RASPBERRY PI BASED SIGN LANGUAGE INTERPRETER

ESS Project Report

By

Samruddhi Patil (211061032)

Flavia Saldanha (211021007)

Yash Bhavnani (211060035)

Sakshi Rathod (211061010)

Dr. R. D. Daruwala

Guide/Supervisor

Electrical Engineering Department

Veermata Jijabai Technological Institute

Mumbai 400 019

2023-24

Electronics in Service to Society

Certificate of Achievement

This is to certify that

Samruddhi Patil

Flavia Saldanha

Yash Bhavnani

Sakshi Rathod

has successfully completed the laboratory session in

Electronics in Service to Society Lab

For the Academic Year

2023-24

_____ _____

Lab Teacher Lab Teacher

Date: _____ Date: _____

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources.

We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission.

We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Samruddhi Patil
(211061032)

Flavia Saldanha
(211021007)

Yash Bhavnani
(211060035)

Sakshi Rathod
(211061010)

Date: Place: Matunga

ABSTRACT

This project report unveils the pioneering development of a Raspberry Pi-based Sign Language Interpreter, a breakthrough in bridging communication barriers for the hearing impaired. Leveraging cutting-edge technology, including the Raspberry Pi 4B, Logitech web camera, Adafruit LCD display, and jack earphone, alongside Bookworm OS, the system showcases the fusion of hardware and software expertise.

Central to this innovation is a meticulously crafted machine learning model, meticulously programmed in Python within Jupyter Notebook. This model enables real-time gesture recognition and interpretation, facilitating seamless communication between sign language users and the broader community.

The Logitech web camera captures and identifies gestures, while the Adafruit LCD display swiftly communicates detected signs. The incorporation of HDMI cable ensures versatile display options, enhancing user accessibility.

User engagement and empowerment are paramount in the design ethos. The incorporation of a jack earphone provides text-to-speech functionality, enabling immediate auditory feedback for detected signs. This holistic approach prioritizes inclusivity and usability, positioning the interpreter as a catalyst for fostering inclusive environments.

In summary, this project epitomizes the transformative potential of electronics in facilitating communication accessibility. By seamlessly merging hardware components and software algorithms, it paves the way for enhanced interaction and understanding within diverse communities.

Keywords: Raspberry Pi, Sign Language Interpreter, Machine Learning, Accessibility, Communication, Inclusivity

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The electronic subsystem within the Raspberry Pi-based Sign Language Interpreter represents the neural network orchestrating its functionality, efficacy, and user engagement. As the project's cornerstone, this subsystem intricately blends high and low-voltage circuits to govern power management, real-time gesture recognition, and seamless communication between components. Its significance lies in propelling the interpreter to transcend communication barriers, facilitating fluid interactions for individuals with hearing impairments. This chapter embarks on a comprehensive exploration of the architecture, functionalities, and impact of the electronic subsystem, illuminating its instrumental role in sculpting the effectiveness and accessibility of the interpreter. From sophisticated machine learning algorithms to intuitive user interfaces, the electronic subsystem stands as the bedrock of this transformative endeavor, heralding a new era of inclusivity and communication empowerment.

1.1 OBJECTIVES

The primary objective of our project is to delve into the intricacies of the electronic subsystems within the Raspberry Pi-based Sign Language Interpreter. This involves a detailed examination of both hardware and software components, with a focus on understanding their interplay in facilitating real-time gesture recognition and interpretation.

Furthermore, we aim to translate this theoretical understanding into practical application by constructing a fully functional Sign Language Interpreter prototype. This hands-on endeavor will encompass the selection and integration of hardware components such as the Raspberry Pi 4B, Logitech web camera, Adafruit LCD display, and jack earphone, along with the implementation of Bookworm OS.

In addition to developing the hardware infrastructure, our project seeks to enhance the interpreter's functionality by fine-tuning the machine learning model within a Jupyter Notebook environment. This iterative process will involve training the model to accurately recognize and interpret a wide range of sign language gestures, thereby fostering seamless communication between sign language users and the wider community.

Moreover, we aim to prioritize user accessibility and engagement by incorporating features such as text-to-speech output via the jack earphone, ensuring immediate auditory feedback for detected signs.

In summary, our objectives encompass a comprehensive exploration of the electronic subsystems underlying the Sign Language Interpreter, culminating in the development of a practical and user-centric communication tool for individuals with hearing impairments.

CHAPTER 2: LITERATURE REVIEW

2.1 LIBRARIES

2.1.1 OPENCV (Open Source Computer Vision Library)

OpenCV is a versatile and widely-used open-source computer vision library that provides various functionalities for image and video processing tasks. In your project, OpenCV plays a crucial role in several aspects:

1. Webcam Integration : OpenCV enables the integration of the webcam to capture live video streams. By utilizing OpenCV's `VideoCapture` module, you can access the webcam's feed and process each frame in real-time. This functionality is essential for capturing hand gestures and feeding them into the machine learning model for classification.
2. Image Preprocessing : Before feeding the captured frames into the machine learning model, preprocessing steps may be necessary to enhance the quality of the images and extract relevant features. OpenCV provides a wide range of image processing functions, such as resizing, cropping, filtering, and thresholding, which can be applied to the captured frames to improve their suitability for gesture recognition.
3. Hand Detection and Tracking : OpenCV can be used in conjunction with other libraries or algorithms to detect and track hand gestures within the captured frames. Techniques such as contour detection, blob analysis, or template matching can be implemented using OpenCV to identify the hand region and track its movement across frames. This process is crucial for isolating the hand gestures from the background and extracting features for classification.
4. Visualization : OpenCV facilitates real-time visualization of the captured frames and the results of gesture recognition. You can use OpenCV's drawing functions to overlay graphical elements, such as bounding boxes or lines connecting key points, onto the frames to visualize the detected hand gestures. This allows for intuitive feedback to the user and aids in debugging and optimizing the gesture recognition algorithm.
5. Integration with Machine Learning Model : Once the hand gestures are detected and preprocessed using OpenCV, the resulting images or feature vectors can be fed into the machine learning model for classification. OpenCV provides utilities for converting images to numerical arrays compatible with

machine learning frameworks like TensorFlow, enabling seamless integration between the computer vision and machine learning components of the project.

By leveraging the capabilities of OpenCV, our project can achieve real-time hand gesture recognition and interpretation, paving the way for the development of an effective and user-friendly sign language interpreter. The integration of OpenCV with other libraries and algorithms ensures robustness, efficiency, and accuracy in recognizing and interpreting a wide range of sign language gestures.

2.1.2 NUMPY (Numerical Python)

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. In your project, NumPy plays several critical roles:

1. Data Representation : NumPy's primary data structure is the `ndarray` (n-dimensional array), which allows for efficient storage and manipulation of numerical data. In our project, NumPy arrays can be used to represent images captured from the webcam for gesture recognition. Each frame captured by OpenCV can be converted into a NumPy array, allowing for easy manipulation and processing.
2. Data Preprocessing : NumPy provides a wide range of functions for data preprocessing, including normalization, scaling, and reshaping. These preprocessing steps may be necessary to ensure that the input data fed into the machine learning model is in a suitable format and range. For example, you can use NumPy to normalize pixel values in the image arrays to a specific range or reshape the arrays to match the input dimensions expected by the model.
3. Array Operations : NumPy offers a rich set of mathematical functions and operations for array manipulation. These operations include element-wise arithmetic operations, matrix multiplication, and statistical functions. In our project, NumPy can be used to perform mathematical computations on the image arrays, such as calculating mean or median pixel values, extracting image features, or performing transformations like Fourier transforms.
4. Integration with Machine Learning : NumPy arrays seamlessly integrate with popular machine learning libraries like TensorFlow and scikit-learn. After preprocessing the image data using NumPy, you can convert the arrays into tensors or feature vectors compatible with the machine learning model. This

integration enables the efficient processing and analysis of image data within the machine learning pipeline, ultimately leading to accurate gesture recognition.

5. Efficiency and Performance : NumPy's underlying implementation is optimized for performance, with many of its operations implemented in C or Fortran for speed. This efficiency is crucial for handling large volumes of image data and performing complex computations in real-time. By leveraging NumPy, our project can achieve high performance and responsiveness, even when processing streaming video data from the webcam.

By leveraging the capabilities of NumPy, your project can efficiently handle and preprocess image data, perform complex mathematical computations, and seamlessly integrate with machine learning frameworks. This enables the development of a robust and efficient sign language interpreter capable of accurately recognizing and interpreting hand gestures in real-time.

2.1.3 MATPLOTLIB

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a wide range of functions for creating various types of plots, charts, and graphs, making it suitable for visualizing data and results in your project. Here's how Matplotlib contributes to your sign language interpreter project

1. Visualization of Hand Gestures : Matplotlib can be used to visualize the hand gesture recognition process in real-time. For example, you can create a live plot or graph that displays the position of key points on the hand detected by OpenCV or Mediapipe. This visualization aids in understanding the performance of the gesture recognition algorithm and provides feedback to users.
2. Connecting Nodes with Lines : In certain applications, it may be useful to connect key points on the hand detected by the computer vision algorithms with lines. Matplotlib provides functions for drawing lines and shapes on plots, allowing you to connect nodes representing key points such as fingertips, palm center, or wrist. This visual representation can help users understand the spatial relationships between different parts of the hand and the gestures being performed.
3. Displaying Image Data : Matplotlib can also be used to display images, which is useful for showing the raw frames captured by the webcam or the

preprocessed images before they are fed into the machine learning model. You can create subplots or interactive figures that display the original image alongside additional information such as detected hand regions or bounding boxes.

4. Analyzing Model Performance : Matplotlib is valuable for analyzing the performance of the machine learning model trained to recognize hand gestures. You can create plots or charts that visualize metrics such as accuracy, loss, or confusion matrices, allowing you to assess the model's performance on validation or test data. These visualizations help identify areas for improvement and guide the optimization of the model.
5. User Interface Elements : Matplotlib can be used to create interactive user interface elements such as buttons, sliders, or dropdown menus. While these may not be directly related to gesture recognition, they can enhance the user experience by allowing users to interact with the application, adjust parameters, or switch between different modes of operation.

By leveraging the capabilities of Matplotlib, your project can create informative and visually appealing visualizations that aid in understanding the hand gesture recognition process, analyzing model performance, and enhancing the user experience. These visualizations contribute to the overall effectiveness and usability of your sign language interpreter, making it a valuable tool for individuals with hearing impairments.

2.1.4 MEDIAPIPE

Mediapipe is an open-source library developed by Google for building machine learning pipelines for various perception tasks, including hand tracking, pose estimation, and object detection. In your project, Mediapipe serves several critical functions:

1. Hand Detection and Tracking : One of the key functionalities of Mediapipe is its ability to detect and track hands in real-time video streams. Using pre-trained machine learning models and algorithms, Mediapipe analyzes each frame captured by the webcam to identify the presence and location of hands. This capability is essential for isolating the hand gestures from the background and extracting relevant features for classification.
2. Feature Extraction : Once hands are detected and localized within the frames, Mediapipe extracts a variety of features from the hand region, such as key points (landmarks) representing the fingertips, palm center, and wrist. These

landmarks provide valuable spatial information about the hand's pose and configuration, which can be used as input features for the machine learning model responsible for gesture recognition.

3. Pose Estimation : In addition to hand tracking, Mediapipe can perform pose estimation to infer the overall pose of the human body from video data. While this functionality may not be directly utilized in your project, it demonstrates the versatility of Mediapipe and its potential for future extensions or enhancements of the sign language interpreter.
4. Real-time Processing : Mediapipe is designed for real-time processing of video data, making it well-suited for applications requiring low latency and high frame rates. By leveraging optimized algorithms and hardware acceleration, Mediapipe can efficiently analyze streaming video from the webcam and provide rapid feedback on hand gestures, enabling smooth and responsive interaction with the sign language interpreter.
5. Integration with TensorFlow : Mediapipe seamlessly integrates with TensorFlow, Google's deep learning framework, allowing for easy integration of hand tracking and gesture recognition capabilities into machine learning models built using TensorFlow. This integration simplifies the development workflow and enables interoperability between different components of the sign language interpreter.

By leveraging the capabilities of Mediapipe, our project can achieve robust and efficient hand gesture recognition, enabling seamless communication for individuals with hearing impairments. The integration of Mediapipe with other libraries and algorithms, such as OpenCV for video capture and TensorFlow for machine learning, forms a comprehensive pipeline for real-time sign language interpretation.

2.1.5 TENSORFLOW

TensorFlow is an open-source machine learning framework developed by Google for building and training deep learning models. It provides a comprehensive ecosystem of tools and libraries for developing various machine learning applications, including image recognition, natural language processing, and gesture recognition. In our project, TensorFlow serves several critical functions:

1. Model Development : TensorFlow allows you to design, build, and train machine learning models for hand gesture recognition. You can define the architecture of the neural network, including the number and type of layers, activation functions, and optimization algorithms. TensorFlow provides a

high-level API, such as Keras, which simplifies the process of model development and experimentation.

2. Training : Once the model architecture is defined, TensorFlow facilitates the training process using labeled data. You can use TensorFlow's built-in functions for data loading, preprocessing, and augmentation to prepare the training dataset. TensorFlow also supports distributed training across multiple GPUs or TPUs, enabling faster convergence and scalability for large datasets.
3. Transfer Learning : TensorFlow supports transfer learning, a technique where pre-trained models are fine-tuned on a specific task using a smaller, domain-specific dataset. This approach can significantly reduce the amount of labeled data required for training and accelerate the development process. In our project, you may leverage pre-trained models, such as convolutional neural networks (CNNs) trained on image classification tasks, and fine-tune them for hand gesture recognition.
4. Inference : Once the model is trained, TensorFlow allows you to perform inference, i.e., make predictions on new, unseen data. You can deploy the trained model within the sign language interpreter application to recognize hand gestures in real-time video streams from the webcam. TensorFlow provides optimized runtime environments, such as TensorFlow Lite for mobile and embedded devices, which ensure efficient inference on resource-constrained platforms like Raspberry Pi.
5. Optimization and Deployment : TensorFlow offers tools for model optimization and deployment, allowing you to optimize the size and performance of the trained model for deployment on edge devices. Techniques such as quantization, pruning, and model compression can be applied to reduce the model's memory footprint and computational overhead while maintaining accuracy. TensorFlow also supports integration with TensorFlow Serving or TensorFlow Lite for deploying models in production environments.

By leveraging the capabilities of TensorFlow, your project can develop and deploy state-of-the-art machine learning models for hand gesture recognition, enabling real-time sign language interpretation on devices like Raspberry Pi. The integration of TensorFlow with other libraries and tools, such as OpenCV for image preprocessing and Mediapipe for hand tracking, forms a comprehensive pipeline for building an effective and efficient sign language interpreter.

2.1.6 SKLEARN

scikit-learn is a popular machine learning library in Python that provides simple and efficient tools for data mining and data analysis. While scikit-learn is primarily focused on traditional machine learning algorithms rather than deep learning, it offers a wide range of functionalities that can complement the capabilities of TensorFlow and other libraries in our project. Here's how scikit-learn contributes to your sign language interpreter project :

1. Data Preprocessing : scikit-learn provides a comprehensive set of functions for data preprocessing, including feature scaling, normalization, and dimensionality reduction. These preprocessing techniques can be applied to the feature vectors extracted from hand gesture images before feeding them into the machine learning model. For example, you can use scikit-learn's `StandardScaler` to scale feature values to a standard range or `PCA` for dimensionality reduction if necessary.
2. Model Selection and Evaluation : scikit-learn offers tools for model selection and evaluation, allowing you to compare the performance of different machine learning algorithms and hyperparameter settings. Techniques such as cross-validation, grid search, and hyperparameter tuning can be employed to optimize the performance of the gesture recognition model. scikit-learn also provides metrics for evaluating classification models, such as accuracy, precision, recall, and F1-score, which can help assess the model's performance on test data.
3. Auxiliary Models and Pipelines : scikit-learn supports the construction of machine learning pipelines, which enable the chaining together of multiple preprocessing steps and estimators into a single workflow. For example, you can create a pipeline that includes feature extraction, preprocessing, and classification stages, allowing for streamlined model training and evaluation. Additionally, scikit-learn offers auxiliary models for tasks such as feature selection, outlier detection, and clustering, which can be integrated into the pipeline as necessary.
4. Interoperability with TensorFlow : While scikit-learn and TensorFlow are separate libraries with different focuses, they can be used together seamlessly within your project. For instance, you can use scikit-learn for data preprocessing, model selection, and evaluation, and then integrate the selected model into the TensorFlow-based pipeline for training and inference. This

interoperability allows you to leverage the strengths of both libraries and build a robust machine learning system for hand gesture recognition.

5. Ease of Use and Documentation : scikit-learn is known for its user-friendly interface and extensive documentation, making it accessible to both beginners and experienced machine learning practitioners. The library provides clear and concise documentation, along with numerous examples and tutorials, which can help you quickly get started with building and evaluating machine learning models for your sign language interpreter project.

By leveraging the capabilities of scikit-learn, your project can benefit from advanced data preprocessing techniques, model selection and evaluation tools, and streamlined machine learning workflows. The integration of scikit-learn with other libraries such as TensorFlow and OpenCV enhances the overall functionality and effectiveness of your sign language interpreter, enabling accurate and efficient recognition of hand gestures for communication accessibility.

2.1.7 PYTTSX3

pyttsx3 is a text-to-speech conversion library in Python that supports multiple TTS engines, including SAPI5 on Windows and NSSpeechSynthesizer on macOS. It provides a simple and straightforward interface for converting text strings into spoken audio, allowing your sign language interpreter project to provide auditory feedback for detected hand gestures. Here's how pyttsx3 contributes to your project :

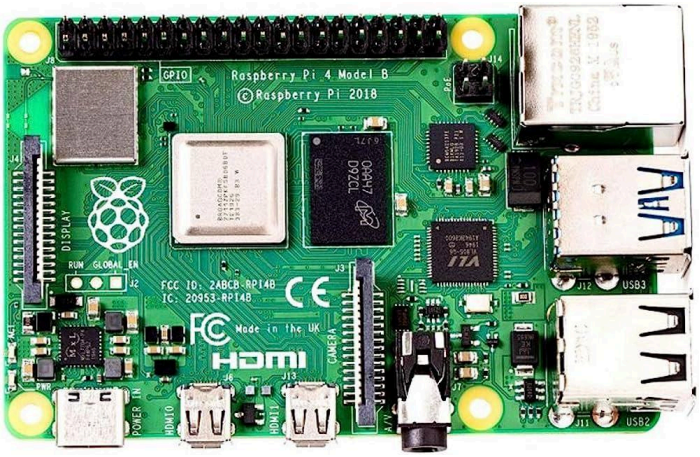
1. Text-to-Speech Conversion : The primary function of pyttsx3 is to convert text strings into audible speech. In your project, pyttsx3 can be used to generate spoken output corresponding to the recognized hand gestures. For example, when a particular sign language gesture is detected and interpreted, the corresponding text representation of the gesture can be converted into speech using pyttsx3 and played through the jack earphone connected to the Raspberry Pi.
2. Customization and Configuration : pyttsx3 offers various configuration options and parameters that allow you to customize the speech synthesis process according to your preferences. You can adjust parameters such as speech rate, volume, and voice selection to tailor the synthesized speech to the specific requirements of your sign language interpreter application. Additionally, pyttsx3 supports the use of custom voice files and pronunciation dictionaries, enabling further customization of the synthesized speech output.

3. Platform Compatibility : pyttsx3 is platform-independent and can be used on different operating systems, including Windows, macOS, and Linux. This cross-platform compatibility ensures that your sign language interpreter application can provide consistent auditory feedback across different environments and devices. Whether running on a Raspberry Pi with Bookworm OS or a Windows-based computer, pyttsx3 enables seamless integration of text-to-speech functionality into your project.
4. Ease of Use : pyttsx3 features a simple and intuitive API that makes it easy to integrate text-to-speech functionality into your Python scripts. The library provides functions for initiating the TTS engine, setting parameters, and converting text strings to speech with a single method call. This ease of use reduces development time and effort, allowing you to focus on the core functionality of your sign language interpreter application.
5. Real-time Feedback : By leveraging pyttsx3 in your project, you can provide real-time auditory feedback for detected hand gestures, enhancing the user experience and accessibility of the sign language interpreter. As soon as a gesture is recognized and interpreted, the corresponding spoken output is generated and played through the connected jack earphone, providing immediate feedback to the user.

By incorporating pyttsx3 into your sign language interpreter project, you can enhance its accessibility and usability by providing real-time auditory feedback for detected hand gestures. The simplicity, customization options, and platform compatibility of pyttsx3 make it an ideal choice for integrating text-to-speech functionality into your Python-based application.

2.2 COMPONENTS

2.2.1 RASPBERRY-PI 4B



Specifications:

- Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory: 1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC Connectivity: 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
- GPIO: Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
- Video & sound: 2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
- Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
- SD card support: Micro SD card slot for loading operating system and data storage Input power: 5V DC via USB-C connector (minimum 3A1) 5V DC via GPIO header (minimum 3A1) Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
- Environment: Operating temperature 0–50ºC

2.2.2 USB WEB-CAMERA :

Specifications:

- Model Name: C270-1.5m cable
- Display size: 2 inches
- Video Resolution: 720p
- Shooting modes: Automatic
- Lens type: Zoom

2.2.3 I2C LCD DISPLAY

Specifications:

- Display capacity: 16 character x 2 row
- Display color: Blue backlit
- Character size: 2.95 mm wide x 4.35 mm high
- Character pixels: 5 W x 7 H
- Voltage requirements: 5 VDC +/- 0.5V

- Current requirements: 2 mA @ 5 VDC
- Connection: 4-pin male header with 0.1″: spacing
- Communication: I2C
- Overall dimensions: 3.15 x 1.42 x 0.51 in (80 x 36 x 13 mm)
- Operating temperature range: 32 to +131 °F (0 to +55 °C)

2.2.4 WIRED EARPHONE

The purpose of integrating wired earphones into the sign language interpreter project on Raspberry Pi is to ensure real-time, private, and seamless communication between the interpreter and the user. Wired earphones offer reliability in audio transmission, eliminating potential latency issues commonly associated with wireless connections. By providing a direct, secure audio channel, wired earphones enhance the interpreter's ability to accurately interpret and convey sign language gestures, facilitating effective communication for individuals with hearing impairments.

CHAPTER 3: MODEL DEVELOPMENT

3.1 OVERVIEW

The project involves setting up a Raspberry Pi 4B to create a sign language recognition system. It includes three stages: Raspberry Pi Setup, Modelling, and User Interface. In Raspberry Pi Setup, the Pi is initialized, required libraries are installed, and the image storage path is specified. Modelling involves dataset creation for Indian sign language gestures, designing a model structure utilizing pose-based deep learning with LSTM, and achieving 85-90% accuracy on test data. Testing includes importing Mediapipe libraries for real-time gesture recognition, integrating the recognition model, and interfacing an LCD display for visual text output. User Interface tasks include installing I2C LCD display libraries, interfacing the display with Raspberry Pi, setting up text-to-speech functionality, enabling

users to hear interpreted signs via wired headphones. Hardware components include Raspberry Pi 4 Model B, USB webcam, I2C LCD display, and wired headphones.

3.2 SCHEMATIC

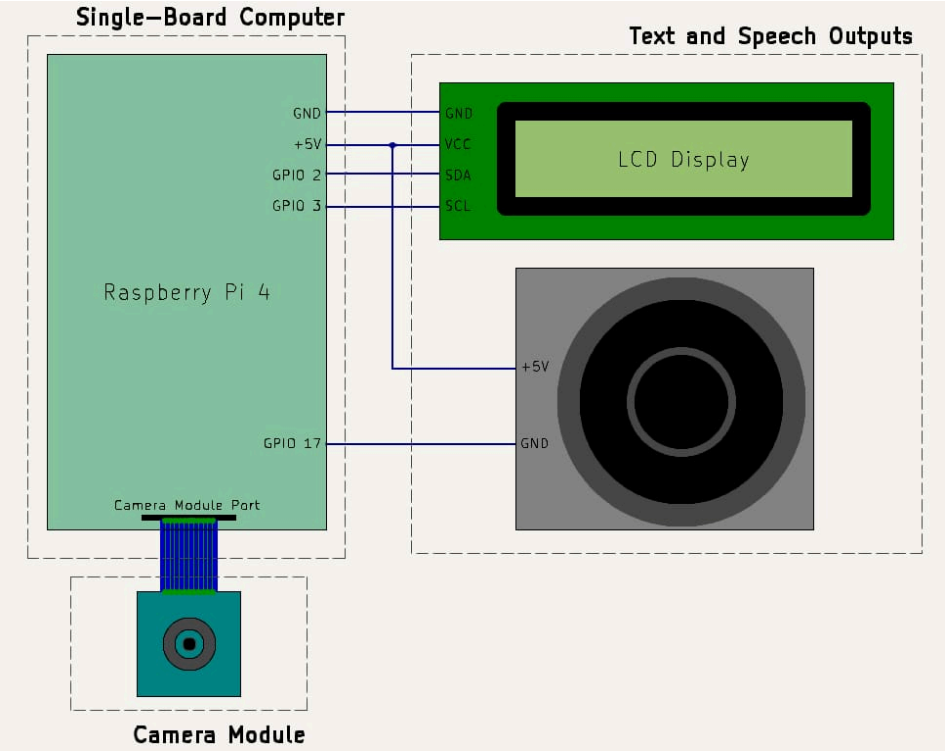


Figure 3.1 : Old Layout

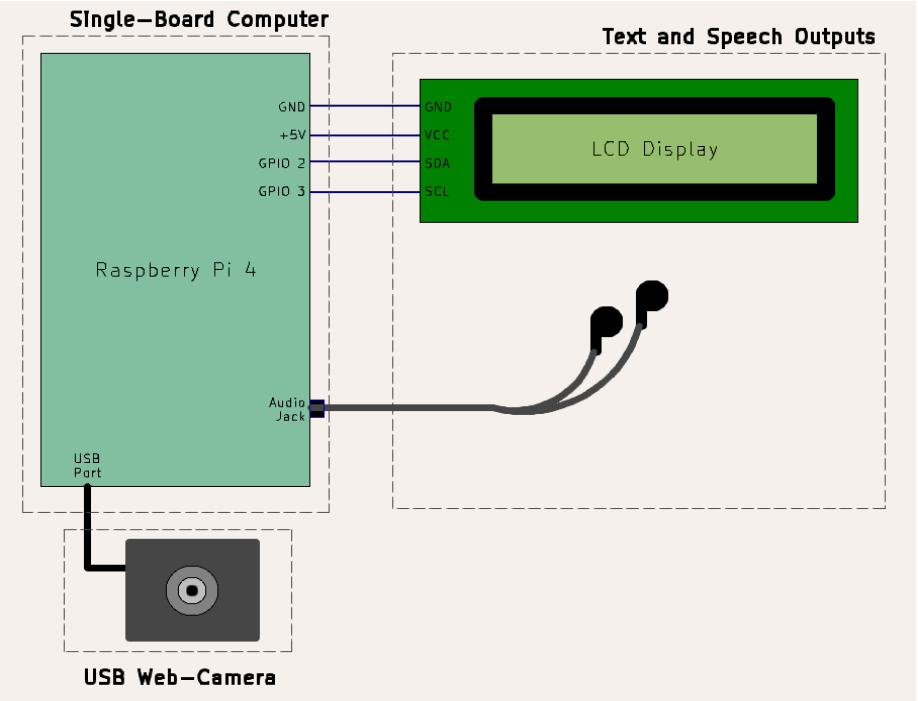
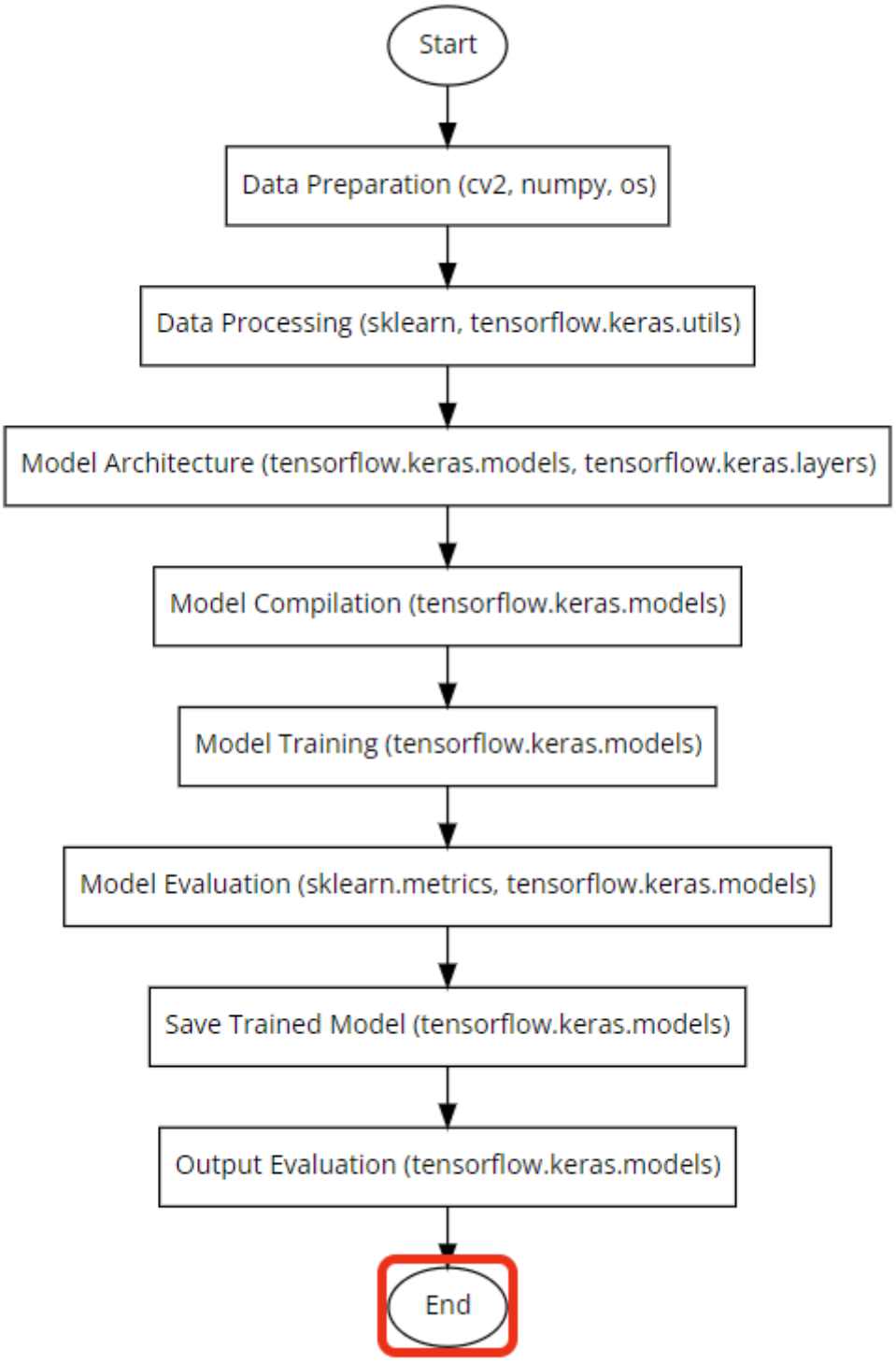


Figure 3.2 : New Layout

3.3 MODEL STRUCTURE



3.4 CODE

3.4.1 PYTHON CODE FOR MODEL DEVELOPMENT [1]

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
import tensorflow as tf
DATA_PATH = os.path.join("/home/raspberrypi/Downloads/Data")
actions = np.array(["HOW ARE YOU","THANKS","WORK","PLEASE","SORRY",
"HELLO", "AFTERNOON", "DEAF", "GOOD", "BAD"])
no_sequences = 40
sequence_length = 20
label_map = {label:num for num, label in enumerate(actions)}
sequences, labels = [], []
for action in actions :
 for sequence in range(no_sequences):
      window = []
      for frame_num in range(sequence_length):
                        res = np.load(os.path.join(DATA_PATH, action, str(sequence),
"{}.npy".format(frame_num)))
         window.append(res)
      sequences.append(window)
      labels.append(label_map[action])
label_map
np.array(sequences).shape
X = np.array(sequences)
y = to_categorical(labels).astype(int)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 30)
y_test.shape
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# Designing Neural Network
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(20,258)))
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'))
# Compiling
model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])
# Fitting
model.fit(X_train, y_train, epochs=20)
# Summary
model.summary()
res = model.predict(X_test)
# Saving the trained Model
model.save('isl6.h5')
yhat = model.predict(X_test)
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
multilabel_confusion_matrix(ytrue, yhat)
accuracy_score(ytrue, yhat)
from sklearn.metrics import precision_score, recall_score
precision_score(ytrue, yhat,average='macro')
recall_score(ytrue, yhat,average='macro')
loss, accuracy = model.evaluate(X_test, y_test)
print('Test Loss: {}, Test Accuracy: {}'.format(loss, accuracy))
```

3.4.2 PYTHON CODE FOR TESTING THE MODEL [1]

```
import cv2
import os
import time
import numpy as np
import mediapipe as mp
from IPython.display import clear_output
from tensorflow import keras
from matplotlib import pyplot as plt
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    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)
    return image, results
def draw_landmarks(image, results) :
     mp_drawing.draw_landmarks(image,results.pose_landmarks,    mp_holistic.POSE_-
CONNECTIONS)       # Draw pose connections
   mp_drawing.draw_landmarks(image,results.left_hand_landmarks,    mp_holistic.HA-
ND_CONNECTIONS)  # Draw left hand connections
  mp_drawing.draw_landmarks(image,  results.right_hand_landmarks,  mp_holistic.HA-
ND_CONNECTIONS) # Draw right hand connections
def draw_styled_landmarks(image, results) :
mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CO
NNECTIONS,
   mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
   mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2))
    # Draw left hand connections
   mp_drawing.draw_landmarks(image,   results.left_hand_landmarks,   mp_holistic.HA
ND_CONNECTIONS,
    mp_drawing.DrawingSpec(color=(0,255,127), thickness=2, circle_radius=4),
    mp_drawing.DrawingSpec(color=(0,255,127), thickness=2, circle_radius=2)
```

```
    # Draw right hand connections
 mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.H
AND_CONNECTIONS,
   mp_drawing.DrawingSpec(color=(0,255,127), thickness=2, circle_radius=4),
   mp_drawing.DrawingSpec(color=(0,255,127), thickness=2, circle_radius=2))
# Calling the Trained Model
model=keras.models.load_model('isl6.h5')
colors                                                                    =
[(245,23,16),(245,23,16),(245,23,16),(245,23,16),(245,23,16),(245,23,16),(245,23,16),(
245,23,16),(245,23,16),(245,23,16)]
def prob_viz(res, actions, input_frame, colors):
   output_frame = input_frame.copy()
   for num, prob in enumerate(res):
               cv2.rectangle(output_frame,  (0,60+num*40),(int(prob*100),90+num*40),
colors[num], -1)
               cv2.putText(output_frame,        actions[num],        (0,        85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255), 2, cv2.LINE_AA)

   return output_frame
actions = np.array(["HOW ARE YOU","THANKS","WORK","PLEASE","SORRY",
"HELLO", "AFTERNOON", "DEAF", "GOOD", "BAD"])
import RPi_I2C_driver
mylcd = RPi_I2C_driver.lcd()
import pyttsx3
engine = pyttsx3.init(driverName = 'espeak')
# 1. New detection variables
sequence = []
sentence = []
threshold = 0.7
cap = cv2.VideoCapture(0)
# Set mediapipe model
with                                mp_holistic.Holistic(min_detection_confidence=0.7,
min_tracking_confidence=0.7) as holistic:
   while cap.isOpened():
      # Read feed
```

```
ret, frame = cap.read()

# Make detections
image, results = mediapipe_detection(frame, holistic)
# Draw landmarks
draw_styled_landmarks(image, results)
# 2. Prediction logic
keypoints = extract_keypoints(results)
sequence.insert(0, keypoints)
sequence = sequence[:20]
if len(sequence) == 20:
    res = model.predict(np.expand_dims(sequence, axis=0))[0]
    clear_output(wait=True)
    mylcd.lcd_clear()
    # Check if the output probability is greater than 0.95
    if np.max(res) > 0.95:
        print(actions[np.argmax(res)])
        mylcd.lcd_display_string(actions[np.argmax(res)], 1)
        engine.say(actions[np.argmax(res)])
        engine.runAndWait()
# 3. Viz logic
    if np.max(res) > threshold:
        if len(sentence) > 0:
            if actions[np.argmax(res)] != sentence[-1]:
                sentence.append(actions[np.argmax(res)])
        else:
            sentence.append(actions[np.argmax(res)])
    if len(sentence) > 5:
        sentence = sentence[-5:]
    # Viz probabilities
    image = prob_viz(res, actions, image, colors)
# Show to screen
cv2.imshow('Prediction', image)
# Break gracefully
if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
            break
        cap.release()
        cv2.destroyAllWindows()
    mylcd.lcd_clear()
```

CHAPTER 4: RESULT AND DISCUSSION

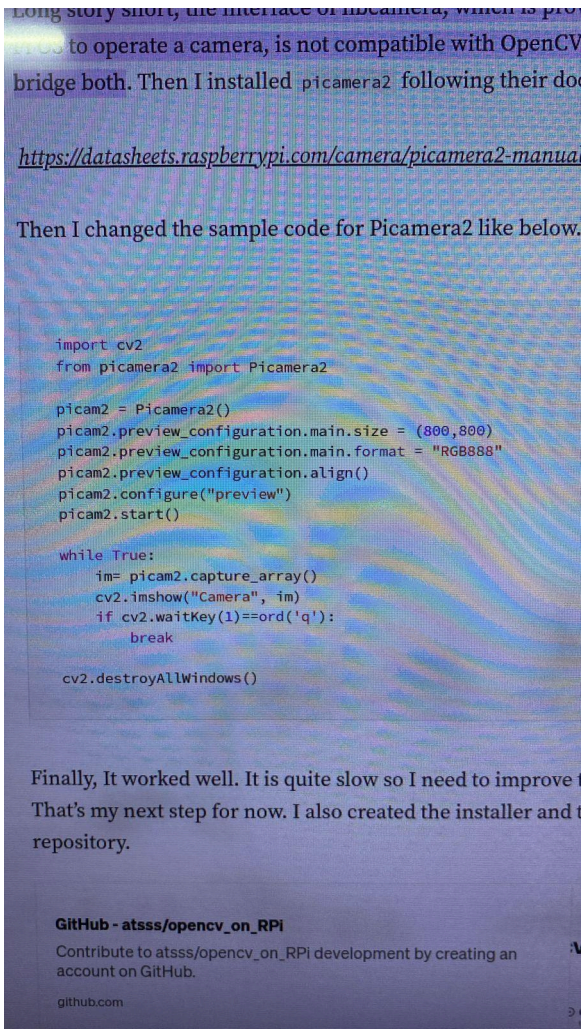
4.1 HARDWARE SETUP

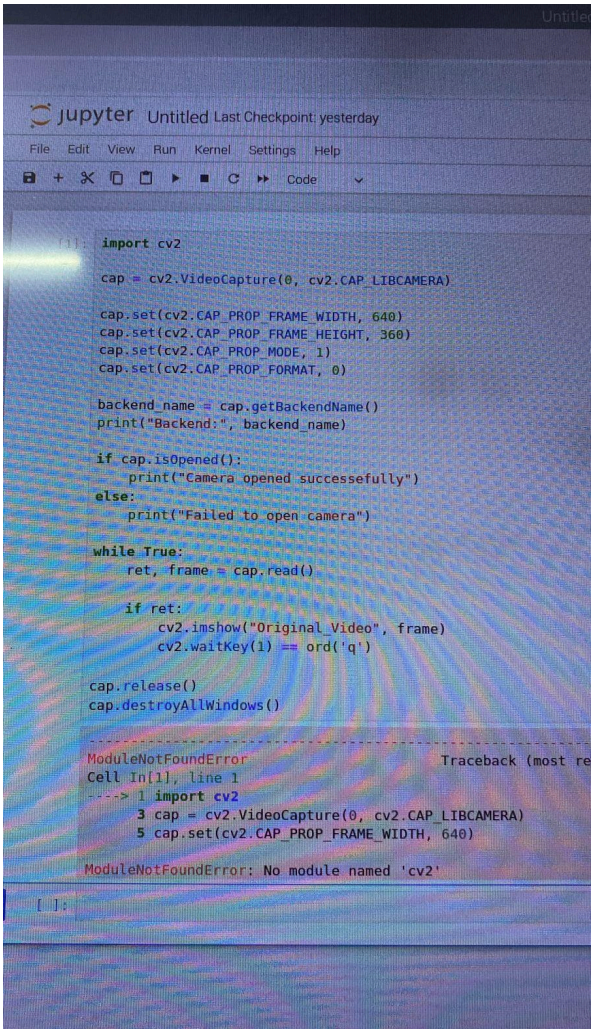
The hardware setup of a Raspberry Pi 4B-based Sign Language Interpreter involves assembling the necessary components and connecting them to the Raspberry Pi. Here's a detailed description of the hardware setup:

1. Raspberry Pi 4B: The Raspberry Pi 4B serves as the main processing unit for the Sign Language Interpreter. It provides the computing power to run the interpreter software and interface with the external hardware components.
2. I2C LCD Display: The I2C LCD display is used to show text or graphical output from the interpreter software. It communicates with the Raspberry Pi using the I2C protocol, which allows for easy connection and data transfer.
3. Wired Headphones: Wired headphones are used to provide audio output from the interpreter software. They connect to the audio output jack of the Raspberry Pi and allow the user to hear spoken interpretations of sign language gestures.

4. USB Webcam: A USB webcam is used to capture video input of sign language gestures. The webcam connects to one of the USB ports on the Raspberry Pi and provides a live video feed to the interpreter software.
5. MicroSD Card: A microSD card is used to store the operating system (such as Raspberry Pi OS) and the interpreter software. It is inserted into the microSD card slot on the Raspberry Pi.
6. Power Supply: A suitable power supply is required to provide power to the Raspberry Pi and the connected hardware components. The power supply should meet the power requirements of the Raspberry Pi and the other components.
7. HDMI Cable (optional): An HDMI cable can be used to connect a monitor to the Raspberry Pi for debugging and testing purposes. It is optional and not required for the basic functionality of the Sign Language Interpreter.

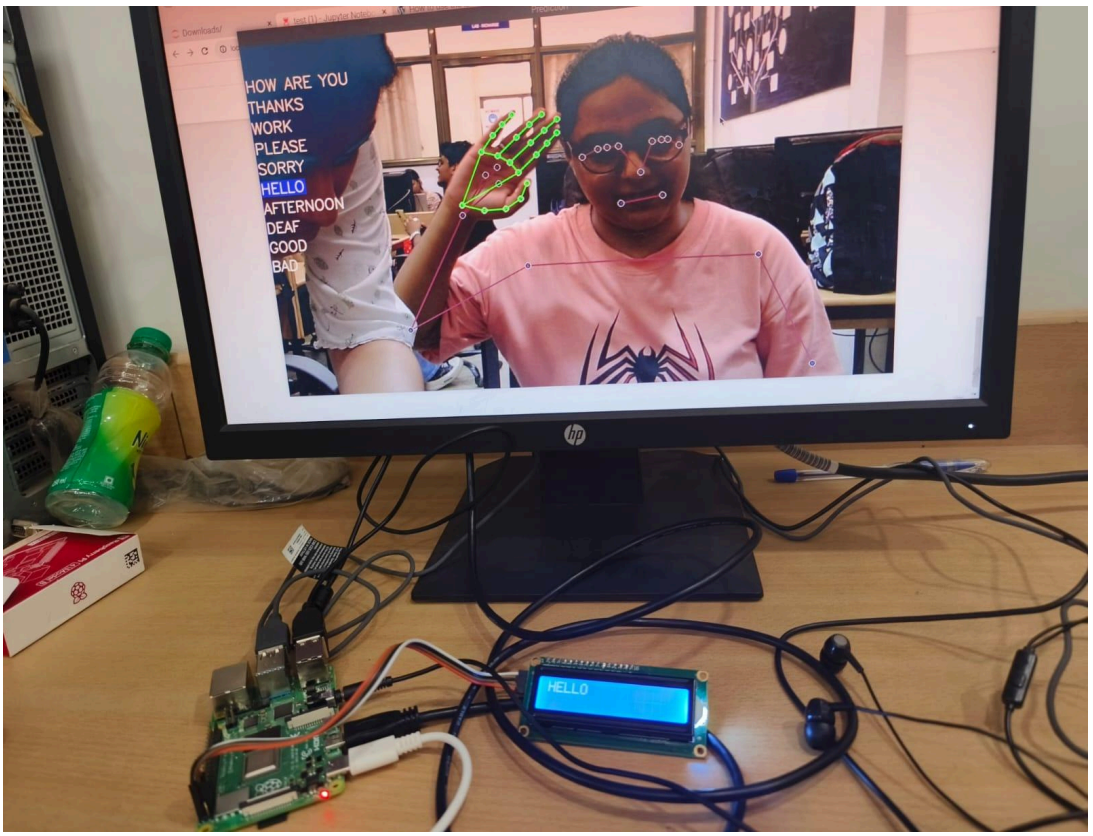
4.2 ERRORS ENCOUNTERED

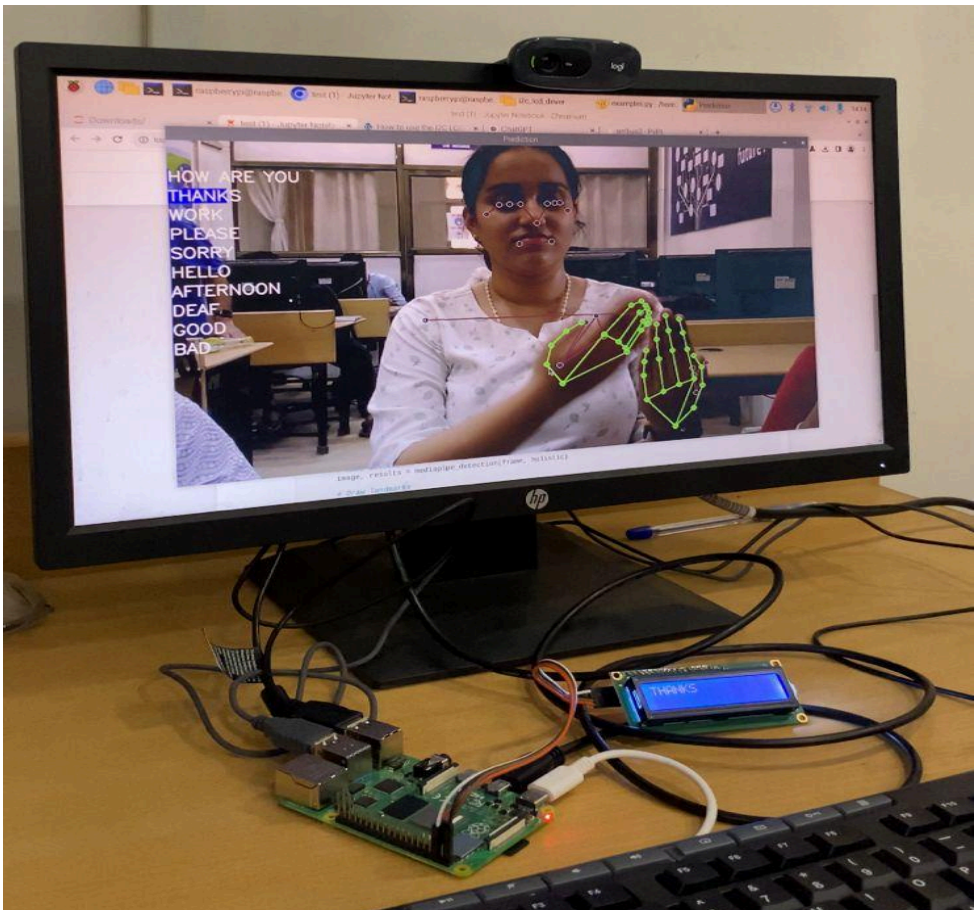




We encountered a lot of problems during the installation of OpenCV library integrated with Libcamera library, one of them was a GPU error, which we resolved by installing some other recommended packages. Even though our camera was working after all the troubleshooting, it didn't work with our code which involved the usage of Mediapipe library for pose estimation, we decided to switch to a lower version of Raspberry Pi OS named Bullseye, but that didn't work either because there were issues with installation of OpenCV library integrated with Libcamera library, so we decided to use a web camera instead and the code ran perfectly with no errors. We encountered no issues in interfacing the LCD Display with Raspberry Pi and Text-to-Speech conversion.

4.3 TESTING RESULTS





CHAPTER 5: CONCLUSION

The Raspberry Pi 4B-based Sign Language Interpreter project demonstrates the potential of using affordable and accessible hardware to create a valuable tool for the hearing and visual impaired community. By leveraging the capabilities of the Raspberry Pi 4B along with an I2C LCD display, wired headphones, and a USB webcam, the project provides a platform for real-time interpretation of sign language gestures.

The hardware setup of the project involves connecting the components to the Raspberry Pi 4B and configuring them to work together seamlessly. The I2C LCD display provides visual output, the wired headphones deliver audio output, and the USB webcam captures video input of sign language gestures. The Raspberry Pi 4B serves as the main processing unit, running the interpreter software that recognizes and interprets the gestures.

Through this project, users can communicate with the Sign Language Interpreter by performing sign language gestures in front of the webcam. The interpreter software analyzes

the gestures using machine learning techniques and provides corresponding text or audio output on the LCD display and headphones, enabling communication with hearing individuals.

The project highlights the importance of accessibility in technology and demonstrates how Raspberry Pi-based solutions can address specific needs within the community. The flexibility and affordability of the Raspberry Pi make it an ideal platform for developing assistive technologies like the Sign Language Interpreter.

In conclusion, the Raspberry Pi 4B-based Sign Language Interpreter project showcases the power of technology to bridge communication gaps and improve the quality of life for individuals with hearing impairments. It serves as a testament to the creativity and innovation possible with open-source hardware and software solutions.

REFERENCES

[1] https://github.com/Swaroop-Srisailam/Continuous-Indian-Sign-Language-Recognition.git

[2] https://gist.github.com/advait-0/0d514a9c4328a28a29b52b297d555c43

[3] Stack Overflow

[4] Raspberry Pi Forums