Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Gesture-based Smart Home Control for Special Needs Individuals Using Computer Vision

Kareem Mohamed

kmohamed35@gatech.edu

Georgia Institute of Technology

Walaa Abo Elenin

Georgia Southern University

Yiming Ji

Kennesaw State University

Research Article

Keywords: Smart Home Control, Vision-Based Interaction, Hand Gesture Recognition, Real-Time Control Systems, Assistive Technologies, Speech Impairment Accessibility, Embedded Systems, Human–Computer Interaction

Posted Date: July 15th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10197900/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Gesture-based Smart Home Control for Special Needs Individuals Using Computer Vision

Kareem Atef Mohamed*
College of Eng. and Computing
Georgia Institute of Technology
Atlanta, Georgia
kmohamed35@gatech.edu

Walaa Abo Elenin
College of Eng. and Computing
Georgia Southern University
Statesboro, Georgia
waboelenin@georgiasouthern.edu

Yiming Ji
College of Computing and Software Eng.
Kennesaw State University
Kennesaw, Georgia
yji9@kennesaw.edu

***Abstract*—Providing reliable and accessible interaction with smart home systems remains a significant challenge, particularly for individuals with speech impairments and other disabilities that restrict the effective use of conventional voice-based control mechanisms. Building upon our previously published hand gesture recognition model in [1], this paper extends the gesture classification framework into a complete vision-based smart home control system that performs real-time camera-based gesture recognition, user intent interpretation, and action execution within a smart home environment. The proposed framework employs the previously developed CNN-based gesture recognition model, trained on a curated subset of a public hand gesture dataset, as the perception component of an end-to-end control architecture. The trained model is integrated into an OpenCV-based real-time inference pipeline that processes live video streams from a standard RGB camera and translates recognized gestures into high-level control intents. These intents are subsequently mapped to smart home actions through a modular action-handling layer that supports both local hardware validation and extensible REST API–based smart home integration. Experimental evaluation demonstrates high gesture classification performance, reliable real-time operation of the inference pipeline, and successful end-to-end validation using an Arduino-based hardware prototype and simulated smart home control scenarios. The results demonstrate that the proposed framework provides an effective and scalable approach for accessible, non-verbal smart home interaction, extending vision-based gesture recognition beyond classification to practical real-time control of connected devices and services.**

***Index Terms*—Smart Home Control; Vision-Based Interaction; Hand Gesture Recognition; Real-Time Control Systems; Assistive Technologies; Speech Impairment Accessibility; Embedded Systems; Human–Computer Interaction.**

I. Introduction

The increasing adoption of smart home technologies has transformed how users interact with and control physical living environments, placing greater emphasis on effective and inclusive interaction mechanisms [1]; however, commonly adopted interaction modalities such as voice commands, mobile applications, and touch-based interfaces are not universally accessible. Such interaction mechanisms present significant barriers for individuals with speech impairments [30], for whom voice-dependent and conventional input methods can limit independent and efficient system use [8] [12]. Gesture-based interaction has emerged as a viable non-verbal input modality, enabling more intuitive system control for users who cannot effectively rely on speech and for those who face challenges with conventional interfaces [4] [12] [21]. Such limitations extend beyond usability concerns, as reliance on traditional interaction mechanisms can reduce independent access to smart home technologies and constrain daily autonomy for users with accessibility challenges [3] [12].

Our previous work in [1] introduced a vision-based hand gesture recognition framework as a foundational step toward assistive computing. The proposed system employed a deep learning–based convolutional neural network trained on a public hand gesture dataset comprising 24 gesture classes and demonstrated strong classification performance across multiple evaluation metrics. That work focused primarily on the gesture recognition component as a perception layer for assistive human–computer interaction and established a conceptual foundation for gesture-driven smart home applications.

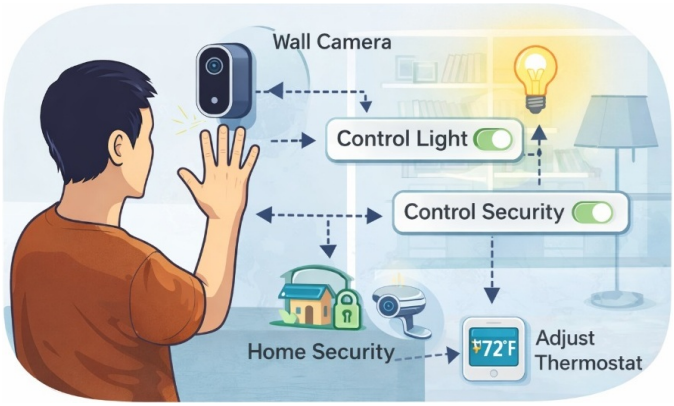


Fig. 1. Conceptual overview of the proposed vision-based smart home control framework, where hand gestures captured by an RGB camera are translated into lighting, climate, and security control actions.

Building upon this foundation, the present work extends gesture recognition beyond perception to a complete smart home control framework, as illustrated in Fig. 1. The proposed system integrates real-time camera-based gesture recognition, intent interpretation, and control execution within a unified architecture that operates using only standard RGB camera input, eliminating the need for specialized sensors or wearable devices [9] [11] [25]. Leveraging advances in deep learning

[35] and computer vision [20], the framework translates recognized gestures into smart home actions through a modular control layer that supports both Arduino-based hardware prototyping and extensible REST API–based integration. This design enables practical, accessible, and scalable non-verbal interaction for users with speech impairments while providing a deployable pathway toward real-world smart home environments.

The main contributions of this work are as follows:

- Extension of our previously developed 24-class gesture recognition model into a complete smart home control framework.
- A conceptual system-level framework that maps smart home control functions to gesture-based user intents, spanning intent capture, recognition, and action execution.
- A hardware prototype, in which Arduino-based LED actuation serves as a physical simulation of the framework's action layer, validating the intent-to-action pipeline under controlled conditions.
- A REST API–based integration pathway, illustrated through three representative use cases (door lock, security, and lighting control) grounded in existing smart home communication literature.
- Comprehensive evaluation including classification performance, real-time inference, prototype action execution, and a nine-participant user study.

The remainder of this paper is organized as follows. Sec. II reviews relevant background and related work across vision-based gesture recognition, smart home control systems, and assistive interaction technologies. Sec. III describes the gesture recognition foundation underlying the system, including dataset preparation, model architecture, and training. Sec. IV introduces the proposed smart home control framework at the conceptual level, defining the system architecture, intent interpretation logic, and the pathway to smart home integration. Sec. V presents the prototype implementation, covering the real-time vision processing pipeline, the Arduino-based hardware prototype, and an analysis of system latency and reliability. Sec. VI details the REST API–based integration pathway, including its technical design and three representative smart home use cases. Sec. VII reports the experimental evaluation, including gesture recognition performance, confusion matrix analysis, prototype validation, and the user study. Sec. VIII discusses design tradeoffs, scalability, and limitations, and Sec. IX concludes the paper with directions for future work.

II. Background and Related Work

This section surveys the existing literature underpinning the proposed framework across three complementary areas. It begins by reviewing vision-based hand gesture recognition techniques and the perception-level tradeoffs they introduce. It then examines gesture-driven smart home control systems, focusing on how prior work integrates gesture recognition with device actuation. Finally, it considers assistive interaction technologies more broadly, positioning hand gesture control alongside other accessibility-oriented input modalities. Collectively, these studies reveal recurring challenges, including tight coupling between perception and actuation, dependence on specialized hardware, and limited emphasis on accessibility, that motivate the modular, end-to-end smart home control framework proposed in this work.

A. Vision-Based Gesture Recognition

Vision-based gesture recognition has been widely explored for real-time device control using low-cost hardware [14]. Yashwanth et al. proposed a real-time hand gesture recognition system that integrates OpenCV and MediaPipe for hand landmark extraction, enabling gesture-driven control of external devices such as LEDs through an Arduino microcontroller [5]. Their approach relies on detecting finger landmarks and applying rule-based logic to map hand configurations to predefined commands, achieving low-latency operation suitable for interactive environments. While effective for basic control tasks, the system tightly couples gesture detection with action execution and relies primarily on landmark heuristics rather than end-to-end learned visual representations.

Broader surveys of vision-based gesture recognition reinforce this distinction: techniques ranging from depth-sensing and skeleton-based approaches to convolutional architectures have been applied to human–computer interaction tasks, with handcrafted and landmark-based pipelines generally trading robustness for interpretability and low latency, while learned convolutional approaches trade some interpretability for stronger generalization across lighting, orientation, and background conditions [20] [21] [28]. In contrast to the landmark-driven and rule-based systems discussed above, the proposed framework employs a fully learned, frame-based convolutional model that operates directly on RGB images. Furthermore, the framework separates gesture recognition from intent interpretation and action handling through a modular architecture. This design allows the perception component to remain independent of the actuation logic associated with any specific smart home device, as described later.

B. Smart Home Control Systems

Several studies have explored vision- and sensor-based gesture control specifically targeting smart home environments rather than general device interaction. Wavale and Sireesha proposed an edge machine learning–based system that employs an MPU6050 inertial measurement unit and ESP32 microcontroller to recognize hand gestures from time-series motion data [6]. Their system performs offline inference using models trained and deployed via the Edge Impulse platform and utilizes ESP-NOW for low-latency, cloud-free communication between devices. While this approach provides strong robustness to lighting conditions and preserves user privacy, it requires users to wear dedicated hardware and constrains interaction to motion-based gestures, in contrast to the passive, vision-based approach pursued in the present work.

Rajanala et al. proposed a vision-based gesture recognition system for controlling electrical appliances using a Raspberry Pi Pico and relay modules, where static hand gestures are captured via camera and mapped to appliance control commands through OpenCV-based processing and predefined gesture logic [10]. Their system demonstrates effective control of lights and fans under controlled indoor conditions and reports satisfactory response times, but its reliance on handcrafted rules and direct gesture-to-actuation coupling limits scalability as the number of devices grows, and performance degrades under varying illumination and complex backgrounds. Vijayaraghavan et al. similarly demonstrated gesture-controlled home automation using a Raspberry Pi platform [9], Shruthi et al. introduced a touchless multimodal interface combining gesture and voice control for smart home interaction [4], and Tsai et al. explored skeleton-based dynamic gesture recognition for home appliance control [13]. Beyond purely gestural interfaces, Leong investigated multimodal gesture-and-speech interaction for AI-driven mobile robots operating in smart home settings [12]. Additional vision-based smart home control systems have examined alternative interaction modalities and application scenarios, including depth- and skeleton-based recognition pipelines and learning-based action recognition frameworks [26], [27], [29], [31]– [34] [37] [39], [40].

Collectively, these systems demonstrate strong potential for touchless smart home control but exhibit recurring limitations related to scalability, environmental robustness, hardware dependency, and tight coupling between perception and actuation. Vision-based approaches often rely on handcrafted rules or landmark heuristics, while sensor-based systems require wearable hardware that constrains user convenience. These gaps motivate the modular, learned-perception framework proposed in this work, in which gesture recognition, intent interpretation, and action execution are explicitly decoupled, as developed in Sec. IV.

C. Assistive Interaction Technologies

Beyond hand-based interaction, alternative gesture modalities and accessibility-focused systems have been investigated to support users with disabilities. Sakthimohan et al. introduced an eye-gesture–driven assistive system that leverages a webcam, OpenCV, and MediaPipe to detect eye movements and translate them into appliance control commands via an Arduino platform [2]. This approach significantly improves accessibility for users with severe motor impairments by enabling hands-free interaction; however, eye-gesture systems require careful calibration, controlled lighting conditions, and impose higher cognitive load during prolonged use, and are inherently limited in gesture vocabulary and command expressiveness.

Neacsău et al. applied YOLOv5-based object detection to hand gesture recognition for people with disabilities, demonstrating the applicability of modern object-detection architectures to accessible gesture interfaces [3]. Mohamed et al. provide a broader review of hand gesture recognition systems, surveying current progress and outlining future directions for accessibility-oriented applications [11], while Melo et al. present a computer vision based methodology specifically aimed at improving interaction for individuals with motor and speech impairment [29]. Adaptive vision-based interfaces have also been explored for elderly and bedridden care, including personalized lighting control through computer vision [15] [16].

In contrast to eye-gesture and object-detection-based approaches, the proposed system focuses on hand-based static gestures, which offer richer semantic expressiveness and lower cognitive overhead than eye-tracking interfaces, while remaining accessible through passive, camera-only interaction that requires no wearable hardware. By supporting real-time inference, modular system design, and scalable smart home integration, the proposed framework balances accessibility, usability, and deployment practicality across a broader range of users and environments than the systems surveyed above.

III. Gesture Recognition Foundation

A consistent commonality runs through existing systems: gesture perception and actuation are built as a single, inseparable unit, which limits how easily any one component can be reused or scaled. The framework proposed here starts from the opposite premise. Before any control logic is introduced, gesture recognition is developed and validated entirely on its own terms, as a standalone capability with no dependency on a specific actuation target. Dataset preparation, model architecture, and training are therefore presented in isolation here, with the smart home control logic deferred to a later section.

A. Dataset and Preprocessing

The gesture recognition model was trained using the publicly available ASL Alphabet Dataset obtained from Kaggle, which contains RGB images representing American Sign Language hand gestures. The original dataset contains 26 alphabet gesture classes; however, the dynamic gestures J

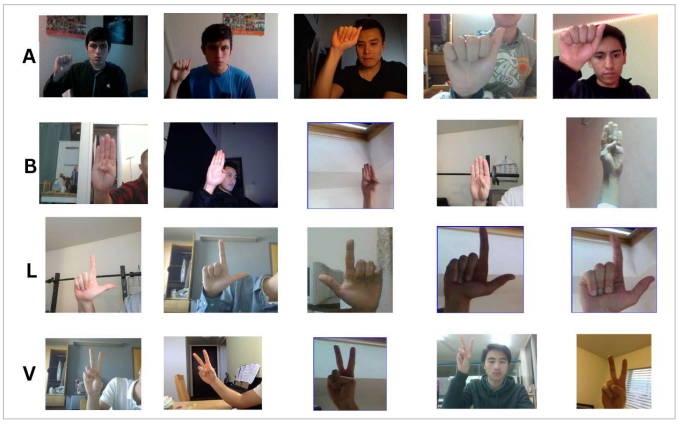


Fig. 2. Representative sample images from the filtered ASL hand gesture dataset illustrating the four selected static gesture classes (A, B, L, and V) under varying lighting conditions, hand orientations, and background appearances.

and Z were excluded because their recognition depends on temporal motion patterns rather than static hand configurations [38]. Since the proposed system operates on individual video frames and does not incorporate temporal or sequence-based modeling, only static gestures were considered [32] [33]. Consequently, the final training dataset consisted of 24 static gesture classes, supporting efficient real-time recognition with low computational overhead. The dataset follows a directory-based organization in which each gesture class is stored in a separate folder, enabling automatic label assignment and efficient dataset management using standard deep learning frameworks.

The complete 24-class static gesture set was used to train the gesture recognition model, resulting in approximately 190,000 labeled images. The dataset was partitioned into training and validation subsets using an 80/20 split, yielding approximately 153,000 training images and 38,000 validation images. As illustrated in Fig. 2, the dataset exhibits substantial visual diversity, including variations in lighting conditions, hand orientation, scale, and background appearance, which helps improve model generalization and robustness under real-world operating conditions.

Prior to training, all images were resized to 160×160 pixels to match the input requirements of the MobileNetV2-based architecture. Pixel values were normalized using the same preprocessing strategy employed during real-time inference, ensuring consistency between offline model training and online deployment. No explicit background subtraction, hand segmentation, or handcrafted feature extraction techniques were applied; instead, the convolutional neural network was trained to learn discriminative visual features directly from the input images [19].

While the gesture recognition model was trained to classify all 24 static ASL gesture classes contained in the preprocessed dataset, only four representative gestures (A, B, L, and V) are mapped to smart home control intents in the proposed framework. Dynamic gestures were intentionally excluded from consideration throughout, since the system operates on individual video frames without temporal sequence modeling, reducing computational complexity while supporting reliable real-time interpretation.

B. MobileNetV2-Based Recognition Model

The gesture recognition component is implemented as a convolutional neural network developed using TensorFlow and Keras [18]. Its architecture, shown in Fig. 3, is based on MobileNetV2, selected specifically for its hardware-efficient computational structure rather than peak classification accuracy alone [9] [17] [18] [36]. MobileNetV2 employs depthwise separable convolutions, which factorize a standard convolution into a depthwise spatial convolution followed by a pointwise 1×1 convolution. This factorization reduces the multiply-accumulate operation count by a factor of $\frac{1}{N} + \frac{1}{D_K^2}$, where N is the number of output channels and D_K is the kernel size, yielding an 8–9$\times$ reduction in computational cost relative to standard convolutions of equivalent receptive field. For a

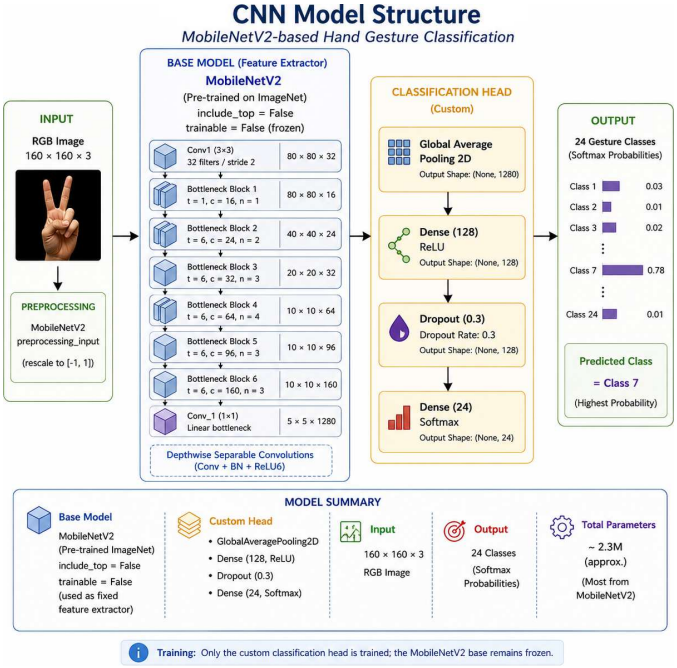


Fig. 3. Architecture of the utilized MobileNetV2-based CNN model for hand gesture recognition, consisting of a pre-trained MobileNetV2 feature extractor and a custom classification head with global average pooling, dense, dropout, and softmax layers for 24-class gesture classification.

3×3 kernel and typical channel depths, this corresponds to a backbone operation count of approximately 300 million multiply-accumulate operations per forward pass, a workload sustainable at 30 frames per second on standard consumer CPU hardware without dedicated neural processing hardware or GPU acceleration.

The pretrained MobileNetV2 backbone serves as a fixed feature extractor: its original classification layers are removed, and the convolutional base is frozen during training to preserve low- and mid-level visual features, such as edges, contours, and hand shape patterns, learned from ImageNet pretraining. High-level feature maps produced by the backbone are reduced to a compact representation through Global Average Pooling, then passed through a fully connected layer of 128 neurons with ReLU activation, a dropout layer with rate 0.3 for regularization, and a final softmax layer with 24 output neurons corresponding to the trained ASL gesture classes. This transfer-learning approach reduces training time and computational overhead while preserving strong feature representation [24], making the architecture well-suited to real-time gesture recognition. Although the model is trained to recognize all 24 gesture classes, only four gestures (A, B, L, and V) are used for control mapping in the proposed framework, as discussed later.

C. Model Training

The gesture recognition model is trained using the Adam optimizer with a learning rate of 0.001, and sparse categorical cross-entropy is employed as the loss function, consis-

TABLE I
SMART HOME CONTROL FUNCTIONS AND GESTURE MAPPING

Function	Gesture	Demonstrated In
Door Lock/Unlock	V	Hardware (Sec. V), REST Case 1
Security Arm/Disarm	L	Hardware (Sec. V), REST Case 2
Lighting Control	A	Hardware (Sec. V), REST Case 3
Climate Control	B	Hardware (Sec. V)

tent with the integer-encoded class labels produced by the directory-based dataset structure. Training is conducted in mini-batches of 16 samples for 10 epochs, with validation performance monitored at each epoch to assess generalization behavior. Early stopping and model checkpointing are applied throughout training to reduce overfitting and preserve the best-performing weights observed during validation.

Because the MobileNetV2 backbone is frozen and only the custom classification head is updated, the model converges rapidly, with stable validation accuracy observed early in training. This configuration provides a favorable balance between recognition accuracy and computational efficiency, aligning with the system's requirement for responsive real-time operation. Quantitative training and validation results, including accuracy and loss curves, are reported in Sec. VII-A.

IV. PROPOSED SMART HOME CONTROL FRAMEWORK

Smart home control is treated here as a mapping problem: given a finite set of functions a user may wish to invoke, the framework defines a structured pathway from human intent to physical or networked action. This pathway is organized around four design steps, described independently of any specific hardware or network implementation:

1) **Mapping user intent to gestures.** Each smart home function targeted for gesture control is associated with a single, geometrically distinct static hand gesture, selected to minimize visual ambiguity and cognitive load for the user.
2) **Capturing gestures from user.** A standard RGB camera continuously observes the user, requiring no wearable sensors or specialized hardware, consistent with the accessibility goals outlined in Sec. I.
3) **Identifying intent from the captured gesture.** The trained recognition model classifies the observed gesture and, subject to a confidence threshold, resolves it into a discrete control intent.
4) **Integrating intent into physical action.** A software action-handling layer translates the resolved intent into either a local hardware command or a networked REST API request, depending on the deployment target.

Table I illustrates this mapping for a representative set of smart home control functions relevant to users with speech impairments, distinguishing between the four-gesture set used in the hardware prototype and the subset additionally illustrated through REST API use cases.

A. System Architecture

The high-level interaction concept is illustrated in Fig. 4, in which a user's hand gestures, captured by a standard RGB camera, are processed by a vision-based gesture recognition system and interpreted as control intents for lighting, climate, and security functions. Fig. 5 expands this concept into a complete system architecture organized into three sequential phases: (1) offline model development, in which a curated subset of a public hand gesture dataset is used to train the MobileNetV2-based recognition model described in Sec. III; (2) real-time gesture recognition, in which live camera frames are processed continuously through an OpenCV-based inference pipeline; and (3) action execution, in which recognized gestures are resolved into control intents and dispatched through one of two parallel pathways, local hardware prototyping or REST API–based smart home control.

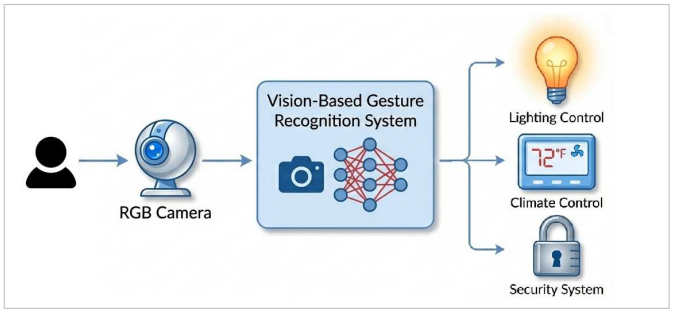


Fig. 4. Conceptual overview of the proposed vision-based smart home control framework. A user performs hand gestures that are captured by a standard RGB camera and processed by a vision-based gesture recognition system. Recognized gestures are interpreted as control intents and mapped to smart home functions, including lighting, climate, and security control, enabling real-time, touchless, and non-verbal interaction without requiring specialized sensors or wearable devices.

This architecture deliberately separates offline model training from online gesture interpretation and action handling. Because the recognition model is trained once and treated as a fixed component during deployment, the system can be extended with new control mappings or new actuation targets without retraining, an architectural decision revisited in the prototype implementation.

B. Intent Capture from Users

Building on the architecture established in Sec. IV-A, this subsection addresses the second design step introduced at the start of this section: capturing a gesture from the user before it can be interpreted. The framework requires only a standard RGB camera as the capture device, consistent with the accessibility goal of avoiding wearable sensors or specialized hardware. Capture is continuous rather than triggered, meaning the camera observes the user at all times during system operation, and gesture presentation is user-initiated: no explicit start signal, button press, or voice cue is required to begin a capture event. This passive, always-on capture model shifts the responsibility for detecting meaningful gestures from the user to the gesture recognition and intent interpretation components. Consequently, users are not required to perform an explicit activation gesture before issuing commands. This design improves accessibility by eliminating unnecessary motor or verbal coordination.

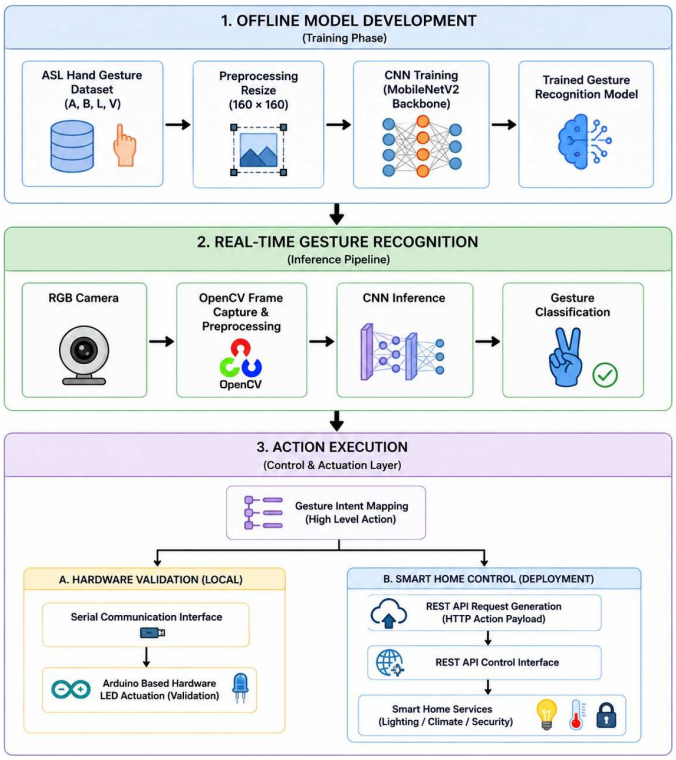


Fig. 5. End-to-end system pipeline for the proposed vision-based smart home control framework, illustrating offline dataset preparation and model training, real-time camera-based gesture inference, and parallel action handling through local hardware validation and extensible REST API–based smart home control.

The implementation details of the capture process, including frame rate, image resolution, and the OpenCV-based acquisition loop, are presented in the prototype implementation section.

C. Intent Interpretation and Action Mapping

Once a gesture is classified, it must be resolved into a discrete control intent before any action is taken. The framework adopts an argmax decision rule over the softmax output of the recognition model, subject to a confidence threshold τ that governs the tradeoff between command responsiveness and false activation rate. At confidence values below τ, the system becomes increasingly susceptible to spurious command generation during transitional hand poses, which occur naturally as a user moves between discrete gesture states; at confidence values well above τ, the system imposes stricter activation requirements that reduce false positives but increase the sustained gesture presentation time required to trigger a valid action. For the present system, $\tau = 0.80$ was selected as the operating point that suppresses inter-gesture transition noise while remaining consistent with the sub-100ms end-to-end response budget targeted in Sec. V-C .

Beyond thresholding, the action mapping layer applies a change-detection rule: a control action is only dispatched when the resolved intent differs from the most recently transmitted intent, preventing command flooding under sustained gesture presentation. This rule applies uniformly across both downstream pathways, ensuring that the hardware prototype and the REST API integration share identical intent-resolution logic and differ only in how a resolved intent is ultimately executed.

D. Smart Home Integration

To validate that a resolved intent can be reliably converted into action, the framework supports two complementary integration pathways, illustrated conceptually in Fig. 5. The first is a local hardware pathway, in which intents are transmitted over a serial interface to a microcontroller that performs physical actuation; this pathway is used in the present work as a controlled, observable simulation of smart home actuation and is described in full in Section V. The second is a networked REST API pathway, in which intents are encapsulated as HTTP requests dispatched to a smart home backend or IoT platform such as Home Assistant or openHAB, enabling integration with commercial smart home ecosystems without modification to the gesture recognition or intent-resolution components; this pathway is described in Sec. VI.

The four functions in Table I are mapped to gestures drawn from the 24-class recognition model described in **Sec.** III. All four functions are demonstrated through the hardware prototype in Section V; three of the four, excluding climate control, are additionally illustrated through REST API use cases in Sec. VI, selected because lighting, security, and access control represent the categories of smart home function most frequently identified in the assistive technology literature as priorities for non-verbal control [3] [8] [12] [30].

V. Prototype Design and Implementation

Of the two integration pathways defined in the proposed framework, this section instantiates the first: a working prototype that converts a resolved intent into physical action through dedicated hardware. Sec. V-A implements the real-time vision processing pipeline that performs the recognition and intent-resolution steps; Sec. V-B describes the Arduino-based hardware prototype that physically simulates action execution; and Sec. V-C characterizes the latency and reliability of the resulting end-to-end pipeline.

A. Real-Time Vision Processing Pipeline

Here, we introduce the real-time vision processing pipeline that transforms continuous camera input into discrete gesture predictions, implementing the second phase of the architecture. The trained model is loaded from disk using Keras' `load_model()` function, and live video is acquired from a laptop-integrated RGB webcam via OpenCV's `VideoCapture` interface. Frames are processed independently in a streaming manner, with no temporal buffering or sequence-based modeling, and inference is throttled to 30 frames per second to maintain stable, low-latency operation on standard consumer hardware.

For each inference cycle, shown in Fig. 6, the captured frame is converted from BGR to RGB color space, resized to 160×160 pixels, and normalized using the

```python
import cv2, time, numpy as np, tensorflow as tf
from tensorflow.keras.models import load_model
model = load_model(r"C:\best_model.h5")
IMG_SIZE = (160, 160)
CLASS_NAMES = [
    "A","B","C","D","E","F","G","H","I","K","L","M",
    "N","O","P","Q","R","S","T","U","V","W","X","Y"
]
cap = cv2.VideoCapture(0)
if not cap.isOpened(): raise RuntimeError("Webcam not found")
print("Webcam ON. Press Q to quit.")
TARGET_FPS, last_t = 5, 0.0
dt = 1.0 / TARGET_FPS

while True:
    t = time.time()
    if t - last_t < dt: continue
    last_t = t

    ret, frame = cap.read()
    if not ret: break

    x = cv2.resize(
        cv2.cvtColor(frame, cv2.COLOR_BGR2RGB), IMG_SIZE
    ).astype(np.float32)
    x = tf.keras.applications.mobilenet_v2.\
        preprocess_input(x)[None, ...]

    probs = model.predict(x, verbose=0)[0]
    idx = int(np.argmax(probs))
    label = CLASS_NAMES[idx]; conf = float(probs[idx])

    cv2.putText(
        frame, f"{label} {conf*100:.1f}%", (20, 50),
        cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,255,0), 3
    )
    cv2.imshow("ASL Webcam", frame)
    if cv2.waitKey(1) & 0xFF in (ord("q"), ord("Q")): break

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

Fig. 6. Code implementation of the real-time gesture recognition inference pipeline using OpenCV and the trained MobileNetV2 model.

same `mobilenet_v2.preprocess_input` function applied during training, ensuring consistency between offline development and online deployment. The resulting tensor is expanded along the batch dimension and passed to the recognition model, which outputs a softmax distribution over the 24 trained gesture classes. The class with the highest probability is selected as the predicted gesture, and its confidence score is overlaid directly on the live video stream, providing immediate visual feedback. Fig. 7 presents representative screenshots of the system during live operation, illustrating stable recognition behavior across the four control gestures under normal indoor lighting.



Fig. 7. Screenshot captures of the real-time gesture recognition system during live operation.

The intent-resolution logic introduced conceptually in Sec. IV-B, confidence thresholding at $\tau = 0.80$ and change-detection-based action gating, is implemented directly within this loop, as shown in Fig. 8. Only when a prediction exceeds τ and differs from the previously transmitted command is the corresponding gesture label encoded and forwarded to the

```python
ret, frame = cap.read()
img = cv2.resize(frame, (IMG_SIZE, IMG_SIZE))
img = img / 255.0
img = np.expand_dims(img, axis=0)

preds = model.predict(img, verbose=0)
conf = np.max(preds)
label = CLASS_NAMES[np.argmax(preds)]

if conf > CONF_THRESHOLD and label != last_sent:
    arduino.write(label.encode())
    last_sent = label
```

Fig. 8. Real-time gesture inference and serial transmission code snippet

action-handling layer described in Sec. V-B.

B. Arduino-Based Smart Home Prototype

While the conceptual framework in Sec. IV supports both local hardware actuation and networked REST API control, the present implementation uses Arduino-based LED actuation as a *physical simulation* of smart home action execution rather than as a claim of full smart home deployment. This design choice allows the complete intent-to-action pipeline, from camera input through gesture classification to physical output, to be observed, timed, and debugged under controlled conditions, without requiring integration with a live IoT network. Fig. 9 summarizes this mapping at a conceptual level, showing how each classified gesture (A, B, L, V) corresponds to a distinct LED output channel on the Arduino microcontroller. Sec. VI subsequently shows how the same intent representation extends to networked REST API control without modification to the recognition or intent-resolution components.

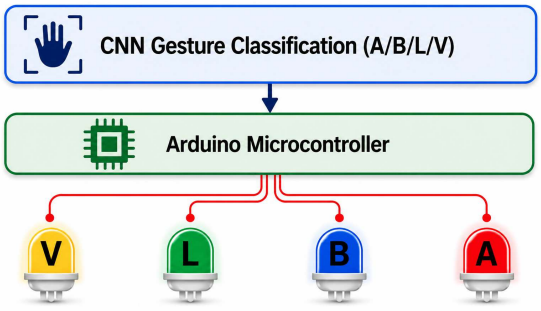


Fig. 9. High-level mapping from CNN-based gesture classification output to Arduino-controlled LED actuation, to four discrete output channels used in the hardware prototype.

Table II summarizes the hardware and software configuration of the simulated prototype.

Fig. 10 illustrates the underlying circuit: an Arduino Mega microcontroller interfaced with four LEDs through current-limiting resistors, with each LED's digital pin mapped to one gesture-derived control command. Fig. 11 shows the corresponding physical assembly on a solderless breadboard.

TABLE II
HARDWARE AND SOFTWARE SYSTEM SPECIFICATIONS

Component	Specification
Microcontroller	Arduino Mega (ATmega2560)
Host Platform	Standard consumer laptop
Camera	Integrated RGB webcam
Camera Resolution	1920 × 1080 pixels (1080p)
Inference Frame Rate	30 frames per second
Serial Interface	USB-to-UART
Baud Rate	9600 bps
Serial Latency	∼1.04ms per command
Model Input Resolution	160 × 160 pixels
Model Architecture	MobileNetV2 + custom head
Model Parameters	∼2.3M (frozen backbone)
Training Framework	TensorFlow / Keras
Inference Framework	TensorFlow / OpenCV
Confidence Threshold	0.80
End-to-End Latency	∼54–64ms

Table III lists the resulting gesture-to-action mapping, consistent with Table I.

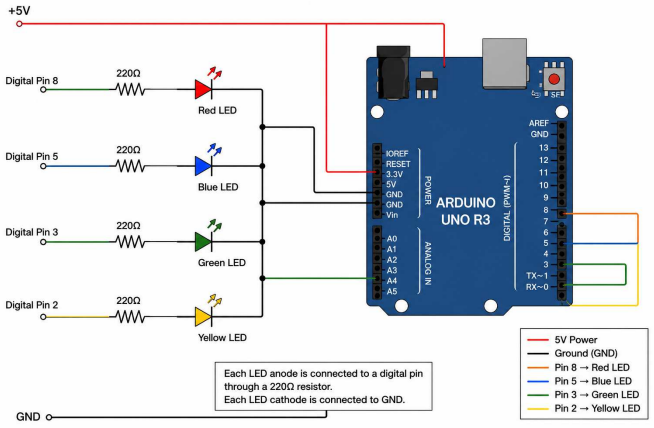


Fig. 10. Schematic diagram of the Arduino-based hardware validation circuit, illustrating the electrical connections between the Arduino Mega and the red, blue, green, and yellow LEDs used for gesture-driven control validation.

TABLE III
EXPERIMENTAL GESTURE-TO-ACTION MAPPING FOR HARDWARE VALIDATION

ASL Gesture	LED Color	Smart Home Intent
A	Red	Lighting Control
B	Blue	Climate Control
L	Green	Security Arm/Disarm
V	Yellow	Door Lock/Unlock

Fig. 12 demonstrates this mapping during live operation, showing each of the four control gestures correctly classified and triggering its corresponding LED on the physical prototype. For each gesture, the overlaid confidence score exceeds the $\tau = 0.80$ threshold established earlier, confirming that the recognition model, intent-resolution logic, and serial-actuation layer operate together as a coherent pipeline confirming reliable, low-latency behavior across the full intent-to-action pathway under live operating conditions. The consistent, high-confidence activation observed across all four gestures, A, B, L, and V, provides direct visual confirmation of the deterministic, low-latency behavior characterized quantitatively later in this work.

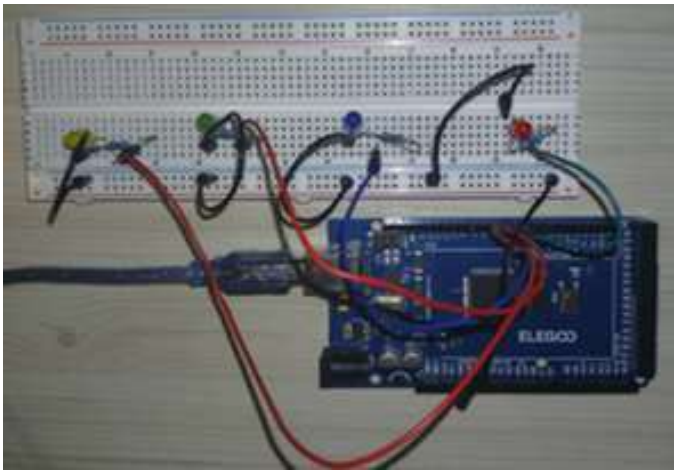


Fig. 11. Physical prototype of the hardware validation circuit, featuring an Arduino Mega connected to a breadboard with red, blue, green, and yellow LEDs corresponding to the gesture-driven control intents.

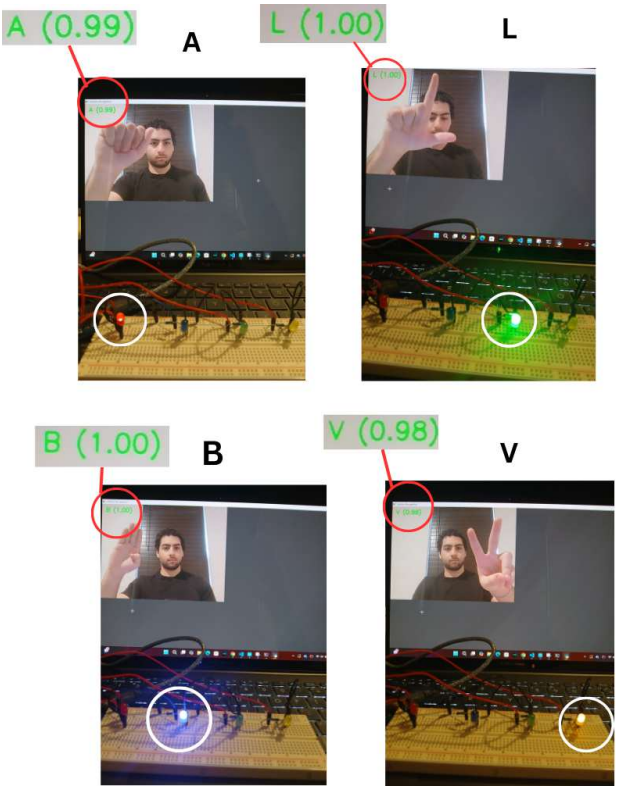


Fig. 12. Live demonstration of the hardware validation interface, illustrating the deterministic mapping from classified gesture input on the host PC to distinct LED actuations via the Arduino microcontroller. Serial commands ('A', 'B', 'L', 'V') trigger the corresponding red, blue, green, and yellow LEDs, respectively, with classification confidence scores overlaid on the live video feed.

On the hardware side, the Arduino Mega [23] continuously listens for incoming serial data, shown in Fig. 13. Upon receiv-

ing a gesture character, the microcontroller activates the corresponding digital output pin; in the absence of valid serial input, all output pins hold their current state, preventing undefined actuation behavior during communication interruptions. Each gesture label is transmitted as a single ASCII character upon detection. This encoding was selected over binary or packet-based alternatives for three reasons: minimal transmission overhead, deterministic decoding on the microcontroller side, and human-readable signal monitoring during development. The absence of a packet wrapper or checksum was accepted as a deliberate simplification, justified by the low-noise USB-to-UART physical medium and by the confidence-threshold and change-detection mechanisms described in Sec. IV-B, which already filter spurious commands at the application layer, independent of the transport layer.

```
if (Serial.available()) {
  char gesture = Serial.read();
  allOff();

  if (gesture == 'A') digitalWrite(LED_A, HIGH);
  if (gesture == 'B') digitalWrite(LED_B, HIGH);
  if (gesture == 'L') digitalWrite(LED_L, HIGH);
  if (gesture == 'V') digitalWrite(LED_V, HIGH);
}
```

Fig. 13. Arduino-based gesture-driven LED actuation code snippet

During testing, gesture detection was configured to trigger an action on a single recognized frame rather than requiring multi-frame confirmation, representing a worst-case responsiveness configuration; the architecture supports straightforward extension to multi-frame confirmation or temporal smoothing for deployment scenarios with stricter safety requirements, as discussed in Sec. VIII-A .

Beyond the actuation pathway itself, the prototype incorporates layered fault-tolerance mechanisms at each pipeline stage, illustrated in Fig. 14. At the perception layer, the $\tau = 0.80$ confidence threshold prevents ambiguous or partially occluded hand poses from propagating to the control layer. At the communication layer, change-detection logic prevents command flooding under sustained gesture presentation. At the actuation layer, the Arduino's deterministic event loop ensures that only explicitly decoded commands trigger output changes. Together, these mechanisms ensure that the simulated prototype degrades gracefully under uncertain input rather than producing unintended actuation, a property equally relevant to the networked REST API pathway, since both pathways consume the same upstream intent representation.

C. System Latency and Reliability

The end-to-end response time of the prototype is defined as the interval between gesture presentation and physical actuation, and can be decomposed into three sequential components: frame acquisition and preprocessing, model inference, and serial command transmission. At 30 frames per second,

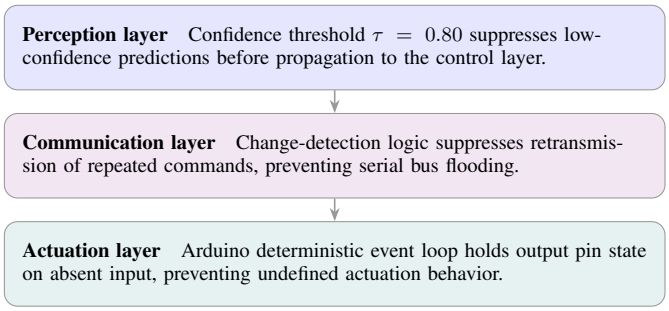


Fig. 14. Layered fault tolerance architecture of the proposed system, illustrating the fault interception mechanisms operating independently at the perception, communication, and actuation layers.

the frame acquisition cycle imposes a maximum interval of 33.3ms between successive gesture evaluations. MobileNetV2-based inference on standard CPU hardware introduces an additional processing delay of 20–30ms, consistent with published benchmarks for this architecture [9]. Serial transmission of a single-character command at a baud rate of 9600, with one start bit and one stop bit, produces a 10-bit frame and a theoretical transmission latency of 1.04ms. Summing these components yields a theoretical end-to-end latency of 54–64ms, well below the 100ms perceptual threshold established for interactive control systems [23]. This theoretical estimate is compared against measured response times from the user study.

At the communication layer, the serial interface's theoretical maximum throughput of 960 characters per second is used at only a small fraction of capacity, since the system transmits at most 30 single-character commands per second, leaving substantial headroom for future protocol extensions such as multi-byte encoding or acknowledgment signaling without requiring a baud rate change. At the actuation layer, the Arduino Mega's 54 available digital I/O pins represent the practical upper bound for directly connected GPIO-controlled outputs in the current LED-based simulation; the four-channel configuration used here utilizes less than 8 percent of this capacity. A more general treatment of scaling beyond the four-gesture demonstration is given in Sec. VIII-B .

VI. REST API-Based System Integration

This section extends the intent-to-action pipeline validated through hardware simulation in Sec. V to a networked deployment pathway, in which recognized gestures are converted into REST API requests rather than serial commands, enabling integration with commercial smart home platforms. Sec. VI-A establishes the technical architecture and design rationale shared by all REST-based use cases, while Sections VI-B through VI-D instantiate this architecture for three representative smart home functions, door lock control, security system control, and lighting control, each motivated by accessibility needs documented in the existing literature and accompanied by an example implementation.

A. REST API: Design Rationale and Technical Architecture

While the hardware prototype in Sec. V demonstrates the feasibility of the intent-to-action pipeline under controlled, local conditions, real-world smart home deployments require integration with networked, often vendor-specific control infrastructure. To support this, the framework incorporates a REST API–based control abstraction that decouples gesture interpretation from device-specific actuation logic, allowing the same upstream intent representation to be consumed by external smart home platforms without modification.

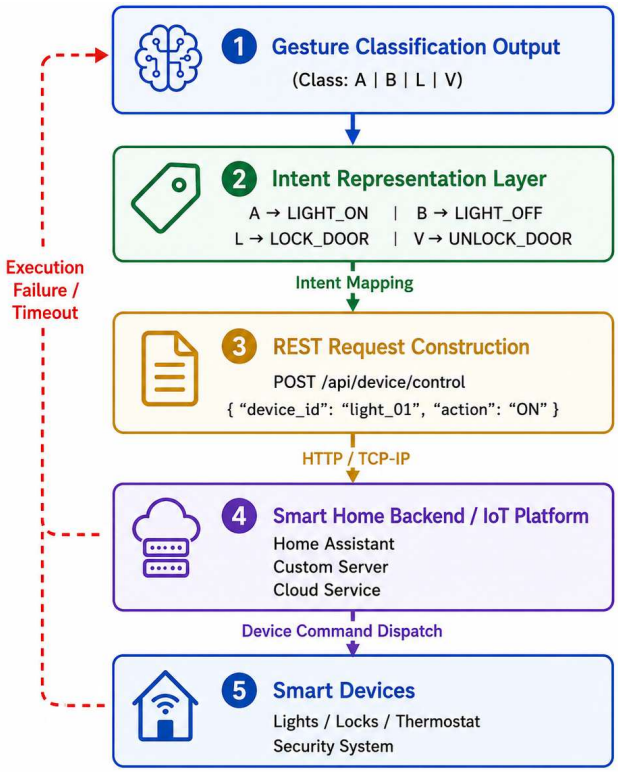


Fig. 15. REST API–based control pipeline illustrating intent abstraction, HTTP request generation, backend dispatch, and failure-handling feedback loop for gesture-driven smart home control.

As illustrated in Fig. 15, a recognized gesture is first transformed into a high-level control intent, independent of any device-specific command syntax. This intent is encapsulated as a RESTful HTTP request and transmitted to a smart home backend or IoT platform, such as Home Assistant or openHAB, both of which expose REST endpoints for programmatic device control. In the event of execution failure or network timeout, control feedback is returned to the interaction layer, ensuring that gesture recognition itself remains decoupled from transient network conditions.

The REST interface follows conventional client–server principles, mapping gesture-derived intents to standard HTTP methods. Enabling or disabling a security feature, or locking and unlocking an access point, can be expressed as POST or PUT requests to corresponding API endpoints, with JSON payloads encapsulating the intended action and target device identifier; downstream services interpret and execute the request according to their own implementation. In practice, this abstraction can be realized using lightweight HTTP frameworks such as Flask or FastAPI hosted on a local gateway, or by targeting the REST endpoints already exposed by open smart home platforms.

```
# Gesture-to-command mapping
gesture_command_map = {
    "A": "SECURITY_ARM",
    "B": "SECURITY_DISARM",
    "L": "DOOR_LOCK",
    "V": "DOOR_UNLOCK"}
```

Fig. 16. Example gesture-to-command mapping structure used to translate recognized gestures into high-level smart home control intents.

Fig. 16 shows the gesture-to-command mapping structure used to translate a recognized gesture class into a high-level intent, consistent with the canonical mapping established in Table I. The three subsections that follow instantiate this abstraction for three representative smart home functions, door lock control, security system control, and lighting control, selected because they correspond to access, safety, and environmental control needs most frequently identified in the assistive technology literature as priorities for non-verbal interaction [3], [8], [12], [30]. Each use case follows the same structure: a brief discussion of why the function matters for users with speech or motor impairments, a review of the communication models and device types relevant to that function, and an example REST request implementing the gesture-derived action.

B. Use Case: Function 1 – Door Lock

Independent control of entry doors is a basic requirement for safe and autonomous living, yet conventional deadbolts and keyed entry require fine motor control and, in many cases, verbal coordination with another person to grant access, both of which present direct barriers for individuals with speech impairments or co-occurring motor limitations [12], [30]. A gesture-activated lock removes the requirement for either spoken confirmation or precise key manipulation, allowing a user to lock or unlock an entry point through a single, low-effort hand gesture captured by the same camera already used for the system's other functions.

Literature on Lock Types and Communication Models. Digital door lock systems in the literature span a range of authentication and communication strategies. Early systems replaced mechanical keys with RFID tags or one-time-password authentication while retaining a short-range wired or wireless link to the lock motor. Subsequent systems moved toward network-connected locks: Wi-Fi–based designs route lock and unlock commands through a smartphone application and a cloud messaging service, removing the need for a dedicated GSM module. Bluetooth-based designs instead favor a direct, hub-free link between a mobile device and the lock for short-range, low-power operation, while ZigBee-based

designs embed the lock as the central coordinating node of a broader home automation network, exchanging status with other sensor nodes over a low-power mesh. A systematic review of these approaches highlights a consistent design tension: locks optimized for convenience and remote accessibility tend to introduce a larger network attack surface, while locks optimized for security tend to require more deliberate, less remotely-accessible interaction, a tradeoff directly relevant to assistive applications where remote, low-effort access is itself an accessibility requirement.

For the proposed framework, this body of work motivates the choice of REST-over-HTTP as the integration layer: rather than committing to one lock communication standard, the gesture-derived intent is expressed as a vendor-agnostic API call, allowing the underlying lock to use any of the communication models surveyed above, Wi-Fi, Bluetooth, or ZigBee, so long as it exposes, or is bridged to, a REST endpoint.

Example Implementation. Fig. 17 shows the REST request logic used to translate a recognized door-lock gesture into an authenticated HTTP request against a door-lock API endpoint, following the gesture-to-command mapping defined in Fig. 16.

```python
def send_door_lock_command(command):
    if command == "DOOR_LOCK":
        api_endpoint = "/api/door/lock"
    elif command == "DOOR_UNLOCK":
        api_endpoint = "/api/door/unlock"

    requests.post(
        url=SMART_HOME_BASE_URL + api_endpoint,
        headers={"Authorization": "Bearer <API_TOKEN>"},
        json={"initiated_by": "gesture_control"}
    )
```

Fig. 17. Example REST API request logic for door-lock control actions.

C. Use Case: Function 2 – Security Enabling

Arming or disarming a home security system conventionally requires either a physical keypad entry or a verbal exchange with a monitoring service, both of which exclude users who cannot reliably produce speech or precise manual input under time pressure [3], [12]. A gesture-based arm/disarm control allows this safety-critical function to be triggered through the same low-effort, camera-based interaction used throughout the system, removing both barriers simultaneously.

Literature on Security System Types and Communication Models. Smart home security literature converges on a small set of recurring sensing and communication patterns. Passive infrared (PIR) motion sensors paired with a low-cost single-board computer, such as a Raspberry Pi, are commonly used to trigger image capture, after which face recognition is applied to distinguish authorized occupants from unrecognized visitors. More recent work integrates AI-driven video processing directly into this pipeline, applying face detection and recognition only after a motion event to balance responsiveness against continuous-monitoring power and privacy costs. Across these systems, the underlying sensing hardware, PIR motion detectors, ultrasonic sensors, and IP-connected cameras, communicates either over a local network (Wi-Fi, Ethernet) to a central hub or directly to a cloud service, with the security system's arm/disarm state itself almost universally exposed through a network-accessible interface rather than a purely local switch, since remote arming and disarming is a core expected feature of modern systems.

This consistent exposure of arm/disarm state through a network-accessible interface is precisely what the proposed REST abstraction targets: the gesture-derived intent does not need to know whether the underlying security platform uses PIR sensing, camera-based detection, or both, only that it exposes an endpoint capable of receiving an arm or disarm command.

Example Implementation. Fig. 18 shows the REST request logic used to translate a recognized security-control gesture into an authenticated HTTP request against a security-system API endpoint, following the gesture-to-command mapping defined in Fig. 16.

```python
def send_security_command(command):
    if command == "SECURITY_ARM":
        api_endpoint = "/api/security/arm"
    elif command == "SECURITY_DISARM":
        api_endpoint = "/api/security/disarm"

    requests.post(
        url=SMART_HOME_BASE_URL + api_endpoint,
        headers={"Authorization": "Bearer <API_TOKEN>"},
        json={"source": "gesture_interface"}
    )
```

Fig. 18. Example REST API request logic for security system control actions.

D. Use Case: Function 3 – Lighting Control

Lighting is among the most frequently cited target functions in assistive smart home literature, both because it is needed continuously throughout the day and because conventional wall switches require a directed reach-and-press motion that may be inconvenient or inaccessible during fatigued or constrained interaction [3], [8]. A gesture-based lighting control allows a user to toggle lighting from wherever the camera has a view, without locating or physically reaching a switch.

Literature on Lighting Control Types and Communication Models. Wireless lighting control in the smart home and building-automation literature is dominated by three short-range protocols: Wi-Fi, Bluetooth (including Bluetooth Mesh), and ZigBee. ZigBee, built on the IEEE 802.15.4 standard, is the most extensively documented in academic lighting-control work, owing to its low power consumption, mesh-network self-healing, and established use in residential and street-lighting deployments. Bluetooth Mesh offers comparable mesh resilience without requiring a dedicated hub, favoring smaller, retrofit-style installations, while Wi-Fi-connected fixtures trade higher power consumption for direct integration with existing home network infrastructure and cloud services. Across all three, the lighting fixture's on/off and dimming state is ultimately exposed through a hub- or cloud-mediated API, regardless of the underlying radio protocol, which is again the integration point the REST abstraction in this work targets directly.

```python
def send_light_command(command):
    if command == "LIGHT_ON":
        api_endpoint = "/api/light/on"
    elif command == "LIGHT_OFF":
        api_endpoint = "/api/light/off"

    requests.post(
        url=SMART_HOME_BASE_URL + api_endpoint,
        headers={"Authorization": "Bearer <API_TOKEN>"},
        json={"source": "gesture_interface"}
    )
```

Fig. 19. Example REST API request logic for lighting control actions.

Fig. 19 shows the REST request logic, structured identically to Fig. 17 and Fig. 18, used to translate a recognized lighting-control gesture into an HTTP request against a lighting API endpoint.

VII. Experimental Evaluation

With the architecture, prototype, and REST API integration now in place, attention turns to empirical performance, starting at the level of the gesture recognition model and working outward to full system behavior, from offline classification performance and confusion matrix analysis to live operation of the real-time pipeline and hardware prototype, closing with a nine-participant user study that offers an initial measure of usability beyond these controlled conditions.

A. Gesture Recognition Performance

The gesture recognition model was trained for 10 epochs using an 80/20 split of the curated 24-class ASL dataset described in Sec. III-A, yielding roughly 153,000 training images and 38,000 validation images. The model achieved a validation accuracy of 98.6%, with a corresponding validation loss of 0.0445, indicating strong convergence and generalization. Table IV reports additional evaluation metrics, including a macro F1-score of 0.970, a Matthews Correlation Coefficient of 0.968, and a Cohen's Kappa of 0.967, all consistent with the high accuracy figure and indicating balanced, well-separated classification behavior across the full 24-class vocabulary rather than performance driven by a small number of dominant classes.

TABLE IV
Performance evaluation metrics for the proposed gesture recognition model.

Metric	Value
Accuracy	0.986
Macro F1-score	0.970
Matthews Correlation Coefficient (MCC)	0.968
Cohen's Kappa	0.967
Log Loss	0.0445

Fig. 20 shows training and validation accuracy across epochs: training accuracy increases steadily while validation accuracy stabilizes early, with close alignment between the two curves indicating minimal overfitting. Fig. 21 shows the corresponding loss curves, which decrease sharply during initial epochs and converge smoothly, consistent with the use of a frozen, pretrained backbone and a restricted, well-separated gesture vocabulary. Precision, recall, and F1-score computed across all 24 classes were consistently high, with most per-class F1-scores exceeding 0.85.



Fig. 20. Training and validation accuracy across epochs, demonstrating rapid learning, early convergence, and strong generalization performance.

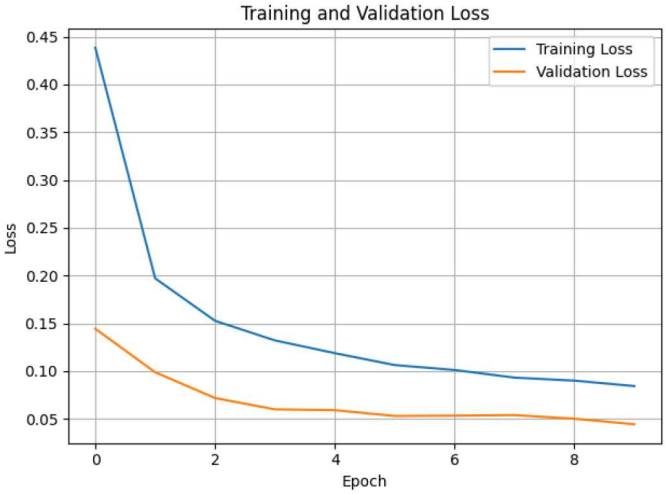


Fig. 21. Training and validation loss across epochs, demonstrating rapid convergence, stable optimization behavior, and minimal overfitting.

These results, together with the model architecture described in Sec. III-B and the training configuration in Sec. III-C, establish the recognition model as a reliable perception component for the real-time prototype evaluated in the remainder of this section.

B. Confusion Matrix Analysis

Fig. 22 illustrates per-class classification performance across the full 24-class ASL gesture vocabulary used to train the recognition model. The matrix demonstrates strong class separability, with each gesture overwhelmingly classified into its correct category; minor off-diagonal entries indicate that visually similar hand poses occasionally introduce confusion,

but no systematic misclassification pattern is observed across the gesture set.

Gestures used in the smart home control demonstration, A, B, L, and V, achieve consistently high recognition accuracy within this broader matrix, with no dominant confusion trends observed between them or with other trained classes. Gesture L exhibits near-perfect classification performance, consistent with its highly distinctive hand pose, while gestures A, B, and V also demonstrate stable and reliable recognition despite minor variations in hand orientation and lighting conditions. These results confirm that the model's predictions remain sufficiently distinct across the full gesture vocabulary to support deterministic mapping between recognized gestures and downstream control actions, even though only a subset of classes is currently used for actuation, and they directly support the suitability of the system for real-time deployment evaluated in the remainder of this section.

C. Prototype Validation and Action Execution

Beyond offline evaluation, the trained model was deployed within the real-time vision processing pipeline described in Sec. V-A to assess behavior under live operating conditions. The system sustained stable operation at 30 frames per second throughout testing, with no observed degradation in frame rate during extended sessions under normal indoor lighting. When no gesture was presented, or when predictions fell below the $\tau = 0.80$ confidence threshold, the system correctly withheld action, confirming that the intent-resolution logic described in Sec. IV-B suppresses spurious activation during idle or transitional states rather than only during deliberate gesture presentation.

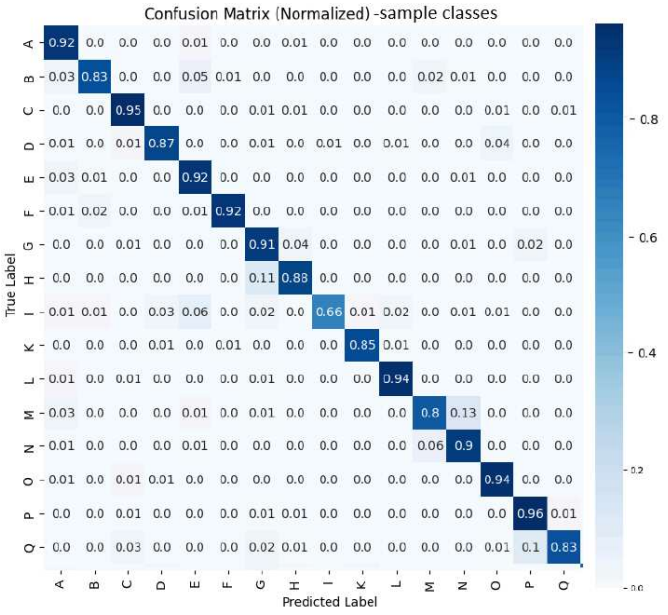


Fig. 22. Confusion matrix (normalized) for the 24-class ASL alphabet scalability evaluation, demonstrating strong diagonal dominance across the full gesture vocabulary.

For the four control gestures (A, B, L, and V), correct LED activation on the Arduino-based prototype described in Sec. V-B was observed consistently across repeated trials, confirming that the complete simulated pipeline, from camera input through gesture classification, intent resolution, and serial transmission, behaves deterministically and without perceptible delay.

Gesture L exhibited the most consistent recognition behavior during live testing, consistent with its near-perfect performance in the offline confusion matrix (Sec. VII-B), while gestures A and V occasionally required a brief re-presentation under oblique hand angles, a behavior examined further as a limitation below.

D. User Evaluation

To complement hardware-level system validation with human interaction data, a preliminary usability evaluation was conducted with nine participants to assess gesture learnability, recognition reliability, and system responsiveness under controlled conditions. This evaluation follows early-stage assistive technology assessment practice [30], in which able-bodied participants are used to validate core system behavior prior to deployment studies with target user populations.

1) Participants and Setup: Nine participants (5 male, 4 female; ages 19–52) with no prior experience using gesture-based control systems were recruited. None of the participants had formal training in American Sign Language. Each participant interacted with the system using the same laptop-integrated RGB webcam and host configuration described in Table II, under consistent indoor lighting conditions. Prior to the evaluation session, each participant received a 2–3 minute familiarization period during which the four target gestures (A, B, L, and V) were demonstrated and practiced without recording.

2) Procedure: Each participant performed 10 trials per gesture class, yielding 40 gesture attempts per participant and 360 total gesture attempts across the full evaluation. For each trial, participants were prompted to produce a target gesture and hold it steadily in front of the camera until the system produced a classification output. The system's response was recorded as correct if the recognized gesture class matched the prompted target and the confidence threshold of $\tau = 0.80$ was satisfied. Response time was measured as the interval between gesture presentation onset and hardware-level LED actuation. Following the gesture trials, each participant completed the System Usability Scale (SUS) [40], a validated ten-item questionnaire scored on a 0–100 scale, to provide a standardized measure of perceived usability.

3) Results: Table V summarizes the per-participant evaluation results across completion rate, average response time, and SUS score. Across all nine participants and 360 gesture attempts, the system achieved a mean task completion rate of 95.6%, indicating reliable gesture recognition under real-world interaction conditions with novice users. The mean end-to-end response time of 59.6ms is consistent with the theoretical latency model presented earlier and falls well below the 100ms

TABLE V
PER-PARTICIPANT USER EVALUATION RESULTS

Participant	Completion Rate (%)	Avg. Response Time (ms)	SUS Score
P1	97.5	57	84
P2	95.0	61	79
P3	100.0	55	88
P4	92.5	63	76
P5	97.5	58	82
P6	95.0	60	81
P7	90.0	66	74
P8	97.5	57	85
P9	95.0	59	80
Mean	**95.6**	**59.6**	**81.0**

perceptual threshold for interactive control systems [24]. The mean SUS score of 81.0 exceeds the established threshold of 68 for above-average usability [40], indicating that participants perceived the system as intuitive and responsive despite having no prior gesture control experience.

Per-gesture analysis of completion rates revealed that gesture L achieved the highest individual recognition rate across participants, consistent with its geometric distinctiveness as identified in the offline confusion matrix analysis (Sec. VII-B). Gestures A and V exhibited slightly lower completion rates for some participants, attributable to subtle inter-participant variation in hand pose geometry and finger extension angle during unconstrained interaction. These findings are consistent with the orientation variability failure mode discussed as a limitation in Sec. VIII-C and reinforce the need for training data augmentation with diverse pose variants in future system iterations.

Overall, the user evaluation confirms that the proposed system supports reliable and learnable gesture-based interaction for novice users under controlled conditions, providing an initial evidence base for system usability that motivates future evaluation with the target population of individuals with speech impairments.

VIII. DISCUSSION

The evaluation in Sec. VII demonstrated that the proposed framework performs reliably under the controlled conditions tested, both at the level of gesture classification and at the level of end-to-end prototype behavior. This section steps back from those results to interpret what they imply about the framework's design. It examines the tradeoffs underlying key design decisions, considers how the architecture scales beyond the conditions tested, and closes by acknowledging the limitations of the current evaluation and the work required before real-world deployment.

A. Design Tradeoffs

The high classification accuracy achieved by the recognition model reflects both the effectiveness of the transfer learning strategy and the visual distinctiveness of the gesture set as a whole. By leveraging a MobileNetV2 backbone pretrained on ImageNet, the model benefits from rich low- and mid-level feature representations that generalize well to hand shape classification across all 24 trained gesture classes, without requiring full end-to-end training from scratch. Among these, the four gestures used in the smart home control demonstration, A, B, L, and V, illustrate this discriminative capability well, exhibiting substantially different spatial configurations that minimize inter-class feature overlap. Gesture L in particular demonstrates near-perfect classification performance, consistent with its highly distinctive hand pose, an extended thumb and index finger forming an unambiguous angular configuration that produces a unique spatial feature signature under MobileNetV2's convolutional filters.

The decision to map only four gestures to control actions, selected from the full set of 24 trained gesture classes, was motivated by the need for a focused, reliable control vocabulary aligned with smart home interaction scenarios rather than broad gesture-to-action mapping. Beyond reducing engineering complexity at the control layer, this constraint carries accessibility value in its own right: prior work on assistive gesture interfaces has noted that expanding gesture or command vocabularies tends to increase cognitive load and calibration burden for users with motor or cognitive impairments [2]. A small, geometrically distinct control vocabulary lowers the demand placed on users during sustained or fatigued interaction, which is particularly relevant for the target population of individuals with speech impairments who may also experience co-occurring motor or cognitive limitations. The four-gesture control mapping therefore reflects an intentional accessibility-first tradeoff rather than a purely engineering-driven simplification, preserving the underlying model's capacity to support additional gestures if future use cases require expanded control vocabularies.

The REST API abstraction layer introduced in Sec. VI was designed specifically to decouple the gesture recognition pipeline from device-specific actuation logic, ensuring that the core interaction framework remains platform-independent and extensible to heterogeneous smart home ecosystems. The choice of ASL-based gestures, while motivated by dataset availability, also carries practical accessibility implications: ASL gestures are standardized, well-documented, and familiar to a significant portion of the target user population, reducing the learning curve associated with adopting the system.

B. Scalability and Deployment Considerations

The architecture supports scaling beyond the four-gesture demonstration without modifying the perception or control pipeline. Since the model is already trained on 24 static ASL gesture classes, expanding the number of actuated control gestures requires no retraining, only an extension of the host-side mapping logic and, for the hardware prototype, the Arduino's actuation table. Inference cost stays fixed regardless of how many of the 24 trained classes are actively mapped, since the dominant computation occurs in the frozen MobileNetV2 backbone rather than the classification head.

This headroom differs in kind between the two integration pathways introduced earlier. On the hardware pathway, scaling is bounded by physical I/O: the current four-LED prototype

uses less than 8% of the Arduino Mega's 54 digital pins (Sec. V-C), and deployments requiring more than roughly 54 directly addressable channels would need hierarchical expansion through I^2C or SPI multiplexing [22]. On the REST API pathway, no such hardware ceiling exists, since each additional gesture-to-function mapping corresponds only to an additional entry in the intent dictionary (Fig. 15) and a corresponding API endpoint on the target platform; the practical limit is determined by the smart home backend's own device capacity rather than by anything intrinsic to the proposed framework. This distinction is itself an argument for the REST pathway as the primary path to real-world scale, with the hardware prototype serving its intended role as a bounded, observable proof of the underlying intent-resolution logic rather than as the deployment target.

C. Limitations

Several limitations of the current work should be acknowledged. The system was evaluated exclusively under controlled indoor lighting conditions using a single public dataset, and performance under varying illumination, complex backgrounds, or outdoor environments remains to be assessed. During live operation, gesture orientation variability was identified as a source of occasional misclassification, particularly in cases where gestures were presented at oblique angles or with reduced articulatory precision, conditions that may arise naturally during fatigued or casual interaction. This behavior is consistent with the composition of the training dataset, which predominantly captures gestures performed in canonical, upright orientations and does not sufficiently represent the full distribution of pose variations encountered in unconstrained real-world use. Augmenting the training data with rotation, affine transformation, and perspective distortion variants, or incorporating pose-invariant feature learning strategies, would likely improve robustness to such orientation-dependent failure cases.

While the underlying model supports 24 gesture classes, the smart home control mapping is currently limited to four representative gestures, which constrains the range of controllable smart home functions and may not fully address the interaction needs of all users. A preliminary usability evaluation with nine able-bodied participants demonstrates promising interaction reliability, but evaluation with individuals with speech impairments remains the most critical gap between the present framework validation and real-world deployment readiness. Hardware validation was performed using LED actuation as a simulation of smart home device control, and, the REST API use cases are presented as an architectural and literature-grounded contribution rather than an empirically benchmarked one; full networked integration with a live IoT platform remains future work. Addressing these limitations through expanded control vocabularies, diverse evaluation conditions, networked deployment, and user-centered studies represents the primary direction for future development of the proposed framework.

IX. Conclusion and Future Work

This paper presented a vision-based smart home control framework designed to enable accessible, non-verbal interaction for individuals with speech impairments. The proposed system integrates real-time camera-based hand gesture recognition with intent interpretation and action handling within a complete, end-to-end conceptual framework, instantiated through both a hardware prototype and a REST API–based integration pathway. By leveraging standard RGB camera input, a lightweight deep learning model, and real-time vision processing, the results suggest the system can provide reliable smart home control under the evaluated conditions without requiring specialized sensors, wearable devices, or voice-based interfaces.

Experimental evaluation demonstrated strong performance across all evaluated stages. The gesture recognition model achieved high classification accuracy and stable generalization on a curated set of control gestures, while real-time validation confirmed reliable frame-by-frame inference and consistent behavior under live operating conditions. Prototype validation, including LED-based hardware actuation, verified that recognized gestures are correctly interpreted and translated into immediate, deterministic control actions, and a nine-participant user study provided initial evidence of learnability and perceived usability. These results confirm that the proposed framework functions as a practical control system rather than a standalone recognition model.

A key contribution of this work is the system-level separation of gesture perception, intent interpretation, and action execution within a unified conceptual architecture. By introducing this separation explicitly and instantiating it through two independent pathways, a hardware prototype and a literature-grounded REST API integration, the framework supports scalable deployment across heterogeneous smart home environments without requiring the gesture recognition component itself to change.

Future work will focus on closing the gap between architectural design and empirical network deployment. One direction involves full integration with a live smart home platform, such as Home Assistant or openHAB, to empirically measure end-to-end network latency, request reliability, and multi-device concurrency for the three REST use cases presented in (Sec. VI), complementing the scope justification given in Sec. VII-D . A second direction involves incorporating multi-frame gesture confirmation to further reduce sensitivity to transient low-confidence predictions in dynamic environments. Additional extensions include expanding the gesture vocabulary beyond the current four control mappings, supporting multiple users, and evaluating the system under diverse lighting conditions and real-world home settings with the target population of individuals with speech impairments.

Overall, the proposed system demonstrates that vision-based hand gesture interaction can serve as an effective, accessible, and scalable alternative to voice-driven smart home control. By enabling intuitive, real-time non-verbal interaction, the

framework contributes toward more inclusive smart living environments for users with speech impairments and other accessibility needs.

Author Contributions

Kareem Mohamed: Conceptualization and overall research vision; design of the complete vision-based assistive framework and experimental methodology; software development of the deep learning model, training pipeline, and real-time OpenCV-based inference system; experimental evaluation and model validation using comprehensive quantitative metrics; formal analysis and interpretation of results; design of all system architecture diagrams, neural network visualizations, and experimental figures; and writing – original draft preparation. Walaa Abo Elenin: Academic and technical supervision; research guidance on experimental design and methodology; project administration; and critical review, editing, and refinement of the manuscript's scientific presentation. Yiming Ji: Investigation, literature review, and comprehensive analysis of related work; visual refinement of manuscript figures and tables for clarity; and critical review, proofreading, and technical editing of the manuscript.

Acknowledgment

AI-assisted tools were used to support manuscript editing, language refinement, figure generation, and software development. All technical content, research design, implementation, experimental evaluation, analysis, results, and conclusions were developed, verified, and approved by the authors, who take full responsibility for the content of this work.

Funding

Funding: Not applicable.

References

[1] K. A. Mohamed and W. A. Elenin, "Vision-Based Hand Gesture Recognition for Assistive Computing," in *2026 7th IEEE International Conference on Artificial Intelligence, Robotics, and Control (AIRC)*, Savannah, GA, USA, Apr. 65–70, 2026.

[2] S. M, S. S, P. R, H. K. R and E. R. G, "Automated Smart Home Assistive System Implementation for Physically Impaired Community Using Eye Gestures," 2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2024, pp. 81-85, doi: 10.1109/ICACCS60874.2024.10717287.

[3] C. Neacşu et al., "Hand Gesture Recognition Using YOLOv5 for People with Disability," 2025 IEEE 25th International Conference on Bioinformatics and Bioengineering (BIBE), Athens, Greece, 2025, pp. 521-525, doi: 10.1109/BIBE66822.2025.00093.

[4] K. Shruthi, G. S. Kiran, K. V. Kumar, M. A. Kumar, P. Meghana and A. Anitha, "Touchless Human-Computer Interaction: An AI-Driven Gesture and Voice Control System," 2025 5th International Conference on Intelligent Technologies (CONIT), HUBBALI, India, 2025, pp. 1-5, doi: 10.1109/CONIT65521.2025.11167682.

[5] Y. D, S. A, V. P. D, S. K. S and V. M, "Real-Time Hand Gesture Recognition for LED Control using CNN and Arduino," 2025 Second International Conference on Intelligent Technologies for Sustainable Electric and Communications Systems (iTech SECOM), Coimbatore, India, 2025, pp. 1-7, doi: 10.1109/iTechSECOM64750.2025.11307582.

[6] A. B. Wavale and K. Sireesha, "An Edge ML-Driven Gesture Recognition System for Smart Home Control," 2025 International Conference on Sensors and Related Networks (SENNET) Special Focus on Digital Healthcare(64220), Vellore, India, 2025, pp. 1-6, doi: 10.1109/SENNET64220.2025.11135995.

[7] S. Nigam, M. Shamoon, S. Dhasmana and T. Choudhury, "A Complete Study of Methodology of Hand Gesture Recognition System for Smart Homes," 2019 International Conference on contemporary Computing and Informatics (IC3I), Singapore, 2019, pp. 289-294, doi: 10.1109/IC3I46837.2019.9055608.

[8] H. Baumgartl, D. Sauter, C. Schenk, C. Atik and R. Buettner, "Vision-based Hand Gesture Recognition for Human-Computer Interaction using MobileNetV2," 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 2021, pp. 1667-1674, doi: 10.1109/COMPSAC51774.2021.00249.

[9] V. Vijayaraghavan, N. T. Uppala, P. Gumma, G. Seelam and P. Darukumalli, "Gesture Control Based Home Automation Using Raspberry Pi," 2025 9th International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2025, pp. 1767-1772, doi: 10.1109/ICISC65841.2025.11188184.

[10] U. K. Rajanala, N. K. B. Chinni, G. Sandaka and R. Shaik, "Hand Gesture Recognition for Smart Control of Electrical Appliances Using Computer Vision," 2024 International Conference on Smart Technologies for Sustainable Development Goals (ICSTSDG), Chennai - 600077, Tamil Nadu, India, 2024, pp. 1-6, doi: 10.1109/ICSTSDG61998.2024.11026353.

[11] N. Mohamed, M. B. Mustafa and N. Jomhari, "A Review of the Hand Gesture Recognition System: Current Progress and Future Directions," in IEEE Access, vol. 9, pp. 157422-157436, 2021, doi: 10.1109/ACCESS.2021.3129650.

[12] W. Y. Leong, "Multimodal Gesture and Speech-Based Interaction Interfaces for AI-Driven Mobile Robots in Smart Homes," 2025 10th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, Japan, 2025, pp. 326-331, doi: 10.1109/ICIIBMS66230.2025.11316778.

[13] T. -H. Tsai, Y. -J. Luo and W. -C. Wan, "A Skeleton-based Dynamic Hand Gesture Recognition for Home Appliance Control System," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 3265-3268, doi: 10.1109/ISCAS48785.2022.9937780.

[14] H. Tang, "Intelligent Vision Testing Technology for Gesture Recognition Based on Artificial Intelligence Algorithms," 2024 International Conference on Data Science and Network Security (ICDSNS), Tiptur, India, 2024, pp. 1-5, doi: 10.1109/ICDSNS62112.2024.10691206.

[15] A. Anand, A. Pandey, S. Mehndiratta, H. Goyal, P. Kaushik and R. Rathore, "Smart AI basedVolume Control System: Gesture Recognition with OpenCV & Mediapipe Integration," 2023 International Conference on Smart Devices (ICSD), Dehradun, India, 2024, pp. 1-6, doi: 10.1109/ICSD60021.2024.10751362.

[16] Y. Javeed and H. M. Mehdi, "A Computer Vision Approach to Personalized Light Dimming for Elderly and Bedridden Care," 2025 International Conference on Emerging Technologies in Electronics, Computing, and Communication (ICETECC), Jamshoro, Pakistan, 2025, pp. 1-6, doi: 10.1109/ICETECC65365.2025.11070273.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4510–4520, doi: 10.1109/CVPR.2018.00474.

[18] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, USA, 2016, pp. 265–283.

[19] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2021, doi: 10.1109/TNNLS.2021.3084827.

[20] S. S. Rautaray and A. Agrawal, "Vision Based Hand Gesture Recognition for Human Computer Interaction: A Survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015, doi: 10.1007/s10462-012-9356-9.

[21] D. Mocrii, Y. Chen and P. Musilek, "IoT-Based Smart Homes: A Review of System Architecture, Software, Communications, Privacy and Security," *Internet of Things*, vol. 1–2, pp. 81–98, 2018, doi: 10.1016/j.iot.2018.08.009.

[22] H. K. Kondaveeti, N. K. Kumaravelu, S. D. Vanambathina, S. E. Mathe and S. Vappangi, "A Systematic Literature Review on Prototyping with Arduino: Applications, Challenges, Advantages, and Limitations," *Computer Science Review*, vol. 40, p. 100364, 2021, doi: 10.1016/j.cosrev.2021.100364.

[23] C. Attig, N. Rauh, T. Franke and J. F. Krems, "System Latency Guidelines Then and Now – Is Zero Latency Really Considered Necessary?" in *HCI International 2017 – Posters' Extended Abstracts*, Cham, Switzerland: Springer, 2017, pp. 3–11, doi: 10.1007/978-3-319-58475-1_1.

[24] M. Iman, K. Rasheed and H. R. Arabnia, "A Review of Deep Transfer Learning and Recent Advancements," *ArXiv preprint*, arXiv:2201.09679, 2022. [Online]. Available: https://arxiv.org/abs/2201.09679

[25] F. Weichert, D. Bachmann, B. Rudak and D. Fisseler, "Analysis of the Accuracy and Robustness of the Leap Motion Controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013, doi: 10.3390/s130506380.

[26] J. Suarez and R. Murphy, "Hand Gesture Recognition with Depth Images: A Review," *Proceedings of the IEEE RO-MAN Conference*, Paris, France, 2012, pp. 411–417, doi: 10.1109/ROMAN.2012.6343787.

[27] A. Oudah, A. Al-Naji and J. Chahl, "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques," *Journal of Imaging*, vol. 6, no. 8, p. 73, 2020, doi: 10.3390/jimaging6080073.

[28] S. S. Konyar, S. Konyar and S. Sarikaya, "Hand Gesture Recognition Methods: A Literature Review," *International Journal of Applied Mathematics, Electronics and Computers*, vol. 7, no. 3, pp. 88–94, 2019, doi: 10.18100/ijamec.629578.

[29] P. Melo, A. Veloso and P. Dias, "Computer Vision-based Methodology to Improve Interaction for People with Motor and Speech Impairment," *ACM Transactions on Accessible Computing*, vol. 13, no. 4, pp. 1–31, 2020, doi: 10.1145/3408300.

[30] N. Neverova, C. Wolf, G. Taylor and F. Nebout, "ModDrop: Adaptive Multi-Modal Gesture Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1692–1706, 2016, doi: 10.1109/TPAMI.2015.2461544.

[31] K. Devineau, F. Moutarde, W. Xi and J. Yang, "Deep Learning for Hand Gesture Recognition on Skeletal Data," *Proceedings of the IEEE International Conference on Automatic Face & Gesture Recognition (FG)*, Xi'an, China, 2018, pp. 106–113, doi: 10.1109/FG.2018.00025.

[32] V. John, A. Boyali, S. Mita, M. Imanishi and N. Sanma, "Deep Learning-based Fast Hand Gesture Recognition Using Representative Frames," *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Gold Coast, Australia, 2016, pp. 1–8, doi: 10.1109/DICTA.2016.7797060.

[33] O. K. Oyedotun and A. Khashman, "Deep Learning in Vision-Based Static Hand Gesture Recognition," *Neural Computing and Applications*, vol. 28, no. 12, pp. 3941–3951, 2017, doi: 10.1007/s00521-016-2294-8.

[34] Y. LeCun, Y. Bengio and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[35] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[36] R. Elahi, A. Rehman, S. Ur Rehman and T. Saba, "Efficient Human Action Recognition Using a Deep Learning Architecture for Real-Time Applications," *IEEE Access*, vol. 10, pp. 1–17, 2022, doi: 10.1109/ACCESS.2022.3141359.

[37] A. Jalal, N. Khalid and J. T. Kim, "Automatic Recognition of Human Posture and Daily Life Activities: A Machine Learning Approach," *IEEE Access*, vol. 8, pp. 68985–68999, 2020, doi: 10.1109/ACCESS.2020.2986521.

[38] L. Pigou, A. Van Den Oord, S. Dieleman, M. Van Herreweghe and J. Dambre, "Beyond Temporal Pooling: Recurrence and Temporal Convolutions for Gesture Recognition in Video," *International Journal of Computer Vision*, vol. 126, no. 2, pp. 430–439, 2018, doi: 10.1007/s11263-016-0957-7.

[39] M. Oudah, A. Al-Naji and J. Chahl, "Human Action Recognition via Key Frame Extraction and CNN," *Mathematics*, vol. 10, no. 7, p. 1028, 2022, doi: 10.3390/math10071028.

[40] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester and I. L. McClelland, Eds. London, UK: Taylor & Francis, 1996, pp. 189–194.