Journal of Neonatal Surgery
ISSN(Online): 2226-0439
Vol. 14, Issue 32s (2025)
https://www.jneonatalsurg.com

OPEN ACCESS

Hand Gesture Control Apps Using Hand Movements

Gundubogula Vidhyanjali[1], Dr. Priyanka Kumari Bhansali[*2], Dr. S K Hiremath[3], Syed Mujib Rahaman[4]

[1]MCA, Department of CS & SE, Andhra University College of Engineering(A), Visakhapatnam, AP-03, India
[*2]Assistant Professor, Department of CS & SE, Andhra University College of Engineering(A),Visakhapatnam, AP-03, India
[3]Associate Professor, Department of Computer Science & Engg.(AI & ML), CMR University, Bangalore, Karnataka, India
[4]Research Scholar, Faculty of Computer Engineering ,Pacific Academy of Higher,Education and Research University, Udaipur, Rajasthan, India
Corresponding Author:
[*]**Corresponding author:**
Dr. Priyanka Kumari Bhansali
Email ID: dr.priyankabhansali@andhrauniversity.edu.in

Cite this paper as: Gundubogula Vidhyanjali, Dr. Priyanka Kumari Bhansali, Dr. S K Hiremath, Syed Mujib Rahaman, (2025) Hand Gesture Control Apps Using Hand Movements. *Journal of Neonatal Surgery*, 14 (32s), 6266-6274.

ABSTRACT
This project presents an innovative and touch-free method for controlling a computer using simple hand gestures. With the help of a webcam, users can perform everyday actions—like moving the mouse cursor, clicking, scrolling, or adjusting system settings such as volume and brightness—just by waving or positioning their hands in specific ways. At the heart of this system is Mediapipe, a powerful computer vision library developed by Google. It accurately detects hand landmarks and tracks finger movements in real-time. By analyzing how the fingers are arranged, the program interprets different gestures and converts them into commands for the computer. This gesture-controlled interface not only makes computer interaction more intuitive but also enhances accessibility, especially for individuals with physical limitations or those looking for hygienic, contactless interaction. It's a meaningful step toward making human-computer interaction more seamless, inclusive, and futuristic.

Keywords: *hand gesture recognition, hand-computer interaction, computer vision, mediapipe, cursor control, touchless interface, real-time tracking, gesture-based navigation, assistive technology, Python automation*

1. INTRODUCTION

In today's fast-paced digital era, the ways we interact with computers are constantly evolving. From traditional input methods like keyboards and mice to modern interfaces like touchscreens and voice assistants, each advancement strives to make technology more seamless and intuitive. One of the emerging frontiers in this journey is **gesture control**— the ability to operate digital devices using simple hand movements, with no physical contact required.

This project explores that frontier by enabling users to control a computer using only hand gestures, captured in real-time through a standard webcam. Whether it's moving the mouse, clicking, scrolling through content, or adjusting settings like volume and brightness, everything can be done with natural gestures such as pinching or forming a fist.

To achieve this, the system leverages **Mediapipe**, a powerful real-time framework that detects hand landmarks and analyzes finger positions with high accuracy. By interpreting the spatial arrangement and movement of the hand, the system converts gestures into commands—essentially turning your hand into a smart, touch-free remote control.

2. LITERATURE SURVEY

Over the last decade, gesture recognition has gained significant traction as a means to build more natural and contactless ways of interacting with digital systems. The concept of controlling devices through hand gestures appeals to both usability and accessibility, especially in environments where conventional tools like keyboards or mice may be inconvenient or unusable.

A notable early advancement in this space was Microsoft's **Kinect**, which introduced mainstream gesture control by allowing users to interact with games and applications using full-body and hand movements. Kinect's depth-sensing capabilities enabled robust tracking of motion in three-dimensional space [1]. However, despite its effectiveness, such Hardware-driven solutions are often expensive and lack portability, limiting their widespread adoption. This led researchers to explore **vision-based gesture recognition** using standard RGB cameras, which offer a more affordable and flexible alternative. Various

techniques have been studied, such as analyzing **motion trajectories** [2], using **glove-based tracking systems** [3], or implementing **color segmentation approaches** [4].

While these methods showed early potential, they frequently encountered challenges like inconsistent lighting, low accuracy, and the inconvenience of requiring external wearables.

The field gradually shifted toward more advanced techniques that involve **hand landmark detection** and **automated feature extraction** through artificial intelligence. For instance, Pavlovic et al. presented a detailed review of gesture recognition techniques in Human-Computer Interaction (HCI), emphasizing the need for systems that are not only precise but also capable of operating in real time [5]. As machine learning evolved, researchers began leveraging **neural networks** to interpret gestures more effectively, removing the reliance on manual feature engineering [7].

A breakthrough came with the release of **Google's Mediapipe**, an open-source framework capable of detecting 21 hand landmarks in real-time using a standard webcam [8]. This significantly simplified the implementation of gesture-based interfaces by removing the need for depth sensors or complicated calibration processes.

To bridge the gap between gesture detection and system control, tools like **PyAutoGUI** have played an important role. They allow the mapped gestures to be directly converted into computer actions—such as mouse movements, clicks, or scrolling—making the interaction seamless and highly responsive [12].

There has also been growing interest in using gesture recognition to enhance **accessibility**. For users with physical disabilities, touchless systems provide an empowering alternative to traditional input devices [6]. Researchers have contributed by creating diverse **gesture datasets** and defining **evaluation benchmarks** to improve recognition accuracy and usability [14].

Recent innovations are extending these capabilities into **smart system control**, where gestures can be used to adjust environmental settings like screen brightness or audio volume using intuitive motions like pinching [15]. These developments not only enhance everyday interactions but also pave the way for smart, hygienic, and fully contactless computing experiences.

3. SYSTEM OVERVIEW

The fundamental goal of this system is to enable users to interact with their computer through **natural hand gestures**, eliminating the need for physical contact. Using just a standard webcam, it detects and interprets specific hand movements in real-time, mapping each gesture to corresponding computer functions such as **mouse movement**, **clicking**, **scrolling**, or adjusting **volume** and **brightness** levels.

At the core of the system lies **Google's Mediapipe framework** [8], a powerful tool that identifies **21 key landmarks** on each hand—including fingertips, knuckles, and joints. These landmarks provide accurate spatial data, allowing the program to determine the hand's position, orientation, and finger configurations with remarkable precision.

Once a hand is detected in the camera feed, the system performs **landmark analysis**, calculating distances and angles between key points. By interpreting which fingers are raised or curled, it can classify the hand pose using a **binary encoding scheme**, where each finger's state (open or closed) contributes to a unique gesture code.

For example:

- A **closed fist** could signal a "click" or "grab."
- A **two-finger 'V' sign** might be mapped to cursor movement.
- A **pinch gesture** can trigger scrolling or modify system settings like brightness and volume, depending on its direction.

To manage this efficiently, the gesture recognition logic is divided into two main components:

- **1. Hand Landmark Analyzer**: This module extracts and interprets landmark positions, determines which fingers are extended or bent, calculates inter-finger distances, and recognizes gesture patterns.

2. Command Controller: Based on the gesture identified by the analyzer, this module translates it into system commands using automation libraries such as **PyAutoGUI** [12]. These commands can control the mouse Pointer, simulate clicks, or interact with operating system functions.

To ensure **accuracy and user comfort**, the system incorporates **gesture confirmation through temporal filtering**. A gesture must remain stable for a predefined number of frames before it is executed. This prevents accidental triggers due to momentary hand movement or detection noise, making the experience smoother and more reliable.

An additional feature is the **support for dual-hand recognition**. The system can differentiate between the **primary (dominant) hand** and the **secondary hand**, enabling more complex interactions. For instance, the dominant hand can handle navigation, while the secondary hand can activate system-level commands or shortcuts—broadening the range of possible control actions.

In essence, this gesture-controlled system offers a **real-time**, **contact-free**, and **intuitive interface** for interacting with a computer. It holds great potential in diverse scenarios—whether it's assisting users with physical limitations, enhancing touchless interaction in smart environments, or simply providing a futuristic alternative to traditional input devices.

4. METHODOLOGY:

The approach adopted in this project follows a structured, multi-stage pipeline—from capturing hand gestures to translating them into actionable system commands. Each step is optimized for **real-time responsiveness**, **accuracy**, and **user comfort**, ensuring a seamless touchless interface.

1. Video Capture

The process begins with the system accessing the webcam through **OpenCV**, which streams live video frames continuously. To create a more intuitive user experience, each frame is **horizontally flipped**, producing a mirror-like effect where the user's movements directly correspond to on-screen behavior. This step ensures a natural interaction, similar to how one views oneself in a reflection.

2. Hand Detection Using Mediapipe

Each incoming video frame is processed using **Mediapipe's Hand Tracking module** [8]. This advanced model detects one or both hands in the frame and identifies **21 precise landmarks** per hand, including finger joints, tips, and palm centers. The detection operates in real time and adapts to hand orientation, scale, and motion with minimal latency.

Providing robust performance even under dynamic conditions.

3. Gesture Recognition

After detecting hand landmarks, the system proceeds to **analyze the spatial configuration** of the hand. It calculates distances and angles between key points to determine the pose and state of each finger.

A **binary encoding scheme** is applied to simplify gesture classification: each finger is marked as either "open" or "closed," creating a unique pattern for each gesture. These patterns are then matched against a predefined gesture library, including:

- **First** → Simulates a click or grab action
- **Open Palm** → Interpreted as idle or neutral (no action)
- **Pinch** → Controls brightness or volume based on movement direction
- **V Shape** → Enables cursor navigation
- **Two-Finger Close** → Triggers a double-click event

To ensure reliability, gestures must be **stable across several consecutive frames** before the system recognizes and executes them. This reduces false positives and avoids accidental commands.

4. Command Execution

Once a gesture is identified and validated, it is mapped to a corresponding system-level action using **PyAutoGUI** [12], a Python automation library. This module acts as the interface between gesture recognition and the operating system. Examples of gesture-to-action mappings include:

- **Cursor Movement**: Based on real-time coordinates of hand landmarks
- **Clicking**: Detected fist gesture simulates mouse click (left/right based on finger position)
- **Scrolling**: Achieved through vertical or horizontal pinch gestures
- **Volume/Brightness Control**: Triggered by a pinch gesture followed by finger movement, where the distance traveled adjusts the level up or down.

5. Hand Role Classification

To support more sophisticated interactions, the system leverages Mediapipe's **handedness detection** to classify hands as either **major (dominant)** or **minor (non-dominant)**. Typically, the right hand is treated as the primary control (e.g., cursor movement), while the left hand may trigger secondary functions like media control or shortcuts. This dual-hand functionality Enhances flexibility and usability, particularly in multitasking environments.

6. Stability and Filtering Mechanisms

To provide a smooth and responsive user experience, the system integrates several **stabilization techniques**:

- **Frame Count Filtering**: A gesture must persist for a set number of frames (e.g., 3–5) to be considered valid. This helps filter out momentary or accidental poses.
- **Movement Dampening**: To eliminate jittery behavior, especially during fast hand movement, a smoothing algorithm reduces erratic changes in position. This results in **fluid cursor control** and improved overall system stability.

Overall, the methodology enables a **real-time, touchless interface** for controlling a computer using only hand gestures. It combines powerful vision-based tracking with intelligent filtering and automation, making it suitable for a wide range of use cases—from **daily computing** to **assistive technology** and **smart environments**.

5. SYSTEM ARCHITECTURE / DESIGN OVERVIEW:

The system is built around a modular architecture that enables real-time detection, interpretation, and execution of hand gestures to control various computer functions. Each module in the pipeline performs a specific role, and together, they create a smooth,

responsive, and user-friendly gesture control system. The design emphasizes speed, accuracy, scalability, and minimal hardware requirements.

1. **Input Handling – Webcam Feed Acquisition.** The gesture recognition process starts with capturing live video from the user's webcam. This is accomplished using the **OpenCV library**, which continuously reads frames in real time. These frames are horizontally flipped to simulate a mirror view, making it easier for users to interact naturally—when they move their hand to the left, the on-screen cursor also moves left.
2. **Hand Detection – Mediapipe Tracking** The flipped video frames are processed through **Mediapipe's Hand Detection module** [8]. This stage is responsible for identifying hands within the frame and extracting **21 landmark coordinates** for each hand. These landmarks represent critical points such as fingertips, knuckles, and the wrist. This precise Tracking is the foundation for recognizing different hand gestures. Mediapipe can detect **single or dual-hand input**, allowing for more versatile interactions. It also performs well in varied lighting conditions and provides real-time output with low computational overhead.
3. **Gesture Analysis and Classification.** Once the hand landmarks are extracted, the system enters the **gesture recognition phase**. It analyzes the relative positions of the fingers to determine which are extended and which are folded. Using a **binary representation system**, the status of each finger (open or closed) is encoded to form a gesture ID. Each gesture is mapped to a specific computer command, such as:
   - Cursor navigation
   - Single and double clicks
   - Right-click
   - Vertical or horizontal scrolling
   - Volume and brightness control via pinch detection

   To ensure reliable performance, the system incorporates **gesture validation across multiple frames** to prevent accidental detections due to hand jitter or rapid transitions.
4. **Hand Role Assignment – Major vs. Minor Hand.** The architecture supports the use of **both hands** simultaneously. Using Mediapipe's handedness classification, the system labels each detected hand as either the **major (dominant)** or **minor (supporting)** hand. Typically, the dominant hand is used for core actions like moving the cursor or clicking, while the supporting hand performs auxiliary tasks like adjusting settings. This dual-hand strategy enables more advanced control without increasing complexity.
5. **Action Execution – PyAutoGUI Integration** After a gesture is recognized and verified, it is translated into an actual system-level command using the **PyAutoGUI** library [12]. PyAutoGUI handles:
   - Mouse cursor control
   - Simulated mouse clicks
   - Scroll commands
   - System volume or brightness adjustment (via custom scripts). For instance:
   - A **fist** can simulate a click or drag action.
   - A **V-sign** can trigger cursor movement.

A **pinch** gesture modifies system parameters depending on the direction and scale of movement.

1. Real-Time Visual Feedback

To assist the user and improve usability, the system includes a **visual display window** that shows the webcam feed along with **highlighted hand landmarks and outlines**. This real-time feedback helps users see how the system interprets their hand gestures, making the experience more intuitive and interactive.

Key Design Advantages

- **Modularity**: Each function is independently managed, making the system easy to debug, upgrade, or expand.
- **Efficiency**: Optimized for real-time performance with minimal delay, suitable for laptops or low-resource devices.
- **Scalability**: Easily supports new gestures or command mappings without overhauling the entire system.
- **User-Centric Design**: Real-time feedback ensures that users always know how their gestures are being interpreted.

Fig.1. Hand Gesture Control System Flowchart

6. FUNCTIONAL OVERVIEW:

The proposed system introduces an intuitive, contact-free method for controlling a computer using **simple hand gestures**, eliminating the need for a traditional keyboard or mouse. With only a webcam, the system functions in real-time and can accurately interpret various hand poses to trigger specific system-level actions.

1. Hand Movement Capture

The process begins with the **webcam capturing live video** of the user's hand. This continuous video stream serves as the primary input for the system. It tracks the user's hand as it moves across the frame, preparing the data needed for gesture detection.

2. Hand Detection and Landmark Extraction

Each video frame is processed using **Google's Mediapipe framework** [8], which specializes in real-time hand tracking. The system detects the presence of one or two hands and extracts **21 specific landmark points** per hand. These include finger tips, knuckles, and palm positions, which are essential for analyzing hand posture and movement dynamics.

3. Gesture Recognition

Once the landmarks are identified, the system analyzes finger positions to determine which fingers are extended or folded. This is encoded using a **binary logic system**, creating a unique signature for each gesture. These patterns are then matched to a predefined gesture set, including:

- **First** → Simulates click or drag operations
- **V-Shape (two fingers open)** → Controls cursor movement
- **Pinch Gesture** → Adjusts brightness or volume
- **Two Fingers Closed** → Triggers a double-click

This process ensures that every gesture has a distinct and recognizable pattern, minimizing confusion or overlap.

4. **Role Classification: Major and Minor Hand.** To enhance control flexibility, the system supports **dual-hand detection**. It classifies one hand as the **major hand**, typically used for primary navigation and interaction, while the **minor hand** is used for secondary tasks like scrolling or adjusting settings. This distinction allows users to perform **multitask operations** with ease.
5. **Command Mapping and Action Triggering** Once a gesture is confirmed, it is mapped to a specific system function through the **PyAutoGUI** library [12], which allows the program to simulate keyboard and mouse inputs. Common mappings include:
    - Cursor movement based on hand position Mouse click triggered by a closed fist
    - Brightness/volume control via pinch movement
    - Double-click when two fingers are held closely together

These mappings allow seamless integration between **gesture detection** and **system control**, providing a real-time, interactive experience.

6. **Execution and Visual Feedback** After the gesture is validated and the command is triggered, the system provides immediate **visual feedback** to the user. The camera feed displays hand outlines and detected landmarks using Mediapipe's drawing tools, allowing users to see how the system interprets their gestures in real-time. This **real-time visualization** helps build user confidence and improves usability.

7. RESULTS AND ANALYSIS:

Fig.2. Launching the Gesture Controller

In this stage, the system is successfully launched from the command line using Python. The script Gesture_Controller.py begins execution, and the webcam feed is activated in a window titled "Gesture Controller." This confirms that the system is ready to detect hand gestures. The message Created TensorFlow Lite XNNPACK delegate for CPU indicates that the Mediapipe backend is properly initialized, and real-time gesture processing is set to begin.

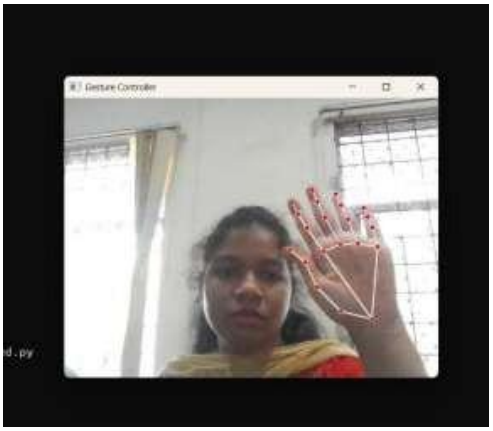

Fig.3. Gesture Detection and Cursor Control

Here, the system detects the user's raised hand and displays the 21 Mediapipe hand landmarks, connecting key points on the fingers and palm. This confirms that the hand is properly recognized and tracked. The landmarks are continuously updated as the hand moves, enabling the system to interpret specific gestures. In this case, the system responds to finger positions and moves the cursor based on the hand's location, allowing the user to navigate the computer without touching the mouse.

Fig.4. Application Control with Hand Gestures

In this final stage, the system detects a specific gesture (e.g., "V" gesture or pinch) and triggers a command to open an application. This proves that the gesture recognition logic is accurately mapped to custom actions such as launching files or software.

The hand is tracked with Mediapipe landmarks, and the gesture command is successfully interpreted, showcasing real-time control over the desktop environment using only hand gestures.

8. COMPARING WITH EXISTING SYSTEMS

Over the years, gesture-based systems have evolved significantly, offering new ways to interact with digital environments. However, many of the earlier systems had notable limitations that affected usability, accessibility, and practicality.

One of the most well-known examples is the **Microsoft Kinect** [1], which introduced gesture control using depth-sensing cameras. While it delivered reasonably accurate body and hand tracking, it relied on specialized hardware, making it less feasible for widespread personal or portable use.

Similarly, **glove-based systems** [3] used embedded sensors to precisely detect finger positions and hand movements. Although they offered high precision, these systems were cumbersome, often uncomfortable for extended use, and impractical for everyday tasks like browsing or media control. Early **vision-based gesture recognition techniques** [2][4] also had their drawbacks. Many depended on color segmentation or basic motion detection, which performed poorly under inconsistent lighting conditions or in cluttered environments with background noise. These limitations significantly reduced the reliability of such systems in real-world settings.

In contrast, the system developed in this project is designed for **practicality, ease of use, and wide accessibility**. It eliminates the need for external sensors or wearable devices, relying solely on a **standard webcam** and the **Mediapipe framework.**

[8] to detect and analyze hand landmarks in real-time. This makes it suitable for both personal and professional environments, including home computing, office work, and even accessibility-focused use cases.

Another common issue in older systems was the **lack of real-time performance**, often caused by the use of complex algorithms or reliance on cloud-based processing. The proposed system addresses this challenge by running completely on the local machine, utilizing **TensorFlow Lite** in conjunction with the **XNNPACK delegate** for efficient CPU processing. As a result, gesture recognition is performed with **minimal latency**, ensuring smooth, real-time responsiveness.

What further sets this system apart is its ability to **translate recognized gestures into actual system-level controls** — such as cursor movement, clicks, scrolling, and volume or brightness adjustments. This is made possible through integration with **PyAutoGUI** [12], bridging the gap between recognition and real-world functionality. Moreover, unlike simpler gesture recognition tools, the system also supports **dual-hand interaction**. By identifying the **major and minor hands**, it enables richer, multi-dimensional controls. For instance, the user can navigate with one hand while adjusting volume or scrolling content with the other [14], offering a more natural and versatile human-computer interaction experience.

Table 1: Comparison with Existing and Proposed Systems

Feature	Existing systems	Proposed system
Hardware requirement	External sensors/gloves	Only a webcam
Accuracy	Affected by lighting	Robust with meidapipe landmarks
Gesture to action mapping	Limited or demo only	Fully functional system control
RealTime Response	Some lag or delay	CPU-optimized real-time tracking
Hand Role Classification	Mostly absent	Included (major/minor hands)
Accessibility	Low	High, user-friendly, contactless

9. CHALLENGES AND LIMITATIONS:

While the gesture control system developed in this project shows promising results and offers a contactless way to interact with a computer, several challenges and limitations were encountered during its implementation and testing phases:

1) **Sensitivity to Lighting Conditions.** The performance of the system is significantly influenced by the surrounding lighting. In dimly lit or overly bright environments, the webcam may fail to capture the hand clearly, making it difficult for the Mediapipe model to accurately detect and track hand landmarks. As a result, gestures may be misinterpreted, delayed, or entirely unrecognized. Ensuring consistent and adequate lighting is therefore important for stable performance.

2) **Limited Set of Recognized Gestures.** At present, the system supports only a specific number of hand gestures that are manually defined and hard-coded. This limits the flexibility and scope of

Interactions. Incorporating more diverse or customizable gestures—such as those unique to a user's preference or more complex motion patterns— would require building or training a more sophisticated gesture classification model, potentially involving machine learning or deep learning techniques.

3) **Difficulty with Background Noise and Multiple Users**

The system assumes that a single user's hand is present and visible in the camera frame. In environments where multiple people are present, or if the background contains moving objects or similar hand-like shapes, the system may mistakenly detect gestures from the wrong source. This can result in unwanted commands being triggered or interruptions in user control. Filtering or prioritizing the primary user's hand remains a technical challenge.

10. CONCLUSION AND FUTURE WORK:

This project has successfully implemented a real-time hand gesture recognition system that enables users to control their computer without the need for any physical contact. Utilizing only a standard webcam and the Mediapipe framework, the system efficiently tracks hand landmarks and interprets various predefined gestures to trigger corresponding system actions— such as moving the cursor, clicking, scrolling, and adjusting brightness or volume levels. The inclusion of binary gesture encoding, along with major and minor hand classification, enhances the versatility and responsiveness of the system. Its simplicity, combined with the minimal hardware requirements, makes it accessible for a wide range of users. Notably, this technology offers significant advantages in improving accessibility for individuals with physical impairments and provides a hygienic alternative to traditional input devices—an important factor in public or shared computing environments. By effectively bridging the gap between gesture recognition and everyday usability, the project showcases the potential of intuitive human-computer interaction with low-cost and readily available tools.

Future Work

While the current system is functional and reliable, there are several promising directions in which it can be further enhanced:

- **Customizable Gesture Training:** Introduce functionality that allows users to create and train their own gesture sets. This would enable a more personalized experience and accommodate unique interaction preferences or cultural gestures.
- **AI-Driven Learning and Adaptation:** Integrate machine learning models capable of adapting to user behavior over time. This would help the system improve its accuracy and responsiveness through continuous, offline learning without needing constant updates or retraining.
- **Cross-Platform Integration:** Expand the system's compatibility to work with a wider range of devices, such as smart TVs, mobile phones, AR/VR setups, and IoT-based smart home environments. This would broaden its application in areas like home automation, entertainment, and assistive technology.

REFERENCES

[1] Zhang, Z. (2012). *Microsoft Kinect sensor and its effect*. IEEE Multimedia, **19**(2), 4–10.

[2] Mittal, A., Zisserman, A., & Torr, P. H. S. (2011). *Hand gesture recognition using motion trajectories*. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 411–424).

[3] Wang, R. Y., & Popović, J. (2009). *Real-time handtracking with a color glove*. ACM Transactions on Graphics, **28**(3), 1–8.

[4] Freeman, W. T., & Weissman, C. D. (1995). *Television control by hand gestures*. In *IEEE International Workshop on Automatic Face and Gesture Recognition* (pp. 179–183).

[5] Pavlovic, V. I., Sharma, R., & Huang, T. S. (1997). *Visual interpretation of hand gestures for human-computer interaction*. IEEE Transactions on Pattern Analysis and Machine Intelligence, **19**(7), 677–695.

[6] Ong, S. C. W., & Ranganath, S. (2005). *Automatic sign language analysis: A survey and the future beyond lexical meaning*. IEEE Transactions on Pattern Analysis and Machine Intelligence, **27**(6), 873–891.

[7] Carfagni, M., Furferi, R., Governi, L., & Volpe, Y. (2020). *Real-time gesture recognition using convolutional neural networks and transfer learning*. Procedia Manufacturing, **42**, 526–532.

[8] Google Developers. (n.d.). *MediaPipe Framework*. Retrieved from https://mediapipe.dev

[9] Van den Bergh, M., & Van Gool, L. (2011). *Combining RGB and ToF cameras for real-time 3D hand gesture interaction*. In *IEEE Workshop on Applications of Computer Vision (WACV)* (pp. 66–72).

[10] Arora, M., & Tiwari, P. (2020). *Hand gesture recognition using computer vision*. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, **6**(2), 2456–3307.