International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

# Smart Sign Automated Caption Generator for Sign Language

Sanket Shukla, Abhishek Shukla, Yogesh Tiwari
Department of Artificial Intelligence and Data Science

IIMT College of Engineering, Gretaer Noida, UP, India shuklasanket2004@gmail.com

***Abstract— Design, fabrication, and experimental analysis of microelectromechanical systems (MEMS)-based mass air flow (MAF) sensor especially tailored for use in ventilator devices are presented here. With principles of forced convective heat transfer in mind, the sensor employs a platinum (Pt) micro heater and This paper proposes a novel approach for developing a real-time sign language translation system, aimed at bridging communication gaps between deaf and hearing communities. By leveraging computer vision techniques with OpenCV and deep learning models, this project seeks to create an automated caption generator that can detect, recognize, and translate sign language gestures into readable text or speech. The system's accuracy, real-time performance, and adaptability will be evaluated through the integration of machine learning algorithms for hand and gesture recognition.***

## I. INTRODUCTION

Sign language plays a crucial role in communication within the deaf and hard-of-hearing community. However, its usage is not widely understood by the general population, making it difficult for deaf individuals to interact with people who do not know sign language. Traditional methods of translation, such as human interpreters, are not always readily available. Therefore, there is a need for innovative solutions that can automatically interpret sign language and convert it into captions or voice-based outputs in real-time.
This research aims to develop a "Smart Sign" system, which uses OpenCV for image processing and machine learning techniques for gesture recognition. The goal is to create an efficient and scalable system that can recognize hand gestures and convert them into natural language captions.

## II. PROBLEM STATEMENT

The main challenges faced in the development of an automated sign language recognition system include:

- Variability in Sign Language Gestures: Different sign languages (e.g., American Sign Language, British Sign Language) have distinct signs.
- Hand Gesture Complexity: Hand positions, orientations, movements, and facial expressions all contribute to the meaning of signs.
- Real-time Processing: The system must operate efficiently in real-time, which requires balancing high recognition accuracy with low processing time.

## III. OBJECTIVE

- To design and develop a system capable of recognizing sign language gestures.
- To generate captions from the recognized signs in real-time.
- To evaluate and improve the accuracy of the gesture recognition system.

## IV. LITERATURE REVIEW

In this section, summarize existing works related to:

- Sign Language Recognition Systems: Review the state-of-the-art methods, including approaches using image processing, deep learning (e.g., convolutional neural networks), and sensor-based techniques.
- Computer Vision with OpenCV: Discuss how OpenCV has been utilized in previous gesture recognition systems.
- Machine Learning in Gesture Recognition: Explore the types of machine learning models (e.g., CNN, RNN, LSTM) used for interpreting hand gestures.
- Real-time Translation Tools: Identify current systems that have been developed for real-time sign language translation.

## V. METHODOLOGY

The experimental validation framework comprised a
V.I. Data Collection:
To train the machine learning model, a large dataset of sign language images or videos is required. This could be collected from:
• Public Datasets: Datasets like the "Sign Language MNIST" or "RWTH-PHOENIX-Weather 2014T" dataset could be useful for training.
• Custom Dataset: Capture video or images of various sign language gestures for better model training.
V.II. Preprocessing and Feature Extraction:
• Hand Detection: Use OpenCV for detecting and tracking hand regions. Techniques like background subtraction or contour detection could be useful for identifying the hands.
• Pose Estimation: Utilize OpenCV or libraries like Mediapipe for pose estimation to track the hand's position, angle, and gestures.

• Image Processing: Apply image processing techniques such as resizing, normalizing, and filtering to improve the quality of input data.

V.III. Model Training:
• Deep Learning Model: Train a deep learning model using a CNN or a combination of CNN with RNN (e.g., LSTM) to recognize hand gestures.
• Data Augmentation: To increase the robustness of the model, apply augmentation techniques like rotation, translation, and scaling on the training dataset.
• Model Evaluation: Evaluate the model using metrics such as accuracy, precision, recall, and F1-score.

V.IV .Real-time Gesture Recognition and Caption Generation:
• Real-time Input: Use a webcam or video stream to capture sign language gestures.
• Gesture Recognition: Process the input frame-by-frame to detect and classify gestures.
• Caption Generation: Convert the recognized gestures into captions in real-time using text output.

## VI. TECHNOLOGIES USED

We sincerely thank IIMT College of Engineering and our guide Dr. Pankaj Jha for their support and mentorship. We are also grateful to CSIR-CEERI, Pilani, for providing research resources, and to Dr. Rahul Prajesh (Principal Scientist) for his invaluable guidance as our external mentor.

- OpenCV: For image processing, detecting hand gestures, and managing real-time video input.
- TensorFlow/Keras/PyTorch: To build and train deep learning models for gesture recognition.
- Mediapipe: For hand pose estimation and landmarks detection.
- Python: For overall system development, data manipulation, and model integration.

## VII. RESULTS AND DISCUSSION

Discuss the outcomes of the model's performance:
- Accuracy and Real-time Performance: Evaluate the recognition accuracy and speed of the system, comparing it against benchmarks.
- Challenges Faced: Identify the challenges such as misclassifications, difficulties in recognizing complex gestures, and lighting or background interference.
- Future Improvements: Suggestions for enhancing the system, like integrating multiple camera views for 3D hand tracking or expanding the dataset for other sign languages.

## VIII. CONCLUSION

The research presents a significant step toward improving communication between deaf and hearing communities through automatic sign language translation. Although the system provides real-time gesture recognition with reasonable accuracy, challenges in hand detection and complex gestures remain. With further development, this approach could scale to more complex and diverse sign languages, making it a valuable tool for inclusion and accessibility.

## IX. REFERENCES

Provide references to relevant research papers, articles, and resources used to develop the project. Some example references include:
I. A. Author et al., "Sign Language Recognition Using CNN," Journal of Computer Vision, 2022.
II. B. Author et al., "Hand Gesture Recognition using OpenCV," International Journal of Computer Science, 2021.
III. C. Author et al., "Deep Learning for Sign Language Translation," IEEE Transactions on AI, 2020.

## X. APPENDIX (IF NEEDED)

Include any additional charts, graphs, or code snippets that are relevant to your work.

**Implementation Code (Example):**

```
import cv2
import mediapipe as mp

# Initialize MediaPipe Hand Model
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=2, min_detection_confidence=0.5

# Initialize OpenCV webcam
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Error: Could not read frame.")
        break

    # Flip frame horizontally for natural interaction
    frame = cv2.flip(frame, 1)

    # Convert the frame to RGB for Mediapipe
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process the frame and detect hands
    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:
        for landmarks in results.multi_hand_landmarks:
            # Draw hand landmarks
            mp_drawing.draw_landmarks(frame, landmarks, mp_hands.HAND_CONNECTIONS)

        # Display number of hands detected
        cv2.putText(frame, f"Hands detected: {len(results.multi_hand_landmarks)}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

    # Show the frame
    cv2.imshow("Hand Gesture Recognition", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```