRODUCTION

1.1 Introduction & Problem Domain

Since the beginning of humankind, humans have been communicating through gestures, before the principle of speech and language was introduced. Gestures were the prime method of communication, and it is evident that ancient structures discovered throughout the years have been successful in transferring and storing massive amounts of data. This can be seen through the Egyptian era, where many symbols, structures, and art, was used to communicate the Egyptians' beliefs about the world and their attempt at understanding the world (Mohamed G.K, 2019). Gestures were also heavily used during the stone ages where communication solely relied on gestures and body language. This is also evident in stone drawings created in the stone age of humanity. Since this time humans have evolved to enable the use of their vocal cords more and gestures are still used today, but not as meaningful to some as to others. Gestures are natural and commonly occur as a reflex to certain situations and humans tend to express with gestures during conversations subconsciously. Gestures provide an additional "emotion" when someone is speaking, this can be seen anywhere, just turn on the TV and watch the news presenter subconsciously perform a gesture. Or when you are in a lecture watch the lecturers hands and body be used, for example the lecturer can be pointing at a segment of a screen, gesturing the class to look at that particular area.

But what exactly is considered a "gesture"? A "gesture" is sometimes used to refer to any of a variety of movements, ranging from arm and hand movement to the adjustment of posture and other fiddly movements people do whilst talking (W.M Roth, 2001). Nowadays some computer software has been created to understand and respond to touch through touchscreen and keyboards, and some software have speech recognition and all that's needed is a microphone. Researchers have been able to create software that aid in the recognition of hands.

Hand gesture recognition is an important part of human-computer interaction and has been a big topic among the subject for years. The first hand gesture recognition research dates back beyond the 1980s. The idea was that if a computer was able to recognise the way in which humans move and interact, it will allow for computers to accept it as input, similar to how speech recognition works. Mark Weiser, in 1991, wrote about his vision of ubiquitous computing, where computers would become a seamless part of everyday tasks. It is safe to say he was not incorrect, with the introduction of smart phones and how prominent they are in peoples lives. He also states "there are no systems that do

well with the diversity of inputs to be found in an embodied virtuality” (Mark Weiser, 1999), expressing his vision that input must move beyond just the screen and support a range of future computer forms.

Previously on some mobile devices, there were inherent gesture control commands that performed certain tasks. For example, in some Samsung Galaxy phones (Prior to the Samsung Galaxy S6) you were able to skip songs by simply hovering your hand right over the phone or go to the previous track by hovering left, known as “air-swipe”. There were other simple yet useful features like “Quick Glance” that would turn the screen on if the sensor recognised your eyes looking at the phone. There were several reasons as to why Samsung decided to stop air gesture on their devices, it was clunky and did not recognise certain gestures at times, used up extra space for a special camera, and meant air gestures would constantly run on the background which drained the battery. However, there have been no major attempt at making software compatible with a traditional computer.

Research has been done on hand gesture recognition and have been helpful in aiding in certain tasks and are generally specific to industries. For example, medical students and engineering students have accessed to software which create virtual environments for them to interact with. The use of hand gesture recognition is displayed in Wachs, J.P et al.’s article where in which they present a video-based hand gesture capture and recognition system used to manipulate MRI scans within a graphical interface. In hospital environments using keyboards and pointing devices (e.g mouse) by doctors and nurses are great but this is a common route in which infections can spread. Also having to wash your hands every time before using one of these devices consumes time that doctors could be using to help another patient.

For a typical mouse to be used the mouse must be connected to the computer through USB if wired. Some mice are able to use WIFI or Bluetooth to connect to a computer or even a dongle is needed, this just adds to the hardware components needed for a computer. Similar with keyboards they require some sort of hardware to be able to perform keyboard operations. The way in which people interact with computers is simply through the use of input peripherals. Recent advancements in technology have allowed for different input methods that will be discussed later on in the report.

A great example of a speech automation software is Alexa (by Amazon), Google assistant and Cortana (by Microsoft). These systems are able to comprehend a persons speech and convert their words into a command that is run in the background. These systems are programmed to handle multiple different tasks when given. They are able to perform keyboard input, store calendar information etc, becoming a very useful virtual assistant for users.

The major aim of this project is to be able to minimise peripherals used with computer systems and create a software that allows the user to mimic keyboard and mouse input as well as other normal computer functions. The proposed system will help in avoiding COVID-19 spread and also allow users to operate a computer with only a webcam as hardware needed.

1.2 Scope of the project

The purpose of this project is firstly to conduct research into the field of computer vision and methods of communicating with the computer, through peripherals and vision. To investigate and detail any similar or existing works as well as understand the usability of said software. Then using all the research gathered, a software tool will be created that will allow for real-time hand tracking to be a method of input.

1.3 Report Structure

The report aims to cover all my research and design of the project. Chapter 2 discusses the background research that was conducted for this project. Chapter 3 discusses the system requirements. Chapter 4 discusses the design process and thoughts carried out. Chapter 5 discusses the legal, social, ethical and professional issues faced during the implementation of the project. Chapter 6 discusses the implementation that went into the project. Chapter 7 discusses the results and evaluation. Chapter 8 will then conclude my report.

2 BACKGROUND & LITERATURE REVIEW

2.1 Overview

There are a few methods in which input can be accepted into a computer. A research study was carried out with the intentions of figuring out health problems caused by non-keyboard input devices. The study looks into detail of many different pointing input devices and how users interact with them. The study states that 85% of workplace assessment interviews and 65% of user questionnaires reported that they had muscle pain, or discomfort (Woods.V, 2002). In the next segment I will be focusing on not only the common inputs used for computers, but also the non-standard methods of input. Then I will look into research regarding gesture and hand recognition.

2.2 Standard Methods of Input

2.2.1 MOUSE

The invention of the mouse boosted the relevancy of HCI. Before the 1980s Command Line Interfaces (CLI) was the most used interface for any sort of program, there was a lack of need for a device that would control a pointer object. Until the mouse was first introduced with the Apple Mac in 1984, which had several features of a modern operating system (WDD Staff, 2009). The mouse is a very common hardware used with computers, allowing for users to select text, access menus and interact with programs, files or data that appears on the screen. The mouse is a simple hardware in which you can move the mouse on your desk and the pointer will move according to the direction the mouse is moved. The mouse combines the pointing with the buttons on the mouse to perform either left-click, right-click, double click, drag and scrolling. There are a couple variations of the standard mouse we know today. For example, the trackball which worked with a ball being what operates the pointer, this was advantageous as if your work space was limited then a stationary mouse can help. However, some models lacked buttons and also took some practice to get used to. Another very popular variant of the mouse is the trackpad, where movement across a small area is translated into pointer movements. The trackpad unlike the mouse, can easily build up dust that can alter the performance. Another common mouse variant is the pointers that are located on keyboards. This was a very sensitive and fidgety method of pointing.

2.2.2 KEYBOARD

The keyboard was one of the first peripherals created to work with a computer, and has since seen very little changes in the way it is used. All keyboards have a very similar layout, at times some may include different keys or different characters which would depend on the region the keyboard was bought for. Most keyboard layouts have alphanumeric keys, this includes letters of the alphabet and numbers and also contain special keys that perform different functions (e.g Tab, Caps Lock, Enter, Backspace). Also contains modifier keys which were keys that would modify input of other keys for example when pressing the Shift key and an alphabet key it will capitalise. Function keys (F1, F2, F3 ,....) allows for you to input commands without typing or pressing other keys, for example the F10 key is normally used to full-screen an application, function keys vary based on the system and the program you are using. Cursor movement Keys are keys which help the user navigate the screen, these consist of keys like the arrow keys, Home/End and page up and down keys. Some keyboards also contain special shortcut keys, including insert, delete, esc, print screen and even include media keys. Keyboards work by pressing the key, which will send a scan code for the key to the keyboard buffer which then sends the interrupt request to the system which responds by reading and passing the scan code to the CPU which will process the character onto the screen (Nakaseko.M et al., 1985). The keyboard has become so iconic and just a simple design that it has been incorporated to many other devices. This can be seen from virtual keyboards on mobile devices to projected keyboards. Which will be touched upon in the next section.

2.3 Non-Standard Methods of Input

2.3.1 TOUCH

Touchscreens nowadays are very present in the current technology that people use on their day to day lives. When using a touchscreen the co-ordinates at which the user places their fingers can easily be recognised as it is a 2 dimensional environment. This can be seen through the zoom feature on smartphones, where a user just has to pinch in or out to zoom in or out. With simple recognition algorithms these gestures can be easily recognised. Whether that be their smartphones, tablets or even the self service machines in shops, touchscreen is a perfect example of a polished hand based input with many uses. The first touchscreen originated in the 1960s, they were not as accurate

as present touchscreens. They were not accurate due to the way in which they operated, the touchscreens were made of different materials than modern touchscreens (consisting of an insulator panel with a resistive coating). They worked when a finger touches the surface of the screen it would close a circuit, changing the binary value stored in that position. It was not able to read the position or the pressure of the touch. The prototypes for touchscreen consisted of using cameras behind glass, then the advancement of sensor technology led to multi-touchscreens being small enough to be used with any device, as can see with smartphones today (Chiesa.V et al, 2011). Touchscreens have become commonly used across interfaces, but still developers face the challenge of whether the user will know how to operate the device. These have to somehow be taught to the users, even though touch-screens are used in our daily lives, a user will have to assume how to function the device. Research has been conducted for touch-screens to be applied on any surface (Chris.H, 2011). One problem that is being tackled by researchers is that touchscreens are mostly focused on being for single-user interface and are not able to differentiate between different people. Furthermore, due to the COVID-19 pandemic, sanitary environments are taken more seriously and we can see this by recent research done. On average, there are about 253,857 colony-forming units per square inch on airport check-in screens (Julia. R, 2018), and the best solution to this problem is said to be “having better hygiene” which unfortunately does not resonate with the broader public.

2.3.2 PENS

Pen based interaction is more of a recent advancement in the world of HCI. Pen based systems use a pen or a stylus to allow for input. Mostly used when the user wishes to write or draw on a special screen, but can also be used a pointing device like a mouse. Pen based systems are usually used for inputting signatures and drawing etc. Pens can be used for writing large amount of text but handwriting recognition is complex, but research has been carried out over the recent years to improve its accuracy. Pen input can be considered more accurate than using a finger for touchscreens, this is because the tip of the pen has a much sharper and smaller surface area compared to fingers. Furthermore, with recent advancements such as the Apple pencil, the pressure of the pen on the screen can be measured and used for different purposes. Making different levels of pressure perform different tasks, allows for further user interaction and features. Another thing to note is Samsung’s Galaxy Note series (pun intended), which is a smartphone which comes with a stylus and aims to reach the demographic and has relative success with recent additions to the Note series. There has also been research into using both your fingers and a pen for touch input (Guojun Cheng and Jibin Yin,

2013), with relative success as their experiments proved users were able to pick up the system quickly. A software prototype was created to allow for users to use both pen and hand in a 3d virtual desktop environment which is able to be manipulated like a normal desktop (J. Yin and Y.Gao, 2016).

2.3.3 GESTURES

Hand gesture recognition is based on recognising patterns in hand movement. This can be through 2 dimensional planes (smartphones) or 3 dimensional input. Gesture based interfaces have not been commercially designed and are currently mostly used within medical fields. It becomes difficult to track the 3d movement of a hand as it requires some sort of hardware to recognise gestures and the hand itself in a 3-dimensional manner. An experiment of (Andrew.D, 2006), proposed a recognition algorithm that recognised discrete hand positions, such as the user bringing two fingers together. A research group created software on mobile devices that use only the back RGB cameras to recognise gestures (Jie Song et al., 2014). Electrical Impedance Tomography, radio frequencies and a full machine vision of in air-gestures are all techniques that have been used to attempt to read gestures accurately. Some researchers have constructed a wearable device, called “ViBand” which is worn like a watch, that is able to read gestures effortlessly and can be used for multiple applications (Gierad Laput et al., 2016). The device is able to not only recognise gestures but also uses bio-acoustic sensing to detect vibrations of grasped mechanical or motor-powered objects. This enables for context-aware functionality, allowing for the watch to change functionality depending on the object that is being held. For example, if the user was holding a whisker, they can be provided with a progress bar on how long it must be whisked for. The project is a very amusing concept, but has complications. It requires the user to know what gestures to use, it takes a toll on the battery and is still in its prototype stage so is not commercially ready. Gesture patterns get complicated when the gesture pattern increases, because new gesture patterns have to avoid conflicts with existing gesture patterns (H. Takahashi, 2016). The provided solution built off a proposed solution by Kuribara named “HandyScope”. Frustrated Internal Reflection (FTIR) is a technique used in multi-touch display for tabletop systems. The proposed solution used a mirror to reflect the GUI through a projector. The solution tracks the movement of the tip of a user's finger and supplies an interface that integrates touch gestures.

2.3.4 MICROPHONES

For sound input you need a microphone and a sound card to translate the sound waves to digital code that the computer can store, also known as digitising. Microphone input has come far in the last decade. With the introduction of voice assistants like Siri, Google and Cortana, voice input has become more and more predominant in mobile phones and computers. With a few words you can find out the current news, time, calendar dates etc. A research study of (K.L. Brown, 1995) was done to see the accuracy of speech recognition algorithms with CTIMIT-trained recogniser to have an 82% accuracy rate compared to 58% with TMIT-trained recogniser. Microphones can be used with computers to dictate what to do rather than typing, this proves very useful when needing to type too much or even just to do commands faster. For example it would be much faster to ask Siri to set an alarm than to physically set an alarm through the app. Speech recognition is easily accessible too as most devices nowadays have microphones.

2.3.5 OTHER

There are some input methods that have not been discussed in depth, due to its lack of true input with a computer. Game controllers are an input method but are very much for the single purpose of playing a game that requires only a certain number of inputs (the buttons on the controller). With the addition of the PlayStation 4 and 5 controllers that come with a touchpad at the top, these can be used as pointer devices, but still lacks use for the everyday user of a computer. Another input method is a bar code scanner, this is a good input device for translating bar codes but that's about it, cannot be used as a real-time method of input for computers.

2.4 Hand Tracking

Compared to gesture-based systems which look at patterns in hand motion, hand tracking offers the fidelity to support continuous tracking allowing for the users hands to be used in 3D space that require continuous tracking. Hand tracking systems use a mix of cameras and computer vision techniques to track the hand. Some systems are designed to be able to create an approximate model of hand skeleton allowing for users to view the bones and joints in real-time.

Robert.Y et al. (2009) used an ordinary cloth glove that is imprinted with a custom pattern, to help simplify the pose estimation problem. Using the nearest neighbour approach to track the hands iteratively. The colour glove allows for the hand to be easily

identifiable. The approach only requires a single camera. The flaw of the design is the need for a glove, which may be uncomfortable for the user.

In Chen Z.H et al. (2014) used the background subtraction technique to detect the hand. When the background is removed from the main image only the hand is cropped and shown. Background subtraction becomes more difficult when there is a moving object. The skin colour can be used to ensure that only the hand region is identified and not any moving objects, the colour of the skin is measured by the HSV (hue, saturation, value) model. The output was a binary image, which would be used to identify the palm point and the other points of the hand. The research then goes into recognising how many fingers are held up.

A method to detect hands was through the use of a Kinect Sensor. In L.K Phadtare's (2012) study he proposes a system based on an existing transcription method. The gathered data was applied to ASL (American Sign Language). The system was accurate but struggled when dealing with similar hand gestures. The Kinect is considered a depth camera, allowing for 3d image depth to know the distance from the camera. This allows for a very accurate detection of the hand and has been used for many hand tracking projects. P. Sharma (2016) created a projectable interactive surface using the Kinect. The camera was set up at a distance and a projector would project a screen. As the Kinect is able to sense depth, the y and z co-ordinates of the hand can be converted to x,y co-ordinates on the screen, this enables for recognition of where the user is pointing. In 2021, further research similar to the system above was created (R.Daminadrov, 2021). The system was aimed at creating a touch-less screen. Similar to P.Sharma's (2016) work, the system worked rather than having the user touch the projected surface, the user would keep distance and perform gestures to operate the system. As seen in Figure 1.

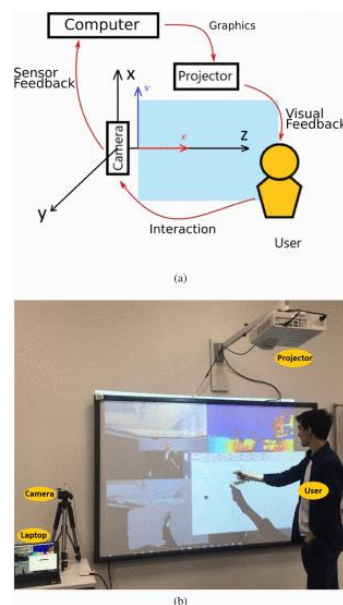


Figure 1:

a) diagram of setup with co-ordinate system, b) real setup

The key difference between these researchers works is that one of them allows for touchless input. Furthermore, both these experiments require the use of a Kinect sensor, which the broader public might not have accessed to. Kinects are also pricey.

A method of hand tracking that was experimented on before was using co-training, a machine learning technique. Haar and HOG features were applied in the co-training method of hand recognition (Y. Fang, 2008). The method demonstrated that large labelled data samples were not needed as some other methods suggested. The system created would be able to function in real-time, but a real-time application was not made.

R. Han (2017) noticed the lack of sensors in previous research in gesture recognition, either using the Kinect, data gloves or visual based recognition, Han proposed a solution using multiple sensors at once. This proved to be a struggle when converting one type of data and comparing it to the another, known as data fusion. The solution solves the issue regarding displaced information from both sensor methods and further enables an accurate reading of hand gestures. The proposed solution is a great step forward, especially in the virtual reality scene.

The camshift tracking algorithm is an algorithm able to reach a robust hand motion tracking. (S. S. Rautaray, 2011), used the algorithm in his project to implement hand gestures into video games. The proposed solution also uses haar as used in the earlier example. Their application maps the gestures recognised to the appropriate action defined by the gesture vocabulary. The program had 4 main moves connected to gestures Punch, Grab, Throw and Move Forward each being triggered when a gesture is made. OpenCV was used in this project to read the web-camera input data and provide computer vision algorithms. Since this time, hand tracking has been a goal for researchers to perfect.

2.4 Existing Work

As seen through the previous chapter, hand tracking can be done through different algorithms and methods. Hand tracking over the years will get more and more accurate. Hand tracking projects have been created before such as S.S. Rautaray (2011) in which a video game system was created using hand tracked data. The research conducted in this upcoming chapter goes over the practical uses of hand tracking in regards to being an input for a computer.

Air-swipe as mentioned previously was a feature on previous Samsung phones that would use a special front camera to sense the movement of the hand and perform commands based on the gesture. T.Sharma (2017) proposed the same Samsung feature

but using the stock camera of an android phone. The system's design was to use cost effective built-in tools to capture the image, using just the front camera of the device.

The use of hand tracking as a means of input for a painting application was suggested and implemented by I.J. Tupai et al.(2021). He proposed a system that would allow the user to decide what object to track, for example a pen. The calibration phase is time consuming as the user would have to manually select 6 different values HUE, Saturation, Value each with a minimum and maximum value. Once the object to track is identified 3 algorithms are run, get pen point, draw on canvas and get final canvas. The problem with this solution was that if there is an object of the same colour in the background, then it would falsely recognise it as the pen. I.J Tupai et al. (2021) also created a second version of the product using MediaPipe's hand tracking solution. This solution consisted of the camera being held up as such a birds eye view of the desk is shown. Using the index finger as the pen. This provided a much more natural feel for writing as it felt similar to writing with a pen on paper. S.Gulati et. al. (2022) created a similar project of a virtual paint board using hands as an input. The system also used OpenCV and MediaPipe's hand tracking solution to allow for drawings to be made. The project used the OpenCV module to overlay the captured image with colours for the user to select, and also had an eraser for the user to play with, as shown in figure 2.



Figure 2:

The Graphical User Interface created for virtual painting (S. Gulati et. al.,2022)

The GUI created using OpenCV lacks variation in colour, but only performs simple paint tasks. The system however shows the capability of overlaying images on the OpenCV captured image, to create a user interface.

Hand tracking has been researched as a method of navigating a virtual museum (W.A.A.Praditasari et al., 2021). The system was able to navigate through museums on google maps as if the hand was a mouse. A similar project was done where in which a virtual globe was controlled using hand tracking. V.Chunduru (2021) created a system which recognised 4 gestures for zooming in and out, rotating and to stop tracking. These

paired with the section of the camera in which the hand is currently oriented, allows for the program to control the globe on google maps.

A unique take for keyboard input can be seen through V.I Saraswati's (2016), eye gaze system. The system uses the centre point of one's iris to be used as a keyboard input method. The duration in milliseconds of which the user blinks is the classifier to what action to take. The system shows the possibility of iris tracking as input, however due to it taking nearly 13x longer than a normal keyboard, it fails to be quick and efficient.

Xu. P (2017) uses a convolutional neural network (CNN) classifier to recognise gestures. Using the Kalman estimator the program estimates the position of the mouse cursor based on a point tracked by the detector. The Kalman filter is used to smoothen the motion of the mouse cursor, although it provides a slight delay it is acceptable. The system would sometimes detect some invalid gestures as if it was an understood gesture. The software created was able to manipulate keyboard events based on the gesture performed. The research was aimed to make a strong case for hand tracking in HRI (Human-Robot interaction) cases.

Using the OpenCV library V.V Reddy (2020) has created a method in which a user can use a virtual mouse with just your hand. They have implemented a coloured circle cap identification algorithm that allows the user to stick a circle on their finger, which is then used as the main basis of the mouse pointer. Each color and combination of colour results in different operations. The use of coloured caps proved to be an efficient method in distinguishing different operations. However, this requires the user to have coloured caps and will require the user to pick up different caps to perform different actions. This makes the entire process longer and less efficient for the user.

M. Ranawat et al (2021) have achieved cursor control by skin detection, hand contour extraction, hand tracking and gesture recognition, similar to other projects, also used the haar method followed by further support from the OpenCV library. Without needing the use of coloured caps, the program uses the user's fingertip location to correlate to the screen. The system uses the understanding of hand gestures through a CNN to allow users to trigger mouse events. With different gestures allowing different events to occur. The problem lies with the unnatural gestures used to perform tasks such as right click

S.R Chowdhury (2020) used convex hull to segment the hand from the background, allowing for the hand to be further segmented using convex defects. After understanding the points of the hand, a virtual mouse and keyboard system is created. The mouse system calculates the number of defects using the convex hull method, depending on how many there are an operation is run (left-click etc). The keyboard feature works by displaying a keyboard to the user, which they are able to scroll through

and works with the mouse system as “left clicking” on a character would input that character onto the screen. Refer to Figure 3 to see the keyboard interface.

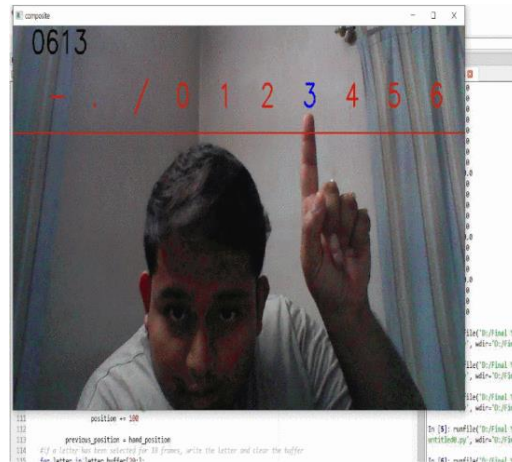


Figure 3:
Virtual Keyboard interface (S.R Chowdhury, 2020)

The study shows the use of the convex hull algorithm with hand tracking. The mouse feature of the system uses gestures to perform and can sometimes be inaccurate with incorrect gestures being identified. The keyboard, although functioning, has limited user functions and could cause discomfort for the user when using. The solution to be proposed will aim to implement a more functionable keyboard.

Fan Zhang (2020) proposes a hand tracking solution using OpenCV and MediaPipe Hands. The solution incorporates two machine learning pipeline models. A palm detector is used to locate palms via an oriented hand bounding box and a hand landmark model, that landmarks each point of the finger. Using OpenCV the hand landmarks can be displayed providing a real-time skeletal model of the hand. The research touches on the use of MediaPipe Hands to be used with hand tracking projects easily, but does not show its functionality as an input method.

A virtual mouse system was created by S.Shriham et al(2021) using MediaPipe Hands. The system uses OpenCV, MediaPipe and AutoPyGUI to control the windows mouse with the skeletal hand. The solution uses a set number of gestures to control the mouse and works very smooth. With an overall accuracy score of 99%, this can be compared to V.V Reddy's (2020) solution which has 92% accuracy with an empty background but a 43% accuracy rate when having a noisy background. MediaPipe hands is able to minimise noise levels to a scale that allows the mouse to work in real-time with little to any gesture based errors. The application lacks a click and drag functionality for the mouse, but also does not look into the other form of input, text entry, which my project aims to build upon.

2.5 Fitts Law

Fitts' law is a model that describes how quickly people can select a target. The law states that the time to acquire the target is a function of the size of the target and the distance to the target. Equation 1 shows Fitt's law formula, where MT is time taken to select a target, a and b are constants set by the type of device, A is the distance from starting point to target and W is the width of target along axis of motion

$$MT = a + b \log_2\left(\frac{2A}{W}\right)$$

Equation 1.

Fitt's Law

Fitts' law aims to reduce interaction cost by reducing the distance the user needs to traverse to get to the endpoint (Guiard.Y , 2004). Fitts' law is a key principle when creating user interfaces and therefore is a key element when creating the virtual input controller environment.

3 SPECIFICATION REQUIREMENTS

This chapter highlights the functional and non-functional requirements of the project. These requirements were generated at the start of the project and were adapted throughout the project creation and based on the time left until the deadline of the project.

3.1 Functional Requirements

1. The software must be able to open the user's camera.
 - a. The software correctly outputs a video stream of the devices' camera.
2. The software must distinguish properties of a hand.
 - a. The software must receive all hands presented in front of the camera in a real-time state, constantly updating.
 - b. The software must be able to distinguish each of the 21 landmarks of the hand.
 - c. The software must connect all the landmarks to provide a skeleton overlay on the hand, can be optionally turned on or off.
 - d. The software is able to distinguish between multiple hands, if presented on camera.
 - e. The software must be able to identify whether the hand is left or right.
 - f. The software must be able to operate no matter how far away from the camera the hand is.
3. The software must be able to make use of the landmarks in each hand/ be able to distinguish gestures. This can be implemented with the following methods:
 - a. Must be able to distinguish the distance between any two point of one or two hands.
 - b. Must be able to tell whether a finger is up or down.
4. The software must have the following operational mouse elements to it.
 - a. The mouse must move the computers cursor, depending on the position of the user's hand/fingers.
 - b. The mouse must be able to left-click, depending on performed gesture.
 - c. The mouse must be able to right-click, depending on performed gesture.
 - d. The mouse must be able to scroll down, depending on performed gesture.

- e. The mouse must be able to scroll up, depending on performed gesture.
 - f. The mouse must be able to click and drag, depending on performed gesture.
 - g. The mouse must be able to double click, depending on performed gesture.
- 5. The software must be able to allow for text-entry input.
 - a. The software must accurately take input.
 - b. The software must enter text into the clicked text entry (whether that be on Google Chrome search bar, Word Document, Notes, any other text input etc).
- 6. The software must be able to perform media key imitation based on gesture.
 - a. The software must be able to Pause current media.
 - b. The software must be able to Play current media.
 - c. The software must be able to Skip current media.
 - d. The software must be able to go Previous on current media.
 - e. The software must be able to control the Volume of the device.
- 7. The software must be able to perform shortcuts based on gesture.
 - a. The software must be able to imitate shortcuts of a keyboard, e.g Alt + Tab, Ctrl + Shift etc.
 - b. The software must allow for the user to customise shortcuts.
- 8. The software must be able to correctly change modes seamlessly.
 - a. The software must make it clear which mode is being used.
 - b. The software must have visual indications of the current mode.

3.2 Non-Functional Requirements

- 1. The software must allow for the user to customise gestures.
- 2. The software must allow for multiple hand detection modes (enabling other modes to be used with two hands).
- 3. The software must have a settings GUI that allows user to customise the entire interface.
- 4. The software must output hand landmarks extremely accurately.
- 5. The software must operate with no latency.
- 6. The software must be easy to deploy, configure and maintain in the future.
- 7. The software produced should meet the highest professional and ethical standards set out by the British Computing Society (BCS).

4 DESIGN

4.1 Assumptions

The following assumptions were made during the development of the project.

1. The user will have a camera connected to the main device (computer), whether that's built-in (laptop) or USB powered.
2. The user will have a microphone connected to the main device/ or the camera has a built-in microphone.
3. Sometimes there may be noise that interrupts the hand recognition causing for random spurts of incorrect hand landmarks/ reading whether the hand is left or right.
4. It is assumed the user has a hand/hands.

4.2 Tools & Technologies Used

The following Tools & Technologies were used during the creation of the project:

4.2.1 PYTHON

The Python programming language has been chosen for this project because of its suitability in computer vision programming tasks. It also has container datatypes that will be useful for the project (lists, dictionaries, sets and tuples). Furthermore, I am quite familiar with the Python programming language

4.2.2 OPENCV

OpenCV (Open Source Computer Vision Library) is an open source machine learning software library based on computer vision. Commonly used for image processing and computer vision tasks. The OpenCV module library was used in this task for landmark detection and for outputting them and the cameras video onto the screen. The library is also used to draw lines, circles, rectangles and provide some overlay images onto the camera stream, allowing you to create an interface around the captured camera stream.

4.2.3 MEDIAPIPE FRAMEWORK

MediaPipe is a framework used for building machine learning pipelines. The framework is cross-platform allowing for any device to make use of it. For this particular project the MediaPipe's hand solution will be used. A pipeline is a graph which contain "calculators" and "streams". Each calculator performs different tasks, putting calculators together allows for different solutions to be made. The pipeline used for MediaPipe's hand solution can be seen in figure 4.

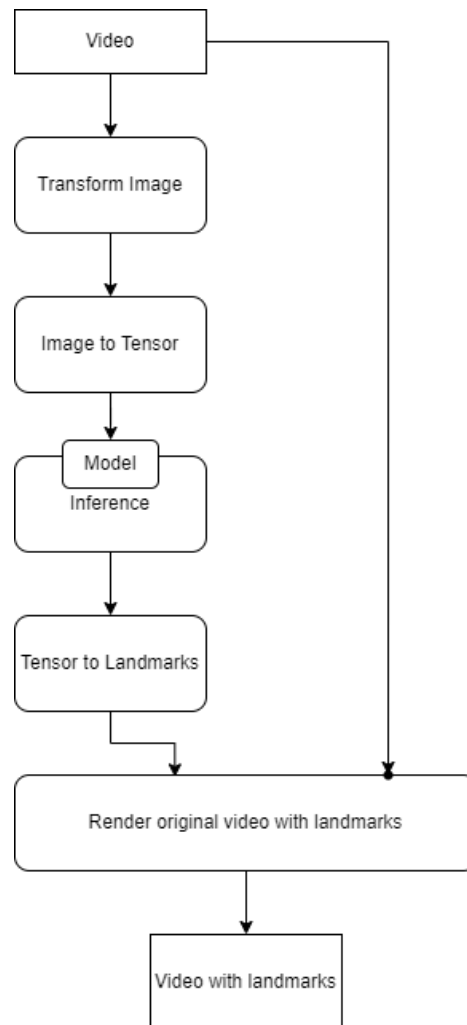


Figure 4.

Flow chart illustrating MediaPipe's hand solution pipeline

In the above figure each rounded square represents a calculator and the arrows represent the stream. There are 2 main models that the MediaPipe hands solution uses in order to recognise the hand. First is the Palm Detection model, which detects the palm at a 95.7% accuracy rate. Then the Hand Landmark model is run, which outputs the 21 hand-knuckle coordinates of the hand. Refer to figure 5 for a diagram of the hand landmarks.

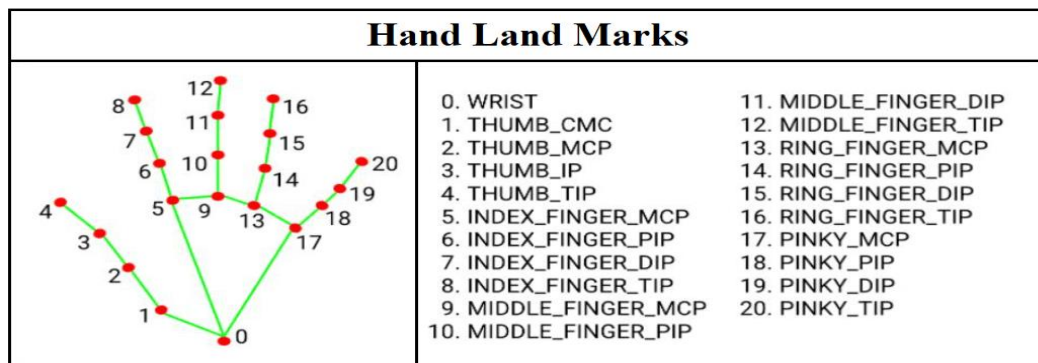


Figure 5.

Co-ordinates or land marks in the hand (MediaPipe Hands)

4.2.4 TIME

The time module allows for various time-related functions to be used. In this project I will be using time to check the framerate of the application.

4.2.5 MATH

The math module provides access to many mathematical functions. Will be used to determine certain values within the code.

4.2.6 NUMPY

NumPy, which stands for numerical python, is a library that is used for various mathematical computations, mainly working with arrays and matrices. Since OpenCV stores images as a matrix this library will be used to help analyse and alter them.

4.2.7 OS

This library provides functionality when interacting with the operating system, allowing for interactions with the file systems.

4.2.8 SYS

This library provides functions and variables to allow manipulation of different parts of the Python runtime environment. This library will be used with PyQt to allow the GUI to be run.

4.2.9 MOUSE (MODULE)

This library allows for full control of the mouse, allowing python to simulate mouse movement and clicks. This library is used to control the mouse pointer.

4.2.10 KEYBOARD (MODULE)

This library allows for full control of the keyboard, allowing python to simulate key presses and much more. This library is used to control the keyboard.

4.2.11 WIN32API AND WIN32CON

Using win32api allows python to communicate with Windows, this allows for various Windows operations to be performed. This library will be used to allow for media key and shortcuts functionalities.

4.2.12 THREADING

This library provides a higher-level threading interface than “Thread” allowing for thread creation. Threading is a way of concurrent programming, allowing for multiple processes to be run in a queue state, will be touched upon further in a later chapter.

4.2.13 PYCAW

Pycaw, which stands for Python Core Audio Windows library, allows for audio playback devices to be recognised and manipulated. This library will be used to allow for volume adjustment on the system.

4.2.14 PYQT5 AND QT DESIGNER

PyQt5 is a free cross-platform graphics user interface (GUI) toolkit implemented as a Python plugin. PyQt includes a substantial set of GUI widgets that are used to construct GUI's. Qt Designer is a software used to build PyQt5 based GUI's in a simpler fashion. Qt Designer allows users to create custom windows, dialogues and forms in a “what you see is what you get” manner. The output is a ui file that can be converted into a python file using the dev tool “pyuic5”.

4.2.15 SPEECHRECOGNITION

SpeechRecognition is a library used for performing speech recognition. The library has support from several engines and APIs allowing for different standards of speech recognition. The API I will be taking advantage of is the “Google speech recognition” engine.

4.2.16 CAMERA & MICROPHONE(HARDWARE)

For the camera I will be using a USB 2.0 powered Nulaxy C900 webcam with an inbuilt microphone. The specifications will be in the appendix.

All links to the above reference documents will be in the appendix.

4.3 Design Principles

The following software engineering principles have been implemented throughout the design and development process.

1. Agile Development

The Agile software engineering methodology is the main approach used within this projects design and implementation stages. The methodology uses an iterative and incremental approach which allows for flexibility in the code. The iterative approach does not need fully fledged requirements, and requirements made can be altered and revised upon, allowing for adaptability. The principle works great with the project, as the main features are implemented accordingly, but then new features are revised and then enter the cycle of design and implementation.

2. Divide and Conquer

Divide and conquer is a core principle in problem-solving situations. It is the process of breaking a problem into smaller “subproblems” and solving them. Solving the smaller bits allow for the larger problem to be solved more efficiently. In this project each there are different modes that are needed (mouse, media etc.), which allow the problem to perfectly split. The first prototype of the program only allowed each individual program to work, then after all the modes were complete, it was much easier to join them all into the

final system. Each subproblem could be further split into “sub-sub-problems” for example splitting the functionality of the mouse into left-click, right-click etc.

3. Reusability and Open/Close Principle (OCP)

Reusability is the a key concept in this project. When implementing the program there are factors that make separating the hands difficult, as reading a left and right hand are slightly different. The OCP principles states that classes, methods and functions should be written in a way that make it so they only have one specific use. This can be seen in the functions designed for each mode within the program.

4.4 Design Decisions

The following design decisions were made throughout the application building cycle that led to the final prototype created:

1. The initial idea for the mouse click gesture was to use the index and middle finger to click. The idea was changed as having the user be able to customise the gesture used would allow for a more user-friendly experience.
2. The initial idea for text input was similar to S.R Chowdhury’s (2020) method of text input through OpenCV’s “cv2.write()” function allowing for a virtual keyboard to appear on the top of the screen. Rather than making the user have to scroll through letters, the idea was to overlay every single key onto the screen allowing for the user to click the keys when the index and middle finger touched. However, this approach not only added too much to the user’s screen but also felt like a lot of code when windows has a built in on-screen keyboard feature. The windows on-screen keyboard can be opened through using the win32api, and using the virtual mouse created would provide a similar experience to just having a lot of keys overlayed on the camera image. This would also allow the user to stay in mouse mode rather than having to change between modes frequently.
3. Another form of text-entry was considered when researching speech recognition. Being able to input text based on speech proved to be much easier as input, and faster than using the on-screen keyboard. The only constraint of this is that the user also then requires a microphone.
4. For the first prototype the decision was made to have the settings be amendable through the settings text file.
5. The decision was made to present icons and mode selection on the top of the camera feed rather than having them be triggered by different gestures. This decision was made as it felt like there would be too many gestures that became confusing to handle.

4.5 System Flow Diagram

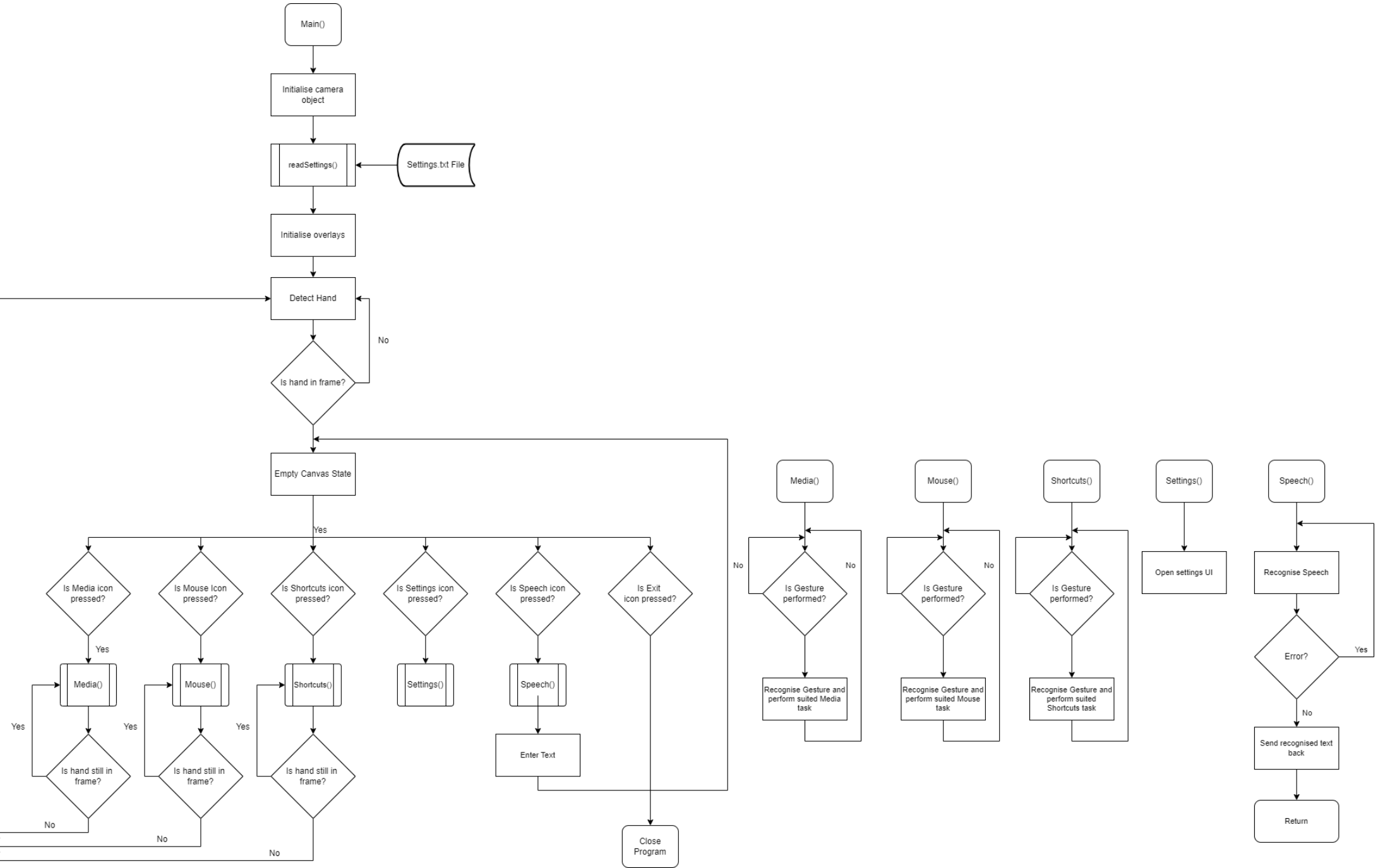


Figure 6.1:
Flowchart of the system flow

4.6 UML Design



Figure 7:
UML Diagram

4.7 Use Case Diagram

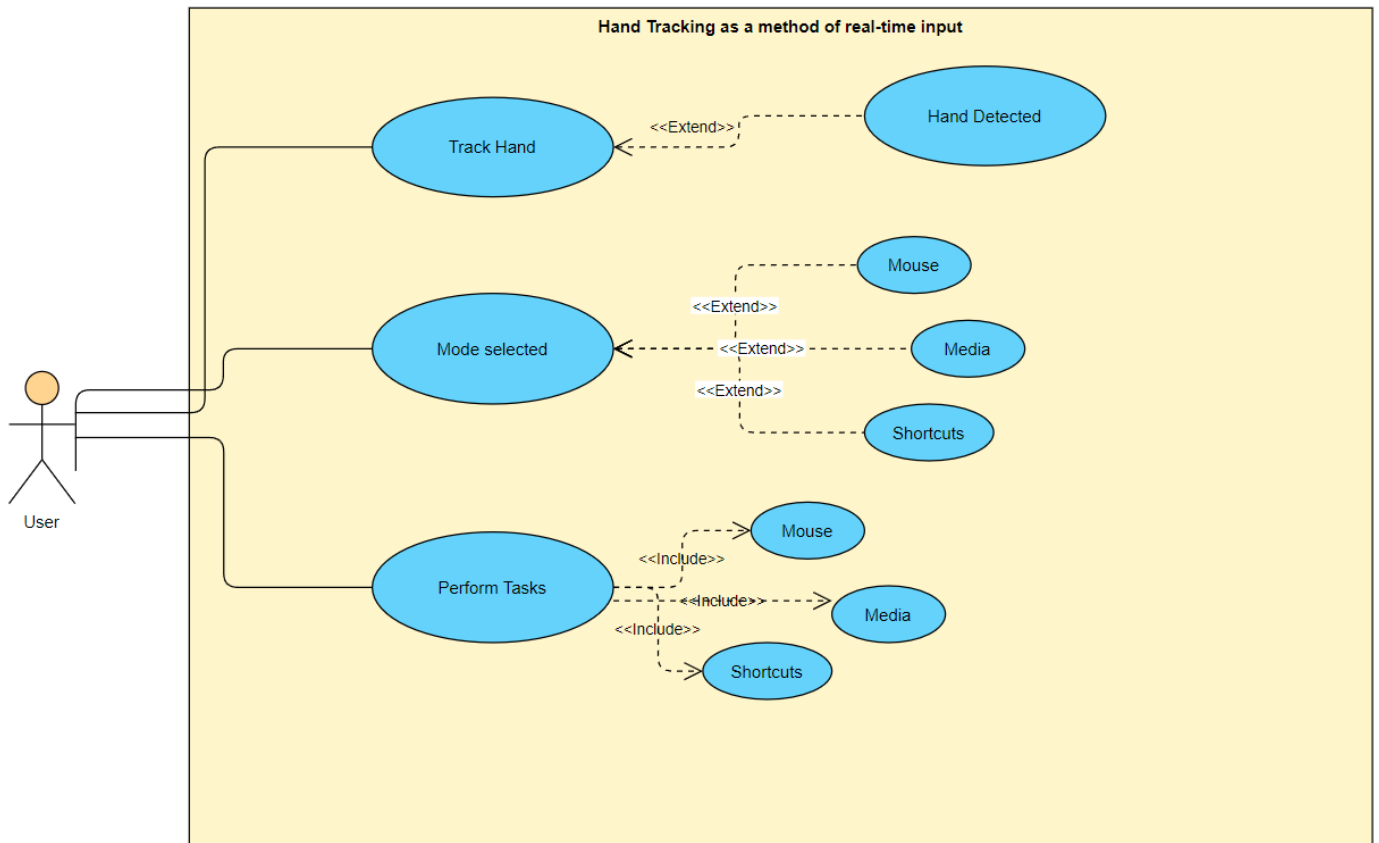


Figure 8:
Use Case Diagram

5 LEGAL, SOCIAL, ETHICAL AND PROFESSIONAL ISSUES

This chapter discusses the legal, social, ethical and professional issues faced during the implementation of this project.

When processing the users' video the video is only displayed to the user and is not stored. Therefore no personal information and no identities are saved within the system.

Additionally, various open-source libraries have been used over the course of the project which were discussed in Chapter 4.2. All sources have been clearly referenced within the code and the report.

This project is not intended to be used commercially.

The written code for the project and the report written is all my own work, and any external sources that I have used have been cited.

6 IMPLEMENTATION

This chapter discusses the implementation of the project. This chapter will go over each step taken to create the final prototype. The section highlights how landmarks were recognised, how each function/mode was implemented and the implementation of the user interface. All snippets of code will be included in the appendix for reference with this chapter.

6.1 Initialise Webcam

The first step that was required was to take input from the user's camera feed. This step was simple, using the OpenCV library's VideoCapture method, a video object was formed and read the user's camera. This object was named "video_object".

VideoCapture takes in one argument, the value of the argument will decide which camera to be used. Using the argument '0' will use the default camera on the system.

The video object is read in a while loop, this is because the object read's the camera frame for frame. The while loop constantly updates the "camera_img", which is the output video frame that is displayed to the user. "camera_img" is stored as a matrix, which stores each pixel of the video object.

6.2 Hand Landmarks Recognition

The next step was to use the mediaPipe framework to recognise the hand/s. In this step we have to make use of the hand solution from mediapipe. Firstly you must extract the solutions for hands and save it as an independent object. Using the object you are able to create another object for the hands, this object makes use of the "Hands" class within the mediapipe solutions. To initialise the hand there are certain parameters used in order to alter the state of recognition. To briefly summarise each parameter:

mode- "static_image_mode" this determines whether to treat the camera image as static or a video stream. Since the application must work in real-time, the parameter is set to False.

maxHands- The maximum number of hands the hands will attempt to recognise.

modelComplexity – The landmark accuracy as well as inference latency go up.

detectionCon – The minimum confidence value from 0.0 to 1.0 for the detection to be considered successful. The solution will attempt to recognise the hands, but if it is not “confident” that the hand landmarks are detected correctly, then it will not be processed. The higher the value of the parameter the more accurate it will be. Changing this parameter allows for less jittery landmarks.

trackCon – The minimum tracking confidence value from 0.0 to 1.0 for the tracking to be considered successful. If the tracking confidence is below the minimum then the detection process will occur again.

Once the hands object is created, we must convert the camera_image into an RGB image, this is because the hands solution only reads RGB images and cv2 automatically creates the object using BGR. This can be done using the OpenCV module to convert the colour. Then process the hands detected in the image.

Then to obtain landmark information to be used, there are 3 different outputs that can be taken from the detected hands. To briefly summarise each output collection:

MULTI_HAND_LANDMARKS- This output is a collection of all the landmarks discovered from the hand (21, stored as a list), with each landmark composing of an x, y and z co-ordinate. With the x and y co-ordinates depending on the image width and height, and z being the depth of the hand.

MULTI_HAND_WORLD_LANDMARKS- This output is a collection of the detected hands where each hand landmark is stored in a list. The difference between WORLD landmarks and normal landmarks is that WORLD landmarks uses real world 3D co-ordinates. The co-ordinates are stored as x, y and z but are stored in metres rather than the image’s resolution.

MULTI_HANDEDNESS- This output is a collection of handedness (whether the hand is left or right). This does not store the x, y and z co-ordinates of each hand landmark, but rather the “label” and “score” of each hand. The label identifies whether the hand is left or right, and the score is an estimation of the predicted handedness. This output information will be most useful for when identifying left and right hands.

To use the results/output of the detected hands, first we have to iterate through each hand, then every landmark. For each hand detected, we must draw the skeletal view. Drawing the skeletal view can be done using OpenCV by connecting each landmark together individually, however this will take a while to implement. Luckily mediapipe has a method that helps draw each of lines connecting each point, “mp.solutions.drawing_utils”. Then to draw each of the landmark positions I had to iterate through each landmark and draw a circle (using OpenCv) on each point. This can be seen in figure 9.

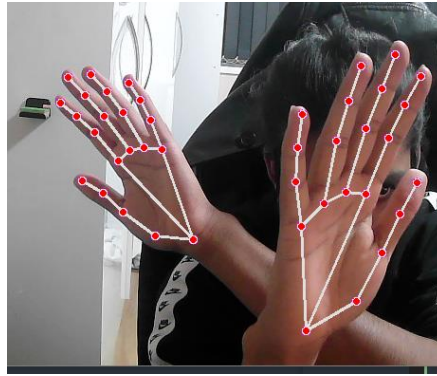


Figure 9:

Drawing the skeleton of the hands using `mediapipe.solutions.drawing_utils`

The first implementation of the hand detection used two separate functions to draw the hand and to detect each landmark position. This proved to be inefficient and made the process of reading each landmark of the hand difficult. Furthermore, only one hand was able to be used for detection. To fix this the decision was made to attempt to convert the needed values into a dictionary that can easily be accessed. However, from further thought I decided to implement an initialiser child class to hold each of the hands' properties allowing for easier accessibility to each hands properties. The class was named "HandProperties()" and contains the `handId`, whether the hand is left or right, the `landmarkList`, bounding box, `rectanglescale`, centre of the hand and the length and height and width of the bounding box. The bounding box was a rectangle that was created to show the hand boundaries, refer to figure 10.

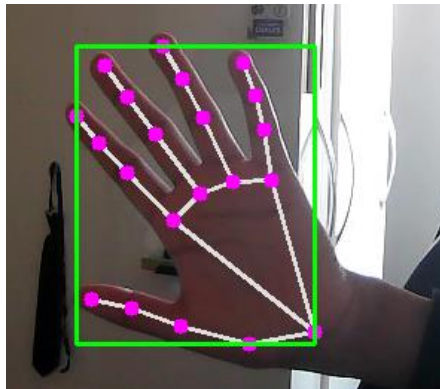


Figure 10:

Drawing the bounding box

The rectangle is created using the lowest x and y and the highest x and y co-ordinates of each hand. Calculating the area of the bounding box allowed me to tell how far the hand was from the camera. Furthermore, using the bounding box's area, I was able to apply scale to functions. Adding scale means that even if I was to be far from the camera, the hand would recognise the distance between two points of a finger as the same if they were close up. This allowed for the user to use any function whether they were close or far.

In the next section the methods and functions of the HandDetection class will be spoken on.

`drawHands()`

The `drawHands` method was created in order to draw and store the landmark information of each hand. The first implementation of the `drawHands` method was only able to draw the skeletal view, and provided a landmark list of all the hands. However, it was not able to distinguish the 2 hands, and would always only take the last hand to enter the frame as its main hand. This led to complications as it couldn't even tell whether the hand was left or right, causing for some issues with other methods.

`findDistance()`

This method is used to calculate the distance between two points. The first implementation of this method would return the length between the two fingers and the x,y co-ordinates of the 2 fingers as well as the middle point of them. The middle point is a necessary co-ordinate to know as this is what is used as the pointer for the cursor of the mouse control. The problem was when returned there were too many variables that became messy to deal with. The solution was to create a subclass which simply held each of the variables, allowing for much easier access to points. Furthermore, when scale was implemented, the `scaledLength` was initialised, allowing for, as previously mentioned, tracking the distance between 2 points even when distant from the camera. Scaling was done using numpy's `interp` function.

`isFingersUp()`

Detecting whether the user's fingers were down or not was previously based on the distance between the tip of the finger and the 2 indexes before that. For example, to calculate whether the middle finger was down, the program would check whether `MIDDLE_FINGER_TIP` (12) (point y) was less than `MIDDLE_FINGER_PIP` (10) (point y), refer to figure 5. The issue with this solution was that when the user's hand would rotate, the fingers would not be recognised correctly. The second implementation of the "`isFingersUp()`" method was to use the "`findDistance`" method to check whether the distance between the tip of the finger and the 2 landmarks before were less than a certain length. However, this implementation also was flawed as orientation was not considered, as well as the fact that the distance between two points is always positive. Therefore, when the tip of the finger goes below a certain point it will consider the finger to be up.

To fix this issue, an algorithm was written to create a mini box inside the hand, named "`cbx`" / "`cby`" "for centre box x" and "centre box y". The idea was to create a rectangle between points 9 and 0 then check whether the x and y co-ordinates of the fingertip are within the box. This implementation can be seen in the "`isFingersUp`" method. This proved to be a better method as it was able to classify whether fingers were up even if the hand orientation was not the desired orientation.

The previous versions of each method or class will be shown in the implementation/appendix section.

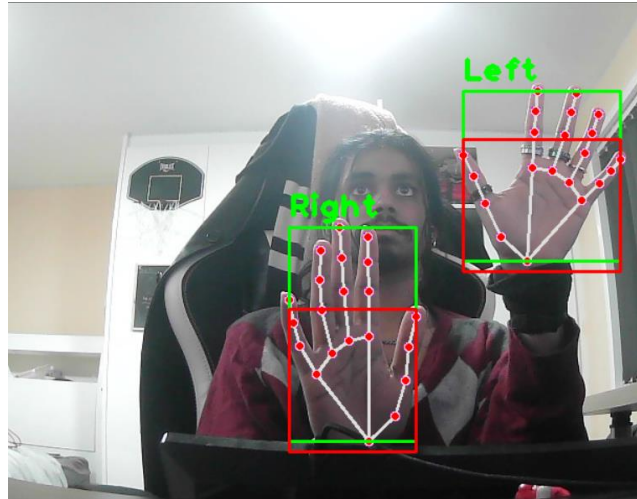


Figure 11:

Implementation of the handProperties class to allow for multiple hand tracking, recognising different hands and the closedFingersBox(red)

6.3 Mouse

The following section will highlight the thought process when creating mouse functionality through hand recognition.

6.2.1 CURSOR CONTROL

In order for cursor control to work, the mouse module was imported. The mouse was designed to use the findDistance() method between two finger tips, and use the centre of them to be used as the cursor point. For the program to understand where the cursor should be on the computer screen in proportion to the camera image, was implemented using numpy's interp function that allowed the cursor to be interpreted.

The equation to calculate the centre point of two points:

$$Cx = x1 + x2 // 2$$

$$Cy = y1 + y2 // 2$$

$$\text{CentrePoint/midpoint} = (Cx, Cy)$$

The initial method to use the mouse was to use the distance between the thumb point and the index finger. This was a comfortable gesture to use as a mouse function and felt right. Reminded of S.Shriham et al(2021)'s mouse interpretation, I also wanted to allow the user to choose between using thumb and index or index and middle finger as the pointer fingers.

Furthermore, rather than using the entire camera image for mouse control, a little portion of the screen is used as the tracking area. This is due to a couple reasons. When the users' hand is below a certain part of the camera's vision the hand is not properly recognised as the full hand is not in frame, and either causes stutters or does not work. Furthermore, allowing the user to change the trackpad size through the settings, lets the user define how sensitive the mouse will be, the smaller the area the more sensitive.

6.2.2 LEFT CLICK

Using the same fingers used in the cursor control function, the mouse is considered "clicked" when the two fingers touch. This can be calculated by checking the distance between the two points and seeing whether the scaledDistance is less than 1, and if so then perform left click.

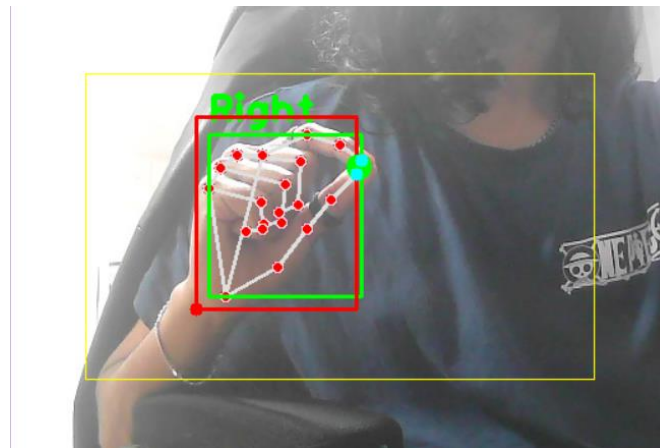


Figure 12:
Mouse Left Click functionality/ indicator

6.2.3 RIGHT CLICK

Creating the right click function proved to be difficult as figuring out the gesture was complicated, there were many variations that were tried. But the variation that was chosen at the end was to use the middle and index finger to perform the right click function. Similar to left click, the distance between both middle and index would be calculated and if less than 1 would be considered a right click. The problem that was encountered with this solution was the mix of this gesture and the scroll gesture. Which will be discussed below.

6.2.4 SCROLL

The scroll feature was implemented also using the middle and index fingers. To scroll up, both index and middle finger must be touching while the ring finger is pointing up (finger is up). This was paired with scroll down which would be called when the finger was pointed down. This provided a seamless scroll up and down gesture, allowing for easy navigation of windows. However, since the right-click function was using the same fingers, problems occurred when attempting to use both. To fix this issue, the right click will only be performed if the user's ring finger is in nor up or down, but facing the camera.

6.2.5 DOUBLE CLICK

The double click functionality was automatically implemented when left click was implemented. To operate the double click, the user simply has to tap their index and thumb finger twice together in the span of a second, similar to normal mouse cases. I have hypothesised that a problem may occur if the user's computer is not strong enough, they may have a lower frame rate. Which might in turn not recognise when the user double clicks. However this is not a problem that was found during testing.

6.2.6 LEFT CLICK DRAG

The left click drag cannot to be performed by a simple left-click followed by the user's cursor control, this is due to the way in which left click functions, as it is only a clicks the left click button, but does not hold it. To solve this issue, I created a separate gesture for when needing to click and drag, the gesture works in parallel with the normal left click gesture. The difference is that it detects whether the 3 other fingers (middle, ring and pinkie) are up. If so then the left click will be considered held rather than a click.

6.3 Text Input

The following section will highlight the thought process when creating text input functionality within the hand recognition software.

6.3.1 ON-SCREEN KEYBOARD

Initially the idea was to overlay each and every key onto the camera image using OpenCV to create several instances of squares with Text to replicate a key. This proved to take up too

many resources. Furthermore, Windows has an inbuilt on-screen keyboard that can be used with the gesture-based mouse. Rather than having the user have to switch between keyboard and mouse mode, the user can simply open on-screen keyboard. The keys are big and therefore easily usable with the virtual mouse.

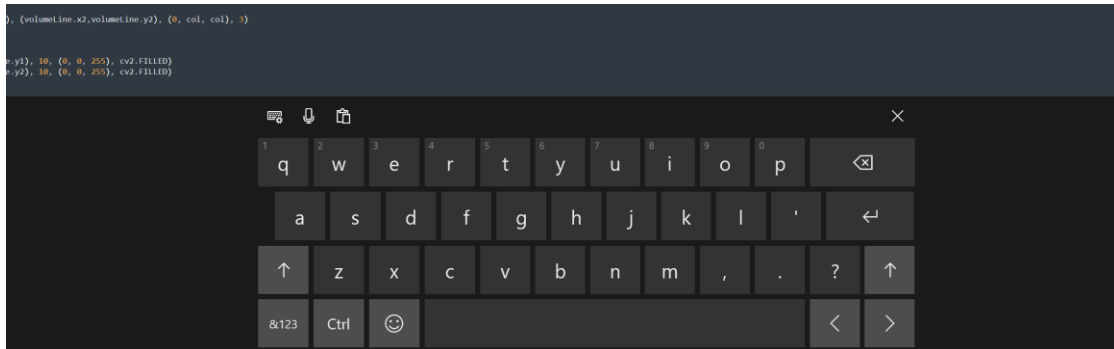


Figure 13:
On-Screen Keyboard, takes up 1/5 of the screen

6.3.2 SPEECH RECOGNITION

Speech recognition was an alternative method that could be used for text entry and would even be faster than a traditional keyboard at times. Furthermore, implementing speech recognition through python is easy and quick. The speech recognition system worked by listening to the user's microphone, and after sometime will output the text onto whatever text entry was last selected. Adding a delay provided a smooth type into the text entry.

6.4 Media Keys

This section of the report will highlight the thought process when creating gestures for the media mode of the system.

6.4.1 PAUSE/PLAY

Playing and pausing media can be done with the same command as they normally work in synchronous as the Windows virtual key for Play and Pause is defined in the same variable, `VK_MEDIA_PLAY_PAUSE`. Therefore, playing and pausing was easy to implement, the gesture was designed to be a closed fist. The system Plays/Pauses whichever media is currently playing, even if the media is in the background. Works identical to a Play/Pause button on a keyboard.

6.4.2 SKIP/PREVIOUS

Skipping and going back on tracks/media was a relatively simple process. Similar to Play and Pause, windows has virtual keys for different keys, the virtual key for skip is “VK_MEDIA_NEXT_TRACK” and the key for previous is “VK_MEDIA_PREVIOUS_TRACK”. Using these virtual keys and applying them to a gesture, allows for skipping and previous functionality within the media mode. Holding up the index finger will skip, Holding up both index and middle finger will go to the previous track.

6.4.3 VOLUME

The initial idea for volume was to use the distance between two fingers to determine the volume level. The closer they got the quieter the volume would get. The issue at first was when walking further from the screen the volume would consider the finger distance minute and consider the volume to be low even if the user’s fingers are wide apart. However, this was fixed after the addition of scaling to allow for seamless volume control. Furthermore, adding an indicator to show the volume level by creating a line between the two fingers, and having the line change colour based on how far or close they are. When the fingers touch, this would mute the device, adding indication to show when the volume is muted. The volume feature makes use of both the thumb and index finger.

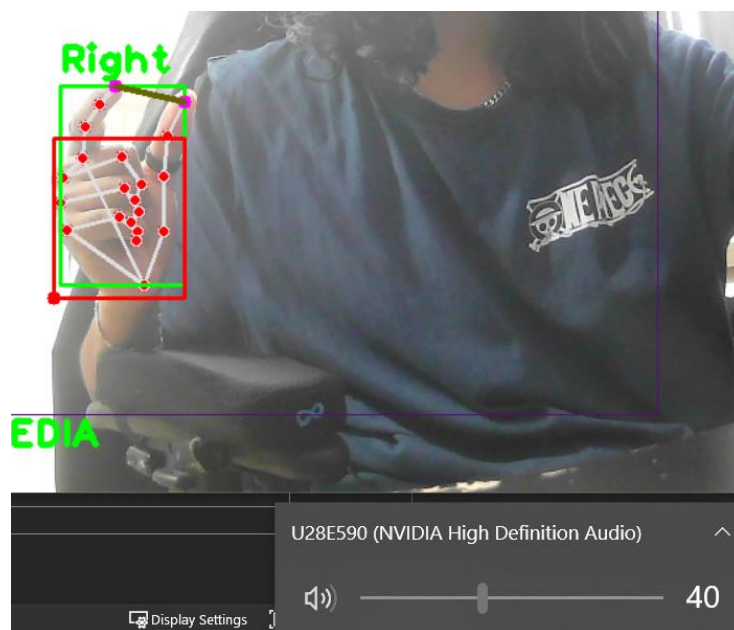


Figure 14.1:
Volume Control

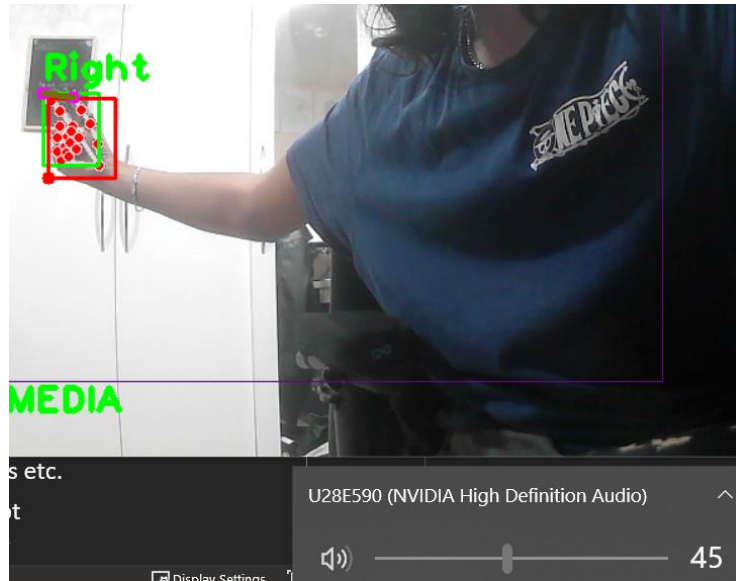


Figure 14.2:
Volume Control showing Scaling

6.5 Shortcuts

This section of the report will highlight the thought process when creating gestures for the Shortcut mode of the system.

6.5.1 KEYBOARD SHORTCUTS

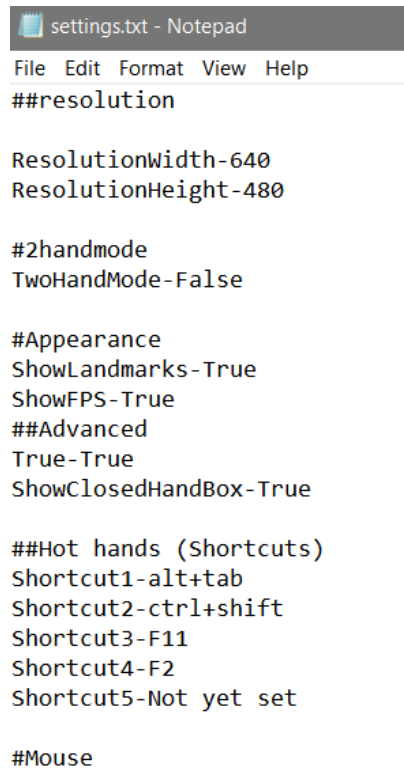
Keyboard Shortcuts are shortcuts that are used by pressing a combination of keys on the keyboard together, e.g “alt+tab”. To create keyboard shortcuts, the user can define what shortcut they would like and add it to the settings to then be used with gestures. The prototype requires the user to manually enter the shortcut term. The gestures are based on which of the 5 fingers are held up, allowing for 5 gestures.

6.5.2 WINDOWS SHORTCUTS

Windows shortcuts refers to shortcuts that can open up applications, perform tasks etc. This type of shortcut was causing issues during implementation and therefore is not present in the current prototype. However, adding this type of shortcut will allow for a clearer system.

6.6 Settings Storage

The settings for the system is stored in a text file, refer to figure 15. The text file is read at the start of the program to ensure that the user's preferences are met. The settings can be altered through the text file itself or the Graphical User Interface.



```
settings.txt - Notepad
File Edit Format View Help
##resolution

ResolutionWidth-640
ResolutionHeight-480

#2handmode
TwoHandMode-False

#Appearance
ShowLandmarks-True
ShowFPS-True
##Advanced
True-True
ShowClosedHandBox-True

##Hot hands (Shortcuts)
Shortcut1-alt+tab
Shortcut2-ctrl+shift
Shortcut3-F11
Shortcut4-F2
Shortcut5-Not yet set

#Mouse
```

Figure 15:
Settings.txt file

6.7 Graphical User Interface

In this section I will be highlighting the graphical interfaces created for the user's interactivity.

6.7.1 THE REAL-TIME HAND INPUT INTERFACE

The graphical interface was designed with Human Computer Interaction ideologies in mind. Providing icons to indicate what each button does and using Fitts's Law by having

all the buttons near each other for easy mode changing. The buttons can easily be pressed by having both middle and index finger touch where the icons are.

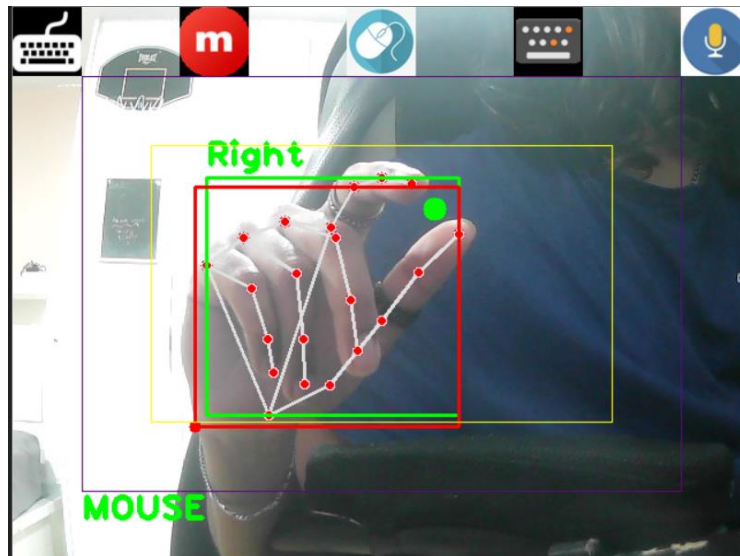


Figure 16:
Main System's GUI

As mentioned previously, the camera image is stored as a matrix. Using the knowledge of matrices I was able to overlay the icons onto the screen.

6.7.1 THE SETTINGS GUI

The GUI built using PyQt allows for settings for the real-time hand input system to be adjusted. The GUI provides a simpler settings alteration method than directly using the settings text file. This is because the GUI is designed to work differently for each setting and is split into submenus allowing for a comfortable view and use of the system.

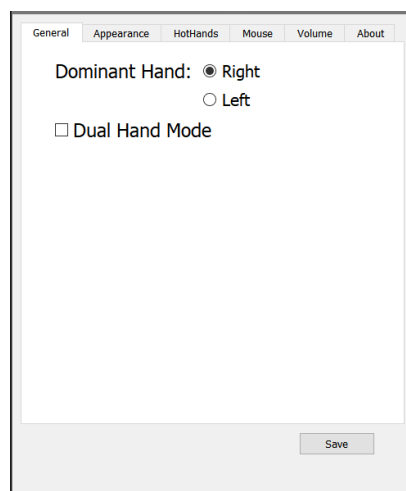


Figure 17.1:
General Tab of settings screen

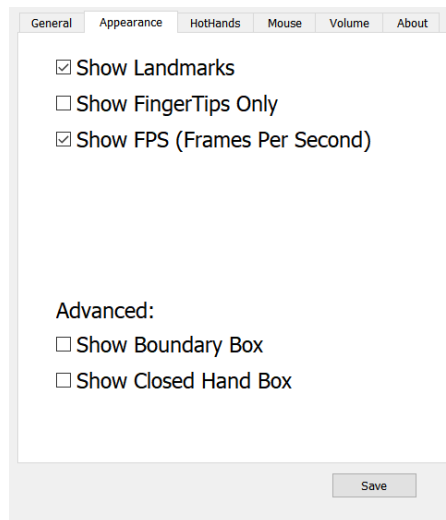


Figure 17.2:
Appearance Tab of settings screen

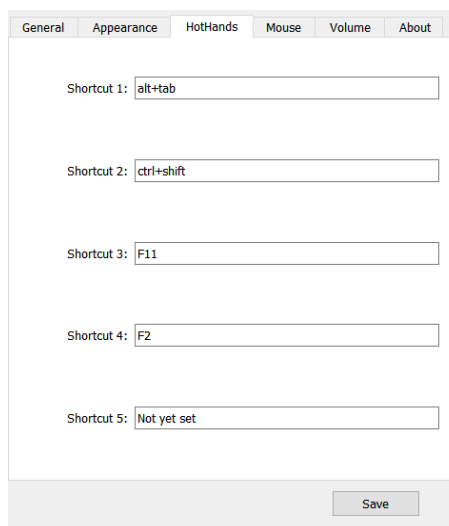


Figure 17.3:
HotHands (Shortcuts) Tab of settings screen

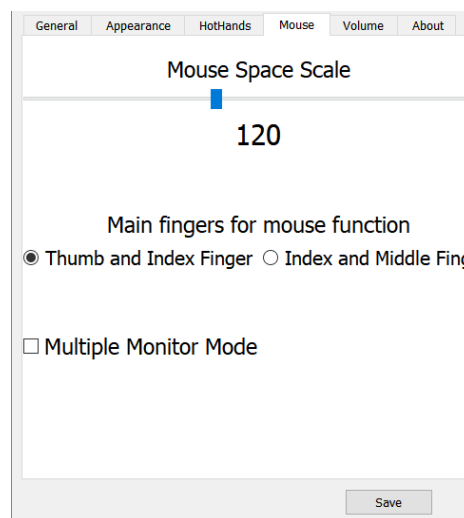


Figure 17.4:
Mouse Tab of settings screen

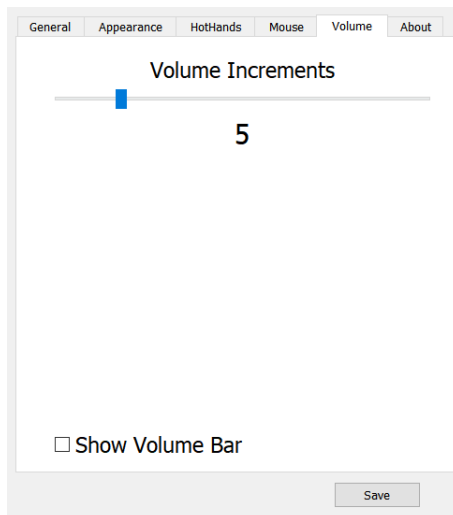


Figure 17.5:
Volume Tab of settings screen

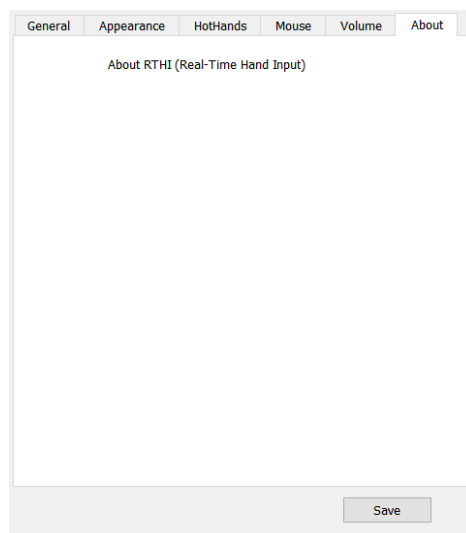


Figure 17.5:
Volume Tab of settings screen

7 RESULTS & EVALUATION

7.1 Results Evaluation

To test the software, each gesture was performed numerous times to validate the accuracy of each function. Figure 18 represents the number of times a gesture was recognised correctly and the accuracy rating.

Gesture	Function	Success	Failure	Accuracy (%)
Thumb and index finger	Mouse Movement/Volume	50	0	100
Thumb and index finger touch	Mouse Click	50	0	100
Middle and index finger touched	Right Click	45	5	90
Middle and index finger touch, while ring finger is up	Scroll up	48	2	96
Middle and index finger touch, while ring finger is down	Scroll down	50	0	100
Thumb and index finger touched but with all other fingers up	Click and Drag	47	3	94
All fingers down	Pause/Play	42	8	84
Index finger up	Skip/ Shortcut2	49	1	98
Index and middle finger up	Previous	48	2	96
Middle finger up	Shortcut3	50	0	100
Ring finger up	Shortcut4	50	0	100
Pinkie finger up	Shortut5	48	2	98
Thumb up/out	Shortcut1	47	3	94

Figure 18:
Experimental results

To further test the capabilities of the system experiments were done using the system from close to the camera and far from the camera. The system shows the same level of accuracy up to a 3m away from the webcam. In theory if the webcam had higher resolution capabilities then it is most likely that the system will be able to detect from even further away, however this is just a theory as I unfortunately do not have the necessary equipment to test this.

Further tests were attempted such as having more than 2 hands presented on the screen. This test was done to check whether the program would falter when multiple hands are present, the test showed that only the first 2 hands that entered the screen are registered. If within the main handDetection class I initialise the maxHands parameter to be more than 2

then the hands can be recognised and will be identified correctly, but only the first 2 hands are considered to be the controller hands.

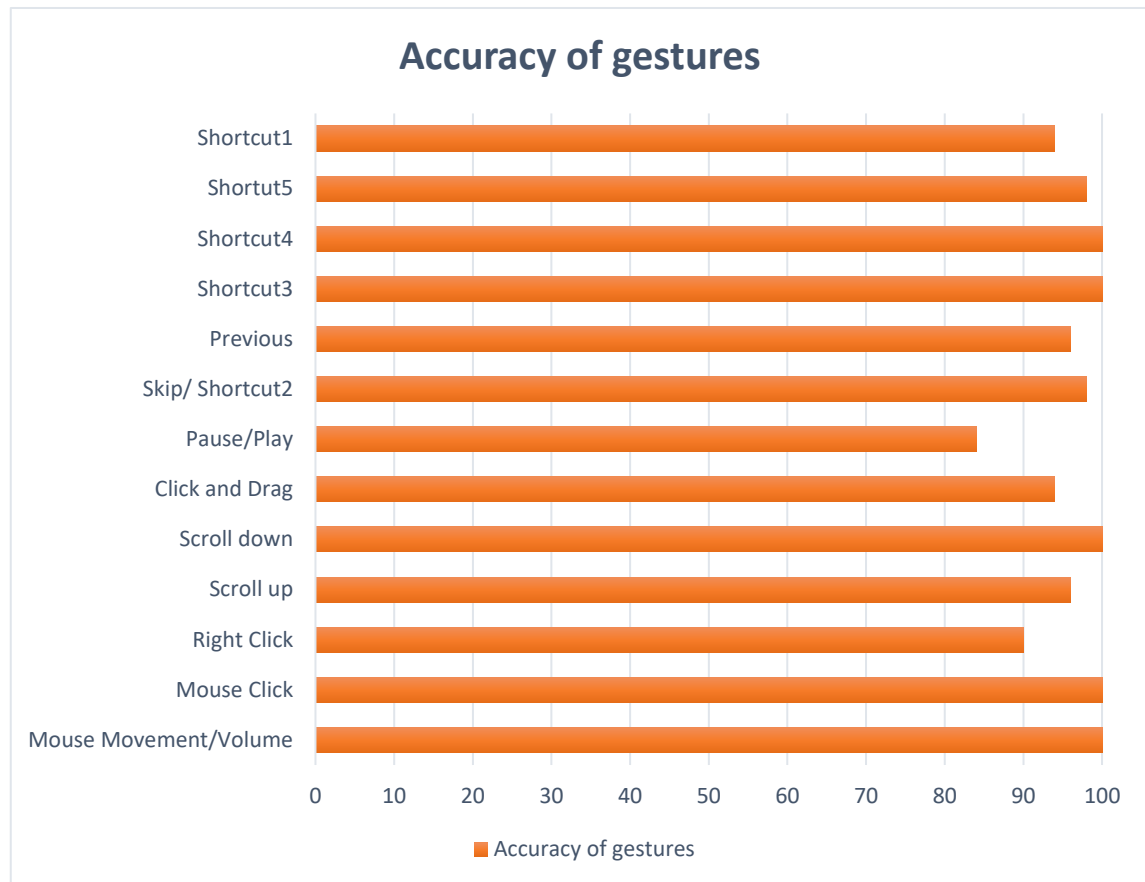


Figure 19:
Graph showing accuracy in percentage of each gesture performed

7.2 Comparison Against Existing Work

When comparing my project with the other projects that was discussed throughout the report, we can see that my product has essentially built upon the virtual mouse designs by others, improving and adapting it for a more user-friendly approach. In comparison to S.R Chowdhury (2020)'s virtual mouse and keyboard design, even though the method of hand tracking was different, the HCI aspect of my project shows an improved version. The keyboard design in particular. S.Shriram's (2021) approach was built upon for the mouse movement, allowing for drag and drop and also a customisable gesture picker.

7.3 Evaluation of the software

In this section of the evaluation I will be going over the previously mentioned system requirements and whether I have met them, I will also evaluate the system design.

7.3.1 REQUIREMENTS

Functional Requirements:

1. The software must be able to open the user's camera.

The software correctly outputs a video stream of the devices' camera.

Yes, the software correctly outputs the device's camera stream

2. The software must distinguish properties of a hand.
 - a. The software must receive all hands presented in front of the camera in a real-time state, constantly updating.
 - b. The software must be able to distinguish each of the 21 landmarks of the hand.
 - c. The software must connect all the landmarks to provide a skeleton overlay on the hand, can be optionally turned on or off.
 - d. The software is able to distinguish between multiple hands, if presented on camera.
 - e. The software must be able to identify whether the hand is left or right.
 - f. The software must be able to operate no matter how far away from the camera the hand is.

All conditions have been met regarding the recognition of the hand.

3. The software must be able to make use of the landmarks in each hand/ be able to distinguish gestures. This can be implemented with the following methods:
 - a. Must be able to distinguish the distance between any two point of one or two hands.
 - b. Must be able to tell whether a finger is up or down.

All the functional requirements for the hand data usage have been met.

4. The software must have the following operational mouse elements to it.
 - a. The mouse must move the computers cursor, depending on the position of the user's hand/fingers.
 - b. The mouse must be able to left-click, depending on performed gesture.
 - c. The mouse must be able to right-click, depending on performed gesture.
 - d. The mouse must be able to scroll down, depending on performed gesture.

- e. **The mouse must be able to scroll up, depending on performed gesture.**
- f. **The mouse must be able to click and drag, depending on performed gesture.**
- g. **The mouse must be able to double click, depending on performed gesture.**

The software does in fact perform all mouse functions, just not with 100% accuracy.

- 5. **The software must be able to allow for text-entry input.**
 - a. **The software must accurately take input.**
 - b. **The software must enter text into the clicked text entry (whether that be on Google Chrome search bar, Word Document, Notes, any other text input etc).**

The software does accept text-entry, however the mouse pointer must already be hovering/clicked on a text enterable box/area.

- 6. **The software must be able to perform media key imitation based on gesture.**
 - a. **The software must be able to Pause current media.**
 - b. **The software must be able to Play current media.**
 - c. **The software must be able to Skip current media.**
 - d. **The software must be able to go Previous on current media.**
 - e. **The software must be able to control the Volume of the device.**

The software can control the media keys allowing for skipping, playing and pausing of either videos or music. Furthermore, the volume functionality works, but at times will not interpret the mute button.

- 7. **The software must be able to perform shortcuts based on gesture.**
 - a. **The software must be able to imitate shortcuts of a keyboard, e.g Alt + Tab, Ctrl + Shift etc.**
 - b. **The software must allow for the user to customise shortcuts.**

The software performs windows shortcuts, however the customisation is not to its best of its abilities. Will be touched on in the next section

- 8. **The software must be able to correctly change modes seamlessly.**
 - a. **The software must make it clear which mode is being used.**
 - b. **The software must have visual indications of the current mode.**

The software does correctly change and show which mode it is on. The software unfortunately freezes during the speech recognition mode.

Non-Functional Requirements

- 1. The software must allow for the user to customise gestures.**
- 2. The software must allow for multiple hand detection modes (enabling other modes to be used with two hands).**
- 3. The software must have a settings GUI that allows user to customise the entire interface.**
- 4. The software must output hand landmarks extremely accurately.**
- 5. The software must operate with no latency.**
- 6. The software must be easy to deploy, configure and maintain in the future.**
- 7. The software produced should meet the highest professional and ethical standards set out by the British Computing Society (BCS).**

The software meets some of these criteria, the customisation can be improved upon, the software does enable two hand mode but is not perfected. The software is however easy to maintain, and can easily be provided updates.

7.3.2 SOFTWARE DESIGN

In this section I will be highlighting the software's design elements and where I could have made changes and improvements, but also where I have made correct decisions.

When first creating the software the idea was to have many different gestures that would switch modes, however with further thought the idea was scrapped. This was due to how complicated it would make the software for users. The idea to overlay icons on the screen was implemented and I believe that it was a much better decision.

Furthermore, the original idea was to also have a settings icon on the screen for the user's to interact with. The settings icon would open up the PyQt Settings GUI. However, when attempting to implement this there was several crashes and errors caused by PyQt that it would have taken too much time to implement.

An improvement that could have been made regarding the GUI is that rather than having PyQt GUI and the OpenCV camera image separately. PyQt has OpenCV support allowing for the camera image to be displayed within the GUI. Implementing this would have allowed for a working settings button, and would have a merged system, rather than what the final prototype is, Settings and Main system being separate.

There was a drawback which was that the camera image would freeze every time the microphone was needed to pick up audio. To fix the error, an attempt was made to use

multiprocessing to allow for microphone use to be run through a separate process, but this still caused freezes. Threading was also attempted, but proved not to work. The freeze could be due to the camera's mic being used and therefore would cause the video stream to pause, but I am not sure.

Another small issue I wanted to address was the icons sometimes will appear with transparent backgrounds and sometimes without. This problem was difficult to fix as I had no idea why it would do this.

Overall, however, the design decisions and changes made throughout the projects design and implementation were made appropriately and accordingly allowing for the final prototype to be much greater than what I had anticipated.

7.4 Personal Evaluation

Overall personally I am quite happy with the prototype that I ended up with. Despite being very long and time consuming the process and design of the software was fun to say the least. There were faults in my project due to lack of time, which could have been better managed with better planning. Furthermore I did not expect the research element of the project to take so long, this was mainly due to the lack of similar projects I could find, all though this provided a fun challenge, it was difficult to find reassurance through other projects. The chosen Agile methodology proved to be very useful as breaking down my project into smaller problems, such as splitting the modes/features, proved to be very beneficial in making sure each feature was polished.

I believe that the product is a very good start for what hand recognition can accomplish as a real-time input to replace the normal peripherals, and especially during the time of the covid-19 pandemic can be very useful. I believe that as hand tracking techniques and algorithms improve, more and more usable software will be created for developers like me to have fun and create interactive projects.

Throughout the entire process I was able to make use of many of the skills I have gained across the last 3 years. From using my software engineering management skills to my algorithms and paradigms knowledge, I was able to test the waters of what being a software engineer will be like.

8 CONCLUSION & FUTURE WORK

8.1 Conclusion

The aim of this project was to create a software that would enable the use of a webcam as input for a traditional computer, allowing the user to mimic mouse and keyboard movements and presses. The project then further developed to allow for media key usage as well as allowing for shortcuts to be performed. The software even allows for users to change the system based on their preferences. The literature review discussed methods of input and similar existing works, outlining various methods of hand detection and possible use cases. With some of these methods implemented in the final prototype.

8.2 Future Work

This project was fun for me to create and develop, and therefore in all honesty I may end up further working on the project after the submission. There is so much possibility for a software like this, being able to control my computer from a distance would be so beneficial for some people. Several features could be added, such as usage with smart products as mentioned in Gierad Laput's (2016) report about the "ViBand", or a paint mode as seen in S. Gulati's work (2022), or a presentation mode allowing teachers and lecturers to move through slides through simple gestures. The use and demand for computer vision related products, is only going to go up.

REFERENCES

Mohamed, G.K., Kandil, H. and Zaki, H.E.E.D., 2019. The Gestures and Symbolism through the Collection of the Egyptian Museum. *Journal of the Faculty of Tourism and Hotels-University of Sadat City*, 3(2).

Roth, W.-M. (2001) 'Gestures: Their Role in Teaching and Learning', *Review of Educational Research*, 71(3), pp. 365–392. doi: 10.3102/00346543071003365.

Wilcox, S., 2004. Gesture and language: Cross-linguistic and historical data from signed languages. *Gesture*, 4(1), pp.43-73.

Wachs, J.P., Stern, H.I., Edan, Y., Gillam, M., Handler, J., Feied, C. and Smith, M., 2008. A gesture-based tool for sterile browsing of radiology images. *Journal of the American Medical Informatics Association*, 15(3), pp.321-323.

Chen, Z.H., Kim, J.T., Liang, J., Zhang, J. and Yuan, Y.B., 2014. Real-time hand gesture recognition using finger segmentation. *The Scientific World Journal*, 2014.

L. K. Phadtare, R. S. Kushalnagar and N. D. Cahill, "Detecting hand-palm orientation and hand shapes for sign language gesture recognition using 3D images," 2012 Western New York Image Processing Workshop, 2012, pp. 29-32, doi: 10.1109/WNYIPW.2012.6466652.

Y. Fang, J. Cheng, J. Wang, K. Wang, J. Liu and H. Lu, "Hand posture recognition with co-training," 2008 19th International Conference on Pattern Recognition, 2008, pp. 1-4, doi: 10.1109/ICPR.2008.4761066.

H. Takahashi and Y. Kitazono, "Integration of Hand Gesture and Multi Touch Gesture with Glove Type Device," 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD), 2016, pp. 81-86, doi: 10.1109/ACIT-CSII-BCD.2016.027.

V. V. Reddy, T. Dhyanchand, G. V. Krishna and S. Maheshwaram, "Virtual Mouse Control Using Colored Finger Tips and Hand Gesture Recognition," 2020 IEEE-HYDCON, 2020, pp. 1-5, doi: 10.1109/HYDCON48903.2020.9242677.

S. S. Rautaray and A. Agrawal, "Interaction with virtual game through hand gesture recognition," 2011 International Conference on Multimedia, Signal Processing and Communication Technologies, 2011, pp. 244-247, doi: 10.1109/MSPCT.2011.6150485.

Doe-Hyung Lee and Kwang-Seok Hong, "Game interface using hand gesture recognition," 5th International Conference on Computer Sciences and Convergence Information Technology, 2010, pp. 1092-1097, doi: 10.1109/ICCIT.2010.5711226.

R. Han et al., "Multi-sensors Based 3D Gesture Recognition and Interaction in Virtual Block Game," 2017 International Conference on Virtual Reality and Visualization (ICVRV), 2017, pp. 391-392, doi: 10.1109/ICVRV.2017.00091.

S. R. Chowdhury, S. Pathak and M. D. A. Praveena, "Gesture Recognition Based Virtual Mouse and Keyboard," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), 2020, pp. 585-589, doi: 10.1109/ICOEI48184.2020.9143016.

M. Ranawat, M. Rajadhyaksha, N. Lakhani and R. Shankarmani, "Hand Gesture Recognition Based Virtual Mouse Events," 2021 2nd International Conference for Emerging Technology (INCET), 2021, pp. 1-4, doi: 10.1109/INCET51464.2021.9456388.

Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.L. and Grundmann, M., 2020. Mediapipe hands: On-device real-time hand tracking. arXiv preprint arXiv:2006.10214.

Woods, V., Hastings, S., Buckle, P. and Haslam, R. (2002) Ergonomics of using a mouse or other non-keyboard input device, Loughborough University, [online] Available at: <https://hdl.handle.net/2134/2559>.

WDD STAFF, 2009. Operating system interface design between 1981-2009. Webdesignerdepot. Available at: <https://www.webdesignerdepot.com/2009/03/operating-system-interface-design-between-1981-2009/>

Nakaseko, M. et al. (1985) 'Studies on Ergonomically Designed Alphanumeric Keyboards', Human Factors, 27(2), pp. 175-187. doi: 10.1177/001872088502700205.

Chiesa, V. and Frattini, F., 2011. Commercializing technological innovation: Learning from failures in high-tech markets. *Journal of Product Innovation Management*, 28(4), pp.437-454.

Chris Harrison, Hrvoje Benko, and Andrew D. Wilson. 2011. OmniTouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11)*. Association for Computing Machinery, New York, NY, USA, 441–450. <https://doi.org/10.1145/2047196.2047255>

Ries, J., 2018. Dangerous bacteria on public touch screens. Healthline. Available at: <https://www.healthline.com/health-news/want-to-avoid-dangerous-bacteria-dont-use-touch-screens>

Guojun Cheng and Jibin Yin, "The study of a novel gesture technique based on "Pen+touch" in multi-touch interfaces," *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013, pp. 1130-1134, doi: 10.1109/MEC.2013.6885233.

J. Yin and Y. Gao, "The design and research of 3D desktop interface based on the Pen + Touch," *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2016, pp. 129-133, doi: 10.1109/RCAR.2016.7784013.

Jie Song, Gábor Sörös, Fabrizio Pece, Sean Ryan Fanello, Shahram Izadi, Cem Keskin, and Otmar Hilliges. 2014. In-air gestures around unmodified mobile devices. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 319–329. <https://doi.org/10.1145/2642918.2647373>

Andrew D. Wilson. 2006. Robust computer vision-based detection of pinching for one and two-handed gesture input. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06)*. Association for Computing Machinery, New York, NY, USA, 255–258. <https://doi.org/10.1145/1166253.1166292>

Gierad Laput, Robert Xiao, and Chris Harrison. 2016. ViBand: High-Fidelity Bio-Acoustic Sensing Using Commodity Smartwatch Accelerometers. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. Association for Computing Machinery, New York, NY, USA, 321–333. <https://doi.org/10.1145/2984511.2984582>

K. L. Brown and E. B. George, "CTIMIT: a speech corpus for the cellular environment with applications to automatic speech recognition," *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1995, pp. 105-108 vol.1, doi: 10.1109/ICASSP.1995.479284.

Robert Y. Wang and Jovan Popović. 2009. Real-time hand-tracking with a color glove. *ACM Trans. Graph.* 28, 3, Article 63 (August 2009), 8 pages.

<https://doi.org/10.1145/1531326.1531369>

Sharma, Pratyusha & Joshi, Ravi & Bobby, Riby & Saha, Subir & Matsumaru, Takafumi. (2015). Projectable Interactive Surface Using Microsoft Kinect V2: Recovering Information from Coarse Data to Detect Touch. 10.1109/SII.2015.7405081.

R. Damindarov, C. A. Fam, R. A. Bobby, M. Fahim, A. Klimchik and T. Matsumaru, "A depth camera-based system to enable touch-less interaction using hand gestures," 2021 International Conference "Nonlinearity, Information and Robotics" (NIR), 2021, pp. 1-7, doi: 10.1109/NIR52917.2021.9666090.

T. Sharma, S. Kumar, N. Yadav, K. Sharma and P. Bhardwaj, "Air-swipe gesture recognition using OpenCV in Android devices," 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), 2017, pp. 1-6, doi: 10.1109/ICAMMAET.2017.8186632.

K. H. Shibly, S. Kumar Dey, M. A. Islam and S. Iftekhar Showrav, "Design and Development of Hand Gesture Based Virtual Mouse," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019, pp. 1-5, doi: 10.1109/ICASERT.2019.8934612.

I. J. Tupal and M. Cabatuan, "Vision-Based Hand Tracking System Development for Non-Face-to-Face Interaction," 2021 IEEE 13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM), 2021, pp. 1-6, doi: 10.1109/HNICEM54116.2021.9731873.

S. Gulati, A. K. Rastogi, M. Virmani, R. Jana, R. Pradhan and C. Gupta, "Paint / Writing Application through WebCam using MediaPipe and OpenCV," 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM), 2022, pp. 287-291, doi: 10.1109/ICIPTM54933.2022.9753939.

W. A. A. Praditasari, R. Aprilliyani and I. Kholis, "Design and Implementation of Interactive Virtual Museum based on Hand Tracking OpenCV in Indonesia," 2021 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2021, pp. 253-256, doi: 10.23919/EECSI53397.2021.9624265.

V. Chunduru, M. Roy, D. R. N. S and R. G. Chittawadigi, "Hand Tracking in 3D Space using MediaPipe and PnP Method for Intuitive Control of Virtual Globe," 2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC), 2021, pp. 1-6, doi: 10.1109/R10-HTC53172.2021.9641587.

V. I. Saraswati, R. Sigit and T. Harsono, "Eye gaze system to operate virtual keyboard," 2016 International Electronics Symposium (IES), 2016, pp. 175-179, doi: 10.1109/ELECSYM.2016.7860997.

Xu, P., 2017. A real-time hand gesture recognition and human-computer interaction system. arXiv preprint arXiv:1704.07296.

S. Shriram, B. Nagaraj, J. Jaya, S. Shankar, P. Ajay, "Deep Learning-Based Real-Time AI Virtual Mouse System Using Computer Vision to Avoid COVID-19 Spread", Journal of Healthcare Engineering, vol. 2021, Article ID 8133076, 8 pages, 2021.
<https://doi.org/10.1155/2021/8133076>

Guiard, Y. and Beaudouin-Lafon, M., 2004. Fitts' law 50 years later: Applications and contributions from human-computer interaction. International Journal of Human-Computer Studies.

APPENDIX

Chapter 6 – Implementation Code Snippets to show progress.

Reading the camera data.

```
video_object = cv2.VideoCapture(0)
video_object.set(3,wVideo)
video_object.set(4,hVideo)

while True:
    success, camera_img = video_object.read()

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

Previous implementation of findDistance()

```
def findDistance(self,p1, p2, img, draw=True, r=15,t=3):
    x1, y1 = self.landmarksList[p1][1:]
    x2, y2 = self.landmarksList[p2][1:]
    cx, cy = (x1 + x2) // 2, (y1 + y2)//2
    if draw:
        #cv2.line(img, (x1,y1), (x2, y2), (255, 0, 255), t)
        #cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
        #cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
        #cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)
        length = math.hypot(x2 - x1, y2 - y1)

    return length, img, [x1, y1, x2, y2, cx, cy]
```

FingersUp before changing to fix problem where jitters and errors occur due to hand orientation (version 1)

```
def isFingersUp(self):
    fingers = []

    if self.landmarksList[self.fingerTips[0][0]] [1] > self.landmarksList[self.fingerTips[0][0] - 1] [1]:
        fingers.append(1)
    else:
        fingers.append(0)

    for id in range(1,5):
        if self.landmarksList[self.fingerTips[id][0]][2] < self.landmarksList[self.fingerTips[id][0]-2][2]:
            fingers.append(1)
        else:
            fingers.append(0)
    return fingers
```

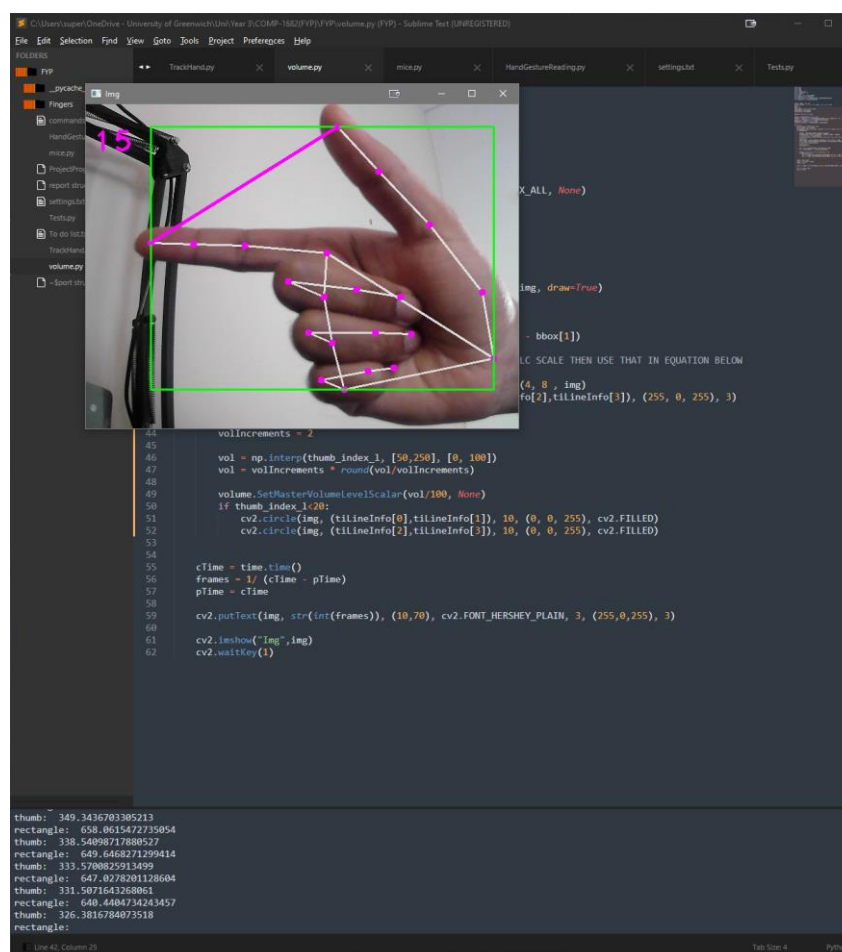
Composition before implementing 2 hand mode and adding hand identifier explain problem with it.

```
def findLMPosition(self, img, handNo=0, draw=True):
    listX = []
    listY = []
    bbox = []
    hbox, lbox = 0, 0
    rectangleScale = 0
    self.landmarkslist = []
    if self.cam_video.multi_hand_landmarks:
        print(self.cam_video.multi_handedness)
        myHand = self.cam_video.multi_hand_landmarks[handNo]
        for id, lm in enumerate(myHand.landmark):
            height, width, channel = img.shape
            centre_x, centre_y = int(lm.x*width), int(lm.y*height)
            listX.append(centre_x)
            listY.append(centre_y)
        # print(id, centre_x, centre_y)
        self.landmarkslist.append([id, centre_x, centre_y])
        if draw:
            cv2.circle(img, (centre_x, centre_y), 5, (255, 0, 255), cv2.FILLED)

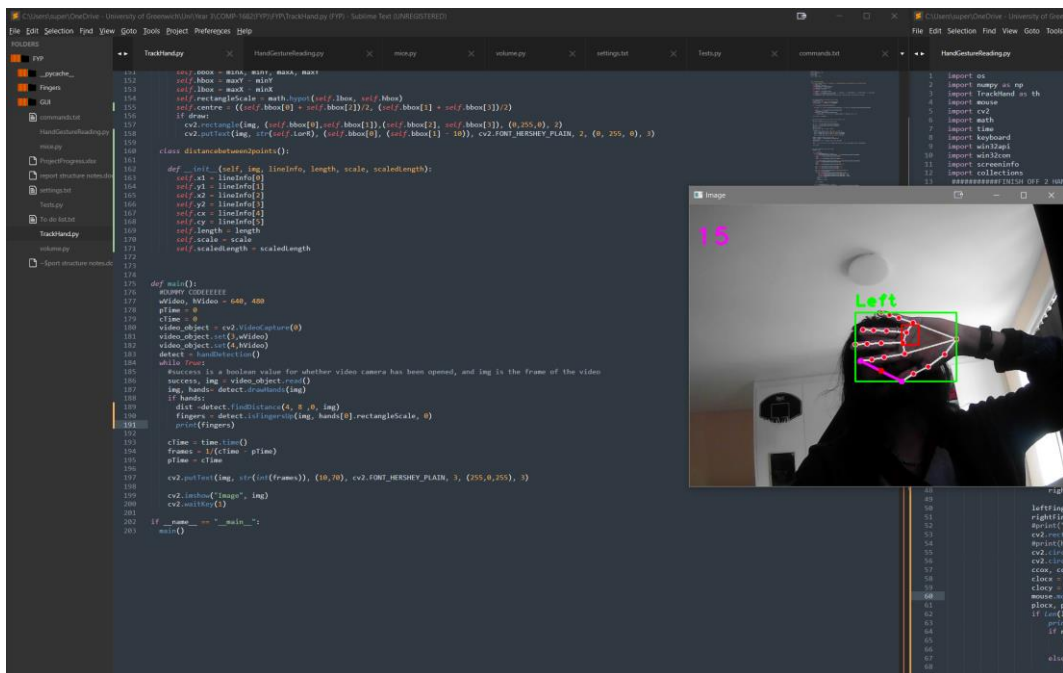
    minX, maxX, minY, maxY = min(listX), max(listX), min(listY), max(listY)
    bbox = [minX, minY, maxX, maxY]
    hbox = maxX - minX
    lbox = maxY - minY
    rectangleScale = math.hypot(lbox, hbox)

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

Addition of scaling with the bounding box for volume



Orientation problem occurring with closingBox



Planning Done:

The use of excel to contain each feature and whether their standards have been met

Feature

Feature	Operations	How it works	How it should work	Expectations met?	Possible improvements
Hand recognition	Landmarks are displayed	Using openmv to open the users camera, displays the landmarks of each hand within frame and also stores each landmarks co-ordinates (updates every frame) in a list. This is used as the basis for other functions to be implemented.	Real time video based off webcam input to show the joints in the hand, and providing landmarks.	Yes	The values should be different based on the users distance from the camera, as it does not scale. Add left hand and multi hand recognition. Make each hand more useable.
Mouse	Cursor control	The middle point between the 2 selected pointer fingers is used as the x,y for the cursor.	The mouse pointer should move around the screen according to where the users' index finger (2) is.	Yes, but causes lag and could be clunky at times	Adding alternative methods of cursor control, or using different fingers to operate it
	Left Click	Meeting the 2 pointer fingers will perform the left click function	Using the thumb (4) the user can tap their finger with the index finger (8) in order to replicate a left click	Yes, but clunky at times	Adding alternative methods of Left click, or using different fingers to operate it
	Right Click	When both fingertip 8 and 12 are close enough the right click function is performed	When 2 fingertips meet or a specific finger e.g if the pinkie finger (20) is held up	Yes, but clunky at times	
	Left Click drag		When moving index finger with thumb/middle finger, the mouse should replicate holding the left click, when dragged this should replicate the drag/select function.		
	Scroll	If the distance between the 2 right click fingers and the centre of the hand are a certain distance apart then the mouse scrolls up, if the 2 fingers are inside the palm and pointing down, the mouse scrolls down.	Using your 2 rightclick fingers, creating a downward waving motion to scroll down, and upward motion to scroll up	Yes, but sometimes sees scrolls as right clicks	Apply time to make scroll speed slower
	Double click	----->	Works the same way the left click works, but the user must make the touching motion twice	Works on occasion	
Keyboard	Keyboard interface	----->	Opens up on screen keyboard on the screen where the mouse is being used.		
	Shortcuts				
	Gesture to open				
	Windows button				
Media key shortcuts	Pause/Play button				
	Skip button				
	Previous button				
Custom					
Gestures		The distance between the thumb (4) and fingertip 8 are shown with a line, this represents the volume level, the further apart the landmarks the louder the volume will be	Depending on the distance between fingertip 4 and fingertip 8 the volume will change according to a volume scale of 0-100.	Yes	make which fingers customisable, and also the sensitivity of the volume distance, also a gesture that understands to lock the volume at the volume shown (this means an indicator for the volume will be needed on the imo).