International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 09 | Sep 2022 www.irjet.net p-ISSN: 2395-0072

Hand Gesture Recognition and Translation Application

Siddhant Shivdikar[1], Juilee Thakur[2], Arnav Agarwal[3]

[1] *Electronics Department, Fr. Conceicao Rodrigues College of Engineering, Bandra, India*
[2] *Electronics Department, Fr. Conceicao Rodrigues College of Engineering, Bandra, India*
[3] *Electronics Department, Fr. Conceicao Rodrigues College of Engineering, Bandra, India*

Abstract - *Often during conferences or casual conversations, the speech-impaired people face issues of understanding what is being conveyed by the orator and also in rightfully putting their opinions in front of others without having any sort of miscommunication. In order to avoid such issues and establish a common ground for a conversation between them, the sign language gestures need to be understood for effective communication. In this project, we are developing an android application which not only detects a person's hand gestures (American Sign Language) in real-time using a pre-trained Machine Learning model (Recurrent Neural Networks) but also translate it into a text format. This text can later be translated into regional languages according to the user's choice. Our aim is to recognise American Sign Language (ASL) alphabets, numbers and a few commonly used phrases.*

Key Words: **Google MediaPipe, Android, ASL, Translation, Hand Tracking, Palm Detection, Machine Learning, Speech Impairment.**

1. INTRODUCTION

A majority of the population are capable of communicating with each other through different verbal methods like speech, with the help of various languages which were developed by people over past centuries. But not everyone is equally gifted to communicate through verbal methods. Such people who don't have the ability to hear and talk are called deaf and mute respectively by society. People who are deaf-mute experience great limitations and inconvenience in their personal and social life. This can hinder their day-to-day tasks such as travelling, working professionally, taking part in social events, etc. These limitations can prove a lifetime disadvantage such as

1. Losing the sense of independence: Relying on others for basic communication and the frustration of not being able to convey themselves adequately can lead to lower self-esteem and confidence.

2. Poor Education: The majority of the time, deaf-mute children are left with no choice but to enroll in mainstream schools. These children not only need a special environment for their education but also individual attention which is generally not available in mainstream schools. This results in lower grades than the normal kids and also eventually dropping out of school due to their environmental and physical shortcomings. Ultimately, they have lower employability and become less self-sufficient.

3. Poor Social interaction and communication: During summits, conferences or public / political speeches, it is observed that a Sign Language Translator is assigned for the speech-impaired people to understand what is being conveyed. Even during day-today conversations, it is difficult for a speech-impaired person to rightfully convey what he/she means with the help of sign language as there are chances of the other person interpreting it differently due to lack of knowledge about sign languages. This can often lead to miscommunication. The deaf-mute often lacks communication skills and hence have a poor social life.

It is shown that deaf people's relations are limited to their community and people who know sign language.

1.1. GESTURES

Gestures are meaningful maneuvers that involve physical motion of different body parts such as hands, fingers, arms, head, neck, eyes etc. Gestures convey meaningful information. Hand gestures in day-to-day life are used in a way by people to express their thoughts and feelings, which also helps to reinforce information delivered in our daily conversation. Hand gestures vary from person to person and are at the same time open for interpretation; thus, implying no structure. Sign language, on the other hand, is a structured form of hand gestures involving visual motions and signs, which are used as a communication system.

1.2. SIGN LANGUAGE

For speech-impaired as well as deaf people, sign language serves as a beneficial tool for day-to-day interactions. Sign language involves the use of different body parts such as our fingers, head, body, hand, arm and facial expression to deliver any kind of information. Using these parts of the body different gestures are done to represent concepts and ideas thus making Sign Language is a visual language of a kind. Primarily, sign languages are used by people who are deaf but they can also be used by others, such as people who can hear but are speech-impaired and also

helpful for children with Autism Spectrum Disorder (ASD) who struggle developing verbal communication. Hand gestures are usually distinguished into various types such as controlling gestures, conversational gestures, manipulative gestures, and communicative gestures. Sign language comes under the type of communicative gestures. Since sign language is highly structural, it is suitable to be used as a test-bed for computer vision algorithms.

Like spoken languages, there is no single global sign language that is used across the world. These sign languages are developed naturally through a variety of groups of people interacting with each other resulting in a large variety of sign languages. There are somewhere between 138 to 300 different types of sign languages used across the world. After thorough research, it was observed that the most common sign languages used around the world are American Sign Language (ASL), Indian Sign Language (ISL) and Arabic Sign Language (ArSL). Even though countries like Britain, Australia and America which have English as their first language have their individual sign languages.

1.2.1. AMERICAN SIGN LANGUAGE

American Sign Language (ASL) is a hand gesture-based language that does not depend upon speech or sound. ASL has been used and developed by the deaf community over centuries as a means of not only communication on a daily basis but also a sign of cultural unity and pride. The most common misconception about ASL is that it is a signed version of English. ASL has its own grammar, sentence formation, slangs, and also regional variation which is much more flexible than English due to which it cannot be stated as a representation of the English language. ASL users are as capable as other lingual users in conveying abstract or complex ideas.

1.2.2. SIGN LANGUAGE GESTURE RECOGNITION

Gesture's recognition involves complex processes such as motion modelling, motion analysis, pattern recognition and machine learning. It consists of methods that involve the use of manual and non-manual parameters. The structure of environments such as background illumination and speed of movement affects the predictive ability. The difference in viewpoints causes the gesture to appear different in 2D space. Gestures and sign language recognition includes an entire process of tracking and identifying various signs that are being performed and converting them into connotationally meaningful words. Earlier, dated around 1993, efforts were made on achieving gesture recognition. Back then gesture recognition techniques were adapted from speech and handwriting recognition techniques. To achieve this, Dynamic Time Warping (DTW) was used. Dynamic Time Warping utilizes information on the trajectory of the hand to compare a query sign with examples that were in a database.

Later on Hidden Markov Models (HMM) were proposed to distinguish and classify the orientation of the hand, its information of the trajectory and the resultant shape of the gesture depicting the one in sign language. Adaption of this concept comes from speech recognition and its innate properties make it ideal to be used in gesture recognition.

The use of HMMs alone had several limitations in training models. One such limitation was the three-dimensional translation and rotation data of the signs. This was overcome with the use of 'Flock of Birds' developed by Ascension Technologies. The Flock is Ascension's most popular tracker. Researchers, developers and end-users alike use it in all real-time visualization and interactive applications. Accompanied with this was using bigram and epenthesis modelling that helped in achieving higher accuracy. All these methods were applied in mostly overcoming a barrier in conversations at places of social gathering. Keeping this ideology in mind, efforts are being made even today to make advancements in gesture recognition so that this feature can be made easily available for everyone to use.

In places of gatherings that involve social interaction, efforts have always been made to make an impaired person feel less overwhelmed. Thus, to overcome the issue of the communication barrier, a common ground is to be established between people for effective communication to take place.

1.3. PROBLEM STATEMENT

Our aim is to develop an android application that uses a Machine Learning model to recognize the American Sign Language (ASL) gestures in real-time and convert them to text. Further, the detected-gesture text is converted into regional language depending upon the preference of the user.

The next chapter contains a detailed description of the research of the software and hardware methodologies used in hand gesture recognition and also some novel solutions of the same.

2. LITERATURE SURVEY

In this section, different approaches related to gesture recognition and translation have been discussed. The majority of the work for Sign Language Recognition can be generally be categorized:

1. Sensor-based gesture recognition

This approach requires the use of sensors, instruments to capture the motion, position, and velocity of the hand.

A. Inertial measurement unit (IMU): Measure the acceleration, position, degree of freedom and acceleration of the fingers. This includes the use of a gyroscope and accelerometer.

B. Electromyography (EMG): Measures human muscles electrical pulses and harnesses the bio-signal to detect fingers movements.

C. Wi-Fi and Radar: Uses radio waves, broad beam radar or spectrogram to detect in air signal strength changes.

D. Others: Utilizes flex sensors, ultrasonic, mechanical, electromagnetics and haptic technologies.

2. Vision-based gesture recognition

Vision-based approaches require the acquisition of images or video of the hand gestures through a video camera.

A. Single-camera: Webcam, video camera and a smartphone camera.

B. Stereo-camera: Using multiple monocular cameras to provide depth information.

C. Active techniques: Uses the projection of structured light. Such devices include Kinect and Leap Motion Controller (LMC).

D. Invasive techniques: Body markers such as colored gloves, wrist bands, and LED lights.

2.1. SENSOR-BASED SIGN LANGUAGE RECOGNITION

The first category is hardware-based solutions for sign language recognition. These solutions make use of special dedicated hardware, like sensors or gloves, that the user especially either has to wear or carry alongside which will, in turn, help the developers to get that extra variable they need to get more accurate results.

A primitive and most popular way of recognizing sign language is to use flex sensors attached to a glove. A flex sensor is basically a variable resistor whose terminal resistance changes with the change in bent angle. The resistance at the terminal increases linearly as the flat strip of the sensor is bent at higher angles. Kadam et al proposed that when flex sensors are mounted on the fingers of the glove, it changes the resistance as the position of the finger changes. These resistance values of each finger sensed by the flex sensor are then stored in the EEPROM of a microcontroller and can be saved and labelled as a gesture. The LCD display is used as a reference for how much a finger bends to correctly sign a letter. In this work, the authors have programmed the system to work in two modes viz teaching and learning modes. The user can flip a switch to toggle between the two modes. The teaching mode is where the user can input a gesture wearing a glove and can register that gesture into the system enabling the user to add user-defined gestures. On the other hand, the learning mode can be used for students to practice the registered gestures in the system. According to this implementation, the accuracy of the selected signs was around 86%. [1] Harish et al proposed a similar solution to recognize ISL with an addition of an accelerometer to include further parameters such as rotation, angle tilt and direction changes of the hand. [2] Chuang et al proposed a wireless solution for flex sensor gloves wherein a gated recurrent unit (GRU) algorithm was used to spot gestures using the sensory data obtained from the smart gloves. The motions correlated with various fingers and the changes between two consecutive gestures were taken into account during the GRU training period. For the final gesture grouping, the maximum a posteriori (MAP) calculation was used based on the gesture spotting data. [3]

A more complex way of recognizing dynamic hand gestures is by using RGBD cameras with three-dimensional convolutional neural networks (3D-CNN). Molchanov et al proposed a hand gesture recognition system that utilizes depth and intensity channels with 3D convolutional neural networks. It was interleaved between the two channels to build normalized Spatio-temporal volumes, and train two separate subnetworks with these volumes. To reduce potential overfitting and effective Spatio-temporal data to improve the generalization of the gesture classifier, the proposal was an augmentation method to deform the input volumes of hand gestures. The augmentation method also incorporates existing spatial augmentation techniques. [4] Tran et al proposed an updated version of this system by further detecting fingertips of the user using a similar RGBD camera and 3D CNN approach. [5]

In recent years there has been the development of novel sensors such as Leap Motion Controller and Microsoft Kinect which doesn't require the user to wear the apparel. The Leap Motion Controller (LMC) is an optical hand tracking module that captures the movements of the user's hands. It is a small USB peripheral that uses two monochromatic IR cameras and three infrared LEDs which covers a somewhat hemispherical area. The LMC sensors have a range approximation of 1 meter and the user needs to position their hands above the sensor in the range of the hemispherical area. The LMC then calculates the 2D frames generated by the cameras and gives the 3D positioning of the hands. Yang, Chen and Zhu when presented their work, they proposed the duo of an LMC and the two-layer Bidirectional Recurrent Neural Network (BRNN) has been used for the first time to detect Dynamic ASL gestures which the authors claim to be much more accurate. In addition to the two-layer BRNN, their framework included 26 discriminative features based on angles, positions, and distances between the fingers. In

this work, an LMC is used to extract features of dynamic gestures which are then processed by some specific methods and then given to the two-layer BRNN model which outputs the recognized gesture. The authors of this paper managed to get an accuracy of about 95% to 96% for 480 samples and 360 samples respectively which was higher amongst other significant works in the same field. [6]

These solutions are a great way to get fast and accurate results but they tend to be expensive and cumbersome. These special hardware devices are not integrated into our day to day lives. Thus, they would serve more of a special/limited purpose. Some drawbacks of using these solutions like the use of RGBD cameras, Leap Motion Controllers are that they all usually come in hefty sizes whereas solutions like flex sensors need the computers to process/compute their received data. As said, our aim is to maintain a common ground for communication for impaired people at any given point in time and such solutions won't prove to be effective as they need to thus carry this dedicated hardware.

2.2. VISION-BASED SIGN LANGUAGE RECOGNITION

The second category for hand gesture recognition is Computer Vision-Based. The advancement in image capturing methods and image processing techniques, making it one of the most popular and preferred ways of recognizing hand gestures. Vision-based methods have multiple benefits including that they can capture texture and color parameters which is difficult to achieve using sensor technology in 2D and 3D study of hand gesture recognition. Badhe and Kulkarni proposed the following steps in their work:

1. Data Acquisition: Acquiring the gestural data from the user. This can be done in two ways either with still images or in a form of a video. For a real-time system, video input is required which is then processed frame by frame by the system.

2. Preprocessing the Image: The raw images/frames from the camera undergo various methods of preprocessing to extract useful/needed parameters which means to eliminate the redundant, meaningless noise and superficial information that does not contribute to a vital matter of the feature extraction. This is a very important step to make the system work more efficiently and more accurately. Some of the most common preprocessing techniques are:

A. Image Enhancement and segmentation

B. Color filtration and skin segmentation

C. Noise Removal: Erosion and Dilation

D. Thresholding

E. Blob Detection

F. Contour detection

3. Feature Extraction: The feature components of the preprocessed gesture images are extracted and collected in form of data which is usually as vectors.

4. Template Matching: The stored vectors are then compared with the existing vectors that are stored in the reference vectors database.

5. Classification: Based on the output of the template matching the classification will be done as per the nearest match found in template matching.

6. Gesture Recognition: This block recognizes the gesture completely and produces the appropriate output.

The authors tested their system for 26 Alphabets, 9 numbers and up to 10 phrases made in ISL. The accuracy ranged between 95-100%, 85-100% and 85- 95% was achieved in alphabets, numbers and phrases respectively. [7]

Wang et al proposed gesture-based human-machine interaction can be achieved through precise hand segmentation using skin color and background information. In complex backgrounds with identical skin colors and non-uniform lighting, skin color based hand segmentation using skin color templates performs poorly. The method proposed by Wang thus improves accuracy by using related information to split the image on the reversed side. The findings of the experiments reveal that this method outperforms the method that only uses the skin color model. [8]

Further advancement of the previous work was done by Hsiang-Yueh Lai and Han-Jheng Lai. Lai et al. proposed in the work on Real-time Dynamic Hand Gesture Recognition that the processing of video streams is done where the stream is fed through the RGB camera. Webcams were used with resolution 640x480p to capture RGB images. Further, these images were converted to YCbCr format. Thresholds were set such that the skin color region was turned to white and the non-skin color (background) region was turned to black. Further morphological operations were done on the image to convert it into a complete binary image. The authors used OpenCV's list storage method and a sequence of contour points method to delete the noise and the face. The condition that the face's bottom contour is wider than the wrist contour was set to split the hand contour from the binary image. The center of the palm is the first to be detected as it resembles a rough square shape and is easiest to detect. Further fingers are detected and the angle between the

joints are defined. After the hand detection, the convex hull was calculated to detect the convex defect points. These points were used to calculate the angle between the finger spacing. Aculeate points were also detected in a close contour curve as fingertips. The hand gestures are recognized by the fingertip positions and angles between fingers. [9]

2.3. NOVEL SOLUTIONS FOR HAND GESTURE RECOGNITION

Around the world, researchers are always looking for ways to develop and modernize systems involving human-computer interactions with the growing advancement in the field of communication systems. The usual focus is on the exclusion of any special devices or vision-based technology. And the most fundamental application involving this application is hand gesture recognition. Thus, there are a few such unique solutions to Hand Gesture Recognition. Amongst these, two of the solutions were worth mentioning as the technologies used or the concepts involved would provide great help in the project as well as would widen the opportunity for future upgrades.

The first solution that we came across was WiGeR: Wi-Fi-Based Gesture Recognition System. In the research paper, Al-qaness et al. discussed the focus of this project is majorly based on the Wi-Fi signals. It is a novel segmentation method based on wavelet analysis and Short-Time Energy (STE). This solution includes Channel State Information (CSI) as the metric of the designed system. Earlier efforts were made to track human motion using the Wi-Fi based mechanisms that used Received Signal Strength (RSS) from the wireless MAC layer. Though this failed as the distance increased because the value of RSS decreased which led to multipath fading in environments that were a bit complex. In contrast to this, CSI adapts to the environmental changes and also uses O-FDM making it more robust. Its algorithm intercepts CSI segments and analyzes the variations in CSI caused by hand motions, revealing the unique patterns of gestures. To recognize the gestures through the walls, the algorithm involved effective segmentation and fast classification. To make this system more secure in terms of different signals sensed, the system has a pre-decided gesture for user authentication to commence detection. The makers of this WiGeR project claim accuracy up to 99% but only under certain conditions. The security is compromised and there are limitations on sensing the gestures through multiple obstructions such as a wall. Also, it requires sophisticated Wi-Fi (transmitters and receivers) which is difficult and not user-friendly. So it's still at a research stage and is not a viable option. [10]

The second novel solution is an app called Hand Talk Translator. This app has a 3D avatar of an interpreter. Input to the app is given in the form of text in the text box or in the form of speech that is later converted to text. The entered text is then translated by the avatar using animation of the same into an American Sign Language (ASL) gesture. The app allows a choice of two languages: English and Portuguese to be converted to ASL. For security purposes, the app requires authentication of the user. While setting up the profile, the app has a category that lets us choose if the user is hearing impaired or not. The app thus lets the users personalize the experience and store the history of the recent translations done. Using the history, the user can save the translations by marking them favorite under the user details entered for authentication, thus letting offline access to the translations anytime. The app also lets the users rate the gesture and if possible, provide feedback from time to time after the translations are done in order to enhance the overall experience.

In this chapter, some widely used methods for hand gesture recognition were discussed that involve hardware as well as software-based approaches along with some novel ideas. In the next chapter, the proposed implementation is discussed along with some of the preliminaries.

3. IMPLEMENTATION

As discussed in the literature survey above, vision-based techniques prove more beneficial in the lower-level case of Hand Gesture Recognition. Most of these projects and studies were implemented and executed on the computer. This resulted in some limitations while accessing the system as it was not user-oriented in terms of convenience.

In both academia and business, media review is a hot topic. A media decoder separates audio and video streams from a media file or camera content, which are then analysed separately. TensorFlow, PyTorch, CNTK, and MXNet are neural net engines that represent their neural networks as directed graphs with simple and predictable nodes, i.e. one input generates one output, which allows very efficient execution of the compute graph consisting of such lower level semantics. Beam and Dataflow, for example, are graph processing and data flow systems that run on clusters of computing machines. While the dependencies of each operation are often specified as graphs, Beam and Dataflow manage large chunks of data in a batching rather than a streaming manner, making them unsuitable for audio/video processing.

3.1. INTRODUCTION TO MEDIAPIPE

The existing state-of-the-art methods focus mainly on efficient desktop environments for inference whereas an android based solution was needed which will not only be lightweight in terms of computation but also be robust and accurate to achieve realtime performance. While looking

for a solution to the above-mentioned requirements, we discovered MediaPipe.

MediaPipe is an open-source platform for creating inference pipelines for arbitrary sensory data. A perception pipeline can be constructed as a graph of modular components using MediaPipe, which includes model inference, media processing algorithms, and data transformations, among other things. The graph contains sensory details such as audio and video sources, as well as perceived details such as object localization and face landmark streams.

MediaPipe is a tool for machine learning (ML) professionals, also including students, teachers, and software developers, who want to create technology experiments, deploy production-ready ML programs, and publish code for their study. Rapid prototyping of vision pipelines with inference models and other modular elements is the key use case for MediaPipe. It lets us build a model and deploy it on any platform (i.e., Android, iOS, web, IoT).

Deploying a model on a new platform requires various dependencies to be added so that the code can execute as intended to. Whereas with MediaPipe, we don't need to worry about the deployment of the ML models across platforms, it can be added like a module, and we can build the solution around it. MediaPipe is currently in the alpha stage at v0.7. Google to date continues to make and break API changes, guaranteeing stable APIs by v1.0. Version 0.7 of MediaPipe has ML Solutions like Human Pose Detection and Tracking, Face Mesh, Hand Tracking, Hair Segmentation, etc. which are implemented under Apache 2.0 License.

Compared to the other neural network engines mentioned above MediaPipe functions at a much higher level of semantics and allows for more complex and diverse actions, such as one input generating zero, one, or several outputs, which neural networks cannot model. Because of its sophistication, MediaPipe excels at interpreting media at higher semantic levels.

3.2. FRAMEWORK CONCEPTS OF MEDIAPIPE

In order to understand the way MediaPipe works, some basic framework concepts need to be understood to get a better understanding of how the processing happens.

3.2.1. PACKETS

A Packet is a basic unit of data flow in the MediaPipe Framework. A packet consists of a shared pointer to an immutable payload as well as a numerical timestamp. The payload can be of any C++ type, and the class of the payload is often known as the packet type. Packets are treated as value classes, which means they can be copied at a minimal cost. The ownership of the payload is shared by each copy. The timestamp of each copy is unique.

3.2.2. STREAMS

Each node in the graph has a Stream that connects it to another node. A stream is a set of packets whose timestamps must be increasing uniformly. Any number of input streams of the same kind may be attached to an output source. Each input stream generates a different copy of the packets via an output stream and retains its queue such that the receiving node can ingest them at its frequency.

3.2.3. GRAPH

A graph is made up of nodes that are connected by coordinated connections and through which packets will flow. The graph also contains the scheduler for the execution of nodes. A graph is usually specified in a distinct file called a graph configuration, or it can be constructed programmatically in code. Graphs help know the flow of the MediaPipe framework and as they represent themselves like a flowchart can thus prove helpful to debug through the nodes.

A 'GraphConfig' is a standard that specifies a MediaPipe graph's topology and features. The graph can be identified as a Subgraph to modularize a GraphConfig into sub-modules. The subgraph will then be used in a GraphConfig like it was a calculator. When a MediaPipe graph is mounted, the accompanying graph of calculators replaces each subgraph node. As a consequence, the subgraph's terminology and output are similar to that of the related calculator graph. MediaPipe graph execution is decentralized: there is no global clock, and different nodes can process data from different timestamps at the same time. Pipelining provides for higher throughput.

3.2.4. CALCULATORS

A Calculator is used to execute each node in the graph. The majority of graph execution takes place within its calculators. A calculator may receive zero or more input streams and can output the same amount of streams. Each calculator in a framework is documented with the system so that it can be defined by name in the graph configuration. All calculators stem from the same core Calculator class, which has 4 fundamental methods: GetContract(), Open(), Process(), and Close().

In GetContract(), the authors of the calculator define the desired types of inputs and outputs. When a graph is generated, the system calls a static method to validate whether the associated inputs and outputs' packet types complement the details in this definition.

The system calls Open() when a graph begins (). At this point, the calculator has access to the input side packets. Open() interprets node setup operations and prepares the per-graph run state of the calculator. Packets can also be written to calculator outputs using this tool. Any error during Open() will end the graph run.

When at least one input stream has a packet accessible, the system calls Process() continuously for a calculator with inputs. When concurrent execution is permitted, several Process() calls may be made at the same time.

If an error happens during Process(), the framework calls Close(), which ends the graph run. The system calls Close() until all calls to Process() have completed or all input streams have closed. And if the graph run is halted due to an error, this method is only called if Open() was called and succeeded and if the graph run is halted due to an error. The calculator should be called a dead node until Close() returns.

3.3. OVERVIEW OF THE PROJECT

The solution that we implemented was divided into two principal stages:

Stage 1 - Development on the Desktop

Stage 2 - Deployment on Android

3.3.1. STAGE 1

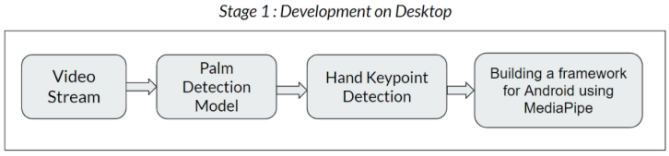


Fig -1: Stage 1: Development on Desktop

Fig 1. Shows the development of framework on desktop. The initial step of the first stage is to acquire a live video feed from the user via camera. The next step is the detection of the palm. This is implemented on MediaPipe using TensorFlow Lite. Initially, an anchor box is formed around the palm as it resembles a squarish shape which is easier to detect. After the palm is detected, anchor boxes are formed around the joints. After the palm detector defines the cropped image region of the hand, the hand-landmark model operates on it to detect the 21 key points of the hand. The keypoint structure is such that upon joining the adjacent points, they form a skeleton over the hand. This model is then converted to an Android Archive (AAR) file. The AAR file is thus to build an Android Application in the Android Studio.

3.3.2. STAGE 2

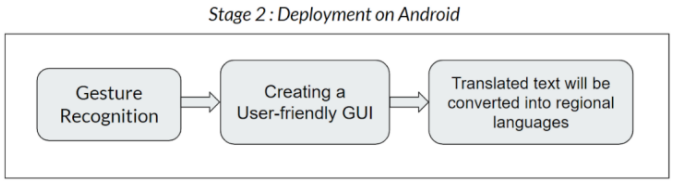


Fig 2 - Stage 2: Deployment on Android

Fig 2 show the Deployment of framework on Android. The AAR file generated in Stage 1 contains the hand-tracking model that outputs the 21 detected hand-key points along with suitable methods to access them using JavaScript. These detected key points are thus used to recognize the ASL gestures. The recognized gesture is then generated and displayed in a text format. A user-friendly interface (GUI) is developed for the Android application. The secondary objective of the project is a translation of the converted text into regional languages depending upon the choice of the user. A script was written to convert the recognized text into regional languages. The script was written to convert the English text into Hindi, Marathi and Gujarati.

3.4. PROPOSED IMPLEMENTATION

The steps listed in the previous section are briefly discussed in this section. The implemented solution utilizes a couple of frameworks working together:

1. A palm detection model (called BlazePalm) that operates on the full image and returns an oriented hand bounding box.

2. A hand landmark model that operates on the cropped image region defined by the palm detection model and returns high fidelity 3D hand key-points.

3. A gesture recognition script that utilizes 3D hand key points as a basis to determine the ASL gesture.

3.4.1. PALM DETECTION MODEL

A single-shot detector model called BlazePalm is designed for mobile real-time to detect initial hand positions. Hand detection is a difficult task: the model must identify occluded and self-occluded hands when working through a wide range of hand sizes with a huge scale distance (>20x) relative to the picture frame. Faces have high contrast patterns, such as around the eyes and lips, whereas hands lack such characteristics, making it more difficult to accurately detect them based on their visual features alone. Providing additional context, such as arm, leg, or individual characteristics, helps with correct hand localization. Fig 3. shows the flow of Palm detection model through MediaPipe Graph.

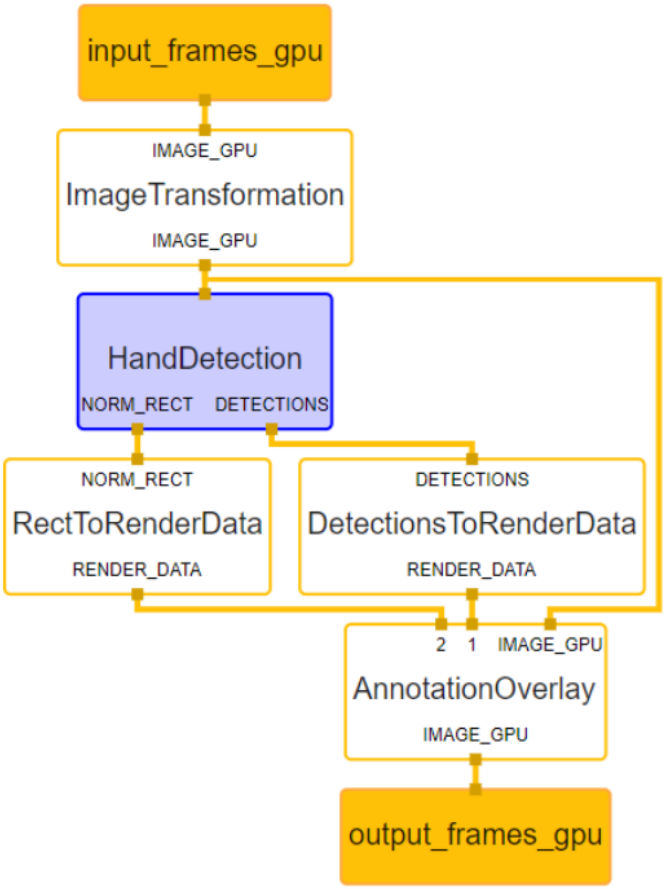


Fig 3- MediaPipe Graph for Palm Detection

First, instead of training a hand detector, we train a palm detector because estimating bounding boxes of rigid structures like palms and fists is much easier than detecting hands with articulated fingers. Furthermore, since palms are smaller specimens, the non-maximum suppression algorithm performs well even in two-hand self-occlusion situations such as handshakes. Furthermore, palms can be modelled using square bounding boxes (anchors in ML terminology) that ignore other aspect ratios, resulting in a reduction of 3-5 anchors. Second, also for small items, an encoder-decoder feature extractor is used for larger scene context understanding. Finally, due to the high size variation, the focal loss is reduced during the preparation to sustain a large number of anchors.

Following is the algorithm of the BlazePalm model. Briefly, the input image on the CPU/GPU is transformed into a 256 x 256 image. Scaling of the input image is done using the FIT function to preserve the aspect ratio. It is then integrated with TensorFlow Lite. Next is the transformation of the input image on GPU into an image tensor stored as a TfLiteTensor. It then takes image tensor and converts it into a vector of tensors (e.g. key points on palm i.e. joint-points). Vectors are then generated of SSD anchors based on specification. It decodes the detection tensors based on SSD anchors into vector detections. Each detection describes a detected object. Suppression is done to remove excessive detections. It then maps the detection label IDs and adjusts detected locations on letterboxed images to corresponding locations on the same image. Next is the extraction of image size from the input images. It converts results of palm detection into a rectangle (normalized by image size) that encloses the palm and is rotated such that the line connecting the center of the wrist and MCP of the middle finger is aligned with the Y-axis of the rectangle. It expands and shifts the rectangle that contains the palm so that it's likely to cover the entire hand.

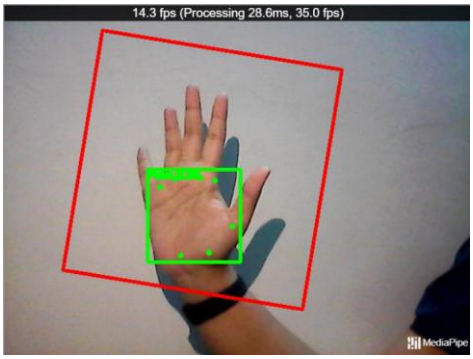


Fig 4- Palm Detection Using MediaPipe

Fig 4. shows the result obtained after implementing the Palm detection model. As seen in the above-obtained result, anchor boxes are generated on the Hand and the Palm. A red anchor box is generated around the Hand for further prediction of the position of the hand. A green anchor box is generated around the Palm.

3.4.2. HAND-TRACKING MODEL

Following palm detection across the entire picture, the hand landmark model uses regression to perform precise key-point localization of 21 3D hand-knuckle coordinates within the observed hand regions, which is known as a direct coordinate prediction. Even with partially visible hands and self-occlusions, the model develops a strong internal hand pose representation. 30K real-world images with 21 3D coordinates were annotated to obtain ground truth results. Z-value was taken from the image depth map if it exists per the corresponding coordinate.

A high-quality synthetic hand model was made over different backgrounds and mapped to the corresponding 3D coordinates to help cover the potential hand poses and provide further supervision on the essence of hand geometry. Purely synthetic evidence, on the other hand, does not generalize well to the real world. A mixed training schema was used to solve this dilemma.

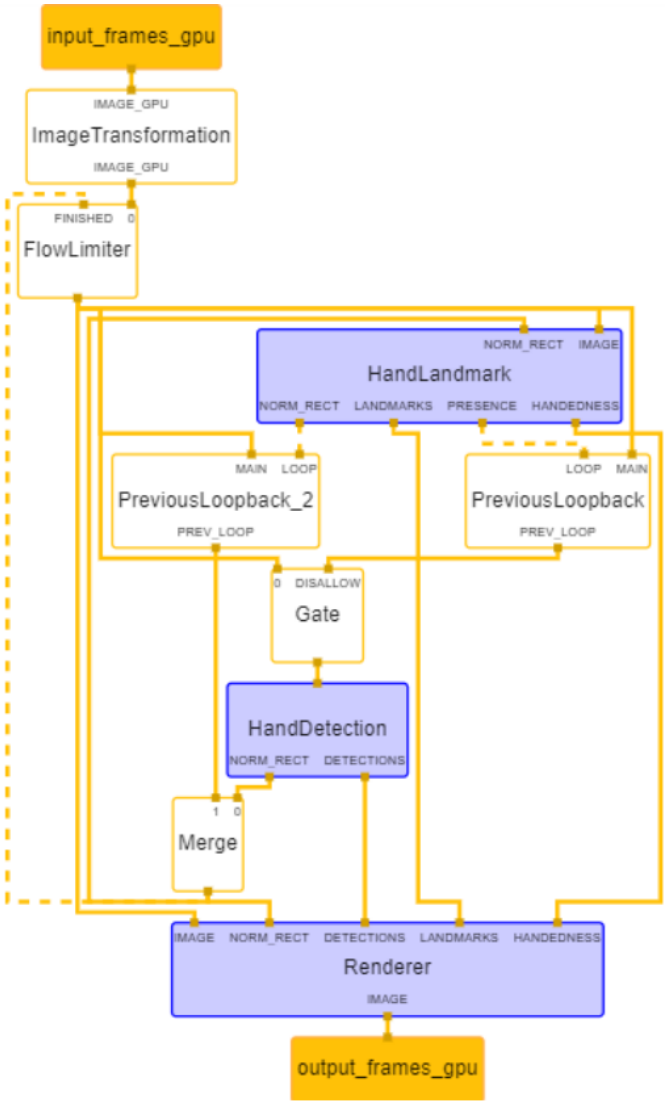


Fig 5- MediaPipe Graph for Hand Tracking

Fig 5. shows the flow of Hand Tracking graph in MediaPipe. This graph throttles the images flowing downstream for flow control. It passes through the very first incoming image unaltered and waits for downstream nodes (calculators and subgraphs) in the graph to finish their tasks before it passes through another image. All images that come in while waiting are dropped, limiting the number of in-flight images in most parts of the graph to 1. This prevents the downstream nodes from queuing up incoming images and data excessively.

The graph caches hand-presence decision feedback from HandLandmarkSubgraph, and upon the arrival of the next input image sends out the cached decision with the timestamp replaced by that of the input image, essentially generating a packet that carries the previous hand-presence decision. The anchor boxes made on the Hand detection part are then used for the Hand tracking part.

The MergeCalculator merges a stream of hand rectangles generated by HandDetectionSubgraph and that generated by HandLandmark Subgraph into a single output stream by selecting between one of the two streams. The annotations and overlays are rendered on top of the input images.

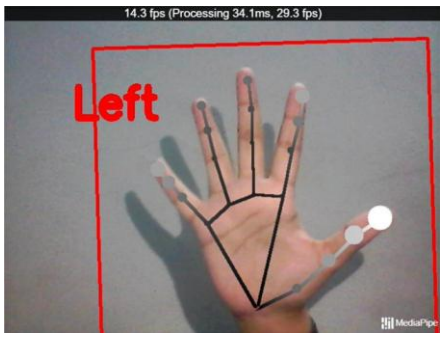


Fig 6- Hand Tracking Using MediaPipe

Fig 6 shows the result of Hand Tracking Model with the help of MediaPipe. From the results shown above, 21 key points detected on the hand can be observed. The adjacent points are joined together to form a Skeleton over the hand. These detected points i.e., the skeleton is used to detect the gesture.

3.4.3. GENERATION OF AAR FILE

The architecture of an Android library is the same as that of an Android app module. It comprises source codes, resource directories, and an Android manifest, among other things. An Android library, on the other hand, compiles into an Android Archive (AAR) file that can be used as a dependency for an Android app module, rather than just an APK that runs on a device. AAR directories, unlike JAR files, have the following features for Android applications:

- In addition to Java classes and methods, AAR files contain Android resources and a manifest file, enabling one to bundle in shared resources like templates and drawables.

- C/C++ libraries for use by the app module's C/C++ code can be found in AAR directories.

When you're creating an app with different APK versions, such as a free and premium version, and you need the same key components in both, AAR comes in handy. It is also beneficial when you're designing different apps that share common components like events, utilities, or UI templates.

The MediaPipe Android Archive (AAR) library is an easier way to use Android Studio and Gradle. MediaPipe does not have a generic AAR for all projects to use. Instead, developers might have to add a MediaPipe_aar() target to

their project to create a custom AAR file. This is important in order to provide unique resources for each initiative, such as MediaPipe calculators.

Following steps to build and integrate AAR into Android Studio:

1. MediaPipe_aar() target is created.

2. AAR file is built and generated using Bazel commands.

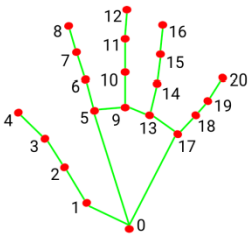
3. The AAR file generated is saved in the libraries in the Android Studio.

4. Assets are saved into the assets folder in the main file.

5. OpenCV JNI libraries are saved into a new libraries folder. 6. build.gradle file is modified to add MediaPipe AAR and MediaPipe dependencies.

3.4.4. GESTURE RECOGNITION

The MediaPipe AAR file, which was introduced to the Android project's repositories, is inserted into the Android project's MainActivity file. Other required functions from the package, such as BasicActivity, Landmark, Structure, and Format Files, were declared. The Hand Landmarks (containing 21 key points) shown in Fig. 8 obtained from the AAR file are stored as a list in the MainActivity file which extends the BasicActivity.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig 7- 21 Hand Keypoints

A Hand Gesture Calculator was created to utilize the Hand Landmarks. Initially, these Hand Landmarks were used to detect the state of each finger, i.e., whether it is open or close. A Boolean value was set to false by default for the state of each of the fingers. For the first finger to be open, a reference point of the same is taken which is the key point 6 in this case. This reference point is then compared with the rest of the key points of the first finger. The example for this is given below in Fig 8.

```
pseudoFixKeyPoint = landmarkList.get(6).getY();
if (landmarkList.get(7).getY() < pseudoFixKeyPoint && landmarkList.get(8).getY() < landmarkList.get(7).getY()) {
    firstFingerIsOpen = true;
}
```

Fig 8- Example of the state of the finger

If the conditions are true the boolean is set to true. To set the boolean value of the state to true, corresponding comparisons were performed in a similar way for the rest of the fingers. For each gesture, a group of conditions were set to check the state of each finger; whether it is open or close. Based on these states, the gestures are recognised. For example, to recognise the gesture 'HELLO' we check the state of all the fingers and the thumb based on the action of the gesture. In this case, we need the state of all the fingers and thumb to be open. The conditions for the same are shown in Figure 9.

```
// Hand gesture recognition
if (thumbIsOpen && firstFingerIsOpen && secondFingerIsOpen && thirdFingerIsOpen && fourthFingerIsOpen) {
    return "HELLO";}
```

Fig 9- Example of the Gesture Recognition for "Hello"

There are few gestures that require the relative positioning of the hand landmarks as the information regarding the state of the finger is insufficient. In such cases, Euclidean distance needs to be measured between the key points. If the Euclidean distance is less than 0.1 it states that the key points are in closer vicinity of each other thus letting us detect the relative positioning of fingers. For the static ASL gesture 'Money', the thumb and the first finger need to be near each other. This is where the Euclidean distance is calculated between the first finger and the thumb and checked if the distance is lesser than the threshold. If the condition is set to be satisfied, the boolean of the same is set 'true'. Along with the Euclidean distance condition, the state of the remaining fingers is also verified to detect the desired ASL gesture. In this project, we successfully implemented 10 static American Sign Language Gestures as shown in Figure 10.

Fig 10- Implemented Static ASL Gestures

3.4.5. TRANSLATION INTO REGIONAL LANGUAGES

To help multilingual users, the static ASL gestures that were identified had to be translated into regional languages. Services like Google Cloud Translation API helps developers to do so at a fixed pricing chart. As the aim of this project was to make an app that could be available to maximum users with no or minimal costing, our approach was to make a dictionary for each of the languages.

In the Android application, an options bar (spinner) was added in order to facilitate the user to select the language of their choice. The spinner sends the selected option to the dictionary and thus the appropriate language is displayed. Three regional languages namely Hindi, Marathi and Gujarati were implemented along with the default language English. Figure 11 shows the implementation of the language spinner in this project.

Fig 11- Language Spinner

When the user's hand is brought in front of the camera, the feed starts tracking the hand that is displayed over the app in the form of a skeleton. In a split second, the gesture is detected and the output is displayed in green color at the top-center of the screen in the English language by default. For the user to change the language according to his/her preference a dropdown menu is displayed at the bottom-center of the app. Upon choosing the preferred language, the language of the recognized gesture displayed on the top-center is changed accordingly.

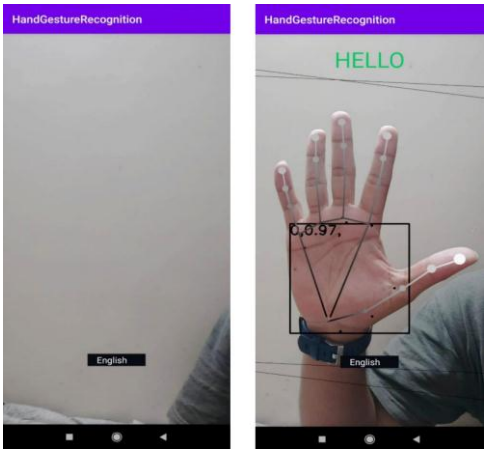


Fig 12- Hand Gesture Recognition Application Layout

4. RESULTS

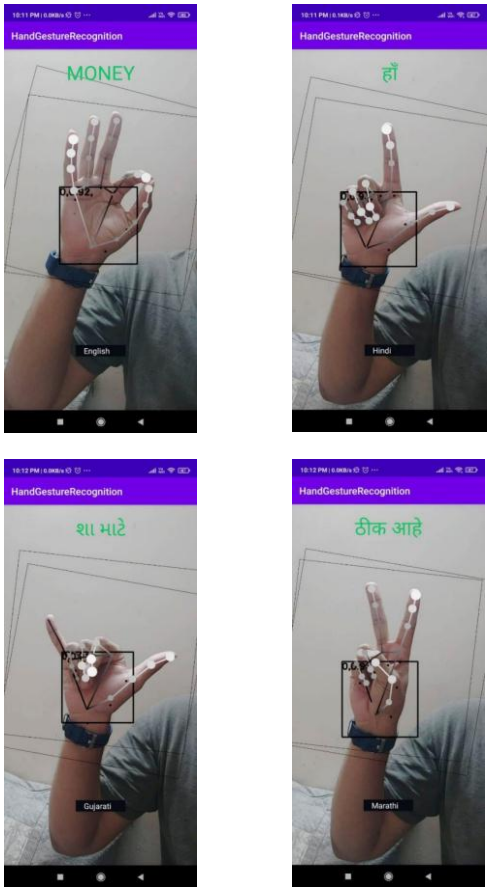


Fig. 13- ASL Gestures along with regional translations

The Hand Gesture Recognition App was run and tested on the following 4 devices with different hardware specifications and Android versions which are mentioned in the Table 1.

Table -1: Test Device Specifications

Device	Processor – RAM	Android Version	Screen Ratio (cm)	Camera Resolut-ion (MP)
Mi Note 9 Pro	Octa-core Max 2.32Ghz - 8 GB	Android 11	16.6 x 7.7	32
Redmi K20 Pro	Octa-core Max 2.84Ghz - 8GB	Android 10	15.7 x 7.4	20
Samsung Galaxy Note 8	Octa-core (4x2.3GHz + 4x1.7GHz) - 6 GB	Android 8	16.3 x 7.5	8

OnePlus 5	Octa-core (4x2.45 GHz + 4x1.9 GHz) – 6 GB	Android 10	15.4 x 7.4	20

The app was tested in debugging mode on all of the 4 devices using Android Studio's Profiler tool. The Android Profiler tool provides real-time data that helped us understand how the app uses CPU, memory, network, and battery resources. The debugging process was done to accumulate data for a time period of 2 minutes. Results for one of the devices are shown below in Fig 14.

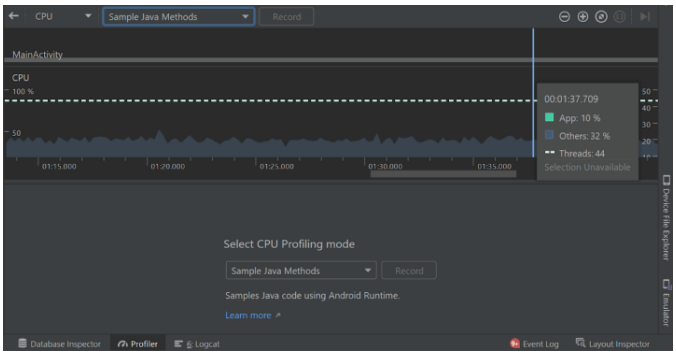


Fig 14- CPU utilization of application

The above figure shows the CPU utilization of the device while running the app. It was observed that the application utilized 10 to 15% out of the total CPU capacity.

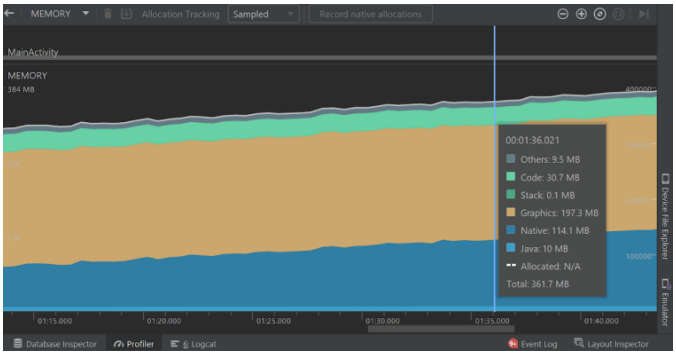


Fig 15- Memory usage of application

The figure 15 represents statistics of memory usage of the Application. Over the two-minute period of testing it was observed that 400MB peak was used by the application, which is between 8-10% of the 4GB RAM available in the device.

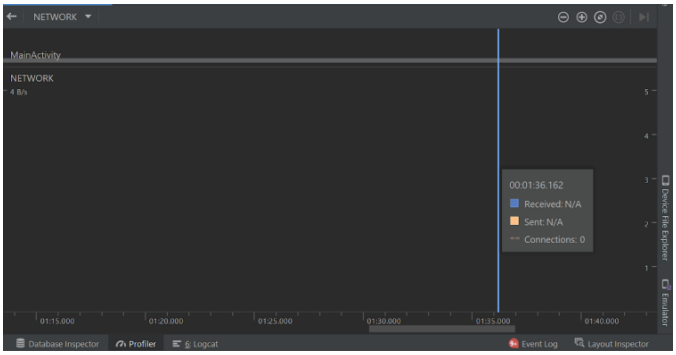


Fig 16- Network usage of application

The above Fig 16 shows the network usage of the application. As the application is completely self-sufficient and does not require any internet provided services, no activity was seen in this analysis.

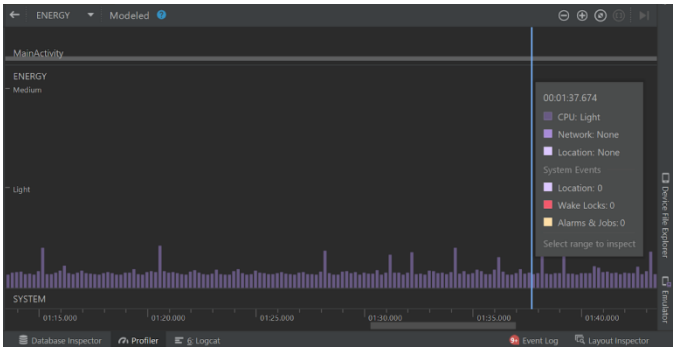


Fig 17- Energy usage of application

The figure 17 shows the overall load across the CPU, Networks and Location Services. It was observed that overall CPU load was low and no network or location services were used which makes this app power friendly and efficient for mobile devices. Similar results were seen across other 3 devices. As observed from the above statistics, the application was lightweight in terms of computation which resulted in real time response.

Across all 4 devices, the application was tested for 100 inputs i.e., 10 inputs of each of the 10 gestures. The inputs given were in real time with variable light conditions and across real life background. Each of the samples of hands differed in skin complexion. The test was done to check the accuracy of the gesture recognition system under real life circumstances.

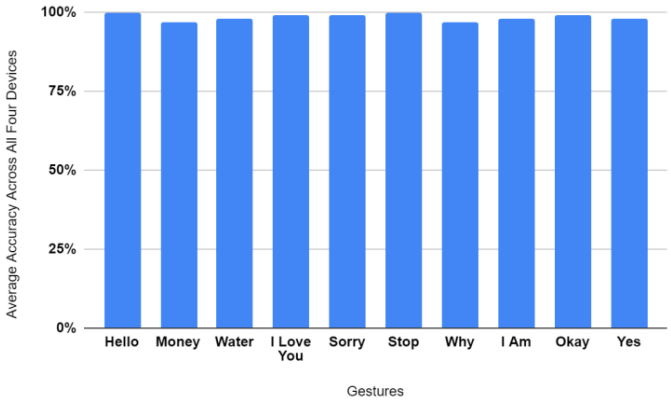


Fig 18- Average accuracy result of each static ASL gesture

The graph in Fig 23 shows the average accuracy obtained across all four devices for each of the 10 gestures. An overall accuracy of 98.5% was achieved. Also, insignificant or no performance drop was observed while translation of the recognized gesture.

5. CONCLUSION

The Gesture Recognition Application was thus implemented to detect 10 static American Sign Language Gestures performed by speech-impaired people. The recognized gestures are then converted to text format that are displayed on the top-center of the screen. These gestures can also be translated into three different regional languages namely Marathi, Hindi and Gujarati by selecting the language from the drop-down menu displayed on the bottom-center of the screen.

The application aims to help the society in terms of better communication aspects. The application enables people with no knowledge of the American Sign Language to understand the gestures being performed by someone who is trying to communicate using Hand Gestures. Beyond this, the scalability of the App ensures support to people around the world who are unable to communicate due to a lack of understanding of American Sign Language.

5.1. LIMITATIONS

The gestures are mainly divided into two categories: Static gestures and Dynamic Gestures. Dynamic movements include strokes, pre-strokes, postures, and stages, while static gestures only include poses and configurations. The dynamic gestures often include movement of body parts. Depending on the context of the gesture, it can also include emotions. Aside from the action phenomenon, the inclusion of feelings is the second dividing characteristic between static and dynamic gestures. Emotions are incorporated into the dynamic gestures. For example, to express 'The Mountain was big,' one will use only static gestures, while to express 'The Mountain was this big,' one would need to use arm movements that fall into the category of dynamic gestures.

There are a few gestures that include use of face for e.g. Mother is expressed using chin and Father is expressed using forehead. To recognize these gestures, facial detection is required along with its key points to determine the position of the hand. Facial key points are also required in order to determine the emotions of the person. As of now our android application has implemented a palm detection and hand tracking model which limits us to recognize hand gestures alone.

Our project thus limits us to recognize the static ASL hand gestures that are independent of other parts of the body.

5.2. FUTURE SCOPE

The future of the Hand Gesture Recognition App is being able to add support for dynamic gestures that are gestures which are performed over a function of time (a few seconds) which is more complex to detect as the system needs to be able to detect the start of a dynamic gesture vs the start of a static gesture. Support for gestures wherein the gestures are performed with a combination of hand movements, body pose and facial expressions. The App can be modified in the future to include a feature to enable Text to Speech, increasing the functionality of the App. Beyond these, support for additional multiple languages can be added to the app, perhaps with the help of Google API, enabling support for more than a hundred different languages.

Additionally, Using MediaPipe, the Hand Gesture Recognition App can be made cross platform, making deployment on iOS a possibility.

REFERENCES

[1] Kunal Kadam, Rucha Ganu, Ankita Bhosekar, Prof.S.D.Joshi, “American Sign Language Interpreter”, 2012 IEEE Fourth International Conference on Technology for Education, pp 157- 159.

[2] Mrs. Neela Harish , Dr. S. Poonguzhali, “Design and development of hand gesture recognition system for speech impaired people”, 2015 International Conference on Industrial Instrumentation and Control (ICIC) College of Engineering Pune, India. May 28-30, 2015, pp. 1129.

[3] Wei-Chieh Chuang, Wen-Jyi Hwang, Tsung-Ming Tai, De-Rong Huang, and Yun-Jie Jhang, “Continuous Finger Gesture Recognition Based on Flex Sensors”, Sensors 2019, vol. 19, pp. 3986.

[4] Pavlo Molchanov, Shalini Gupta, Kihwan Kim,Jan Kautz, "Hand Gesture Recognition with 3D Convolutional Neural Networks", NVIDIA, pp. 1.

[5] Dinh-Son Tran, Ngoc-Huynh Ho, Hyung-Jeong Yang, Eu-Tteum Baek , Soo-Hyung Kim and Gueesang Lee, "Real-Time Hand Gesture Spotting and Recognition Using RGB-D Camera and 3D Convolutional Neural Network", Applied Science 2020, vol. 10, pp. 722.

[6] Linchu Yang, Jian Chen, Weihang Zhu, "Dynamic Hand Gesture Recognition Based on a Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural Network", Sensors 2020, vol. 20, pp. 2106.

[7] Purva C. Badhe, Vaishali Kulkarni, "Indian Sign Language Translator Using Gesture Recognition Algorithm, 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), pp. 195 – 200.

[8] Wei Wang, Jing Pan, "Hand Segmentation using Skin colour and Background Information", 2012 International Conference on Machine Learning and Cybernetics, pp. 1487.

[9] Hsiang-Yueh. Lai, Han-Jheng. Lai, "Real-Time Dynamic Hand Gesture Recognition", 2014 International Symposium on Computer, Consumer and Control, pp. 658 - 661.

[10] Mohammed Abdulaziz Aide Al-qaness, Fangmin Li, "WiGeR: WiFi-Based Gesture Recognition System", ISPRS Int. J. Geo-Inf. 2016, vol. 5, 92, pp. 1- 17.