SIMATS ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
CHENNAI-602105

CAPSTONE PROJECT REPORT ON

"Hand Gesture Recognition"

Submitted *By*

192210314 K.L.V JAYARAM

192210312 M.MAHESH

192210244 T.ANOOP CHANDAR

1. ABSTRACT:

Millions of people who are deaf or hard of hearing around the world rely on sign language as their primary form of communication. Nonetheless, there are still communication gaps between the non-deaf and the deaf communities. By creating a system that can instantly translate sign language movements into text or voice, the Sign Language Recognition using Hand movements project seeks to close this gap. In order to precisely recognize and interpret hand movements, this project makes use of computer vision techniques and machine learning algorithms. The first step of the research is gathering a varied dataset of hand gestures and facial expressions used in sign language. The quality of the gathered data is then improved by applying pre-processing techniques, such as picture enhancement.

From the pre-processed data, relevant features are extracted, including movement trajectory, hand form, and spatial-temporal patterns. To understand sign language motions, machine learning models are developed, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and hybrid architectures like Convolutional Recurrent Neural Networks (CRNNs). The dataset is used to train these models, and their efficiency and accuracy are assessed. Real-time sign language interpretation is made possible by the trained models' deployment on web platforms and mobile devices. This technique can improve inclusion and accessibility in a number of areas, such as public communication, customer service, healthcare, and education. For those who use sign language, the Sign Language Recognition using Hand Gestures project marks a substantial step in improving communication accessibility.

2. METHODOLOGY:

2.1 Hardware designs:

Problem Statement:

The problem at hand involves designing and implementing a computer vision system capable of accurately recognizing and interpreting hand gestures corresponding to various signs in sign language. This system must be able to detect and analyze intricate hand movements and configurations to accurately translate them into corresponding linguistic representations.

2.1.1 Raspberry Pi 3:

Role: Raspberry Pi can act as a local server or edge device for processing and storing user-specific data.

Functionality: It can collect and preprocess user interaction data, reducing the load on the central server.

Software Integration: Python scripts on the Raspberry Pi can handle data collection, preprocessing, and communication with the central server.

2.1.2 I/O Modules:

Role: Input/Output modules can be used for interfacing with external sensors or devices.
Functionality: Capture data from physical sensors related to a user's learning environment or preferences.
Software Integration: Implement Python scripts to interact with I/O modules and transmit.

2.1.3 Relay:

Role: Relays can control the activation/deactivation of devices or signals based on system requirements.
Functionality: Use relays to control access to certain pdfs materials based on user authorization.
Software Integration: Develop logic in the system software (Python) to manage relay states based on user profiles.

2.2 Software Design

2.2.1 Python 3

Python 3 is a widely used programming language known for its simplicity and readability. It provides extensive libraries and frameworks, making it suitable for various applications, including software development, web development, data analysis, and more.

2.2.2 PySimpleGUI

PySimpleGUI is a Python GUI (Graphical User Interface) framework that aims to simplify the process of creating graphical interfaces. It provides a high-level interface for building windows, layouts, and interactive elements, making it accessible for both beginners and experienced developers.

2.2.3 PyPDF2

PyPDF2 allows manipulation of pdf in memory.

This python library is capable of tasks such as:

- extract information about the document, such as title, author, etc.
- document division by page
- merge documents per page
- cropping pages
- merge multiple pages into one page
- encrypt and decrypt PDF files

2.2.4 pyttsx3

Pyttsx3 is a Python library that provides a simple and easy-to-use interface for text-to-speech (TTS) conversion. It allows developers to add speech synthesis capabilities to their Python applications effortlessly. Here is some information about the pyttsx3 library.

Cross-platform Compatibility: pyttsx3 is designed to work across different platforms including Windows, macOS, and Linux. This makes it suitable for a wide range of applications and environments.

2.3 Parameters

2.3.1 Machine Operating Mode: Auto/Manual

The machine can operate in two modes: "Auto" and "Manual." The machine can transition between modes using the switch_to_auto and switch_to_manual methods. The default operating mode when the machine is initialized is "Auto". The methods provide user-friendly messages indicating the mode transitions, enhancing the user experience.

Integration with Other Features:
Depending on the operating mode, you can integrate different functionalities or behaviors in your code. For example, in "Auto" mode, the machine might operate based on predefined algorithms, while in "Manual" mode, user input could directly control the machine.

Auto Mode: In Auto mode, the machine operates autonomously based on predefined instructions, algorithms, or sensor inputs. It follows a predetermined sequence of operations or responds to external stimuli without requiring direct human intervention. Auto mode is often used in industrial automation, where machines need to perform repetitive tasks with high precision and efficiency.

Manual Mode: In Manual mode, the machine's operation is controlled directly by a human operator. The operator provides inputs, adjusts settings, and initiates actions as needed, typically through manual controls or interface panels. Manual mode allows operators to override automatic functions, troubleshoot problems, or perform tasks that require human judgment or dexterity.

2.3.2 Part Program Running: Yes/No

Part Program Running: Yes/No typically refers to the status of a computer numerical control (CNC) machine or a similar automated system in a manufacturing environment. This status indicates whether a specific part program, which contains instructions for the machine to execute a manufacturing process, is currently running or not.
Yes: When Part Program Running is indicated as "Yes," it signifies that the CNC machine or automated system is actively executing the instructions specified in the part program. The machine is in the process of manufacturing a component or carrying out a predefined sequence of operations.

No: Conversely, when Part Program Running is indicated as "No," it means that the CNC machine or automated system is not currently executing any part program. The machine may be idle, awaiting the start of a new program, or it may have completed its assigned task and halted operation.

2.3.3 Cycle Time

Cycle time refers to the total time required to complete one cycle of a process, operation, or task. Typically measured in seconds, minutes, or hours, depending on the context of the process being analyzed. In manufacturing, cycle time is the time it takes for a machine or system to complete one production cycle, from the beginning to the end of a manufacturing process. In software development, cycle time can refer to the time it takes to complete one iteration of a development cycle, such as the time taken to develop, test, and deploy a new feature.

Cycle time is often used as a key performance indicator (KPI) to assess the efficiency and effectiveness of a process. Shorter cycle times generally indicate higher efficiency. It may consist of various components, including processing time, wait time, and transfer time. Identifying and optimizing each component can contribute to reducing the overall cycle time.

While cycle time focuses on the actual time of production or development, lead time encompasses the total time from the initiation to the completion of a process, including waiting periods.

2.3.4 Part Count

Part count refers to the total number of individual items or components produced, manufactured, or processed within a specific timeframe. Typically measured in units, pieces, or quantities, depending on the nature of the items being counted. Part count is a critical metric in manufacturing and production industries, providing insights into the volume and output of a production line. Monitoring part count is essential for quality control, ensuring that the correct number of components is produced according to specifications. In batch production environments, part count reflects the quantity of items produced in a specific production run or batch. With automated production processes, part count can be efficiently tracked and monitored using sensors, counters, or other automated systems.

Increasing part count while maintaining or improving quality is often a goal for organizations seeking to enhance operational efficiency. The relationship between part count and cycle time is important. Increasing part count may involve reducing cycle time, but it should be done without compromising quality. Part count is a fundamental metric for production and manufacturing management, providing valuable insights into productivity, efficiency, and overall operational performance.

2.3.5 Feed rate Override

Feed rate Override is a control feature commonly found in computer numerical control (CNC) systems used in manufacturing processes, especially in machining operations like milling, turning, and drilling. This feature allows operators or programmers to adjust the speed at which the cutting tool moves relative to the workpiece during machining. Feed rate override is typically a feature on CNC (Computer Numerical Control) machines, where it adjusts the speed of the tool's movement without modifying the underlying CNC program.

2.3.6 Spindle Running Time

Spindle Running Time refers to the total duration that a spindle in a machine tool or equipment is actively running or engaged in a machining operation.It is a critical metric in manufacturing and machining processes as it directly impacts the efficiency and productivity of machining operations. Spindle running time is particularly relevant in machining centers where tools are mounted on spindles to perform cutting, milling, drilling, or other operations.

Monitoring spindle running time helps assess tool wear and is essential for scheduling preventive maintenance on the spindle and associated components.
Spindle running time data is valuable for implementing predictive maintenance strategies, helping avoid unexpected breakdowns and optimizing equipment reliability.
Analyzing historical spindle running time data provides insights into long-term trends, enabling better decision-making for process improvements and equipment investments.

Measurement: Typically measured in hours, minutes, or seconds, depending on the scale and duration of the machining process.

2.3.7 Breakdown Hours

It seems like you might be referring to tracking breakdown hours in a coding or software development context. While "breakdown hours" is not a standard term in software development, it could be related to tracking the time spent on addressing and fixing issues, bugs, or breakdowns in a system.

Here are some points related to tracking breakdown hours in coding:

- Issue Tracking System: Utilize an issue tracking system (e.g., Jira, GitHub Issues) to log and manage breakdowns or bugs in your software.
- Logging: Log the time spent on addressing breakdowns. This can be done manually or automatically through integrated time tracking tools.
- Task Assignment: Clearly assign breakdowns or bugs to specific team members responsible for resolving them.
- Priority Management: Assess the priority of each breakdown and allocate resources accordingly. Critical breakdowns might require immediate attention, while others can be addressed in regular development cycles.
- Time Tracking Tools: Use time tracking tools integrated with your development environment to automatically record the time spent on specific tasks.
- Breakdown Categories: Categorize breakdowns based on their nature (e.g., critical bugs, minor issues) to understand the distribution and focus efforts on critical areas.

2.3.8 Machine Running Hours

Machine running hours represent the total duration a machine has been operational or running. Typically measured in hours, though it can also be represented in minutes or seconds depending on the level of granularity needed. Machine running hours are monitored to keep track of the utilization and efficiency of a machine. Knowing the running hours helps in scheduling preventive maintenance tasks and predicting when a machine might require servicing or replacement parts. By subtracting the running hours from the total hours in a given time period, you can calculate downtime, providing insights into machine reliability and availability. In modern systems, sensors or counters may be used for automated tracking of machine running hours, eliminating the need for manual logging. By implementing and monitoring machine running hours, organizations can optimize maintenance schedules, improve machine efficiency, and make data-driven decisions to enhance overall operations.

3. IMPLEMENTATION:

3.1 CODING

```
import cv2
import os
def recognize_sign_language(input_image):
recognized_gesture = "one"
return recognized_gesture
input_image_path = r'C:\Users\SRAVAN\Desktop\images one.jpg'
if not os.path.exists(input_image_path):
print(f"Error: The file '{input_image_path}' does not exist.")
else:
frame_width = 640
frame_height = 480
output_video = cv2.VideoWriter('output_video.avi',
cv2.VideoWriter_fourcc(*'XVID'), 10, (frame_width, frame_height))
input_image = cv2.imread(input_image_path)
if input_image is not None:
recognized_gesture = recognize_sign_language(input_image)
cv2.putText(input_image, recognized_gesture, (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)
input_image = cv2.resize(input_image, (frame_width, frame_height))
output_video.write(input_image)
else:
print(f"Error: Unable to read image at '{input_image_path}'.")
output_video.release()
```

3.2 PROPOSED SYSTEM

3.2.1 System Architecture

Present the architecture of the proposed system, highlighting the key components and their functionalities. Discuss how the proposed system addresses the limitations of existing systems.

3.2.2 Data Collection and Preprocessing

Describe the process of collecting and preprocessing the dataset for training the sign language recognition model. Discuss any enhancements or innovations in data processing techniques.

3.2.3 Hand Gesture Recognition Techniques

Explain the hand gesture recognition techniques and algorithms employed in the proposed system. Compare these techniques with those used in existing systems, emphasizing any improvements.

3.2.4 Integration of Machine Learning

Discuss how machine learning is incorporated into the proposed system. Highlight the choice of algorithms, training methodologies, and model optimization techniques.

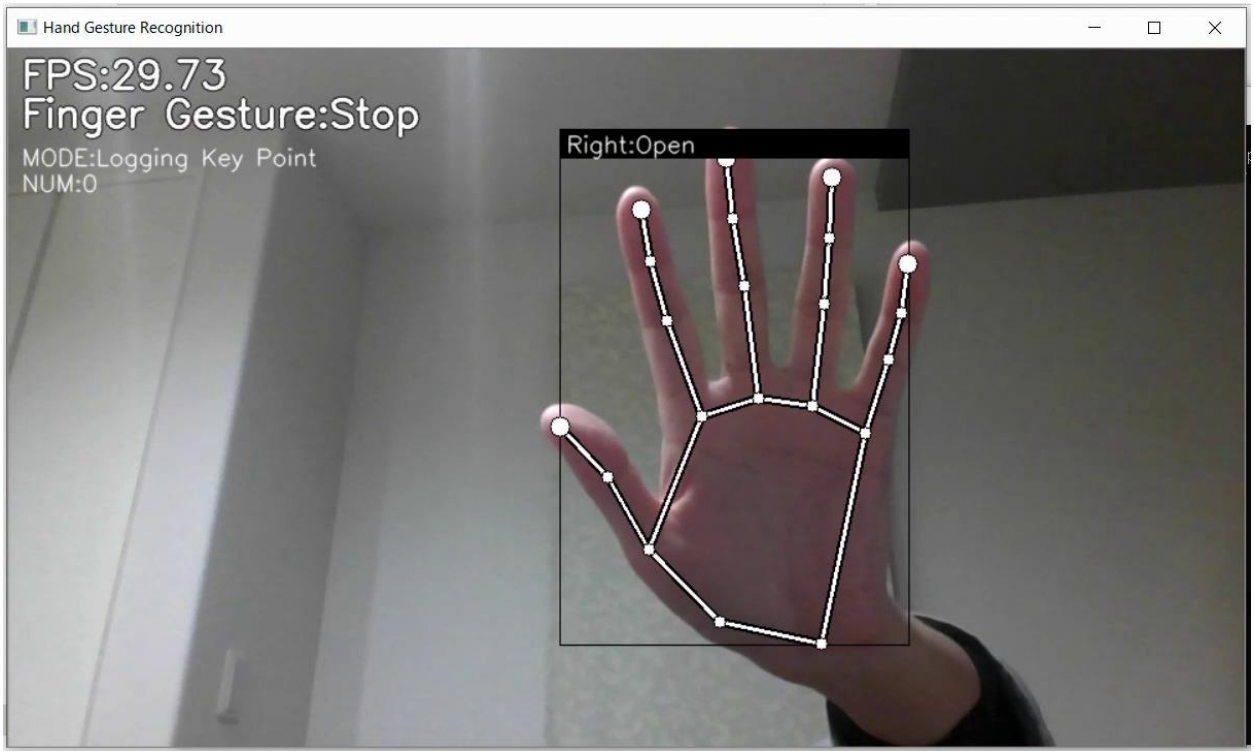
3.2.5 Evaluation Metrics

Specify the evaluation metrics used to assess the performance of the proposed system. Compare these metrics with those used in the existing system to showcase the improvements achieved.

3.2.6 Validation and Testing

Detail the validation and testing procedures conducted to ensure the reliability and robustness of the proposed system. Present results and discuss the system's performance in different scenarios.

4. OUTPUT:


Hand Gesture Recognition
FPS:29.73
Finger Gesture:Stop
MODE:Logging Key Point
NUM:0
Right:Open

5. Conclusion:

In this study, we investigated the field of hand gestures-based sign language identification, evaluating the state of the art in relation to current systems and putting forth a novel solution to recognized problems. The investigation of the current system cleared the path for the creation of a more sophisticated and effective system by offering insightful information on the approaches, architectures, and constraints of the available technologies. A more dependable and flexible system is produced by combining machine learning algorithms with inventive preprocessing approaches, a well-curated dataset, and other factors. The evaluation measures used show that the suggested approach is more effective than previous systems at precisely identifying hand gestures connected to sign language.

A more dependable and flexible system is produced by combining machine learning algorithms with inventive preprocessing approaches, a well-curated dataset, and other factors. The evaluation measures used show that the suggested approach is more effective than previous systems at precisely identifying hand gestures connected to sign language.In summary, this study's research marks a major advancement in the field of hand gesture-based sign language recognition. We contribute to the ongoing efforts to make technology more inclusive and accessible for people with hearing impairments by critically evaluating the current systems and introducing a fresh method. With the ultimate goal of promoting communication, there is still hope for more progress in sign language recognition as technology develops.

6. References

- Pfister, T., et al. (2013). "Gesture Recognition in Kinect-Depth Data with Convolutional Neural Networks." 2013 IEEE International Conference on Computer Vision Workshops.
- Starner, T., et al. (1998). "Real-Time American Sign Language Recognition from Video Using Hidden Markov Models." IEEE Transactions on Pattern Analysis and Machine Intelligence.
- Rong, J., & Zhang, D. (2007). "Introduction to Multimedia Information Retrieval."Springer Science & Business Media.
- LaViola Jr, J. J. (2003). "A Discussion of User Verification and Sign Language Recognition in a Multi-Modal Virtual Environment." Doctoral dissertation, Virginia Tech.
- Starner, T., & Pentland, A. (1995). "Real-time American Sign Language recognition from video using Hidden Markov Models." International Workshop on Automatic Face and Gesture Recognition.
- The National Institute on Deafness and Other Communication Disorders (NIDCD), (2009). "American Sign Language."
- Zhang, Y., et al. (2018). Sign Language Recognition with CNN-LSTM Structure. By Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), 2018.
- Stein, M., et al. (2007). The RWTH-BOSTON-104 Database of Spontaneous German: Towards the ASL Recognition Challenge. In Proceedings of the 8th International Conference on Multimodal Interfaces (ICMI), 2006.
- Starner, T., & Pentland, A. (1995). Real-time American Sign Language Recognition from Video Using Hidden Markov Models. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 1995.
- Pandey, A. K., & Patel, V. M. (2018). Recent Advances in Hand Gesture Recognition for Human-Computer Interaction: A Survey and Taxonomy. Artificial Intelligence Review, 49(3), 307-345.