Indian Institute of Technology Guwahati

Department of Electronics and Communication Engineering

Sign Language Interpreter

Arya Abhay Alvekar 220102015

Harnoor Kaur 220102125

Supervisor: Hanumant Singh Shekhawat

April 23, 2025

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem statement . . . 1
1.3 Aims and Objectives . . . 1

2 Methodology **2**
2.1 List of Components . . . 2
2.1.1 Flex Sensors . . . 2
2.1.2 ESP32 Microcontroller . . . 3
2.2 Sensor Data Collection . . . 3
2.3 Dataset Collection . . . 4
2.4 Machine Learning Model Training and Testing . . . 5
2.5 Code Explanation . . . 6
2.5.1 ESP 32 Code for Sensor Data Transmission . . . 6
2.5.2 Machine Learning Model training . . . 7
2.5.3 Real-Time Sign Prediction Using Sensor Glove . . . 9

3 Results **11**
3.1 Classification Metrics . . . 11
3.1.1 Random Forest Classifier . . . 11
3.1.2 Naive Bayes Classifier . . . 12
3.1.3 Support Vector Machine (SVM) Classifier . . . 12
3.2 Confusion Matrices . . . 12
3.3 Sign Language Interpreter Glove Prototype . . . 14

4 Discussion and Analysis **16**
4.1 Overview of Findings . . . 16
4.2 Sensor Data Analysis . . . 16
4.3 System Limitations . . . 17
4.4 Significance and Implications . . . 17

5 Conclusions and Future Work **18**
5.1 Conclusion . . . 18
5.2 Future Work . . . 18

6 References **19**

1. Introduction

1.1 Background

Communication is a fundamental human need. However, individuals who are mute often face significant communication barriers in daily life. These barriers hinder their ability to express themselves, access services, and participate fully in society. Although sign language provides an effective medium for communication among the deaf and mute community, it requires mutual understanding by both the sender and the receiver. Unfortunately, most people outside this community are unfamiliar with sign language, leading to difficulties in inclusive communication across various contexts such as healthcare, education, and public services.

Technological interventions aimed at bridging this communication gap can significantly improve the autonomy and social integration of the mute community. One promising solution involves wearable electronics that can capture and interpret hand gestures associated with sign language. This project introduces a smart glove that translates hand gestures into textual output, offering a practical, real-time communication aid.

1.2 Problem statement

The primary problem addressed in this project is the lack of accessible and real-time communication tools for individuals who rely on sign language. The absence of such tools makes it challenging for mute individuals to convey their thoughts and needs in environments where others do not understand sign language. There is a need for an affordable, efficient, and user-friendly assistive device that can interpret sign language gestures and present them in a universally understood format—text.

1.3 Aims and Objectives

+

Aim: To develop a wearable glove capable of interpreting hand gestures representing sign language and translating them into real-time textual output.

Objectives:1.Design a glove embedded with flex sensors capable of detecting finger movement.
2.Collect sensor data using an ESP32 microcontroller to serially send to computer.
3.Implement ML algorithms to correctly predict the gestures.
4.Evaluate the system's accuracy.

2. Methodology

2.1 List of Components

Table 2.1 summarizes the key hardware components used in the development of the sign language interpreter glove. Each component plays a critical role in enabling gesture detection, signal processing, and communication.

Component	**Specification/Details**
ESP 32	Microcontroller used for sending sensor data serially
Flex Sensors	Resistance-based bend sensors (2.2" long), used for detecting finger motion
Resistors	10kΩ pull-down resistors at voltage divider nodes
Miscellaneous	Jumper wires, PCB board, cotton glove, soldering for assembly and connections

Table 2.1: Hardware Components Used in the Glove

2.1.1 Flex Sensors

Flex sensors are passive resistive components that change their resistance based on the degree of bending. The specific sensors used in this project are **2.2 inches** long, designed to be lightweight and easily conformable to the shape of human fingers. When mounted on a glove along the length of the fingers, these sensors exhibit a measurable change in resistance as the fingers bend or straighten.

The resistance typically increases with the bend angle — for example, a flex sensor may have a nominal resistance of $\sim 10\,\mathbf{k}\Omega$ when flat and may increase to $\sim 30\text{–}40\,\mathbf{k}\Omega$ when fully bent. This change in resistance is not linear but follows a repeatable pattern, which allows for accurate interpretation when properly calibrated.

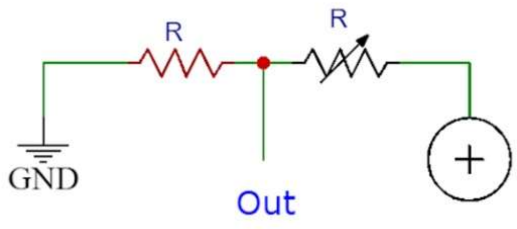


Figure 2.1: Flex sensor acts as variable resistor

Flex sensors are connected in a voltage divider configuration with a fixed resistor to convert resistance changes into analog voltage signals.

2.1.2 ESP32 Microcontroller

The ESP32 is a powerful and versatile microcontroller developed by Espressif Systems. It features:

- Dual-core 32-bit processor
- Integrated Wi-Fi and Bluetooth modules
- Multiple ADC channels with 12-bit resolution
- GPIO pins for analog and digital interfacing
- Low power consumption with various sleep modes

In this project, the ESP32 serves as the central controller for acquiring and processing analog signals from the flex sensors. It reads voltage values from the sensors via its analog-to-digital converter (ADC) pins, digitizes the data, and transmits this data in real-time through serial communication.

2.2 Sensor Data Collection

The sensor data collection process begins with the use of flex sensors mounted along the length of selected fingers (e.g., thumb, index, and middle). These sensors operate on the principle of variable resistance: as the sensor bends, its resistance increases.

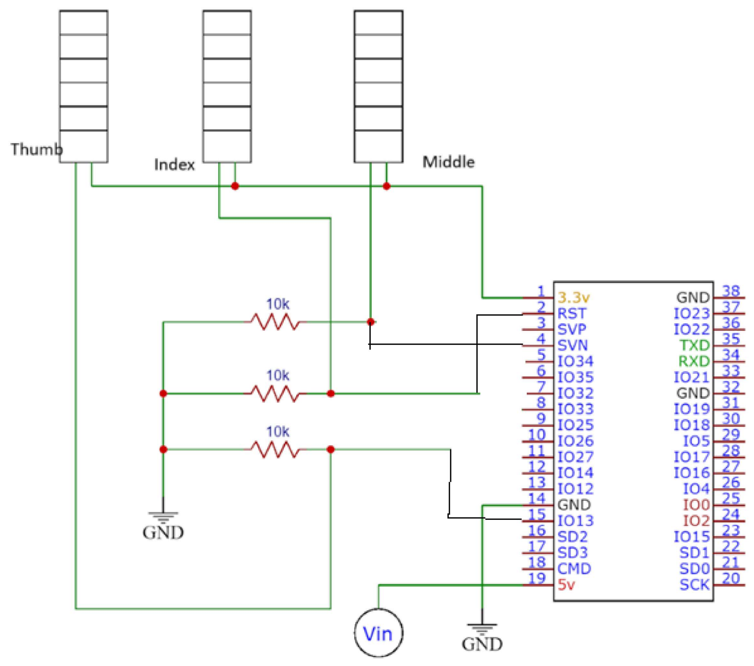


Figure 2.2: Circuit diagram

To convert this change in resistance into a measurable voltage, each flex sensor is integrated into a voltage divider circuit with a fixed $10\,\mathrm{k}\Omega$ resistor. The output voltage of the divider changes in proportion to the bend angle of the finger.

$$V_{\text{out}} = 3.3\,\mathrm{V} \cdot \frac{10\,\mathrm{k}\Omega}{R_{\text{sensor}} + 10\,\mathrm{k}\Omega}$$

This analog voltage is then fed into the analog input pins (ADC channels) of the ESP32 microcontroller. The ESP32's ADC converts the analog voltage into a digital value in the range of 0 to 4095, corresponding to the 12-bit resolution of the ADC. The digital values represent the degree of bending and are used as input features for gesture classification.

The microcontroller continuously samples these inputs in real time and transmits them via serial communication for further processing.

2.3 Dataset Collection

The dataset collection was implemented via a lightweight Python script that interfaces with the ESP32 over serial and logs sensor readings to CSV files.

- **Label and Sample Count:** The user is prompted to enter a gesture label and the total number of samples to record. In our experiments, we fixed this to 1000 samples per gesture class.
- **CSV Logging:** A new CSV file named `data_<label>.csv` is created with the header:

  ```
  thumb, index, middle, label
  ```

  Each subsequent row corresponds to one sample, containing the three flex-sensor readings (integers from the ESP32's ADC) followed by the gesture label.
- **Real-Time Sampling Loop:** The script enters a loop that runs for the specified number of samples. At each iteration:
  1. It reads a line of comma-separated sensor values from the serial port.
  2. Parses the three integer readings (thumb, index, middle).
  3. Appends the readings and label as a new row in the CSV file.
- **Cleanup:** After collecting all samples, the serial port is closed and the file is saved. The resulting CSV file for each gesture class can then be aggregated into the full dataset for subsequent preprocessing and model training.

Dataset Summary:

- Number of gestures (labels): *8*
- Samples per gesture: 1000
- Total samples: 8×1000
- Features per sample: 3 (thumb, index, middle)

2.4 Machine Learning Model Training and Testing

To enable automatic recognition of sign language gestures, machine learning models were trained using labeled flex sensor data. The dataset consisted of multiple samples for each gesture, collected by recording the analog values corresponding to various finger positions using the ESP32 microcontroller. These values were preprocessed and structured as feature vectors for supervised learning.

Three classification algorithms were implemented and evaluated:

- **Random Forest:** An ensemble learning method based on decision trees, which provides high accuracy and robustness by aggregating predictions from multiple trees.
- **Support Vector Machine (SVM):** A discriminative classifier that constructs optimal hyperplanes in a high-dimensional space to separate gesture classes. The SVM is effective in handling small to medium-sized datasets with good generalization performance.
- **Naive Bayes:** A probabilistic classifier based on Bayes' theorem, assuming independence between features. It offers fast training and prediction times and performs well with noisy data.

After training, each model was evaluated on a validation set to measure classification accuracy and performance. SVM showed the highest accuracy for our dataset. Subsequently, the trained models were tested on real-time input data collected live from the glove via the ESP32. The real-time readings were passed as input to the models, and the predicted gesture label was displayed or logged.

This approach enabled accurate recognition of gestures in real time, demonstrating the feasibility of integrating machine learning into embedded systems for sign language interpretation.

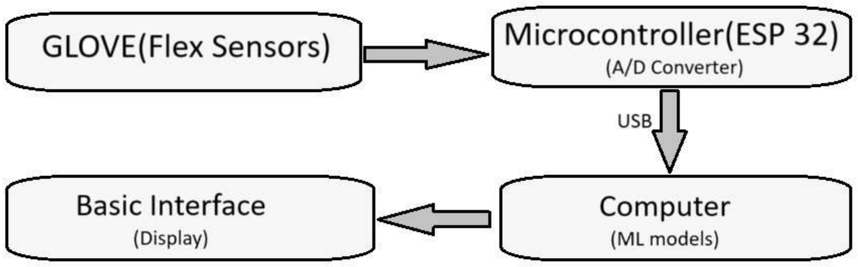


Figure 2.3: System block diagram of the sign language interpreter glove

2.5 Code Explanation

2.5.1 ESP 32 Code for Sensor Data Transmission

1. Setup and Initialization

```
void setup() {
  Serial.begin(115200);
}
```

Listing 2.1: Initializing serial communication

The ‘setup()‘ function initializes serial communication with a baud rate of 115200, enabling efficient data transfer between the ESP 32 and the computer.

2. Reading Sensor Values and Transmitting

```
void loop() {
  int thumb = analogRead(15);
  int index = analogRead(2);
  int middle = analogRead(4);

  Serial.print(thumb);
  Serial.print(",");
  Serial.print(index);
  Serial.print(",");
  Serial.println(middle);

  delay(100);  // 10 readings per second
}
```

Listing 2.2: Reading flex sensor values and sending to serial monitor

In the ‘loop()‘ function, analog values from three flex sensors (connected to pins 15, 2, and 4) are read. These values are sent over the serial port in a comma-separated format (‘thumb,index,middle‘). The ‘delay(100)‘ ensures a sampling rate of approximately 10 Hz (10 readings per second).

2.5.2 Machine Learning Model training

1. Importing Required Libraries

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import joblib
```

Listing 2.3: Importing necessary libraries

2. Loading and Preprocessing the Dataset

```python
data = pd.read_csv('C:\\Users\\Hp\\OneDrive\\Desktop\\C++\\
    sign_language_glove\\data\\sensor_data.csv', header=None)
data.columns = ['thumb', 'index', 'middle', 'label']
```

Listing 2.4: Loading sensor data and assigning column names

The dataset is loaded from a CSV file and column headers are manually set to match the sensor readings and gesture labels.

```python
print("Label distribution:")
print(data['label'].value_counts())

for col in ['thumb', 'index', 'middle', 'label']:
    data[col] = pd.to_numeric(data[col], errors='coerce')

data = data.dropna()
```

Listing 2.5: Validating and cleaning the dataset

This code ensures all data is numeric, and any rows with invalid or missing entries are removed.

3. Feature Selection and Splitting

```python
X = data[['thumb', 'index', 'middle']].values
y = data['label'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Listing 2.6: Separating features and labels

Sensor readings are extracted as features, and gesture labels are used as targets. The dataset is split into training (80%) and testing (20%) sets.

4. Feature Normalization

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 2.7: Standardizing features using StandardScaler

Normalization is crucial to ensure all features contribute equally during model training.

5. Model Training and Evaluation

```
classifiers = {
    'RandomForest': RandomForestClassifier(n_estimators=100,
   random_state=42),
    'NaiveBayes': GaussianNB(),
    'SVM': SVC(kernel='rbf', probability=True, random_state=42)
}

for name, clf in classifiers.items():
    print(f"\n--- Training {name} ---")
    clf.fit(X_train_scaled, y_train)
    y_pred = clf.predict(X_test_scaled)
    print(f"\n{name} Accuracy:", accuracy_score(y_test, y_pred))
    print(f"\n{name} Classification Report:\n",
   classification_report(y_test, y_pred))
    joblib.dump(clf, f'{name.lower()}_model.pkl')
```

Listing 2.8: Defining and training multiple classifiers

Three classifiers are trained: Random Forest, Gaussian Naive Bayes, and Support Vector Machine. Performance metrics are printed, and each trained model is saved for later use.

6. Saving the Scaler

```
joblib.dump(scaler, 'scaler.pkl')
print("\nAll models and scaler have been saved.")
```

Listing 2.9: Saving the StandardScaler instance

The scaler used during training is saved to ensure consistent preprocessing when deploying the model.

2.5.3 Real-Time Sign Prediction Using Sensor Glove

1. Importing Required Libraries

```python
import serial
import joblib
import numpy as np
import time
from collections import Counter
```

Listing 2.10: Importing essential modules

These libraries enable serial communication with ESP 32, model loading (‘joblib‘), numerical operations (‘numpy‘), and ensemble prediction using majority voting (‘collections.Counter‘).

2. Loading Pre-trained Models and Scaler

```python
rf_model = joblib.load('../model/randomforest_model.pkl')
nb_model = joblib.load('../model/naivebayes_model.pkl')
svm_model = joblib.load('../model/svm_model.pkl')
scaler = joblib.load('../model/scaler.pkl')
```

Listing 2.11: Loading machine learning models and scaler

Three pre-trained classifiers and a ‘StandardScaler‘ object are loaded from disk for real-time inference.

3. Establishing Serial Communication

```python
ser = serial.Serial('COM4', 115200)  # Update as per your system
time.sleep(2)  # Allow time for connection to stabilize

print("Listening for sensor values...")
```

Listing 2.12: Connecting to ESP 32 via serial port

A serial connection is established with the ESP 32 (connected at COM4 with 115200 baud rate). A delay ensures the port is ready before reading data.

4. Real-time Data Reading and Prediction Loop

```python
try:
    while True:
        line = ser.readline().decode('utf-8').strip()
        print(f"Raw: {line}")

        try:
            values = list(map(float, line.split(',')))
            if len(values) != 3:
                print("Skipping: wrong number of values")
                continue

            scaled = scaler.transform([values])

            predictions = [
                rf_model.predict(scaled)[0],
                nb_model.predict(scaled)[0],
                svm_model.predict(scaled)[0]
            ]

            final_prediction = Counter(predictions).most_common(1)[0][0]
            print(f"Predicted Sign (Ensemble): {final_prediction}\n")

        except ValueError:
            print("Skipping: could not parse numbers")
```

Listing 2.13: Processing incoming sensor data and predicting gestures

The code continuously listens for data in the format 'thumb,index,middle'. It scales the input, obtains predictions from all models, and applies majority voting to decide the final predicted gesture.

5. Exit

```
except KeyboardInterrupt:
    print("Stopped by user")
    ser.close()
```

Listing 2.14: Closing the serial connection on interrupt

The loop terminates when the user interrupts execution (e.g., via Ctrl+C), ensuring the serial port is properly closed.

3. Results

This section presents the performance evaluation of the machine learning models used for recognizing hand gestures from the sign language interpreter glove. The dataset was split into training and testing sets, and three classifiers—Random Forest, Naive Bayes, and Support Vector Machine (SVM)—were trained and evaluated.

The effectiveness of each model is demonstrated using confusion matrices and detailed classification metrics including precision, recall, and F1-score for each gesture class. These results help assess the reliability and robustness of the models in recognizing real-time sign inputs from sensor data.

3.1 Classification Metrics

The performance of each classifier is evaluated on eight gesture classes (labeled 0 through 7). The metrics reported include precision, recall, and F1-score for each class, as well as overall accuracy.

We see that, for our dataset, Support Vector Machine achieves the top performance across gesture classes, with both precision and recall consistently near 1.0. Random Forest also performs well—though with lower recall on a few classes—and Naive Bayes shows the weakest but still reasonable overall accuracy.

3.1.1 Random Forest Classifier

Table 3.1: Random Forest - Per-Class Performance

Class	**Precision**	**Recall**	**F1-score**
0	1.00	0.94	0.97
1	0.69	1.00	0.82
2	1.00	1.00	1.00
3	0.98	0.54	0.70
4	0.92	1.00	0.96
5	1.00	0.52	0.68
6	1.00	1.00	1.00
7	0.68	0.99	0.81
Accuracy			0.87
Macro Avg	0.91	0.87	0.87

3.1.2 Naive Bayes Classifier

Table 3.2: Naive Bayes - Per-Class Performance

Class	Precision	Recall	F1-score
0	1.00	0.71	0.83
1	1.00	0.80	0.89
2	1.00	0.73	0.85
3	0.40	0.99	0.57
4	0.78	1.00	0.87
5	1.00	0.50	0.67
6	1.00	1.00	1.00
7	0.99	0.50	0.66
Accuracy			0.78
Macro Avg	0.90	0.78	0.79

3.1.3 Support Vector Machine (SVM) Classifier

Table 3.3: SVM - Per-Class Performance

Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	0.67	1.00	0.92
2	0.99	1.00	1.00
3	1.00	0.99	1.00
4	0.97	1.00	0.98
5	1.00	0.51	0.68
6	1.00	0.99	1.00
7	0.99	1.00	1.00
Accuracy			0.94
Macro Avg	0.95	0.94	0.93

3.2 Confusion Matrices

The following figures present the confusion matrices for the three classifiers, showing the number of correctly and incorrectly predicted gestures for each class.

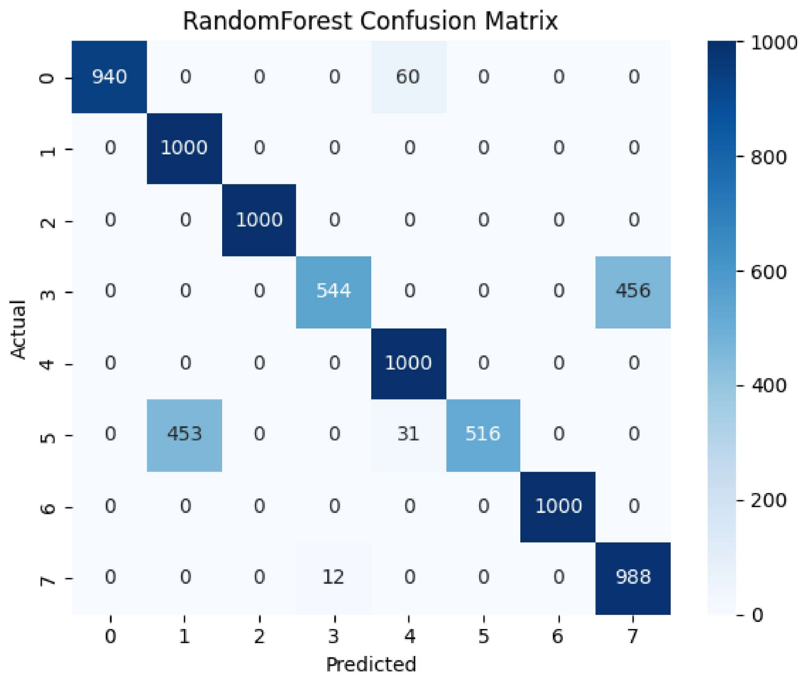


Figure 3.1: Confusion Matrix — Random Forest

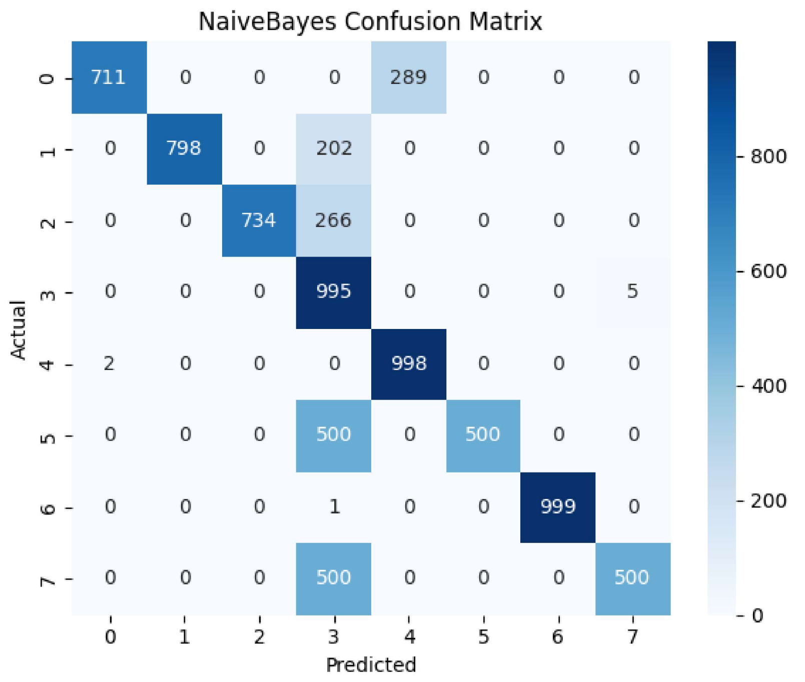


Figure 3.2: Confusion Matrix — Naive Bayes

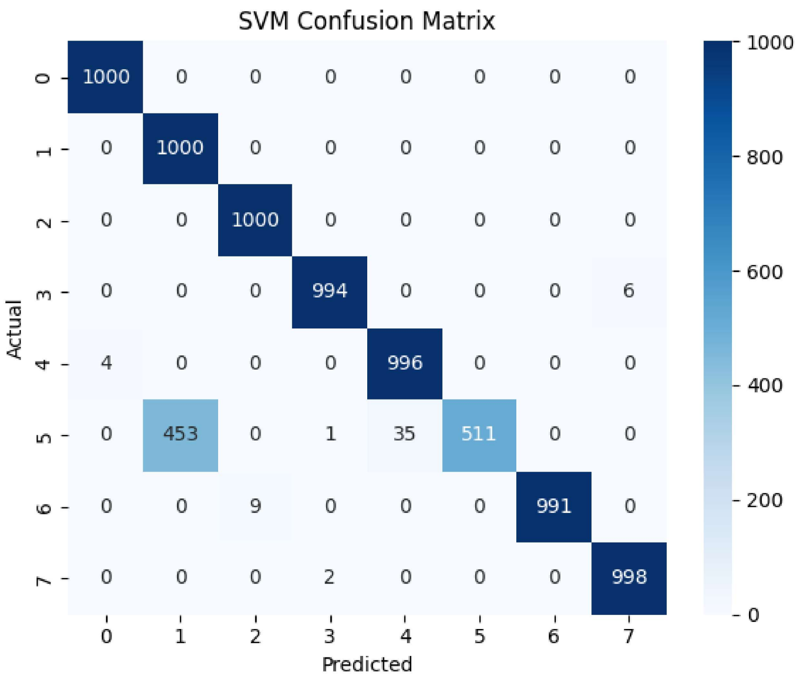


Figure 3.3: Confusion Matrix — Support Vector Machine (SVM)

3.3 Sign Language Interpreter Glove Prototype

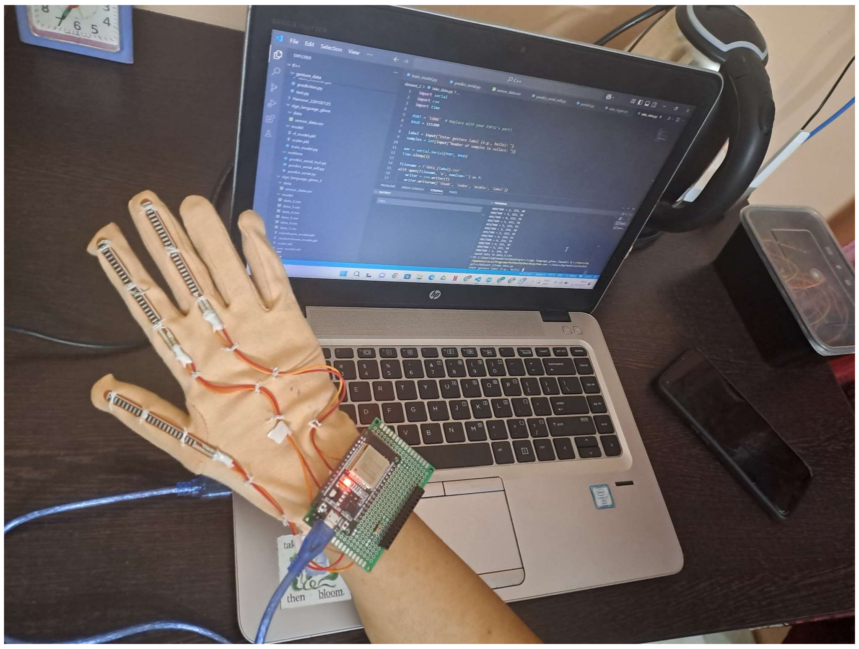
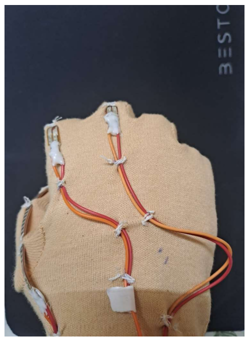
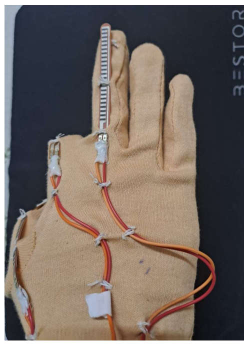
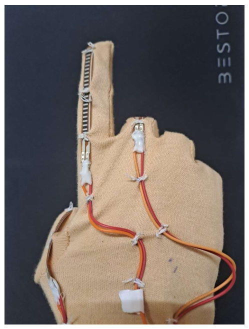
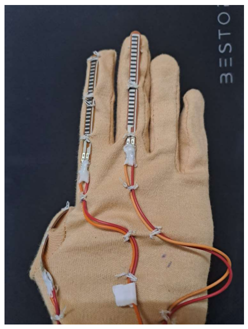


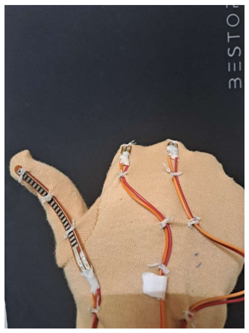
Figure 3.4: Prototype of the sign language interpreter glove

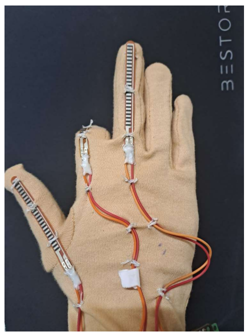

Gesture 0

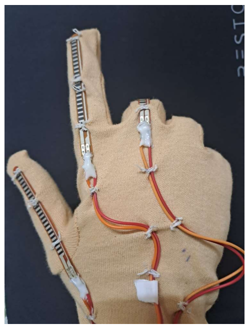

Gesture 1


Gesture 2


Gesture 3


Gesture 4


Gesture 5


Gesture 6

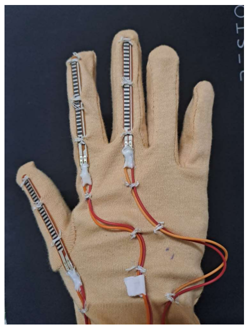

Gesture 7

Figure 3.5: Labeled gestures from 0 to 7 captured by the system

4. Discussion and Analysis

4.1 Overview of Findings

The sign language interpreter glove developed in this project demonstrated effective classification of static hand gestures using sensor data from three flex sensors. There were a few inconsistencies with the flex sensor outputs, which were significantly compensated for with the applied processing methods. The real-time ensemble-based classification system was able to predict signs reliably from live input, validating the practical utility of the prototype.

4.2 Sensor Data Analysis

An analysis of the raw analog input from the glove's flex sensors revealed several important observations:

- **Distinct Value Patterns:** Each gesture exhibited a distinguishable pattern in sensor values, supporting the feasibility of classification using standard machine learning models.
- **Acceptable Signal-to-Noise Ratio:** Although minor fluctuations were observed in raw sensor output due to slight finger tremors or sensor noise, the data was generally stable.
- **Sensitivity to Physical Factors:** Physical conditions, such as sensor bending fatigue or contact with heat during soldering, altered resistance values and introduced variability in readings.
- **Performance Degradation with Wear:** Over extended use, sensors exhibited drift in their output ranges, affecting consistency across sessions.
- **Limited Sensor Coverage:** Only three fingers were monitored, which restricts the resolution of hand pose detection.
- **Poor Sensor Recoil:** Flex sensors exhibited inconsistent return to baseline values after bending, leading to variations in readings even when the finger was fully straightened again. This recoil inconsistency affects data reliability and gesture classification accuracy.

These observations suggest that while the sensor data is suitable for controlled prototype testing, long-term stability and calibration mechanisms would be required for production-level deployment.

4.3 System Limitations

Despite the promising results, the system has several technical and operational limitations:

- **Restricted Gesture Vocabulary:** Only eight static gestures were included in the training set, limiting comprehensive sign language coverage.
- **No Support for Dynamic Gestures:** The system cannot detect gestures that involve motion sequences or temporal dependencies.
- **Lack of Feedback Mechanism:** The system does not notify users when predictions are uncertain or incorrect, which limits usability for communication-critical scenarios.

Addressing these challenges would require improvements in hardware robustness, automatic calibration, and enhanced model adaptability.

4.4 Significance and Implications

This project demonstrates how low-cost sensor systems combined with machine learning can be leveraged to bridge communication gaps for individuals with speech and hearing impairments. The developed glove system has implications in the following areas:

- **Assistive Technology:** Real-time sign recognition systems can enable speech-impaired users to communicate more independently.
- **Inclusive Computing:** Gesture-based input systems offer a new modality for human-computer interaction, especially in accessibility contexts.
- **Educational Applications:** This platform can be extended into interactive learning tools for teaching sign language.

The prototype sets the foundation for future enhancements aimed at full-scale gesture interpretation and seamless assistive communication.

5. Conclusions and Future Work

5.1 Conclusion

This project presented the design and implementation of a wearable sign language interpreter glove that uses flex sensor data and machine learning models to recognize static hand gestures. The glove collects real-time data from three fingers—thumb, index, and middle—and transmits it to a computer for classification.

The system successfully demonstrates the feasibility of combining embedded hardware and machine learning for assistive communication. It has the potential to support individuals with hearing or speech impairments by converting sign gestures into readable or audible formats.

5.2 Future Work

Future developments can focus on the following improvements:

- **Expansion to Full Sign Language:** Extend the gesture set to include more signs, including alphabets, numbers, and commonly used words. This would involve sensors used on all 5 fingers.
- **Dynamic Gesture Recognition:** Integrate accelerometers or gyroscopes to support temporal and motion-based gestures.
- **Portable Output System:** Implement an onboard display or speech module to make the system independent of a connected computer.

With these enhancements, the glove-based interpreter can evolve into a practical, real-world assistive tool, promoting accessibility and inclusivity in communication.

6. References

- Sensor Fusion of Motion-Based Sign Language Interpretation with Deep Learning. Boon Giin Lee, Teak-Wei Chong, Wan-Young Chung. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC7663682/

- "SignAloud": Lemelson-MIT 2017 prize-winning project. Microcontroller and statistical regression to translate from American Sign Language.

- Smart Glove - Sign Language Translator. Bahria University. Available at: https://github.com/WaniaKhance/Smart-Glove-Sign-Language-Translator

- Sign Language Translator Glove. Available at: https://github.com/balpaula/Sign-Language-Translator-Glove

- IIT Bombay. Sign Language Interpretation And Robotic Hand With Natural Control System. Available at: https://github.com/eYSIP-2016/eYSIP-2016-Sign-Language-Interpreter-Leap-Motion-Sensor-