



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

GESTURE RECOGNITION-BASED DEVICE CONTROL

Mudit Garg

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

INTEGRATED MASTERS IN COMPUTER ENGINEERING

Supervisor: Meriel Huggard

April 2024

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Mudit Garg

April 15, 2024

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Mudit Garg

April 15, 2024

GESTURE RECOGNITION-BASED DEVICE CONTROL

Mudit Garg, Integrated Masters in Computer Engineering

University of Dublin, Trinity College, 2024

Supervised by Meriel Huggard

Hand gestures is one of the primary modes of communication after speech and is the key medium of communication employed by the people who are speech or hearing impaired. It acts as a gateway for their interaction with the world. Smart devices offer convenience and efficiency, yet traditional control methods for these smart devices like voice control still present barriers for individuals with disabilities. In this ever-evolving landscape, making the technology more inclusive and accessible to these diverse user groups remains paramount.

This project aims to contribute to this initiative by exploring the development and implementation of a vision-based control system that recognises hand gestures for the purpose of controlling various smart home devices that people use in their day-to-day life. The architecture of the system uses OpenCV to capture the camera feed and Google's Mediapipe for hand key points extraction. It then uses this key-point data compiled and processed to a Numpy array to train an LSTM neural network model which recognises the gestures. Upon recognising the gesture, the model finds the corresponding command according to the user's input and sends the command to the smart device via the local network connection.

This endeavour will contribute to the broader goal of enhancing accessibility of today's technology and bridge the gap between the people who rely on gesture as their primary mode of communication and the technology they daily rely upon. The focus of this study is to enhance the accessibility of the smart devices that are very prominent in modern day homes where most of the traditional appliances like TVs, refrigerators, air conditioning units, speakers, kitchen appliances like coffee makers and toasters are now being replaced by their smart counterparts which can be easily controlled via smartphones or other device control mechanism.

Acknowledgements

First and foremost, I would like to express my sincere gratitude towards my incredible supervisor Meriel Huggard who has always been very supportive with my every initiative and decision and is one of the nicest people I have met in my industry. Her support and guidance throughout the journey of this project has been invaluable and her suggestions have always been genuine and very helpful. I am very confident to say that without her mentorship and support, this project would not have reached to its full potential.

I would also like to extend my gratitude towards Ciaran McGoldrick who had some genuine recommendations towards the project. His contributions led to some very important bits of this project and its write-up. His insights helped me tackle some rough edges of this project and his perspective genuinely helped me improve the project by guiding my attention to some areas of the project which I had overlooked.

Also, I would also like to thank all the people whose work has been credited in my study. Their invaluable contribution and efforts towards the field is the sole reason I was able to complete my study. Without their work, this project would not have been possible.

Thank you for all your contributions towards this study. Direct or indirect contribution, every help counts towards the success of this project.

MUDIT GARG

University of Dublin, Trinity College

April 2024

Table of Content

Introduction.....	11
1.1 Problem Statement	11
1.2 State of the Art	12
1.2.1 Current state of Technologies	12
1.2.2 In Real World Products	15
1.3 Project Objectives	18
1.3.1 Overall Objective	18
1.3.2 Specific Objectives	18
1.4 Document Overview	19
Design	21
2.1 Design Principles	21
2.1.1 Easy to replicate	21
2.1.2 Organised and easy to debug	23
2.1.3 User centric design.....	24
2.1.4 Design simplicity and consistency.....	24
2.2 Design Phases.....	25
2.2.1 Initial Development	25
2.2.2 Project Modularisation.....	26
2.2.3 Frontend version development.....	26
Architecture	27
3.1 High Level Overview	27
3.2 Model Preparation.....	28
3.3 Gesture-based device control	29
3.4 Architecture Limitations	30
3.4.1 Lot of intertwined intricacies for project setup and maintenance.....	30
Implementation	32
4.1 Development Environment	32
4.1.1 Python	32
4.1.2 Jupyter Notebook.....	34
4.2 Technology Stack.....	34
4.2.1 OpenCV	34
4.2.2 Mediapipe	35
4.2.3 Numpy.....	36

4.2.4	Tensorflow	36
4.3	Tool selection	37
4.3.1	Visual Studio Code	37
4.3.2	Git	38
4.3.3	Flask.....	39
4.4	Algorithm Implementation.....	39
4.4.1	Implementation Overview	39
4.4.2	Key-point detection and frame augmentation using OpenCV and Mediapipe ..	40
4.4.3	Data Collection and Preparation	43
4.4.4	Data Preprocessing.....	46
4.4.5	LNN Gesture Recognition Model.....	47
4.4.6	Model Training.....	49
4.4.7	Communication with external devices	53
4.4.8	User Interface Design	54
4.5	Implementation Limitations	55
4.5.1	Security and Privacy Limitations.....	56
4.5.2	Interface Limitations	57
4.6	Implementation Issues.....	58
4.6.1	A separate model for hand key-point recognition and gesture recognition	58
4.6.2	No wait time between the recognised gestures	59
Conclusion	60
5.1	Future Work.....	60
5.2	Final Words	61
Bibliography	63

Table of Figures

Figure 1: Different techniques for hand gestures recognition	13
Figure 2: Vision based sign language models classifications [20].....	14
Figure 3: Automated project setup workflow	22
Figure 4: Project codebase folder structure	23
Figure 5: Scripts folder structure	23
Figure 6: CLI of dataset functions of the project	24
Figure 7: Frontend version of the project	24
Figure 8: Initial Development Phase Approach	25
Figure 9: Architecture for the case of new 'scenario' creation	28
Figure 10: Architecture for the case of smart device control.....	29
Figure 11: Summary of PyPI ecosystem statistics (May 2019) [5]	32
Figure 12: Number of new packages, active packages, new releases, and new authors on PyPI by year; 2019 is a partial year [5]	33
Figure 13: A brief code snippet of the Flask server made using Python.....	33
Figure 14: Jupyter Notebook snippet from the project	34
Figure 15: CPU versus GPU performance comparison [6]	35
Figure 16: Instance of enhanced productivity using VSC IDE using the integrated CLI and Multi-file alignment features during Modularization phase	37
Figure 17: A portion of the Git commit history for the project.....	38
Figure 18: Code snippet for Mediapipe detection.....	41
Figure 19: Mediapipe Holistic model initialisation	41
Figure 20: Final output after Mediapipe detection and frame augmentation.....	42
Figure 21: Code Snippet for frame augmentation with hand key-points.....	42
Figure 22: Summary of the key-point extraction pipeline	43
Figure 23: Create a scenario option in the web interface of the system	43
Figure 24: Code snippet responsible for key-point data collection and storage for every action	44
Figure 25: Starting frame for a gesture record.....	44
Figure 26: Gesture recording in progress.....	45
Figure 27: Key-points extraction function.....	45
Figure 28: Accumulation of all data into a single NumPy array to ready for model training ..	46

Figure 29: Expected shapes of the feature dataset and label dataset for data collection of 3 gestures using 30 videos (with 30 frames each) for each gesture.....	46
Figure 30: Dataset splitting into training and testing dataset.....	47
Figure 31: Summary of model used for gesture recognition	47
Figure 32: LNN Gesture Recognition model architecture	48
Figure 33: Gesture Recognition model compilation.....	49
Figure 34: Highest peak in the epoch loss at 89 epochs	50
Figure 35: Lowest dip in the model categorical accuracy at 115 epochs	50
Figure 36: Epoch loss of the model after the 200-epoch mark	51
Figure 37: Categorical accuracy of the model at and after 186 epoch mark	51
Figure 38: Code Snippet using Test dataset to set the model performance on unfamiliar data	51
Figure 39: Code snippet for testing the trained model with the testing dataset.....	52
Figure 40: Confusion Matrix and Accuracy Score of the trained model with the testing dataset	52
Figure 41: Testing of the trained model in real time.....	52
Figure 42: Code Snippet showing the script that handles the commands for controlling Amazon Fire TV stick using ADB connection.....	53
Figure 43: Demonstration of establishing connection with Fire TV stick using local IP address	54
Figure 44: Code snippet showing a particular endpoint that replicates the create scenario action of the system	55
Figure 45: Button endpoint on the website for create scenario action with appropriate input form.....	55

Acronyms

HCI	Human Computer Interaction
LSTM	Long Short-Term Memory
LNN	LSTM Neural Network
CLI	Command Line Interface
CI/CD	Continuous Integration and Continuous Deployment
GPU	Graphical Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
NPP	NVIDIA Performance Primitives
IDE	Integrated Development Environment
VSC	Visual Studio Code
ReLU	Rectified Linear Unit
BGR	Blue Green Red
RGB	Red Green Blue
LAN	Local Area Network

Chapter 1

Introduction

Sign language is the key medium of communication employed by the people who are speech or hearing impaired [1]. It is one of the key modes of non-verbal communication which makes use of hand gestures to convey a letter, a word or a phrase. As computers become more pervasive in society, facilitating natural human–computer interaction (HCI) will have a positive impact on their use [2]. Hence, there has been growing interest in the development of new approaches and technologies for bridging the human–computer barrier. The ultimate aim is to bring HCI to a regime where interactions with computers will be as natural as interactions between humans, and to this end, incorporating gestures in HCI is an important research area [3]. It is projected that, within the coming decade, there will be more than 50 billion smart objects connected to the Internet of Things (IoT). These smart objects, which connect the physical world with the world of computing infrastructure, are expected to pervade all aspects of our daily lives and revolutionize a number of application domains such as healthcare, energy conservation, transportation, etc [4]. Hence, making technologies like smart devices accessible and inclusive for people from different types of backgrounds becomes paramount. This includes people who are sometimes limited by their inability to use various features of these smart objects.

1.1 Problem Statement

Smart appliances have revolutionised the industry and has become a very big part of every home in today’s modern technological society. Traditional methods of interacting with smart devices, such as voice commands or physical buttons, may pose significant challenges for individuals with disabilities, limiting their ability to independently control and interact with their surroundings. Moreover, existing solutions often overlook the diverse needs and capabilities of users, leading to a lack of inclusivity and accessibility in smart home technology.

One of the primary challenges is the reliance on speech-based interfaces for controlling smart devices, which inherently excludes individuals with speech impairments or those who are non-verbal. Similarly, individuals with hearing impairments may face difficulties in receiving auditory feedback or alerts from smart devices, further exacerbating their challenges in navigating and interacting with their environment. This disparity in access to and usability of smart home technology underscores the pressing need for alternative methods of interaction that cater to the diverse needs of users.

Hence, it is a very prominent shortcoming of this technology to not be accessible to everyone. The main problem that this project intends to tackle is the inability of people with speech

impairment or hearing disability to use the convenience of voice control and the associated features in various smart devices.

Many gesture recognition systems struggle to accurately interpret and classify a wide range of gestures, leading to frustration and inefficiency for users. Additionally, the integration of gesture recognition with smart home devices is an area of technology that has not been explored to a great extent even though it has several benefits and makes the technology that incorporate it, more inclusive and accessible.

The existing solutions for gesture-based device control lack robustness, accuracy, and adaptability, hindering their widespread adoption and effectiveness in real-world settings. Moreover, the current implementations of the gesture-based control systems lack a certain level of flexibility and ease to use that the users need, when using them to perform tasks. Usually, these systems tend to focus on very basic gestures and their recognition which their map to some respective actions. Considering the wide range of smart devices that are being actively developed and maintained in the current industry, these systems do not account for the scalability in terms of gesture options and scenarios. These systems usually pertain to a limited number of gestures that are predefined and pretrained and do not provide any option to expand the capability of adding more gestures to the system.

Moreover, there are ethical considerations surrounding data privacy, consent, and inclusivity that must be addressed in the design and implementation of gesture-based device control systems. Ensuring that user data is handled responsibly and ethically, obtaining informed consent from users, and designing interfaces that are inclusive and accessible to individuals with diverse abilities are paramount concerns that must be addressed to foster trust and acceptance of the technology.

This project encapsulates the multifaceted challenges and opportunities inherent in the development of a gesture-recognition-based device control system and tries to further the technological stand point of the current state. By addressing these challenges and delivering a robust, inclusive, and user-friendly solution, it aims to make significant strides towards enhancing accessibility and usability in smart home environments for individuals with disabilities, ultimately improving their quality of life and independence.

1.2 State of the Art

1.2.1 Current state of Technologies

Linguists consider both communication using speech and gestures to be types of natural language, meaning that both emerged through an abstract, protracted aging process and evolved over time without meticulous planning [9]. Gesture based communication technology has undergone a significant evolution, transitioning from early attempts with simple hand movements to sophisticated systems capable of understanding complex combinations of gestures to convey something. Gesture based communication involves the usage of different

parts of the body, such as fingers, hand, arm, head, body, and facial expression [10]. There are five main parameters in gesture-based communication, which are hand-shape, palm orientation, movement, location, and expression/non-manual signals. To have an accurate convey of an idea, all of these five parameters must be performed correctly.

Most current technologies which are developed, keep verbal communication and commands in mind but there are very few of them which have features that also cater to the speech and hearing-impaired communities who lack the means of access and inclusion due to their shortcoming. Hence, individuals who rely on gestures as their primary means of communication often encounter barriers when interacting with mainstream technological devices. Various research papers based on hand gestures have adopted many different techniques, including those based on instrumented sensor technology and computer vision for HCI.

Previously, hand gesture recognition was achieved with wearable sensors attached directly to the hand with gloves. These sensors detected a physical response according to hand movements or finger bending. The data collected were then processed using a computer connected to the glove with wire. This system of glove-based sensor could be made portable by using a sensor attached to a microcontroller. Although the techniques mentioned above have provided good outcomes, they have various limitations that make them unsuitable for the elderly, who may experience discomfort and confusion due to wire connection problems. In addition, elderly people suffering from chronic disease conditions that result in loss of muscle function may be unable to wear and take off gloves, causing them discomfort and constraining them if used for long periods. Moreover, some sensors are quite expensive. Some of these problems were addressed in a study by Lamberti and Camastra [11], who developed a computer vision system based on coloured marked gloves. Although this study did not require the attachment of sensors, it still required coloured gloves to be worn [12].

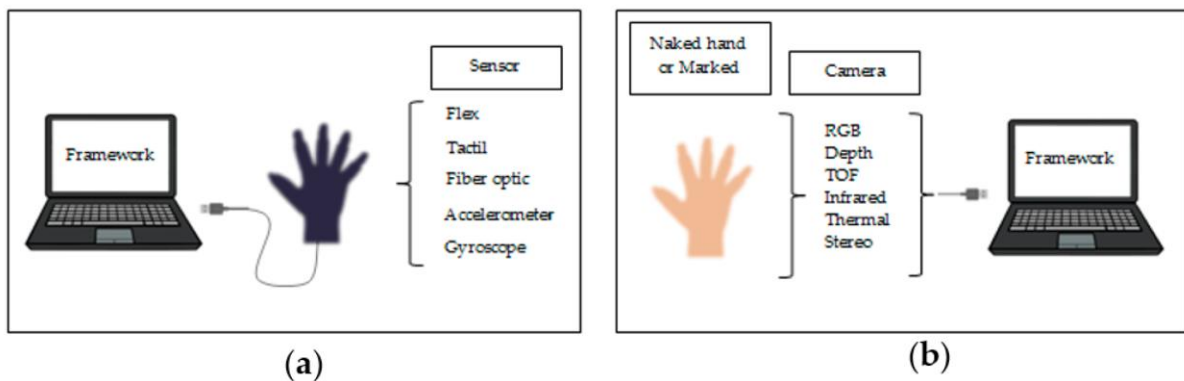


Figure 1: Different techniques for hand gestures recognition

In the above demonstration borrowed from the research paper of Hand Gesture Recognition Based on Computer Vision: A Review of Techniques [12], the part (a) denotes the sensor-based approach using a glove with attached sensors either connected to computer or portable device capable to receive, process and store the data sent over by the glove. The sensor-based approach utilised a plethora of sensors like accelerometer, gyroscope, etc to track the hand movements

and record the spatial and temporal difference data that will be used as gesture. These drawbacks led to the development of promising and cost-effective techniques that did not require cumbersome gloves to be worn. These techniques are called camera vision-based sensor technologies.

In the part (b), the figure demonstrates the computer vision-based approach of using a camera using a marked glove or just a naked hand. This approach utilises the camera to record various properties of the hand as an object in the frame. These properties range from colour spectrum in RGB, the depth and distance of the object in focus from the camera. This approach can utilise the various types of cameras like Infrared and Thermal to capture more data about the object hence preparing more data for denoting the gesture as data within the system. With the evolution of open-source software libraries, it became easier than ever to detect hand gestures that can be used under a wide range of applications like clinical operations [13], sign language [14], robot control [15], virtual environments [16], home automation [17], personal computer and tablet [18], gaming [19].

Visual gesture recognition is a complex research area in computer vision. It encompasses various factors to be considered like variability in hand shape, motion profile, and position of the hand, face, and body parts contributing to each sign. Many models have been proposed by different researchers with significant improvement by deep learning approaches in recent years [20]. The algorithms attempt to segment and detect hand features such as skin colour, appearance, motion, skeleton, depth, 3D model, deep learn detection and more.

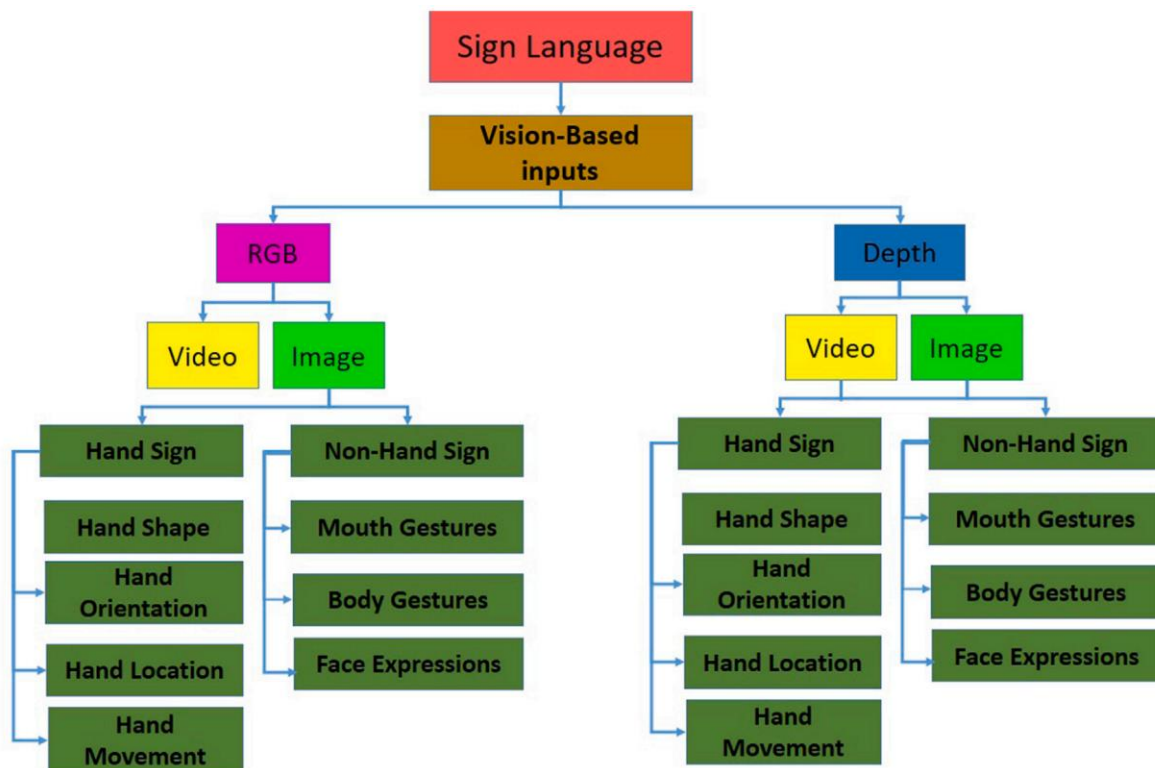


Figure 2: Vision based sign language models classifications [20]

The previous studies done in the field give insight into some gesture recognition systems under various scenarios, and address issues such as scene background limitations, illumination conditions, algorithm accuracy for feature extraction, dataset type, classification algorithm used and application [12].

There have been notable advancements and diverse options in systems used to manage and interact with the wide range of smart home devices available today. These systems span from traditional physical interfaces to advanced voice control systems.

Traditional physical controllers, like remote controls and wall-mounted panels, have been fundamental in smart home technology, providing tactile feedback for managing devices such as smart TVs, thermostats, and lighting systems. However, they have limitations like limited range and the need for a clear line of sight to the signal receiver.

With the emergence of technologies like Wi-Fi, microcontrollers, and short-range options such as Bluetooth, along with the proliferation of smartphones equipped with various sensors and technologies, traditional controllers have been rapidly replaced. A single smartphone app, leveraging APIs and networking protocols, can now effectively control every smart device in proximity.

1.2.2 In Real World Products

Nintendo Wii Motion Controlled Gaming

The Nintendo Wii was released in November 2006 which marked the beginning of its innovative motion-controlled gaming era. The Nintendo Wii revolutionized the gaming industry with its innovative motion-controlled gaming experience. By introducing the Wii Remote controller, equipped with motion-sensing capabilities, Nintendo transformed traditional gaming interactions.

Players could physically engage with games by swinging, pointing, and gesturing with the Wii Remote, immersing themselves in a more intuitive and interactive gaming experience. This motion control technology enabled a wide range of gameplay mechanics, from swinging a virtual tennis racket to steering a virtual racing car, all mimicking real-world movements.

This feature utilised the sensor-based approach for capturing gesture data. It used a combination of various sensors on the Wii remote like the remote's accelerometer sensor to detect hand movements in three dimensions (up-down, left-right, forward-backward) and the remote's gyroscope which measured the rate of rotation or angular velocity, allowing for more precise tracking of movements.

The Wii's motion control implementation democratized gaming, appealing to a broader audience beyond traditional gamers and bridging the gap between physical activity and entertainment. With its accessible and engaging gameplay, the Nintendo Wii redefined the

gaming landscape and inspired future innovations in motion control across various industries beyond gaming.

Xbox Kinect Motion Controlled Gaming [17]

Introduced in November 2010, Microsoft launched their latest addition to their Xbox line of gaming consoles with the introduction of Xbox Kinect. With the introduction of the Kinect, the users were given the flexibility of playing video games on their Xbox without the use of any controllers.

The Kinect implementation of gesture-based gaming combined both the sensor-based capturing of gesture data along with the vision-based capturing of gesture data to implement hands-free controller-free gaming experience.

The Kinect sensor bar contains an infrared (IR) depth sensor, which emits infrared light and captures the reflections from objects in its field of view. By measuring the time, it takes for the light to travel to and from objects, the sensor creates a depth map of the scene, allowing it to detect the distance of objects from the sensor.

In addition to the depth sensor, the Kinect included an RGB camera that captures colour images of the player's movements and surroundings. This camera provides visual context and enables features such as facial recognition and augmented reality experiences.

Double Tap in Apple Watch Series 9 and Apple Watch Ultra 2 [8]

In September 2023, Apple announced the latest iteration in their Apple Watch and the more premium Apple Watch Ultra lineage of products. It was the announcement of the new Apple Watch Series 9 and the new Apple Watch Ultra 2. While these products brought many of the best features of their previous iterations and augmented them along with some new features and performance improvements, these smartwatches had one new feature that wowed the industry.

Apple Watch Series 9 and Apple Watch Ultra 2 had gesture-based control feature called *Double Tap* embedded into these smartwatches. With this feature, the users were allowed to tap their index finger and thumb together twice, to control the device. It also allowed the users to also scroll through widgets, much like turning the digital crown on the side of the watch's dial.

This feature made use of the sensor-based approach for capturing user's gesture data. Double Tap works in combination with the latest Apple Watch accelerometer, gyroscope and optical heart rate sensor, which looks for disruptions in the blood flow when the fingers are pressed together. That data is then processed using a new machine learning algorithm that is pretrained by the engineers at Apple and runs on a faster neural engine, specialized hardware that handles AI and machine learning tasks. When the Apple Watch's display is turned on, the device automatically knows to respond when it senses the fingers are touched together. It essentially

works as a “yes” or “accept” button; that means if a call comes through, you can Double Tap to accept it (covering the watch with your full hand, however, will silence it quickly). If a song is playing, you can pause it by double tapping, and then again to start it.

Apple told CNN that adding Double Tap is more about simplifying the Apple Watch experience. This was one of the most recent applications of the gesture-based approach for controlling smart devices that this project intends to solve for controlling smart home devices.

Air Gesture Control in Redmi 12X 5G launched in India

In the start of April 2024, Redmi, during a livestream event in India, launched their latest flagship smartphone in their Redmi 12 series. It was the Redmi 12X 5G with various flagship level features starting for a very affordable price of 10,999 Indian Rupees which roughly translates to around 125 Euros. The smartphone had a plethora of new and improved features from its predecessor. But one of Redmi’s newest additions to this feature list was the introduction of Air Gesture Control to the smartphone.

Air Gesture Control was introduced to Redmi 12X 5G as a part of their goal of introducing hands-free operation. As a part of this feature, the users could utilise the smartphones camera to provide gestures to the smartphone as input and the gestures can then be recognised by the in-device algorithm to perform tasks in the smart phone ranging from navigation, swiping pages, taking screenshots, turning the screen on or off and confirming an action to perform on the screen.

This feature by Redmi utilised the vision-based approach of capturing the gesture data from around 20-40 cm from the 8 MP front facing camera of the smartphone and utilising the pretrained algorithm housed within the device to recognise the gesture and perform the respective command.

This feature implementation supported the ability to have different gestures for different apps that supported it but there is still a catch with this implementation. The system still uses a list of pretrained gestures and there is no flexibility with the gesture list. While this feature implementation was the closest to this project’s goal, there are still a few areas that this project improves upon these current implementations making it a significant stride in this field.

1.3 Project Objectives

1.3.1 Overall Objective

The objective of this project in a broad high-level perspective is to develop and implement a robust device control system that can be controlled via the user's recognised gestures. This system aims to address the need for enhanced accessibility in smart home environments, particularly for individuals with speech or hearing impairments. By harnessing the power of gesture recognition technology, the project seeks to empower users to control smart devices seamlessly and intuitively through hand gestures, thereby circumventing traditional speech-based interfaces and expanding accessibility options of the smart device network.

1.3.2 Specific Objectives

The central aim of the project, revolving around the development of a gesture-based device control system, can be delineated into specific micro-goals. These micro-goals, once accomplished, collectively contribute to the realization of the project's overarching objective. Throughout the project timeline, particular emphasis was placed on adhering to and achieving these micro-goals during the development phase. Every design decision, architectural layout, and implementation procedure employed in the project was meticulously crafted to push the project towards the completion of these goals. Thus, these specific objectives serve as guiding principles that steer the project's trajectory towards successful completion and the fulfilment of its ultimate purpose.

The primarily objective of this study is to design and deploy a reliable and efficient real-time gesture recognition algorithm capable of accurately identifying and interpreting a set of hand gestures. The system should also offer users the flexibility to add custom gestures to the gesture-recognition system, rather than confining them to a predefined list. Moreover, it should have the capability to store various gestures and their corresponding interpretations for different situations. This means that a single gesture could trigger different commands depending on the context. For example, in one scenario, a gesture might signify one command, while in another scenario, the same gesture could be mapped to a different command.

The recognized gestures should be linked to specific commands or actions related to smart home devices, such as toggling lights, adjusting thermostat settings, or controlling multimedia devices, based on user preferences. Additionally, the system should offer the flexibility to map gestures to different commands for various scenarios. This means that a gesture may trigger one command in a particular scenario but could activate a different command in another scenario, allowing for contextual adaptability and personalized control.

In addition to the technical development aspects, the project also encompasses broader objectives related to usability, user experience, and ethical considerations. User testing and feedback will be conducted to assess the system's usability and user satisfaction, ensuring that the interface is intuitive and easy to use for individuals of varying abilities. Furthermore, ethical considerations, including data privacy, consent, and inclusivity, will be integrated into the project's design and implementation process to ensure the responsible and equitable deployment of the technology.

Overall, the project aims to deliver a comprehensive gesture-recognition-based IoT device control system that not only meets the functional requirements of accessibility and usability but also upholds ethical principles and promotes inclusivity. By achieving these objectives, the project endeavours to make significant strides towards enhancing the accessibility and usability of smart home technology for individuals with speech or hearing impairments, ultimately improving their quality of life and independence.

1.4 Document Overview

This paper intends to summarise the whole research and development process that was done to contribute to the area of concern. The project includes several important aspects which are broken down into various chapters within the paper for a more structured knowledge transfer.

This document starts with the Chapter 1 which provides a complete overview to the project by giving the introduction to the topics dealt within the project. The chapter dives into the problem statement by providing a clear explanation to the problem at hand and the motivation behind the choice of work. It then sheds some light into the current state of the technology in the field, encapsulating all the research that has been done in the related fields. This includes how the technology is currently being used to handle the problem and highlighting some areas where it lacks and how this study intends to improve upon the current methods. This section then sets some clear objectives that are followed throughout the development phase of the project and a brief yet insightful analysis of those objectives.

The paper continues with Chapter 2 where it includes the comprehensive analysis of the design of the project. It first sets stage by explaining the strategy and all the necessary outlines that were kept in mind while making the design choices for the project. The section also gives a brief on how the project goes about implementing those strategies. The chapter also goes on to elaborate on the development phases that the project went through during its timeline and provide a brief analysis on how they contributed towards the achievements of the goals set by the project while following the strategies set during the initial stages.

Following chapter 2, Chapter 3 provides deeper insights into the project by giving a full disclosure about the architecture of the project. This is where the document first provides a high-level overview of the architecture of the whole project. It then proceeds to provide a detailed explanation behind each of these architectures and how they interact with the user to achieve the project goals. The chapter also provides a brief yet informative analysis of the

various limitations the chosen architecture has and how it can be improved in the future development of the project.

The document then proceeds onto Chapter 4 which gives the complete analysis of the implementation of the project on how the design choices and the architectural blueprint collaborate to result in the fruition of the objectives set by the project. It also provides information about the technology stack and various algorithms that were utilised in the project along with the various tools that were selected and the development environment under which the development was done. The chapter at the end provides a brief analysis of the various issues that the project faced during the implementation phase and the limitations that plague the current implementation.

Finally, in Chapter 5, the paper provides a conclusion, highlighting the key components of the project, how it is an improvement over the other work done in the field. It also provides a small peek into the future scope of the project. This includes the limitations faced by the project highlighting the key implementation issues that still need to be tackled. It also provides some improvement prospects for the project that can be worked upon to make it better.

Chapter 2

Design

This chapter deals with the design of the project where we dive into the planning and strategy for the design choices made for the gesture-recognition-based device control system along with the development phases the project went through. It first offers a comprehensive overview of the design principles that were followed throughout the development cycle and how each of them is implemented to contribute towards the project's success. It then provides the complete analysis of the design phases that the project went through to achieve its objectives. The aim of the section is to provide insight into the rationale behind key design decisions and comprehensive explanation of the design and development phases of the project as a whole and how they contributed towards the end goal.

2.1 Design Principles

Design principles form the very basis upon which a system is built, guiding the development process and shaping the system's functionality, usability, and effectiveness in an organised manner. There were a few fundamental design principles that were important to consider during the creation of a robust, intuitive, and inclusive system for controlling smart devices through gesture recognition. The implementation of these guidelines formed the backbone for the development process.

2.1.1 Easy to replicate

One of the most important aspects of a project is its ability to replicate the progress made previously so that most of the future efforts are focused on improving upon it. As the project starts to expand in size, it becomes very hard to keep track of every project dependency and every step taken to successfully run the project. So various documentations and automation scripts were actively maintained throughout development phase to make things easy. All these files are present in the root directory of the project codebase for easy access.

2.1.1.1 README.md

Having a proper knowledge transfer documentation for a project is a very important aspect for any project considering the number of moving parts every project has. Hence, the project

codebase contains a very comprehensive documentation file called '*README.md*' which makes use of the industry standard Markdown language to provide documentation for every part of the project for anyone analysing the project or trying to work on it.

2.1.1.2 requirements.txt

The codebase for Gesture-based device control has many moving parts and racks up a lot of dependencies pretty quickly. To keep track of all of them, Python enables the creation of a dependency list called '*requirements.txt*' which can be used to easily install all the dependencies for the project.

2.1.1.3 project_setup.py

The codebase requires a lot of dependencies and development tools to run and needs various steps to make the project ready to go. To enable this, a '*project_setup.py*' script has been actively maintained throughout the development of the project to automate the process of the first-time setup.

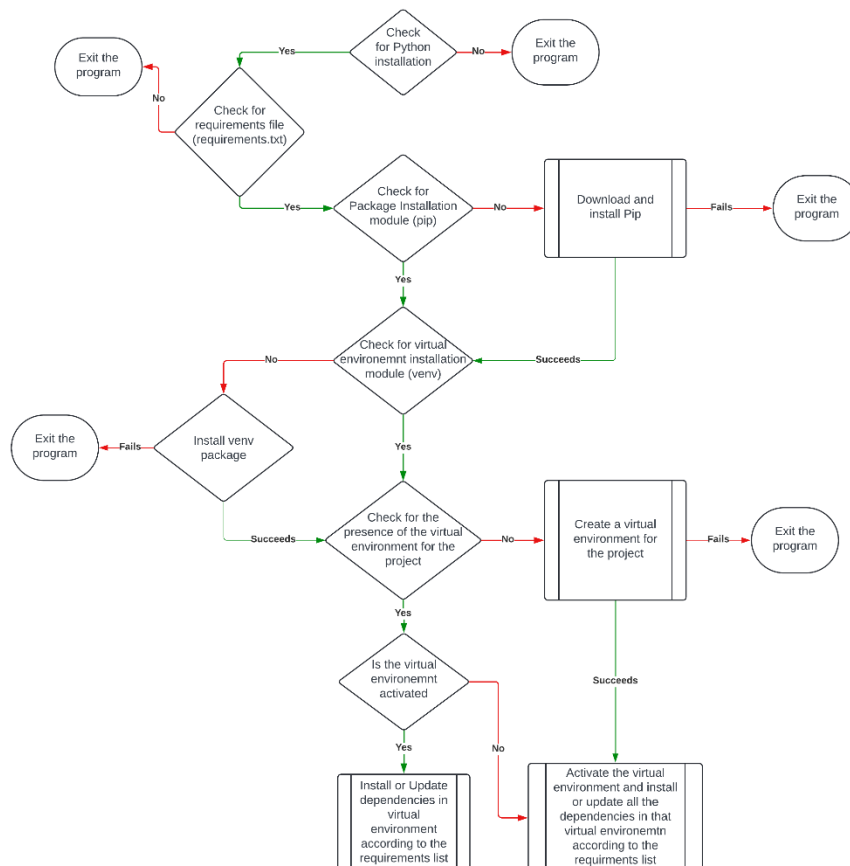


Figure 3: Automated project setup workflow

2.1.2 Organised and easy to debug

To continue the work on a project, it is very important that the project codebase is expressive and readable to enable easy debugging. Hence, since the beginning of the development, the structure of the system's codebase has been made as modular as possible with proper codebase structure.

To implement this, every file is organised into various folders according to their purpose. These files include scripts that perform various tasks, various scenarios' data (training data, gesture data and trained model) accumulated when a user creates one, the Tensorflow model training log files and the files required for the frontend version of the website.

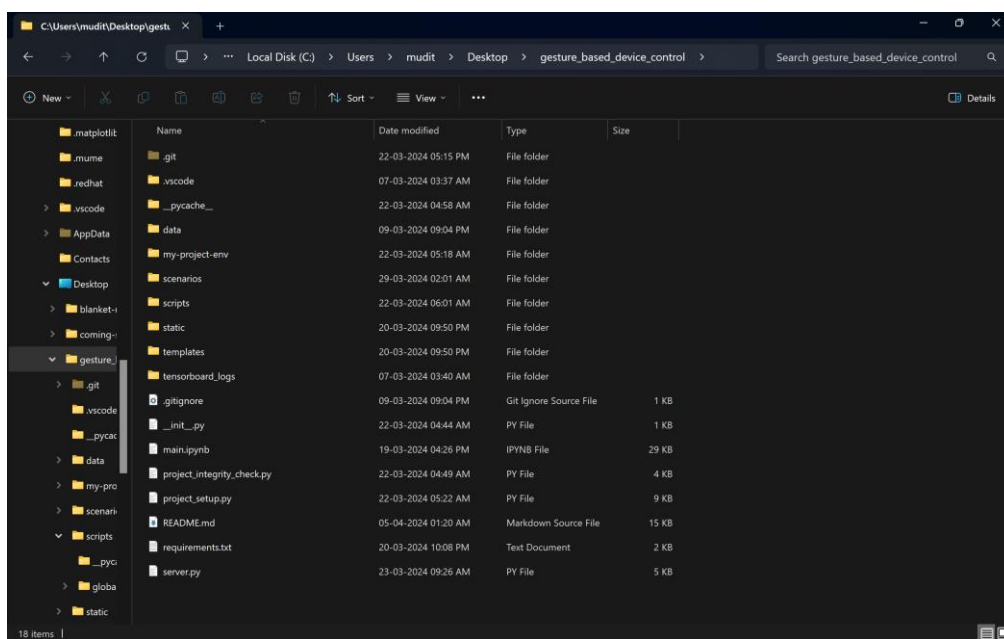


Figure 4: Project codebase folder structure

In addition to this, every task performed by the Gesture recognition-based device control system has been modularised into different functions and every category of functions has been assigned a separate Python script for easy pinpointing during future development and debugging while following proper naming conventions and expressive names.

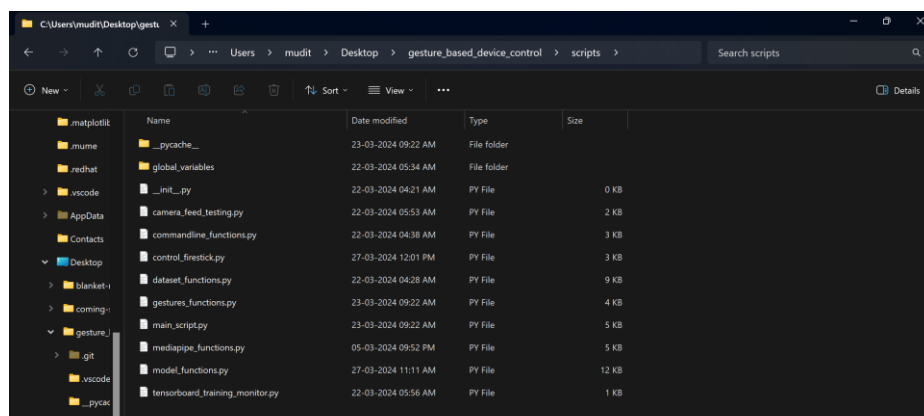
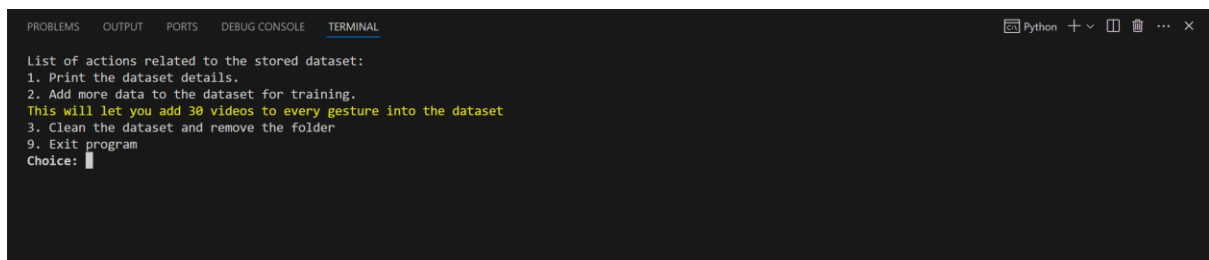


Figure 5: Scripts folder structure

2.1.3 User centric design

It is very crucial for a system like gesture-based device control to be very streamlined and easy to use as possible. This means that the system should be easy to use. To implement this, the system has various endpoints that the users can access to perform various tasks. This includes tasks like project setup, scenario creation, training data management, global variables information, camera feed testing, etc. These endpoints can be easily accessed by running their respective Python scripts if the user is using the command line interface (CLI) of the system or by clicking on the respective buttons in the frontend version of the website.



```
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
Python + - [] ... X

List of actions related to the stored dataset:
1. Print the dataset details.
2. Add more data to the dataset for training.
   This will let you add 30 videos to every gesture into the dataset
3. Clean the dataset and remove the folder
9. Exit program
Choice: 
```

Figure 6: CLI of dataset functions of the project

2.1.4 Design simplicity and consistency

Simplicity and consistency in project design minimizes load, reduces user confusion, and enhances usability. Simple, familiar and intuitive interfaces in both versions of the system makes the project easier to understand and navigate, leading to a more positive user experience. Maintaining consistency in project elements, such as layout, colour scheme, and interaction messages, provides a predictable user experience across the system. This fosters familiarity and reduces user friction.



Figure 7: Frontend version of the project

2.2 Design Phases

The project of gesture-based device control system involved a lot of moving parts and their integration. It was very crucial to adopt a systematic approach to handle the development of the system to ensure that no leaves are left unturned. To achieve this, the whole development journey of the project was divided into several phases.

2.2.1 Initial Development

In the initial stages of development, a systematic approach was adopted for achieving the project objectives. The initial approach focused on leveraging Jupyter Notebook and its feature-rich development environment to continuously add, refine and enhance the system's functionality, guided by the project objectives and outcomes of every iteration.

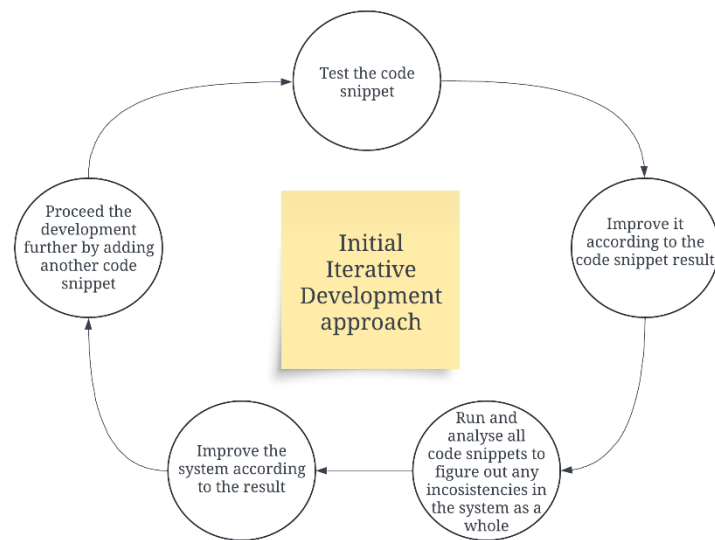


Figure 8: Initial Development Phase Approach

This iterative approach starts with the addition of a code snippet into a new Jupyter cell according to the functionality needed by the project objective. The Jupyter cell is then run to figure out any errors or bugs introduced by the cell and then the code snippet is improved according to it. Once everything works as expected, the whole system is run to figure out if the new added code snippet performs well with the system and improves the functionality of the system as it should. After carefully analysing the whole system with the new cell, the whole system is improved from top to bottom accordingly, removing any bugs or inconsistencies. Once it is ensured that the introduction of the new code works as expected and supports the system, the iteration is completed and the process is repeated for a new code snippet and a new functionality.

This approach of development systematically developed the system while following the project objectives. It broke down the whole initial project development phase into smaller manageable iterations and achievable targets. It also replicated the popular Continuous Integration and Continuous Deployment (CI/CD) strategy of software development and helped to tackle any challenges that were faced in the whole journey. This also allowed for continuous improvement and refinement of the system over time by building upon the previous iterations and enhancing functionality.

2.2.2 Project Modularisation

After the completion of system designing using the iterative approach, a strategic shift was adopted towards code modularisation and function-centric design. The whole monolithic codebase structure of the system written in Jupyter Notebook was then modularised into separate scripts according to their purpose, with each script encapsulating a specific component or functionality of the system. Within each script, the code was further modularized into functions, each tasked with performing a specific operation or task.

This strategic shift focusing on modularity and function centric approach yielded several benefits for the project codebase. Firstly, it significantly improved code maintainability and readability by implementing code reusability, abstraction and structure organization, making it easier to understand, modify, and extend the system as needed. This refactoring of the project structure finalised the incorporation of the design principles that were initially planned and followed partially in the initial phase. Secondly, the modular structure enabled cross-script functionality and communication, allowing for seamless integration and interaction between different components of the system while also removing code repetition and redundancy. Lastly, by exposing certain functionalities as endpoints, the system became more accessible and user-friendly, empowering users to interact with the system in a more intuitive and versatile manner. This phase improved the project significant and indirectly helped in the planning and execution of the frontend interface of the system built using Flask, enabling users to interact with the system programmatically or interactively.

2.2.3 Frontend version development

After proper testing and analysis of the project codebase obtained by the previous modularisation phase, major efforts were employed in developing the frontend interface of the system. The frontend interface was built with a client-server architecture where the server was built using Flask. The backend server was programmed to access the system endpoints for performing various tasks such as accessing the global variables, running project setup and project integrity check, creating a scenario, etc. that were exposed by various scripts built in the previous phase. This was done by using various routes of the website.

Chapter 3

Architecture

The architecture of the gesture-based device control system serves as the blueprint for its design and functionality. This chapter deals with the detailed exploration of the system's architecture, covering the main components and their interactions. The chapter contains a comprehensive overview of the architectures in the project with each aspect of the architecture meticulously examined to provide a comprehensive understanding of the system's underlying structure and functionality to realizing the envisioned solution. The chapter also has a brief analysis of the limitations these architectures and their components, shedding some light on where the system lacks and needs further resources and development.

3.1 High Level Overview

The architecture forms the backbone of the gesture-based device control system which outlines the overall structure and organization of components. This implementation of the system is based on a very simple logic, for every smart device that a user wants to communicate with, there exists different types of gestures which map to different types of commands for the smart device. For the purpose of this project, these are known as *scenarios*. Within the framework of the gesture-based device control system, the concept of *scenarios* emerges as a fundamental component, encapsulating a predefined set of gestures set by the user during the creation of the *scenario* and their corresponding commands, also assigned by the user during the *scenario* creation stage, tailored for specific smart devices. Each *scenario* represents a distinct use case or interaction situation, where users can employ gestures to control various smart devices seamlessly. This section elucidates the essential components and functionalities comprising each scenario, underscoring their significance in enabling intuitive and efficient device control.

In the system, a user can create different *scenarios* in which they select a smart device from the list of supported devices in the system. The user can then input the data for that *scenario* which includes the smart device from the list of supported smart devices and the gesture data for that *scenario*. Then the user proceeds to collect the training data for the LSTM Neural Network (LNN) model. Once the training data is collected for recognising every gesture in the scenario and the data is filtered, processed and prepared, the data is then fed to the LNN to train and the trained model is then stored in the same *scenario* folder.

In the system, a user can use the existing *scenarios* that were created previously to actually communicate with the smart home devices. Once a user selects a scenario, the user is prompted with the appropriate steps to connect with the smart device, depending upon the *scenario* which is stored in the *scenario* folder that was created previously for that *scenario*. Once the

connection is established, the user can then use the stored trained LNN model to recognise the gestures from the hand movements captured and processed by using OpenCV camera feed and Mediapipe library. The recognised gesture is then used to find its corresponding command which is communicated to the smart device. This process of gesture recognition and command communication repeats until the user prompts the system to stop, by closing the camera feed window which in turn also severs the connection with the smart device.

3.2 Model Preparation

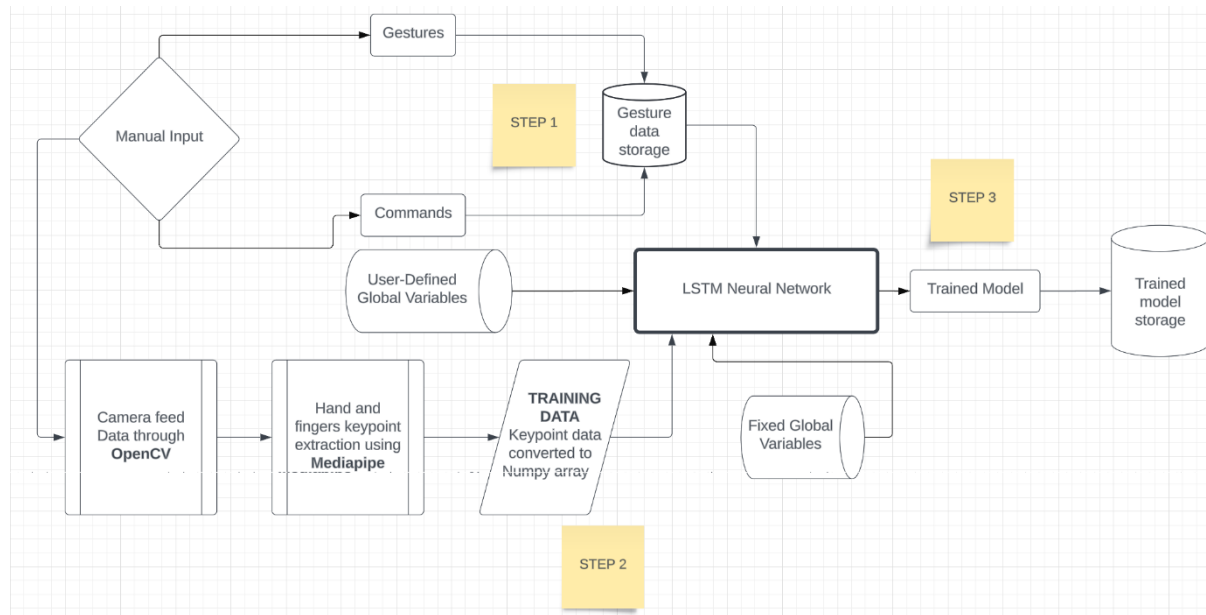


Figure 9: Architecture for the case of new 'scenario' creation

When a user requests the system to create a new *scenario*, a new folder for that scenario is created. The user is first prompted to choose a smart device for which the scenario is. After this, the user manually inputs the gestures that the system is supposed to recognise for that *scenario*. For every gesture, the user is also prompted to choose the corresponding command which will be communicated if that particular gesture is recognised. This 'gesture' data is stored in that *scenario*'s folder as a NumPy array.

The user is then prompted to provide the training data for the entered gesture list. This training data comprises of a series of videos captured by the user's camera feed. For every frame in every video for every gesture, the frame first goes through preprocessing using Google's Mediapipe library to recognise the hand and finger key-points and the key-points are displayed in the camera feed that the user can see on the screen while recording the gestures. Once a video is captured, the key-points generated for every frame in the video are compiled and stored as a NumPy array which, in the future, shall denote the training data for the LNN. This process is repeated for every gesture and the subsequent compiled NumPy array generated is stored as training data in the *scenario* folder. By default, the system records 30 videos for a single gesture, each video having 30 frames.

After this, the system provides the training data as the parameters for the LNN model and the gesture data as the expected output. Both these data along with the LNN architecture come together to train a LNN model for future recognition. Once, the LNN model is trained, the trained model is stored in the *scenario* folder for future use.

As a part of the system's security and privacy considerations, at the end, the system asks the user whether they want to retain the training data or purge it and their choice reflects whether the system stores the training data or just the trained model.

3.3 Gesture-based device control

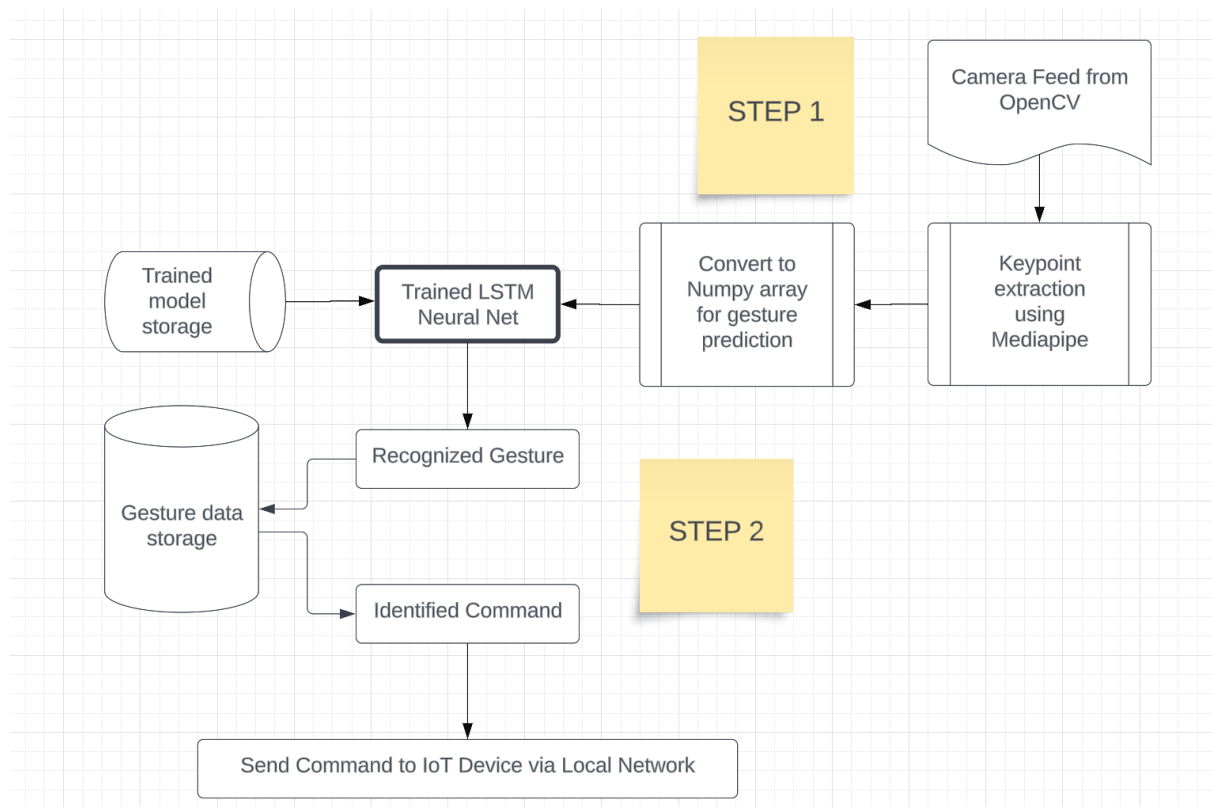


Figure 10: Architecture for the case of smart device control

Once the data for a *scenario* is ready, a user can always use the gesture-based device control system and the *scenario*'s trained LNN model to use their gestures to communicate with their smart device.

The process starts with the user selecting the *scenario*. Once the *scenario* is selected, the user is prompted with appropriate steps to connect to the smart device whose selection was made by the user during the creation of the *scenario*. The appropriate steps to establish connection with a smart device can vary according to the device and their manufacturer's specifications and steps to connect. This can range from scanning the Local Area Network (LAN) to look for the device and allowing the user to make the connection or maybe providing the option to enter the IP address of the smart device to establish the connection. To improve the data security of

the system, the connection details are not saved into the *scenario* folder as this data is very sensitive and if exposed, could be misused.

Once the connection is established successfully, the system opens up the camera feed from the user's device whose every frame is first pre-processed using Google's Mediapipe library to recognise the hand key-points and display them to the user. In the camera feed, the user can perform their desired action whose key-point data generated by Mediapipe is then provided to the trained LNN for recognition. Once, the gesture index is identified, the gesture corresponding to the gesture index in the gesture data is recognised and displayed to the user within the camera feed itself. Meanwhile, the gesture identified is also used to find the corresponding command for the *scenario*'s command and the command is now finally communicated with the smart device via the local area network.

3.4 Architecture Limitations

While the architecture of the gesture-based device control system was very carefully planned while keeping all the project objectives in mind, it was still having some flaws that required attention and resources during the development. In this section, various inherent constraints and challenges encountered in the architectural structure of the gesture-based device control system are discussed in detail, providing insights into areas that require further refinement and optimization. While some of them were tackled during the development phase, there were some that were not tackled in this implementation of the system due to various time and resource constraints. This section intends to shed some light and discussion of these challenges so that they can be tackled in the future iterations.

3.4.1 Lot of intertwined intricacies for project setup and maintenance

The project entailed a considerable amount of setup, encompassing various steps such as tool installation, dependency management, and project configuration, which collectively presented a formidable challenge in terms of project management and accessibility for the new users. The complexity of these setup procedures not only made the project difficult to manage but also posted significant challenges to entry for individuals unfamiliar with the intricacies of the system. Recognizing the importance of facilitating a seamless transition for new users, it became very important to address this aspect of the project to enhance usability and accessibility, which was a very important design strategy that was set during the start of the project.

To tackle this daunting task of project setup, an automation script was meticulously developed and maintained to streamline the entire setup process, removing the burden of manual intervention from the user and ensuring a hassle-free experience. This automation script meticulously carried out each step of the setup journey, from verifying the readiness of prerequisite tools to automatically installing missing dependencies and configuring the project

environment. Throughout the setup process, informative messages were displayed to guide users through each task, providing clarity and transparency every step of the way.

Furthermore, to bolster project manageability and ensure ongoing operational integrity, an additional automation script was devised that could be used to conduct regular project integrity checks. This script would step by step check for the accessibility of all requisite tools and dependencies, ensuring that they remained readily available and functional for the smooth execution of project scripts. By automating the process of project setup and integrity maintenance, the project was not only made more accessible to new users but also enhanced its overall manageability and reliability, fostering a conducive environment for collaborative development and seamless project execution.

Chapter 4

Implementation

In the implementation section, we get into the practical realization of the gesture-recognition-based device control system, providing detailed insights into the development process, technologies used, and methodologies employed. This chapter provides a deeper understanding of the technical intricacies involved in bringing the system to fruition, along with a very comprehensive analysis of how they perform as well as the practical considerations that shaped its development. The chapter also highlights the key implementation limitations that this study faces. It also provides a brief analysis of the challenges encountered during the implementation, and solutions devised. Through a detailed exploration of the implementation journey, it seeks to showcase the iterative nature of software development, from initial prototyping to final deployment, and the iterative and iterative nature of software development. We aim to provide insights into the strategies employed to overcome challenges, optimize performance, and ensure the reliability and scalability of the gesture-recognition-based IoT device control system.

4.1 Development Environment

4.1.1 Python

Python serves as the primary programming language for the development of the gesture-recognition-based device control system. Python is a high level, interpreted language used for general purpose programming due to its highly flexible nature and a very readable and expressive syntax. Python is an open-source language with an extensive ecosystem of packages. As of May 2019, the ecosystem, known as PyPI contained over 178,000 packages and over 1.7 million individual package releases released by over 76,000 unique authors [5].

Statistic	Value
Number of packages	178,952
Number of releases	1,745,744
Number of package classifications	947,896
Number of authors	76,997
Number of maintainers	3,047
Number of licenses (raw)	4,610
Number of imports	156,816,750

Figure 11: Summary of PyPI ecosystem statistics (May 2019) [5]

Year	New Packages	Active Packages	New Releases	Authors
2005	96	96	389	68
2006	367	420	2,324	216
2007	876	1,047	5,301	341
2008	1,702	2,223	10,923	637
2009	2,559	3,727	17,426	974
2010	3,522	5,454	23,474	1,455
2011	4,840	7,479	31,107	1,996
2012	7,235	11,047	47,377	3,062
2013	10,438	16,267	74,003	4,633
2014	13,352	21,555	112,994	6,016
2015	17,355	28,498	173,437	7,742
2016	21,849	36,557	253,262	10,293
2017	29,905	48,223	372,034	12,374
2018	39,351	64,628	502,029	16,064
2019	16,873	18,919	199,664	6,938

Figure 12: Number of new packages, active packages, new releases, and new authors on PyPI by year; 2019 is a partial year [5]

This huge ever-expanding library of packages that Python brings to the table, enables the system to be compatible to use various packages and libraries like OpenCV and Tensorflow. Python in the project is used to perform tasks pertaining to various actions in the system like automating tasks, training data collection, neural network model preparation and communication with smart devices. Python's popular web development framework called Flask is also used in the project to build the backend server for the web version of the project.

```

server.py M X
server.py > refreshGlobalVars
29 def refreshGlobalVars():
30     return data
31
32
33
34 @app.route('/run_project_setup/')
35 def run_project_setup():
36     process = subprocess.Popen([python_command, 'project_setup.py'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
37     output, error = process.communicate()
38     if error:
39         return f"An error occurred: {error.decode()}"
40     else:
41         return '<br>'.join(output.decode().splitlines())
42
43 @app.route('/run_integrity_check/')
44 def run_integrity_check():
45     process = subprocess.Popen([python_command, 'project_integrity_check.py'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
46     output, error = process.communicate()
47     if error:
48         return f"An error occurred: {error.decode()}"
49     else:
50         return '<br>'.join(output.decode().splitlines())
51
52 @app.route('/run_camera_check/')
53 def run_camera_check():
54     from scripts.commandline_functions import success, failure
55     from scripts.camera_feed_testing import main
56     ret_code = main()
57     if ret_code == 0:
58         return "Camera feed accessibility: " + success()
59     else:
60         return "Camera feed accessibility: " + failure()
61
62 @app.route('/create_new_scenario/', methods=['POST'])
63 def new_scenario():
64     from scripts.main_script import scenario_folder_existence, set_new_scenario
65     from scripts.gestures_functions import save_gestures_to_file
66     scenario_name = request.form['scenario_name']

```

Figure 13: A brief code snippet of the Flask server made using Python

4.1.2 Jupyter Notebook

Jupyter Notebook served as a very invaluable tool for the initial phase of development. It has an interactive development, experimentation, and documentation friendly approach towards Python programming. It has support for various developer friendly features like separate code snippet execution along with Markdown support for documentation which enabled the initial iterative approach of development very seamless by allowing assistance in data monitoring, errors pinpointing, and much more.

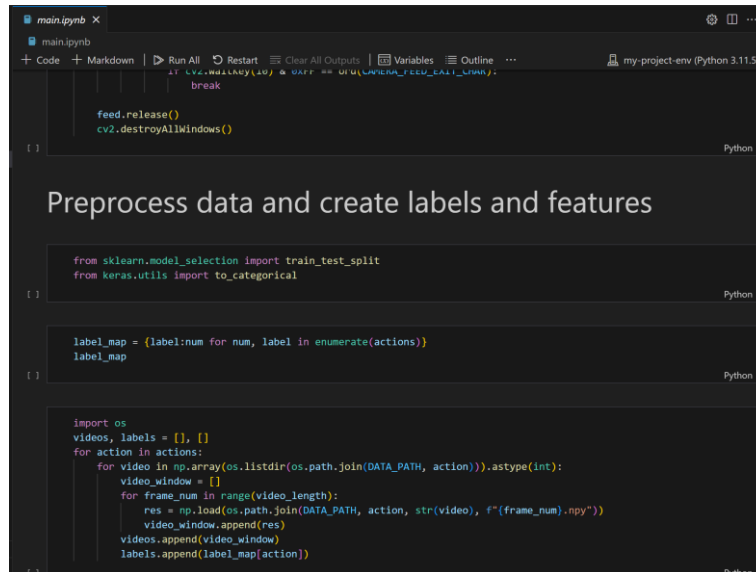


Figure 14: Jupyter Notebook snippet from the project

4.2 Technology Stack

The choice of various technologies and frameworks used during the development of a project have a direct impact on how well the project performs. For the purpose of gesture-based device control, a wide range of frameworks, and alternatives were considered during the project's lifecycle. Shedding light on the rationale behind each choice and its implications for the system's functionality, performance, and scalability, provides insights into the thought process in realizing the envisioned solution and the trade-offs inherent in decision-making.

4.2.1 OpenCV

OpenCV, which stands for Open-Source Computer Vision, is an open-source library designed for real-time computer vision and image processing tasks. It offers a rich set of functions and algorithms for tasks such as image/video capture, feature detection, object tracking, and pattern recognition while also utilising various hardware capabilities of the user's device to provide great performance.

In 2010 a new module that provides GPU acceleration was added to OpenCV. It was implemented using CUDA and therefore benefits from the CUDA ecosystem, including libraries such as NVIDIA Performance Primitives (NPP). The GPU module allows users to benefit from GPU acceleration without requiring training in GPU programming. The below figure is a benchmark demonstrating the advantage of the GPU module. The speedup is measured against the baseline of a heavily optimized CPU implementation of OpenCV [6].

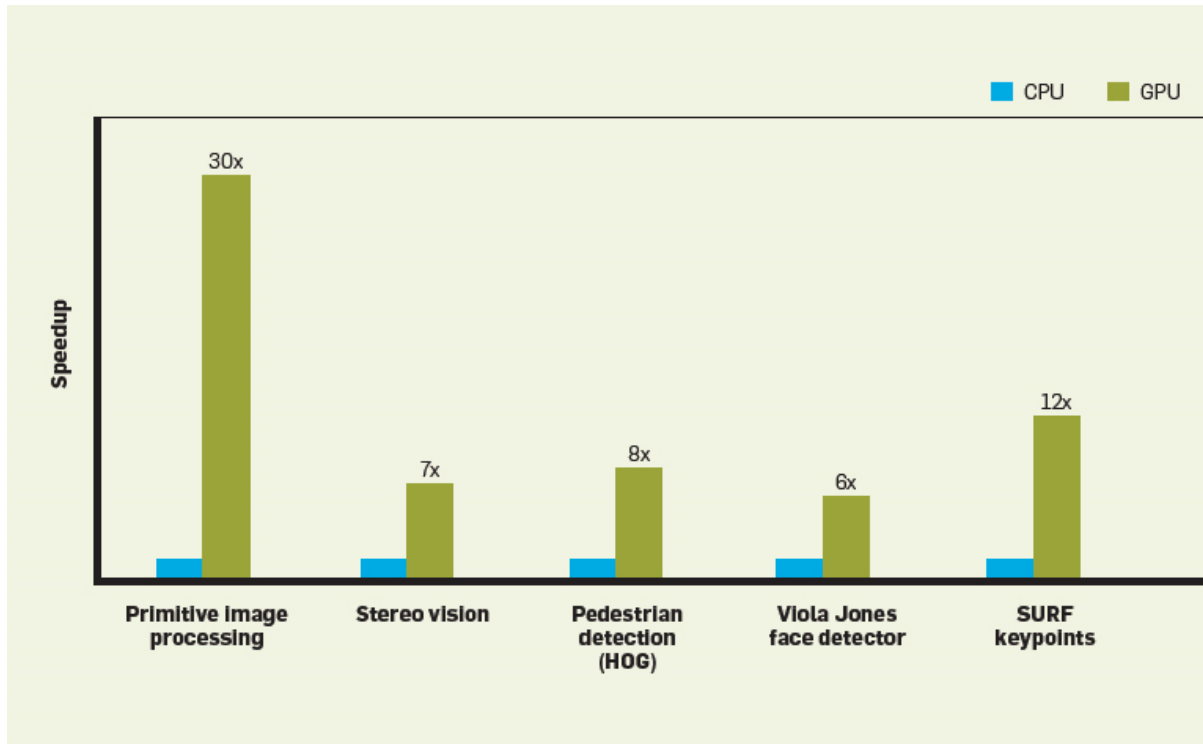


Figure 15: CPU versus GPU performance comparison [6]

OpenCV's versatility and performance make it an industrial standard for developing applications in various domains. For the purpose of this project, OpenCV is used to capture the video feed from the user's device's camera. This camera feed is either used for capturing the training data for gestures or for capturing the gestures for recognition and subsequent smart device communication depending upon the user's request from the system.

4.2.2 Mediapipe

Google Mediapipe is a powerful framework that facilitates the development of machine learning models for various multimedia processing tasks, including hand tracking, pose estimation, and facial recognition. In my project, I leveraged Mediapipe's capabilities for hand tracking, allowing the system to accurately detect and track the user's hand movements in real-time. By incorporating Mediapipe into the project, I was able to implement a highly accurate hand detection pipeline functionality with a relatively easy to read code with the compromise of any of the features. Usage of Mediapipe promoted the quick acceleration of the development

of the system and direct focus on other important aspects of the project. Mediapipe's efficient implementation and ease of integration proved instrumental in enhancing the system's accuracy and responsiveness, contributing to a seamless user experience.

Compared to other similar technologies in the market, Google Mediapipe stands out for its versatility, performance, and ease of use. Unlike some proprietary solutions that may require extensive training data and complex setup procedures, Mediapipe offers pre-trained models and ready-to-use pipelines, simplifying the development process for developers. Additionally, Mediapipe's open-source nature and extensive documentation make it accessible to a wide range of developers, fostering innovation and collaboration within the community. While other frameworks may excel in specific tasks or offer more advanced features, Mediapipe's balance of functionality, performance, and accessibility makes it a compelling choice for developers seeking to integrate machine learning into multimedia applications.

4.2.3 Numpy

NumPy is a fundamental library in Python that provides support for large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays. In my project, NumPy played a crucial role in handling and manipulating the data generated from various sources, including video and subsequent frame image processing. It was also responsible for compiling the whole data and making it ready for the LNN model for training and testing purposes.

By utilizing NumPy's array operations and mathematical functions, I was able to efficiently process and analyse the data, enabling tasks such as feature extraction, data normalization, and model training. Additionally, NumPy's performance-optimized routines contributed to the overall efficiency of the project, ensuring fast and reliable computation even with large datasets.

Compared to other similar libraries, NumPy stands out for its extensive functionality, performance, and widespread adoption within the Python ecosystem. While alternative libraries may offer specialized functionalities or optimizations for specific tasks, NumPy's versatility and comprehensive array operations make it a go-to-choice and an industrial standard for scientific computing and data analysis. Furthermore, NumPy's integration with other Python libraries essential to this project, such as keras and scikit-learn, enhances its utility for a wide range of applications, from numerical simulations to machine learning. Overall, NumPy's rich feature set, performance optimizations, and seamless integration make it an indispensable tool for data manipulation and analysis in my project and beyond.

4.2.4 Tensorflow

TensorFlow is a popular open-source machine learning framework developed by Google that offers a comprehensive ecosystem for building and deploying machine learning models. In my project, TensorFlow served as the backbone for implementing and training deep learning models, including the LNN for gesture recognition. The extensive range of tools and functionalities provided by TensorFlow facilitated the development of complex neural network architectures, enabling the system to learn and recognize intricate patterns in the input data. Moreover, TensorFlow's flexibility allowed for seamless integration with other Python libraries very crucial to this project, such as NumPy and scikit-learn, streamlining the entire machine learning pipeline from data preprocessing to model evaluation.

One notable feature of TensorFlow that was heavily utilized in my project is TensorFlow Monitor, a powerful tool for visualizing and monitoring the training process of machine learning models. By leveraging TensorFlow Monitor, I was able to track key metrics such as loss and accuracy in real-time during model training, gaining valuable insights into the model's performance and behaviour. The interactive visualization capabilities offered by TensorFlow Monitor facilitated rapid experimentation and debugging, enabling me to fine-tune model parameters and optimize performance effectively. Overall, TensorFlow's robustness, scalability, and rich set of tools make it a valuable asset for machine learning projects, empowering developers to build and deploy state-of-the-art models with ease.

4.3 Tool selection

4.3.1 Visual Studio Code

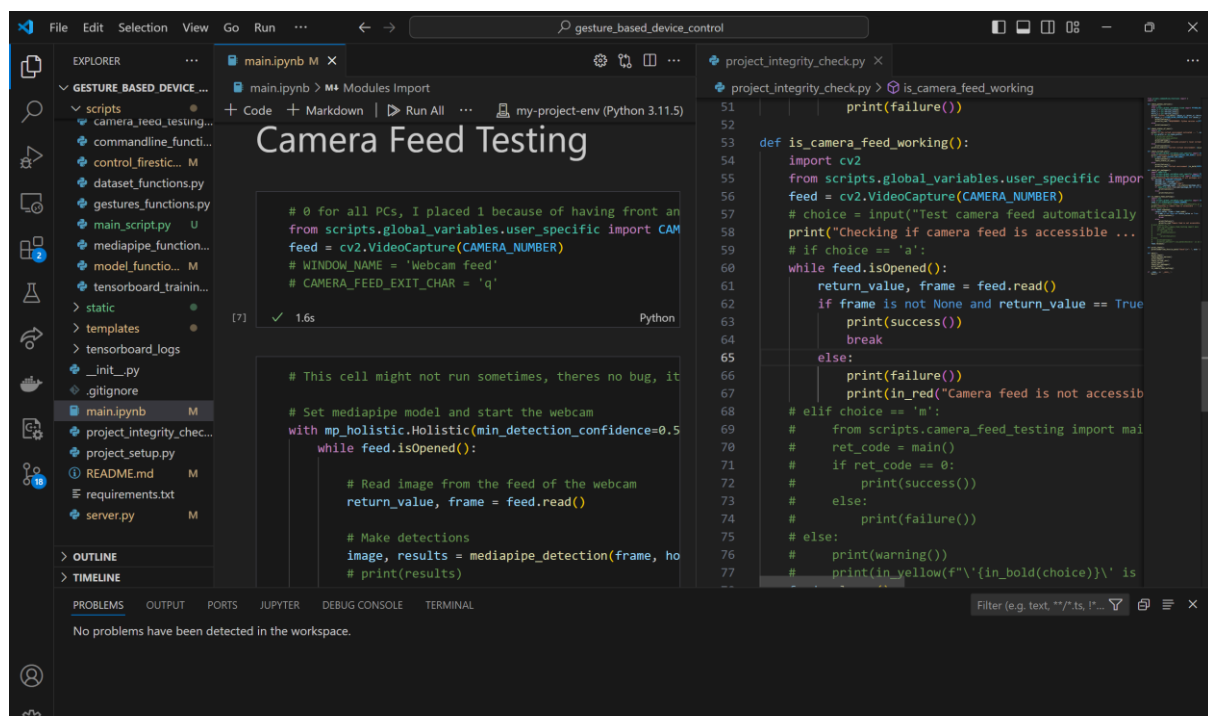


Figure 16: Instance of enhanced productivity using VSC IDE using the integrated CLI and Multi-file alignment features during Modularization phase

Visual Studio Code (popularly abbreviated as VSC) served as one of the most important cornerstones as a part of the project's development toolkit, offering a versatile and intuitive platform for code editing and project management. Throughout the development lifecycle, VSC was instrumental in facilitating various tasks, including coding, modularization, debugging, and version control. Its extensive library of extensions enabled seamless integration with essential tools and frameworks, such as Python, OpenCV, Jupyter, and TensorFlow, to name a few, enhancing the development experience and workflow efficiency.

While exploring alternatives such as PyCharm and IntelliJ IDEA, VSC ultimately emerged as the preferred IDE due to several key advantages. Firstly, its lightweight and fast performance ensured smooth operation even on less powerful hardware, making the development phase a very smooth experience. VSC has a very simple user interface without compromising on any crucial functionality. This enabled project development on VSC a smooth sailing journey with reduced to practically zero friction during development. VSC's robust Git integration provided comprehensive version control capabilities, enabling efficient versioning and code management of the project. Additionally, its extensive customization options and active community support further solidified its appeal, allowing developers to tailor their development environment to suit specific project requirements.

Overall, VSC stood out as the ideal choice for the project's development needs, offering a seamless blend of functionality, performance, and flexibility and did not let the focus shift from the project's main objectives.

4.3.2 Git

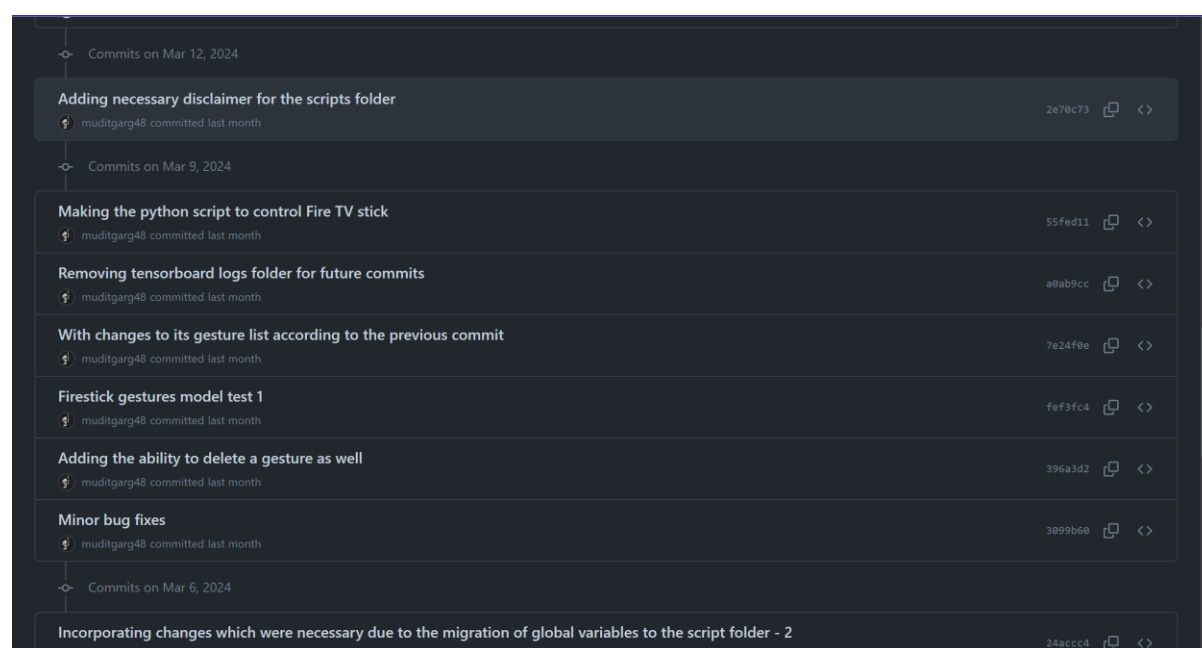


Figure 17: A portion of the Git commit history for the project

Git played a very pivotal role by facilitating versioning system for the project during its implementation phase in the timeline. Utilizing Git, the development was effectively managed with ease in keeping track of code changes, and also acting as a backup in case there is any problem with the project codebase due to breaking changes by allowing to revert back to previous codebase state. By leveraging Git's version control and rollback capabilities, the development of the project maintained a structured development workflow, enabling the implementation of new features and bug fixes in a systematic manner.

4.3.3 Flask

Flask served as the foundational web framework for developing the project's frontend interface and backend services. Leveraging Flask's lightweight and flexible architecture, the project was able to utilise the endpoints exposed by the codebase of the system and deploy web application with minimal overhead. Its intuitive routing system, template rendering capabilities, and built-in development server facilitated rapid prototyping and iterative development. Flask's simplicity and extensibility allowed for easy integration with the system scripts and other Python libraries and packages used within the system, enabling the implementation of custom features and functionalities tailored to the project's requirements.

Alternative web frameworks, such as Django, were considered during the selection process. However, Flask emerged as the preferred choice due to its minimalist design philosophy and emphasis on simplicity and flexibility. Unlike the more opinionated frameworks like Django, Flask provided greater freedom and control over the project's architecture and implementation details, making it well-suited for the project's specific needs. Flask, unlike its counterparts like Django, provides control over the development to a very core level, transferring the responsibility of adding new features and functionalities to the developer. This reduces the number of predefined functions that get packaged with the project and gives control of the size and scalability of the project in the hands of the developer.

4.4 Algorithm Implementation

This section provides a very detailed analysis of the gesture-based device control system and what process it follows from the initial creation of the *scenario* till the very end where the user can use the system and all the data collected to control a smart device via the local area network. This section first glances over the whole process in a brief overview to give an idea of what happens in the system under the hood. The section then gives the complete analysis of each of the component in the implementation and how they work together and pass information to one another to fulfil the project goals.

4.4.1 Implementation Overview

The gesture-based device control system starts with the camera feed captured from the user's device using OpenCV. Each frame from the OpenCV camera feed goes through key-point detection implementation by Mediapipe and the predictions made by Mediapipe are then augmented onto the same frame from OpenCV camera feed and displayed to the user for reference.

This whole process is utilised behind the scenes when the system prompts the user to provide training data for the gesture list of a *scenario*. During this collection, for every frame, the predictions made by Mediapipe are filtered to only obtain the left-hand key-point predictions and the right-hand key-point predictions. If any of the left hand or right hand is not detected, their predictions data is set as a bunch of 0 with their shape ensured to be consistent with others. These predictions are converted to a NumPy array and stored for future training purposes.

Once this process of key-point data collection is complete for the whole list of gestures for that *scenario*, the system then combines all these into one and prepares the dataset for the recognition model. This dataset is then split into a 95-5 ratio where the model is trained on 95% of the dataset created and the remaining 5% of the dataset is used to test the predictions of the model while training. This split and train method ensures that there is no chance of over-fitting of the data. Providing the model with test dataset outside the scope of the training dataset ensures that the model has the flexibility to give correct predictions for unknown or relatively new data.

Once the model is trained, the trained model is stored in the *scenario* folder and the user is asked if they want to purge the training data to ensure data minimisation. This stored trained model can now be used by the system to accurately predict gestures and communicate the corresponding commands to the smart device for that scenario.

A detailed analysis of each of these phases of the system procedure can help provide a deeper understanding of the system and its working and pave way for further development and optimisation.

4.4.2 Key-point detection and frame augmentation using OpenCV and Mediapipe

Google's Mediapipe framework plays a very pivotal role in the project's success. Mediapipe is responsible for grabbing every frame obtained from OpenCV camera feed and identifying various key-points in the user's body including face, body, hands, shoulders, etc. For the purpose of this project, the only key-points of interest are the hand key-points since the main point of reference for any gesture performed by the user is their left and/or right hand. Every frame of the video feed is first passed onto a custom Mediapipe key-point detection function which processes that particular frame.

The Mediapipe Python framework grants direct access to the core components of the Mediapipe as well as hides the technical details of the framework and simply returns the readable model inference results back to the callers. This makes the code readable and allows the developers to focus on the main objective of the project.

```
def mediapipe_detection(image, model):  
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB  
    image.flags.writeable = False                 # Image is no longer writeable  
    results = model.process(image)                 # Make prediction  
    image.flags.writeable = True                   # Image is now writeable  
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR CONVERSION RGB 2 BGR  
    return image, results
```

Python

Figure 18: Code snippet for Mediapipe detection

The function first converts the frame received via OpenCV feed from BGR (Blue-Green-Red) colour channel to RGB (Red-Green-Blue) colour channel because the OpenCV library by default is encoded to process every frame in the BGR spectrum and Mediapipe requires the image to be in the RGB spectrum for proper detection. The function then temporarily converts the frame to non-writable. This step ensures that the system saves a little memory while processing through Mediapipe. Since this process is repeated for every frame in every video, this memory save quickly scales to return a significant amount of storage space in the cache.

Once, the prerequisites are in place, the processed frame is passed to the Mediapipe Holistic model. The Holistic model was initialised when the OpenCV camera feed was opened. This model requires the minimum detection confidence and the minimum tracking confidence properties to be mentioned during the model initialisation.

```
# Set mediapipe model and start the webcam  
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic_model:
```

Figure 19: Mediapipe Holistic model initialisation

Minimum Detection Confidence refers to the threshold accuracy that the model's detection needs to achieve in order to be viable for the initial detection. Minimum Tracking Confidence refers to the threshold accuracy that the model needs to cross when detecting the key-point movement from one point in the frame to another. Both of these properties are taken to be at 50% for the purpose of this project to tackle various scenarios.

Mediapipe Holistic model adopts a unique methodology for key-point detection. It starts with an initial detection and generates its results and then continues to only track the changes in the location of the key-points in the next frame. This methodology helps to reduce the processing time for the key-point detection in every frame making Mediapipe a very performance efficient framework for hand key-point detection.

Once the image is passed through the Mediapipe Holistic model, the results are obtained in the form of a custom Mediapipe object that contains various information including the key-points detected. These results are stored for future processing and the image is again set to writable. Once all this is done, the image is then again converted back to BGR channel for compatibility with OpenCV.

These results, along with the frame obtained from OpenCV is then fed to another custom function that draws these landmarks onto the frame at those points where they were detected for display.

```
def draw_styled_landmarks(image, results):  
    # Draw left hand connections  
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,  
                               mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),  
                               mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)  
                               )  
    # Draw right hand connections  
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,  
                               mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),  
                               mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)  
                               )
```

Python

Figure 21: Code Snippet for frame augmentation with hand key-points



Figure 20: Final output after Mediapipe detection and frame augmentation

This function makes use of the Mediapipe Drawing utility and its *draw_landmarks* function. The function requires the frame on which the key-points are to be augmented along with the key-points detected by the Holistic model, as input to return the same frame with the key-points drawn over them at the correct locations. This frame is then displayed onto the OpenCV camera feed. The results obtained for every frame using this process is accumulated into a single data object for every video and stored as a part of the dataset for a particular gesture.

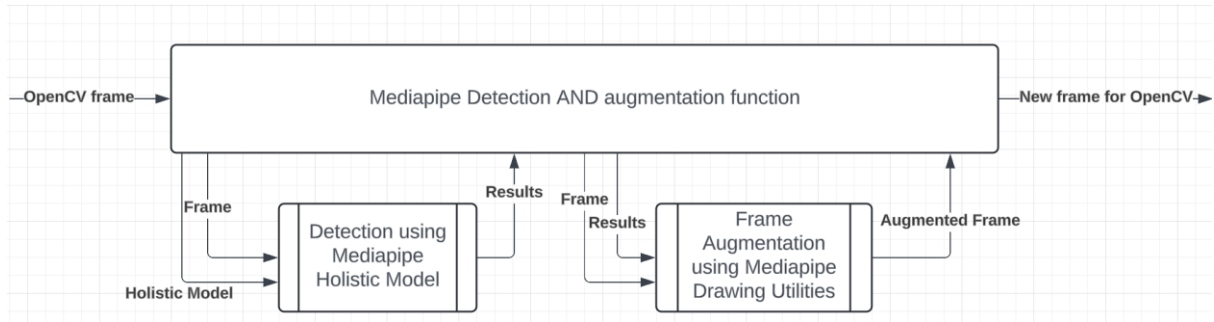


Figure 22: Summary of the key-point extraction pipeline

4.4.3 Data Collection and Preparation

Whenever the user chooses to create a new *scenario*, the user is prompted to choose an appropriate smart device from the list of supported smart devices. This list of smart devices depends upon the availability of their public APIs online and their Python package on PyPI ecosystem.

On choosing an appropriate smart device, the user is prompted to enter the list of gestures that they want the system to recognise in the future.

SCENARIO BASED GESTURE CONTROL
Made by Mudit Garg

[Project Global Variables](#) +

[Initial Project Checks](#) +

[Create new scenario](#) -

This section helps you to create a folder structure for a new scenario for which you can collect and store new training data and then use it to train a new LSTM model which can be used for gesture recognition in further steps. The scenario name will be used as the folder name.

Enter the scenario name:

Enter gesture names in separate lines and
Keep the name lowercase
Try to have one word name or words separated
by underscore only:

gesture1
gesture2
gesture3
gesture4

Create new scenario folder

[View current scenarios](#) +

Figure 23: Create a scenario option in the web interface of the system

This creates a new scenario subfolder by the name of the scenario in the *scenarios* folder of the project codebase. This subfolder is responsible to hold all the data related to that scenario which includes the list of gestures compiled into a NumPy array, the training data that the system is going to collect for each of the gestures entered into this gesture list and also the trained LNN model that is going to be obtained.

Once the scenario folder is setup and is ready with its gesture list, the system now prompts the user to provide the training data for each of the gestures in the gesture list with a series of

videos. The number of videos and the number of frames for each video is stored in the *user defined* global variables file which the user can change before the start of the collection of the training data to alter the amount of training data collected. However, this property remains constant for all the videos for all the gestures in the gesture list.

Once the collection of training data starts, the system opens up an OpenCV camera feed and starts the training data accumulation using Mediapipe Detection and Frame Augmentation pipeline. Before the start of every video for a gesture, the camera feed first displays the message of “STARTING COLLECTION” on a paused frame along with the gesture name and the video number to give time to the user to go to the first position of the gesture. After this, the camera feed records the movement of the hand key-points for a few seconds, according to the number of frames assigned, before again displaying the “STARTING COLLECTION” message for the next video.

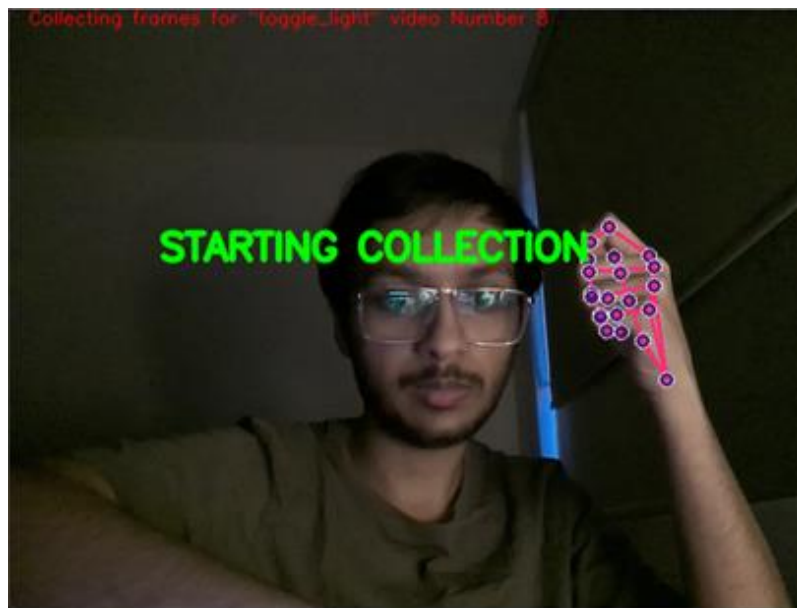


Figure 25: Starting frame for a gesture record

```
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # Loop through actions
    for action in actions:
        # Loop through videos
        for sequence in range(num_of_videos):
            # Loop through video length aka sequence length
            for frame_num in range(video_length):

                # Read feed
                ret, frame = feed.read()

                if ret != True: ...

                # Make detections
                image, results = mediapipe_detection(frame, holistic)

                # Draw landmarks
                draw_styled_landmarks(image, results)

                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(...)
                    cv2.putText(...)
                    # Show to screen
                    cv2.imshow(WINDOW_NAME, image)
                    cv2.waitKey(FRAME_COLLECTION_WAIT_TIME)
                else: ...

                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                full_path_to_frame = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(full_path_to_frame, keypoints)
```

Figure 24: Code snippet responsible for key-point data collection and storage for every action

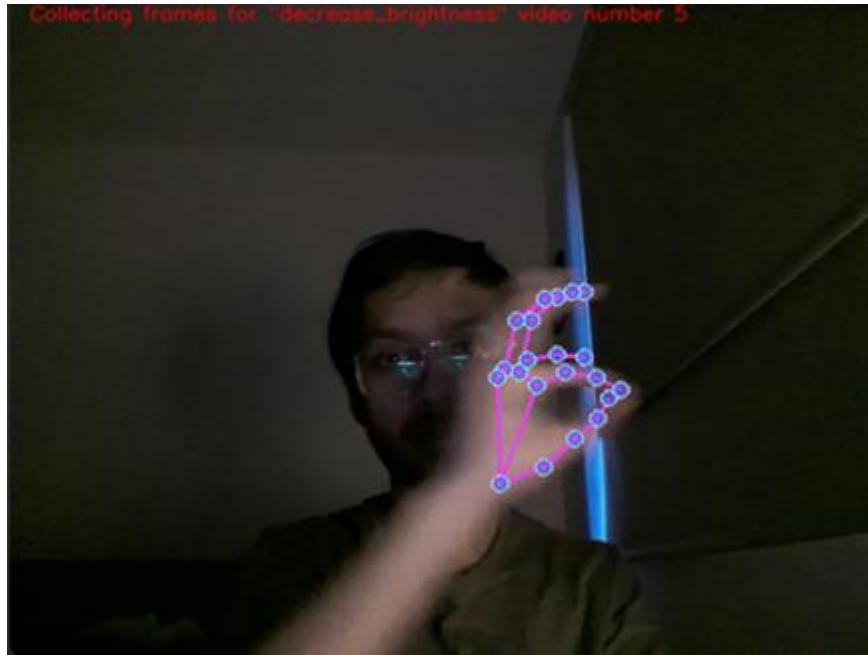


Figure 26: Gesture recording in progress

After the collection of all the frames for a video for a particular gesture, the results returned by the detection pipeline in the form of a Mediapipe custom object, this object is passed to another custom function for the key-point movement data extraction.

Keypoints extraction

```
def extract_keypoints(results):
    if results.left_hand_landmarks:
        left_hand = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten()
    else:
        print("Left hand was not detected")
        left_hand = np.zeros(21*3)

    if results.right_hand_landmarks:
        right_hand = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten()
    else:
        print("Right hand was not detected")
        right_hand = np.zeros(21*3)

    return np.concatenate([left_hand, right_hand])
```

[15] ✓ 0.0s Python

Figure 27: Key-points extraction function

This function reads the *results* object and extracts the left-hand key-points and right-hand key-points. If the results object does not have data in any one of those properties, the function fills their void with a bunch of 0s (null). The function then concatenates both these arrays and compiles them into one single key-point data NumPy array and returns them to the initial loop where they are then saved to the scenario folder.

This process is repeated for all the videos for all the gestures in the gesture list for that particular scenario before concluding the data collection.

4.4.4 Data Preprocessing

After the data is collected, the key-point data that is distributed into different NumPy arrays according to their respective video and respective frame for a respective gesture, the data is accumulated together into one single NumPy array which will be the feature data for the model training. During this, the gesture list for that scenario is also accumulated into one single NumPy array which will act as the label data for the model to be trained.

```
import os
videos, labels = [], []
for action in actions:
    for video in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        video_window = []
        for frame_num in range(video_length):
            res = np.load(os.path.join(DATA_PATH, action, str(video), f"{frame_num}.npy"))
            video_window.append(res)
        videos.append(video_window)
        labels.append(label_map[action])
```

✓ 45.4s Python

Figure 28: Accumulation of all data into a single NumPy array to ready for model training

The shape of the feature data NumPy array can be calculated according to the following formula:

$$(Total\ number\ of\ videos) \times (Video\ length) \times (number\ of\ features)$$

The shape of the label data NumPy array should be equal to the *Total number of videos*.

Here, the total number of videos is equal to the number of videos for each gesture multiplied by the number of gestures. The video length denotes the number of frames in each video. Finally, the number of features denotes the dimension of the features extracted from the *results* custom object which was the output of the detection pipeline.

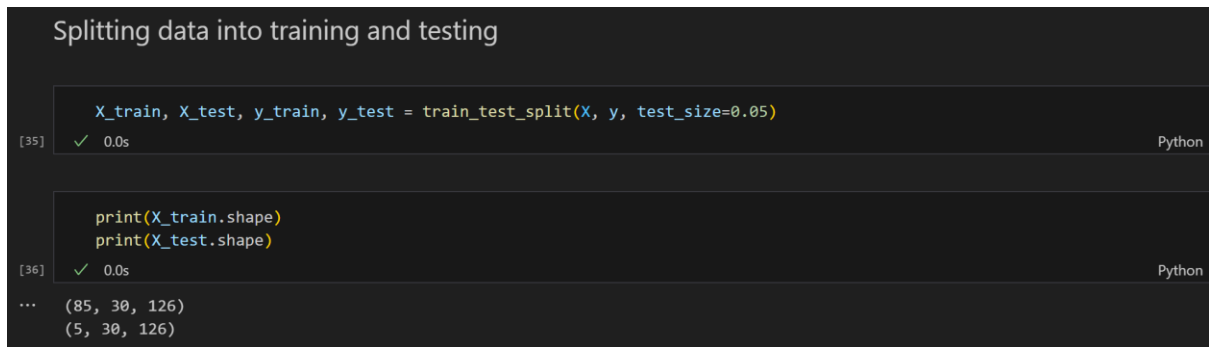
```
nt(np.array(videos).shape) #should be (num_of_video * number of actions, video_length, num of features equal to (the s
nt(np.array(labels).shape) #should be num_of_video * number of actions
```

✓ 0.0s Python

(90, 30, 126)
(90,)

Figure 29: Expected shapes of the feature dataset and label dataset for data collection of 3 gestures using 30 videos (with 30 frames each) for each gesture

The dataset finally generated is now split into a training dataset and a test dataset with a 95-5 ratio.



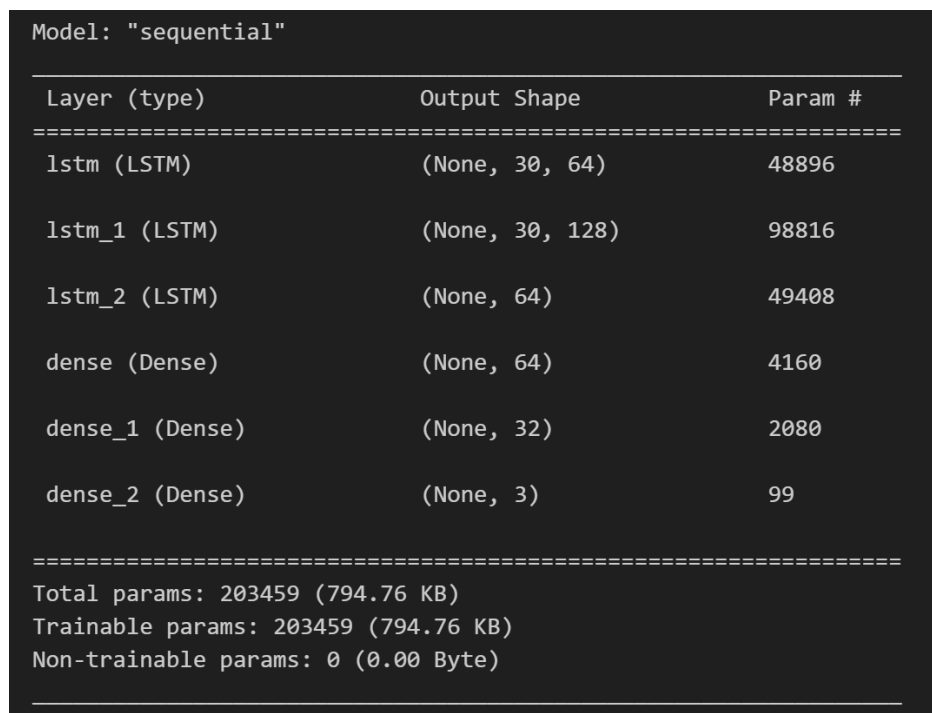
The screenshot shows a Jupyter Notebook titled "Splitting data into training and testing". It contains two code cells. The first cell, labeled [35], executes the code `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)` successfully. The second cell, labeled [36], executes `print(X_train.shape)` and `print(X_test.shape)`, resulting in the output shapes `(85, 30, 126)` for `X_train` and `(5, 30, 126)` for `X_test`.

Figure 30: Dataset splitting into training and testing dataset

Splitting the training data serves several important purposes for the trained model development. The test dataset serves as an independent dataset that is not used during the training process. This enables the evaluation of the model's performance on unseen data ensuring generalization ability of the model and its effectiveness in predicting on new data. Dataset splitting also helps to prevent overfitting which occurs when a model learns to memorize the training data rather than capturing the underlying patterns and relationships. A smaller test dataset (here 5% of the data) helps ensure that a sufficient amount of data is available for evaluation while still maximizing the amount of data used for training.

This concludes the final data preprocessing and makes the datasets ready to feed to the LNN gesture recognition model for training purposes.

4.4.5 LNN Gesture Recognition Model



The screenshot displays the summary of a sequential model. The model is named "sequential". The summary table lists the layers, their output shapes, and the number of parameters. The layers are: `lstm` (LSTM), `lstm_1` (LSTM), `lstm_2` (LSTM), `dense` (Dense), `dense_1` (Dense), and `dense_2` (Dense). The total number of parameters is 203459 (794.76 KB), which are all trainable. There are no non-trainable parameters.

Layer (type)	Output Shape	Param #
<code>lstm</code> (LSTM)	(None, 30, 64)	48896
<code>lstm_1</code> (LSTM)	(None, 30, 128)	98816
<code>lstm_2</code> (LSTM)	(None, 64)	49408
<code>dense</code> (Dense)	(None, 64)	4160
<code>dense_1</code> (Dense)	(None, 32)	2080
<code>dense_2</code> (Dense)	(None, 3)	99

=====
Total params: 203459 (794.76 KB)
Trainable params: 203459 (794.76 KB)
Non-trainable params: 0 (0.00 Byte)

Figure 31: Summary of model used for gesture recognition

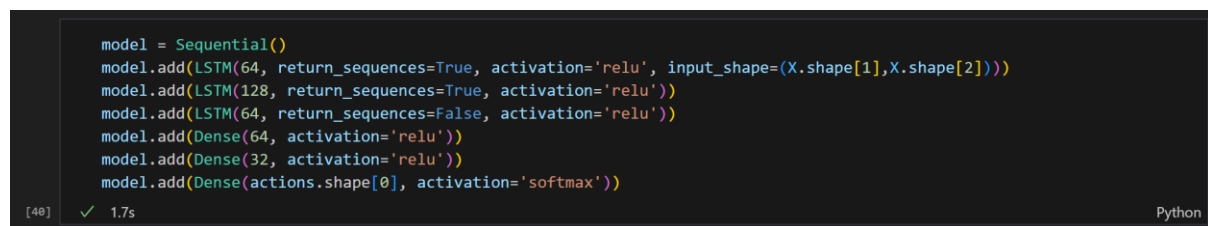
The gesture recognition model employed in the system represents a sophisticated machine learning architecture designed to effectively capture the temporal differences and patterns within sequential data, making it particularly well-suited for this project's task involving time-series analysis of hand movements and their sequential prediction. Comprising multiple sequential layers of LSTM units, the model leverages its recurrent nature to retain and propagate hand key-point information across time steps, enabling it to learn intricate patterns and relationships within the input data.

The model architecture begins with an input layer specifying the shape of the input data, consisting of a sequence of frames with a corresponding number of features. This input layer is fed into a stack of LSTM layers, each configured to process the input sequences while preserving their temporal structure. The first LSTM layer, with 64 units, is configured to take the input shape equal to the properties of each gesture video recorded as a part of the training data. In other terms, the input shape of the first LSTM layer is equal to number of frames in each video of each gesture into the number of features generated by Mediapipe when recognising the key-points of hands. This LSTM layer is followed by additional LSTM layers with increasing complexity (128 units, then back to 64 units), all configured to return sequences to facilitate information flow across layers.

The activation function 'ReLU' (Rectified Linear Unit) is utilized throughout the LSTM layer stack to introduce non-linearity and enable the model to learn complex mappings between input and output sequences. This choice of activation function helps alleviate the vanishing gradient problem commonly encountered in deep neural networks, ensuring more stable and efficient training for the gestures.

Following the LSTM layers, the model incorporates the Densely connected layers to perform feature extraction and abstraction, gradually reducing the dimensionality of the data while preserving essential information. These hidden dense layers, with varying numbers of neurons (64 and 32 units), further enhance the model's ability to capture and represent intricate patterns within the input sequences.

Finally, the output layer of the model utilizes the Soft Max activation function to generate probability distributions over the possible classes, which are the gestures in the gesture list of the *scenario* which are to be identified. This enables the model to make predictions or classifications based on the input data.



```

model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(X.shape[1],X.shape[2])))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))

```

[40] ✓ 1.7s Python

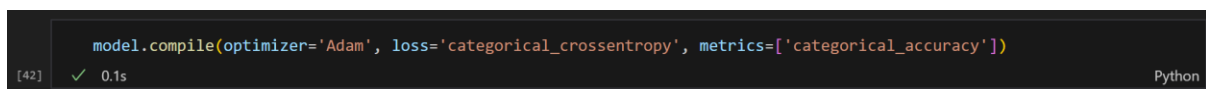
Figure 32: LNN Gesture Recognition model architecture

By compiling the model with these settings and metrics (categorical accuracy), the project ensures robust training and evaluation procedures, ultimately leading to a well-performing

LSTM neural network model capable of accurately predicting and classifying actions based on input sequences.

4.4.6 Model Training

Once, the dataset is finalised, the LNN gesture recognition model is compiled using the LNN architecture described in the above section along with the categorical cross-entropy loss function and the Adam optimizer, which efficiently updates the model parameters to minimize the loss encountered during the training process and improve the accuracy of the predictions.



```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Figure 33: Gesture Recognition model compilation

Once the training of the model is commenced, the Tensorflow Tensorboard Monitor is activated to monitor the training activity of the model. On training the model multiple times on increasing number of epochs, a generalised pattern was found. The training of the LNN gesture recognition model starts off with a random pattern of increasing and decreasing categorical accuracy. During this, the model reaches a very low point with the highest training loss encountered around 89 epochs. But this random pattern only plagues the model's performance up to the limit of around 115 epochs where the model has the lowest categorical accuracy.

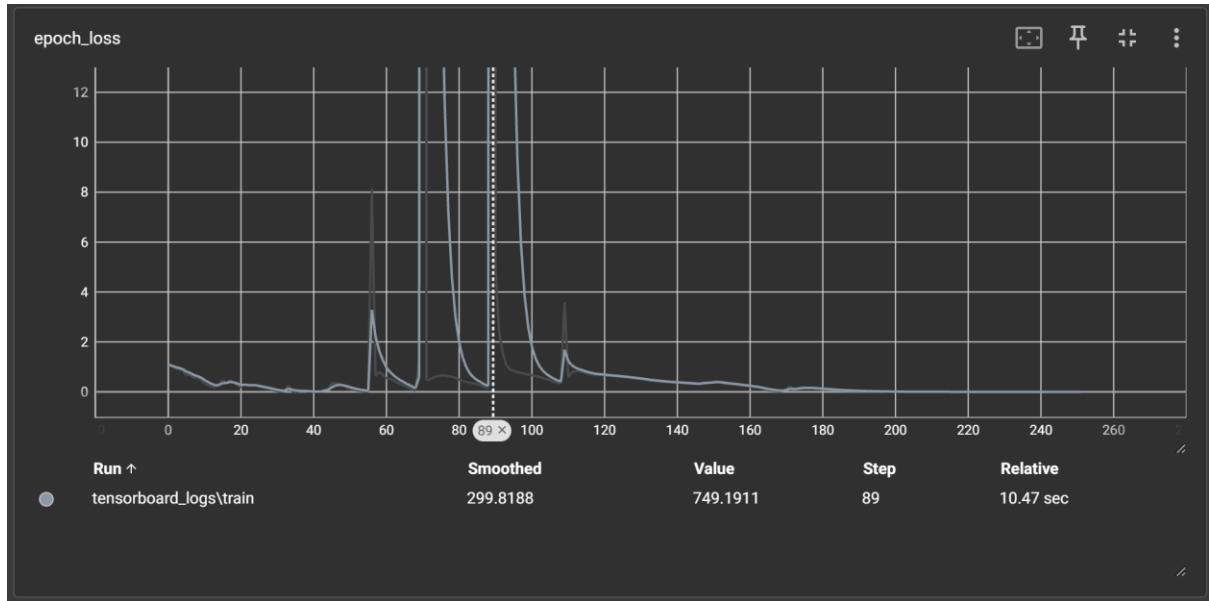


Figure 34: Highest peak in the epoch loss at 89 epochs

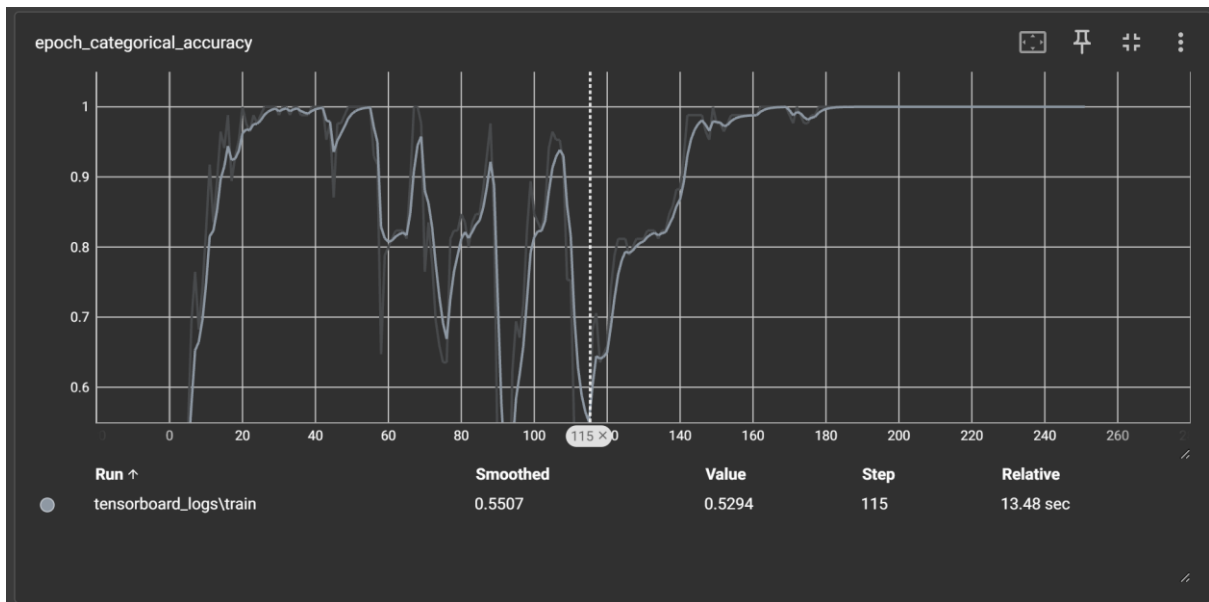


Figure 35: Lowest dip in the model categorical accuracy at 115 epochs

But after that, the training of the LNN model takes a turn for the better and the model starts to recognise the spatial and temporal patterns in the hand key-point movements in the training data and starts to recognise the correct labels for the correct set of features resulting in the increase in categorical accuracy. As the model starts to near the 180 epochs mark, the model training starts to show positive results with achieving near perfect accuracy and very low categorical losses with an order of magnitude of 10^{-5} . As the model cross the mark of 186 epochs, the model training proceeds towards near perfection with the categorical accuracy stabilising at almost 100% and epoch loss even lower with an order of magnitude of 10^{-8} .

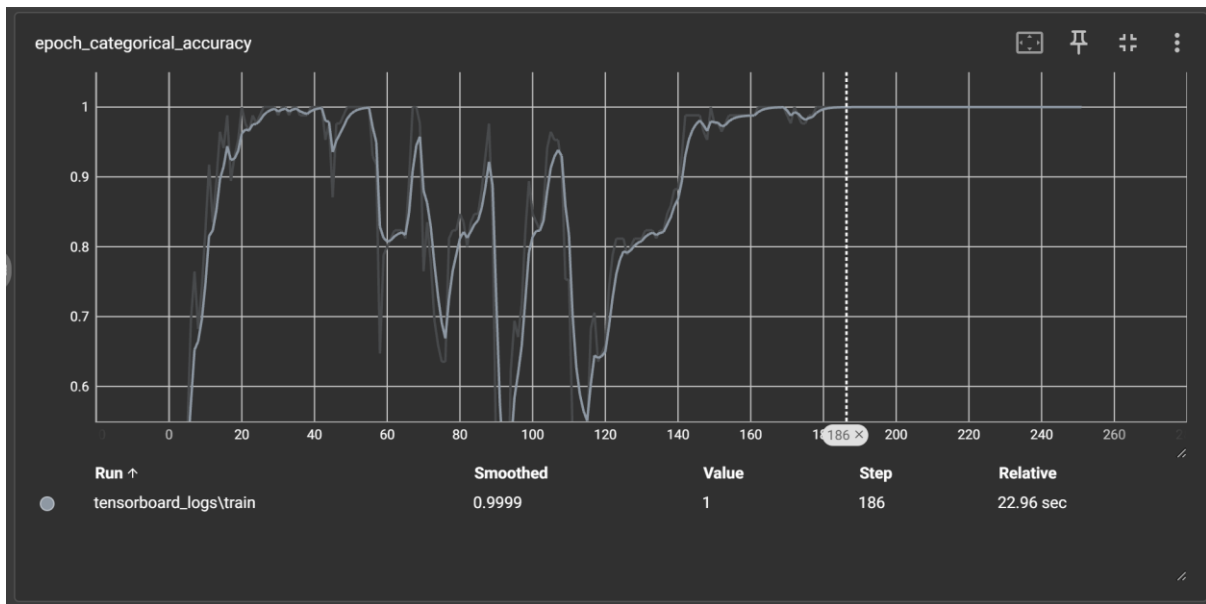


Figure 37: Categorical accuracy of the model at and after 186 epoch mark

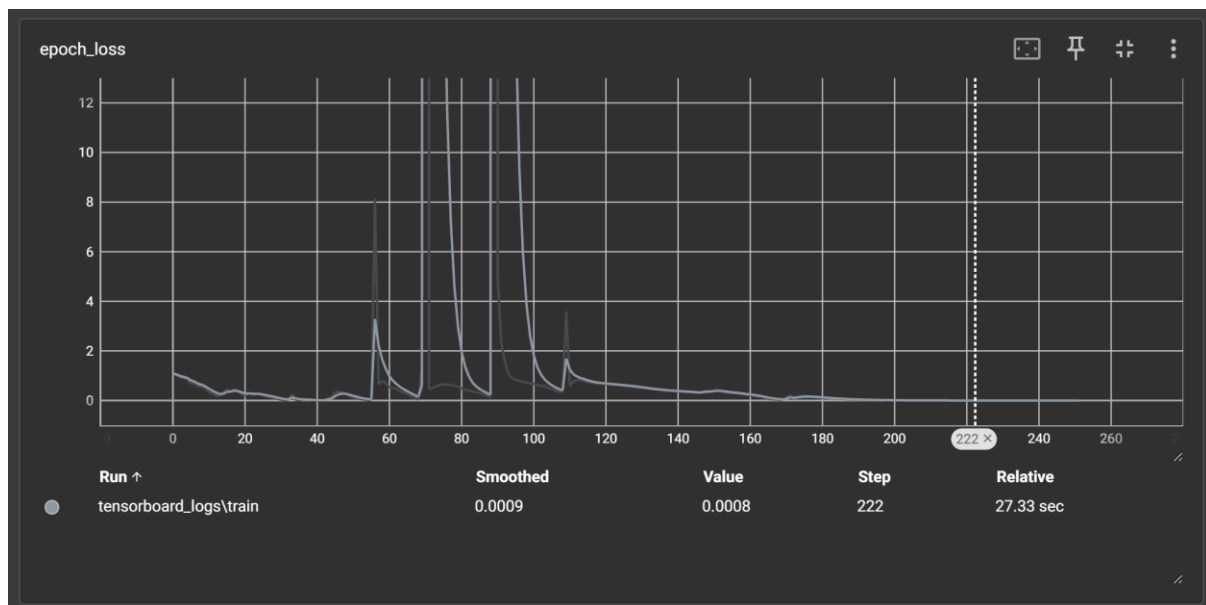


Figure 36: Epoch loss of the model after the 200-epoch mark

Once the model is finally trained, the trained LNN model can be then saved to that scenario's folder from where it can be accessed in the future for gesture recognition. The trained model is now tested against the 5% test dataset that was separated out in the data preprocessing phase.

```
[65] ✓ 0.1s Python
... 1/1 [=====] - 0s 33ms/step

yhat = model.predict(X_test)

ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()

[66] ✓ 0.0s Python
```

Figure 38: Code Snippet using Test dataset to set the model performance on unfamiliar data

The results of the trained model's performance are evaluated using a Confusion Matrix and the Accuracy Score.

```
multilabel_confusion_matrix(ytrue, yhat)
[67] ✓ 0.0s Python
... array([[2, 0],
          [0, 3]],

          [[3, 0],
          [0, 2]]], dtype=int64)

accuracy_score(ytrue, yhat)
[68] ✓ 0.0s Python
... 1.0
```

Figure 40: Confusion Matrix and Accuracy Score of the trained model with the testing dataset

This further proves that the model is performing extremely well with set of data outside the training dataset and the model does not suffer with the problem of overfitting and memorisation of the training data.

The project intends to also provide the opportunity to purge the training data that was collected for the training the LNN model to enable data minimisation and prevent any security or privacy hazard due to any leaks.

The model is also tested in real time using live feed from the user where the user can perform the gesture and the trained model is used to predict the gesture that the user has performed.



Figure 41: Testing of the trained model in real time

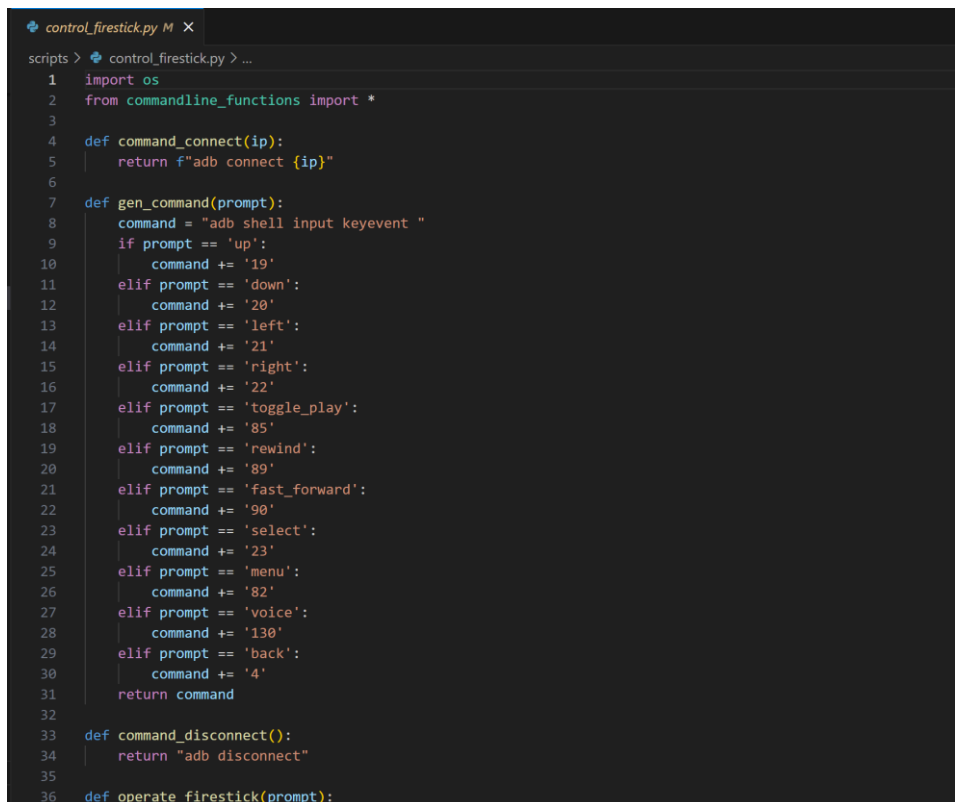
In this window, the system sends every frame through the detection and augmentation pipeline, generating the key-point movement data in space and time. The generated custom data object is then sent to the key-point extraction code snippet where the features are extracted which become the new feature data. This feature data is then provided to the trained LNN model which then determines the probabilities of the performed gestures being one of the gestures in the gesture list. This probability is visualised in the form of a progress bar and if the model feels confident enough, the gesture is recognised and added to the list of recognised gestures displayed at the top of the feed.

4.4.7 Communication with external devices

A robust communication infrastructure is very essential to the project to facilitate efficient control and interaction with various smart devices scattered throughout the home.

To make this possible, the system relies on standard communication protocol of Wi-Fi by utilising the LAN and its devices. In this implementation of the system, the system and the smart device both need to be in the same LAN in order to establish connection and communicate securely and effectively.

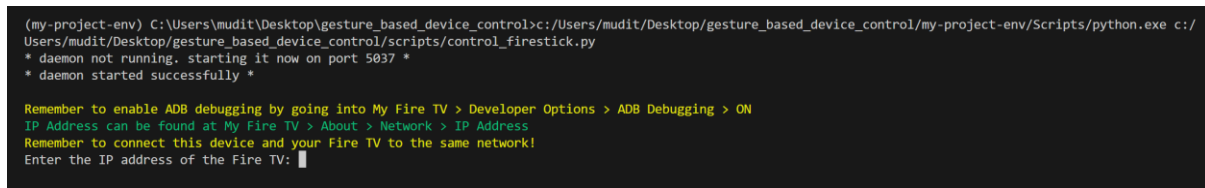
The system utilizes the public APIs and Python packages provided by the manufacturers of these smart devices to access the endpoints of these smart devices to which it can send the corresponding command and perform the action. Due to the adoption of this approach, the



```
control_firestick.py M X
scripts > control_firestick.py > ...
1 import os
2 from cmdline_functions import *
3
4 def command_connect(ip):
5     return f"adb connect {ip}"
6
7 def gen_command(prompt):
8     command = "adb shell input keyevent "
9     if prompt == 'up':
10         command += '19'
11     elif prompt == 'down':
12         command += '20'
13     elif prompt == 'left':
14         command += '21'
15     elif prompt == 'right':
16         command += '22'
17     elif prompt == 'toggle_play':
18         command += '85'
19     elif prompt == 'rewind':
20         command += '89'
21     elif prompt == 'fast_forward':
22         command += '90'
23     elif prompt == 'select':
24         command += '23'
25     elif prompt == 'menu':
26         command += '82'
27     elif prompt == 'voice':
28         command += '130'
29     elif prompt == 'back':
30         command += '4'
31     return command
32
33 def command_disconnect():
34     return "adb disconnect"
35
36 def operate_firestick(prompt):
```

Figure 42: Code Snippet showing the script that handles the commands for controlling Amazon Fire TV stick using ADB connection

codebase needs to have separate scripts or pipelines which can help the system communicate the recognised commands to these smart devices.



```
(my-project-env) C:\Users\mudit\Desktop\gesture_based_device_control>c:/Users/mudit/Desktop/gesture_based_device_control/my-project-env/Scripts/python.exe c:/Users/mudit/Desktop/gesture_based_device_control/scripts/control_firestick.py
* daemon not running. starting it now on port 5037 *
* daemon started successfully *

Remember to enable ADB debugging by going into My Fire TV > Developer Options > ADB Debugging > ON
IP Address can be found at My Fire TV > About > Network > IP Address
Remember to connect this device and your Fire TV to the same network!
Enter the IP address of the Fire TV: █
```

Figure 43: Demonstration of establishing connection with Fire TV stick using local IP address

For the demonstration of the idea, the project currently supports the control of the Amazon Fire TV stick using an ADB connection within the same LAN. In this scenario, the user enters the IP address of the Fire TV stick and once the connection with the device is established, the system is ready to go.

It is very important to note that since the project heavily focuses on the implementation side of the system, the security aspect is not considered to a very great extent. But to enable minimum level of security and privacy, in the current state of the project, the project does not store the connection details like the device properties, the IP address of the smart device and they are required to be entered every time in the start of a scenario to establish connection and to not leave any endpoints vulnerable, the system closes the connection to these devices once the scenario is exited.

4.4.8 User Interface Design

The system in the project is designed in such a way that the project codebase exposes various endpoints that perform various actions related to the actions in the system. These endpoints are accessible if the user calls the appropriate functions or executes the appropriate pipelines.

To replicate the behaviour in a more visually appealing fashion, the visual interface of the system is developed as a website running locally. This web interface is developed using the lightweight web development Python framework of Flask. Flask utilises these endpoints exposed by the project codebase to perform the various tasks that the user would have performed in the CLI version of the system by selecting appropriate options in the menu driven program.

The front-end interface of the web application is made using the HTML template that the Flask back-end server serves which contains all the necessary means of performing actions using endpoints.

These endpoints are accessed in the web application by navigating to particular routes. These routes can be accessed by clicking the appropriate buttons on the website which process the command as a particular route navigation request command. These particular route requests are sent from the front-end to the back-end server. The back-end server calls the appropriate endpoints of the system hence performing the task and returning the appropriate output back.

Create new scenario

This section helps you to create a folder structure for a new scenario for which you can collect and store new training data and then use it to train a new LSTM model which can be used for gesture recognition in further steps. The scenario name will be used as the folder name.

Enter the scenario name:

Enter gesture names in separate lines and
Keep the name lowercase
Try to have one word name or words separated
by underscore only:

Create new scenario folder

Figure 45: Button endpoint on the website for create scenario action with appropriate input form

```
@app.route('/create_new_scenario/', methods=['POST'])
def new_scenario():
    from scripts.main_script import scenario_folder_existence, set_new_scenario
    from scripts.gestures_functions import save_gestures_to_file
    scenario_name = request.form['scenario_name']
    scenario_gestures = request.form['scenario_gestures'].split('\r\n')
    scenario_folder_existence(scenarios_folder_path)
    print("=====")
    print(in_bold("New scenario details: "))
    print(f"Name : {scenario_name}")
    print(f"Gestures: {scenario_gestures}")
    print("=====")
    if folder_exist(os.path.join(scenarios_folder_path, scenario_name), show_output=False):
        print(in_red("Same scenario exists. Delete it to proceed!"))
        return "Duplicate scenario found. Retry!"
    print(in_bold("Creating the new scenario folder ... "),end='')
    try:
        set_new_scenario(
            scenario_folder=scenarios_folder_path,
            scenario_name=scenario_name,
            scenario_gestures=scenario_gestures
        )
        save_gestures_to_file(
            scenario_gestures,
            os.path.join(scenarios_folder_path, scenario_name, scenario_name+"_gestures")
        )
        print(success())
        return "Scenario name and gestures received. Go back to collect training data for this scenario and"
    except:
        print(failure())
        return "Error creating folder structure for new scenario"
```

Figure 44: Code snippet showing a particular endpoint that replicates the create scenario action of the system

4.5 Implementation Limitations

As the development of the gesture-based device control system reached its final stages, a multitude of challenges were realised, revealing the intricate web of complexities inherent in the implementation process. This section delves into the various limitations and constraints

encountered during the execution of the project, shedding light on the hurdles faced and the strategies employed to overcome them. From technical constraints to resource limitations, each obstacle presented unique restrictions to progress, shaping the course of development and influencing decision-making at every turn. By examining these implementation limitations and their potential solutions proposed along with them, a deeper understanding of the project and its components can be achieved.

4.5.1 Security and Privacy Limitations

One aspect of the project that was not prioritized during the implementation phase of the system was the security of the training data that was being collected for the gesture recognition LNN model. This posed significant concerns regarding the protection of sensitive information. Consequently, the project lacked robust mechanisms to safeguard data integrity and confidentiality, leaving it vulnerable to potential security breaches and unauthorized access. In response to this critical limitation, proactive measures were implemented to mitigate security risks and protect user data from exploitation. One such approach involved the implementation of data minimization techniques to reduce the volume of data collected and stored by the system. By limiting the amount of data retained to only essential information necessary for system functionality, the project aimed to minimize the potential impact of security vulnerabilities and mitigate the risk of data misuse by malicious actors. While this approach provided a rudimentary level of data protection, the need for more comprehensive security measures to be integrated into the system architecture is still very high. Moving forward, addressing this implementation limitation will require a concerted effort to prioritize data security considerations and implement robust encryption, access controls, and authentication mechanisms to safeguard sensitive information and ensure user privacy and confidentiality are upheld.

By the design choices of the system, this implementation of gesture-based device control system relies on various public APIs and packages within the PyPI ecosystem to facilitate communication with smart devices, inadvertently exposing the connection to potential security vulnerabilities. Unfortunately, the project initially did not prioritize the implementation of robust security measures to ensure secure communication between the system and smart devices. However, in an effort to mitigate these risks, the project imposes a limitation wherein smart devices must be within the local area network (LAN) for command communication to occur. While this approach restricts the scope of smart device connectivity to the LAN environment, it inherently lacks the robust security protocols necessary to safeguard against potential threats and ensure user privacy. Despite this limitation, confining smart device communication to the local network does reduce the exposure of the connection to external threats and minimizes the risk of unauthorized access. Moving forward, addressing this limitation will require the implementation of comprehensive security protocols, such as encryption and authentication mechanisms, to fortify the connection and protect user data from

potential security breaches. By prioritizing security considerations and implementing stringent security measures, the project can enhance the integrity and confidentiality of smart device communication, thereby safeguarding user privacy and mitigating the risk of security threats.

The management of connection metadata when interfacing with smart devices. Due to concerns regarding data security and privacy, the project adopts a cautious approach by refraining from storing extensive connection metadata, such as IP addresses and device information, which could potentially be exploited if accessed by unauthorized parties. As a result, the system implements small-scale initiatives, such as regularly prompting users to reconnect with smart devices by manually entering connection details. While this approach helps mitigate the risk of exposing sensitive connection metadata to potential misuse, it also introduces usability challenges and inconveniences for users who must repeatedly provide connection details. Additionally, the absence of persistent connection metadata storage limits the system's ability to streamline the user experience and maintain seamless device connectivity across sessions. Moving forward, addressing this implementation limitation requires striking a balance between ensuring data security and user privacy while also enhancing the convenience and usability of the system. By implementing secure storage practices for connection metadata and implementing robust encryption and access controls, the project can better safeguard sensitive information while also providing a more streamlined and user-friendly experience for users interacting with smart devices.

4.5.2 Interface Limitations

The front-end interface of the project faces with one small yet notable implementation limitation. It pertains to the presentation of the OpenCV camera feed within the front-end interface. Presently, the project's front-end interface displays the camera feed in a separate window rather than seamlessly integrating it within the website itself. This approach introduces a notable usability hurdle, as users are required to switch between windows to interact with the camera feed, detracting from the overall user experience and comfort. To address this limitation and enhance user satisfaction, further research and implementation efforts are warranted to seamlessly embed the camera feed directly within the website interface. By integrating the camera feed within the website, users can enjoy a more cohesive and intuitive interaction experience, eliminating the need for window switching and fostering a more immersive and comfortable user experience. This implementation enhancement holds the potential to significantly improve the usability and accessibility of the project, ensuring that users can seamlessly interact with the camera feed without unnecessary disruptions or inconveniences. Through diligent research and strategic implementation, this limitation can be effectively addressed, paving the way for an enhanced and more user-friendly project interface.

4.6 Implementation Issues

The journey of developing the gesture-recognition-based IoT device control system was marked by a myriad of implementation issues that posed humongous challenges along the way. This section serves as a comprehensive exploration of those issues encountered during the development phase of the project, shedding some light on the intricacies and complexities within each of those issues which hampered the progress in bringing the project goal to fruition. From technical hurdles to logistical constraints, each issue presented unique hurdles that demanded creative problem-solving and strategic decision-making. By delving into these implementation issues, we gain valuable insights into the realities of software development, how there is always room for issues to always pop up and planning should always be done with the margin or error considered at all times.

4.6.1 A separate model for hand key-point recognition and gesture recognition

The first objective of the implementation of the gesture-based device control task was to identify the hand and the finger key-points that were going to be tracked for recognising gestures. But the implementation of two machine learning models within the system, one for hand key-point recognition and one for gesture recognition, could have posed significant challenges throughout the development. Introducing separate models for hand key points recognition and gesture recognition added layers of complexity to the system architecture, necessitating meticulous attention to training, testing, and evaluation procedures. This approach would have proved to be resource-intensive, consuming considerable time and computational resources. This approach also would have led to the change in the project's focus since most of the resources would have then been spent on the training, testing, and evaluation of these models which required extensive data preprocessing, like image object isolation to isolate and identify the hand and fingers within each frame. This preprocessing step added to the overall complexity of the system and contributed to longer processing times, exacerbating the challenges associated with model training and evaluation. Moreover, the adoption of two separate machine learning models introduced latency issues that impacted the system's responsiveness and real-time performance. With each model requiring its own set of computations and predictions, the cumulative effect led to increased response times, potentially resulting in noticeable lag within the system. There needed to be enough performance in the system to provide some margin for latencies introduced due to external network factors to maintain seamless interaction between gestures and device commands. As a result, achieving optimal system performance while simultaneously mitigating latency issues emerged as a paramount concern, requiring innovative solutions to address the issue without the loss of any functionality within the system architecture.

To tackle the potentially biggest hurdle of the project, lot of alternative solutions were considered. One solution to tackle the whole situation was to make use of an already powerful model that was capable of the task of hand key-point recognition. This led to the research on

transfer learning and how pretrained models and their parameters can be used and compounded upon to create a more personalised model. This was known as transfer learning [7]. But transfer learning still did not get rid of some issues that the problem of having multiple machine learning models had introduced into the system. The method of transfer learning still required all the tasks related to data processing, model training and evaluation since the model and its parameters, although inspired, but was still a new one that had to be prepared, improved and optimised for the project.

This led to the adoption of a new approach to solve this issue. Trying to outsource the task of hand key-point detection. This led to a bit of research spree where various packages and libraries were discovered that performed the task at hand. The best one that was finally considered was Google's Mediapipe library. Mediapipe was the ideal choice for the project over its counterparts like YOLOv7 and Openpose due to several reasons. Mediapipe had end-to-end optimization, including hardware acceleration, all while being lightweight enough to run well on battery-powered devices. This made the project scalable and less resource intensive as compared to its other counterparts in the industry. It also had simple-to-use abstractions which made the implementation in the codebase very easy, readable and made the code very expressible, complying with the design strategies set during the initial phase of the project timeline.

4.6.2 No wait time between the recognised gestures

One significant implementation issue still plaguing the project is the lack of an effective mechanism to introduce wait time between successive gesture recognitions. While the system demonstrates proficiency in identifying gestures performed by the user in real-time, it fails to incorporate a pause or delay mechanism to allow for seamless communication with the smart device. This absence of a buffer time between two successful gesture recognitions severely restricts the system's ability to effectively interact with the smart device by sending commands and awaiting acknowledgment or success codes. As a result, the system may inadvertently trigger multiple commands in quick succession, leading to potential conflicts or errors in device control operations.

Moreover, the absence of a wait time introduces operational inefficiencies and compromises the overall user experience, as users may encounter difficulties in accurately controlling and coordinating actions with the smart device. Addressing this implementation issue requires the integration of a robust timing mechanism or buffer system within the gesture recognition algorithm, allowing for precise control over the timing and sequencing of command execution. By introducing appropriate wait times between gesture recognitions, the system can enhance its responsiveness, reliability, and usability, thereby improving the overall effectiveness and user satisfaction of the gesture-based device control system.

Chapter 5

Conclusion

5.1 Future Work

The gesture-recognition-based device control technology still has a lot of potential for improvement. While this project was an attempt to further this effort of making one aspect of the daily life technology more accessible for a larger community, there are still a few aspects about the project that require further development and refinement that need to be pursued before deeming it to be ready for everyday use. Some of the aspects of the project that need improvements are being discussed for future reference and development.

The system relies on various public APIs and packages within the PyPI ecosystem to facilitate communication with smart devices. This greatly reduces the scalability and flexibility of the system to incorporate a huge array of smart devices as this is now dependent on the fact that if there is an API or a PyPI package to facilitate connection or communication with that smart device. While a bit of research spree was conducted to figure out a universal solution for all the smart devices and a potential solution was found out to tackle the problem, it had a few issues and was not pursued in the timeline of this project. Homeassistant is an open-source home automation toolkit that is powered by a worldwide community of tinkerers and DIY enthusiasts. It has the ability to control an ever-expanding giant range of smart home devices due to its open-source community support. But Homeassistant according to the latest version during the development of this project, was restricted by various popular platforms and proved to not be flexible and supported on all platforms. Due to this and the clash of this alternative with the project's goals, this option, although having a lot of potential, was not pursued.

Although one of the objectives of the project was to develop an intuitive and user-friendly interface for a gesture-based device control system, certain constraints hindered the completion of this aspect of the work. While the implementation of endpoints for all actions was

successfully achieved, meeting performance expectations, the integration of these endpoints into a cohesive and streamlined interface remains pending. As a result, users are required to invoke the endpoints separately to execute various actions, indicating the need for further development to consolidate these functionalities into a unified interface. Despite these challenges, the project has made significant strides in laying the groundwork for efficient gesture-based device control, with opportunities for future enhancements in interface design and usability.

5.2 Final Words

In today's world, technology needs to be more accessible and free-to-use so that communities from all around the globe can benefit from it and the capabilities new technologies bring with them are utilised by everyone without any restriction or barrier due to very miniscule issues. The development of gesture-based device control system was an attempt to take this idea further by bringing accessibility and inclusivity to the grey area of smart devices control in today's modern homes. It was a very incredible journey ranging from learning from various new technologies that are currently being used as industry standards and understanding the requirements a technology should fulfil to be more useful and accessible for various communities.

Amidst these trials, there were also invaluable lessons and insights gained. With unwavering guidance from many people to sheer determination, the labyrinth of implementation issues was navigated meticulously, leveraging expertise and creativity to forge a path forward. It tested both skills and resilience while also forcing to expand the knowledge horizons and figure out different ways to perform tasks. Despite meticulous planning, unforeseen obstacles emerged, each demanding innovative solutions and strategic decision-making to overcome. Technical complexities, logistical constraints, and unexpected contingencies all contributed to the project's complexity, consuming valuable time and resources along the way. While many challenges were successfully overcome and significant milestones achieved, there remain lingering issues that require further investigation and resolution. These unresolved concerns underscore the ongoing nature of development and the constant pursuit of improvement and refinement.

As we proceed towards a more technologically advanced and inclusive world, it becomes evident that each obstacle represents a pivotal moment in the project's evolution. Through

perseverance and determination, the study not only tried to overcome challenges but also taught and gleaned over many valuable lessons that will inform the future endeavours to work in this area.

Bibliography

- [1] K. K. T. Punsara, H. H. R. C. Premachandra, A. W. A. D. Chanaka, R. V. Wijayawickrama, A. Nimsiri and S. Rajitha de, "IoT Based Sign Language Recognition System," 2020 2nd International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 2020, pp. 162-167, doi: 10.1109/ICAC51239.2020.9357267.
- [2] Ng, Chan Wah, and Surendra Ranganath. "Real-time gesture recognition system and application." *Image and Vision computing* 20.13-14 (2002): 993-1007.
- [3] Pavlovic, Vladimir I., Rajeev Sharma, and Thomas S. Huang. "Visual interpretation of hand gestures for human-computer interaction: A review." *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997): 677-695.
- [4] Jayakumar, Hrishikesh, et al. "Energy-efficient system design for IoT devices." *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2016.
- [5] <https://doi.org/10.48550/arXiv.1907.11073>
- [6] Pulli, K., Baksheev, A., Korniyakov, K., & Eruhimov, V. (2012). Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6), 61-69.
- [7] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... & He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1), 43-76.
- [8] <https://edition.cnn.com/2023/09/20/tech/double-tap-gesture-apple-watch/index.html>
- [9] Irit Meir, Wendy Sandler, Carol Padden, and Mark Aronoff (2010): Emerging Sign Languages. In: Marc Marschark and Patricia Elizabeth Spencer (eds.): The Oxford Handbook of Deaf Studies, Language, and Education, Vol. 2, pp. 267–80.
- [10] Cheok, M., Omar, Z., & Jaward, M. (2017). A review of hand gesture and sign language recognition techniques. *International Journal of Machine Learning and Cybernetics*, 1–23.
- [11] Lamberti, L.; Camastra, F. Real-time hand gesture recognition using a color glove. In *Proceedings of the International Conference on Image Analysis and Processing*, Ravenna, Italy, 14–16 September 2011; pp. 365–373.
- [12] Oudah, M.; Al-Naji, A.; Chahl, J. Hand Gesture Recognition Based on Computer Vision: A Review of Techniques. *J. Imaging* 2020, 6, 73. <https://doi.org/10.3390/jimaging6080073>

- [13] Wachs, Juan Pablo, et al. "Vision-based hand-gesture applications." *Communications of the ACM* 54.2 (2011): 60-71.
- [14] Pansare, Jayashree R., Shravan H. Gawande, and Maya Ingle. "Real-time static hand gesture recognition for American Sign Language (ASL) in complex background." (2012).
- [15] Van den Bergh, Michael, et al. "Real-time 3D hand gesture interaction with a robot for understanding directions from humans." *2011 Ro-Man. IEEE*, 2011.
- [16] Wang, Robert Y., and Jovan Popović. "Real-time hand-tracking with a color glove." *ACM transactions on graphics (TOG)* 28.3 (2009): 1-8.
- [17] Desai, Smit, and Apurva Desai. "Human Computer Interaction through hand gestures for home automation using Microsoft Kinect." *Proceedings of International Conference on Communication and Networks: ComNet 2016*. Springer Singapore, 2017.
- [18] Rajesh, Ram J., et al. "Distance transform-based hand gestures recognition for powerpoint presentation navigation." *Advanced Computing* 3.3 (2012): 41.
- [19] Kaur, Harpreet, and Jyoti Rani. "A review: Study of various techniques of Hand gesture recognition." *2016 IEEE 1st international conference on power electronics, intelligent control and energy systems (ICPEICES)*. IEEE, 2016.
- [20] Rastgoo, Razieh, Kourosh Kiani, and Sergio Escalera. "Sign language recognition: A deep survey." *Expert Systems with Applications* 164 (2021): 113794.