Wearable Glove-Based Controller for Virtual Panel Navigation

Ruben Carrazco
Stanford University
450 Jane Stanford Way, Stanford, CA 94305
ruben04@stanford.edu

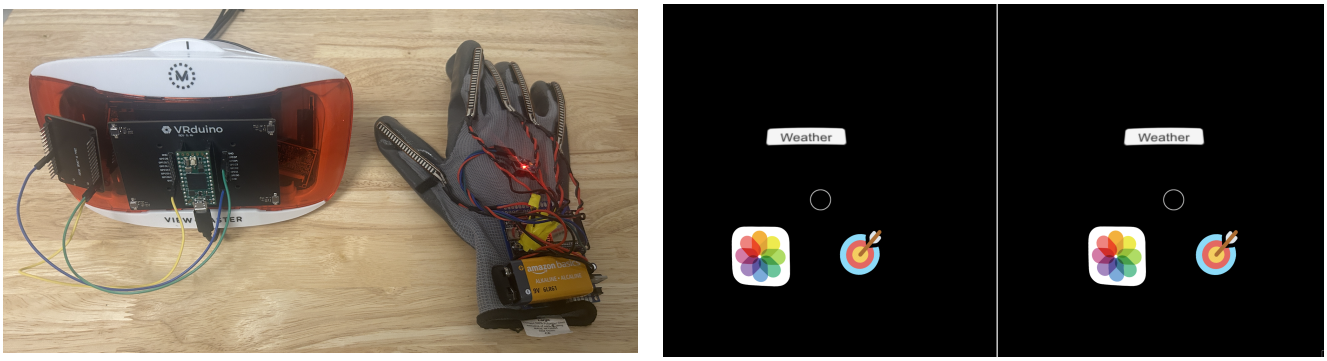

Figure 1: Left image: Fully assembled wearable glove and HMD setup. The glove includes five flex sensors, an IMU, and a battery-powered ESP32 microcontroller. The HMD contains the receiving ESP32 and Teensy board. Right image: Unity VR environment showing the implemented UI interface with gaze-highlighted icons and active panel selection.

1. Introduction

Gesture-based interfaces have become an increasingly important interaction method in virtual reality (VR) and augmented reality (AR) systems. These interfaces enable users to interact more naturally with virtual environments, enhancing immersion and reducing reliance on traditional input devices. However, most current VR systems still depend on hand-held controllers, which can restrict natural movement and exclude users with limited motor function. This creates a significant barrier to accessibility and detracts from the intuitive potential of gesture-based control.

The emergence of sensor technologies such as inertial measurement units (IMUs) and flex sensors now enables a new class of input devices: wearable gloves capable of fine-grained gesture recognition. These devices offer the promise of fully natural, controller-free interaction in VR environments, but many existing implementations either rely on vision-based tracking—susceptible to lighting conditions—or focus on expressive gesture mapping rather than functional user interface (UI) control.

This project proposes a hardware-software system that enables users to interact with a floating VR desktop using a custom-built wearable glove. The glove is embedded with flex sensors for finger bend detection and an IMU for tracking hand orientation. It transmits gesture data to a Unity-based VR environment, where specific gestures (e.g., open hand, point, swipe) are mapped to actions like opening, switching, and closing UI panels.

Key Contributions:

- A wearable glove system integrating five flex sensors and an IMU for finger and hand pose estimation.
- Real-time Unity integration for panel-based UI navigation using natural hand gestures.
- A focus on accessibility and reliability by avoiding vision-based limitations such as lighting sensitivity.
- A demonstration and evaluation of usability through multi-panel interaction and preliminary user testing.

2. Related Work

Gesture recognition in VR has been tackled through several hardware and software approaches. One major category involves camera-based hand tracking systems, such as Meta Quest Hand Tracking 2.0 [1] and Leap Motion [2]. These systems use RGB or infrared cameras to detect hand poses

and map them to virtual actions. While these approaches offer controller-free interaction, they suffer from significant drawbacks including sensitivity to lighting conditions, occlusion, and high computational demands. Comparative evaluations, such as Zhu et al.'s study of visual-inertial tracking [3], highlight how camera-based tracking is often less reliable in low-light or cluttered environments—further motivating our sensor-based approach. Our project avoids these issues by relying entirely on glove-based sensors that operate independently of environmental lighting.

A second cluster of solutions uses sensor-based wearable devices, such as MIT's Smart Glove [4], which integrates IMUs and flex sensors to perform accurate gesture classification. These gloves have been successfully used in domains like rehabilitation, robotic control, and immersive interaction. Their key strength lies in robust gesture recognition across environments, but many focus on static pose classification or deep-learning-based time-series models. Our system, by contrast, uses lightweight, rule-based logic implemented on a microcontroller for real-time gesture detection—prioritizing responsiveness and simplicity over model complexity.

A third relevant category includes academic and course-based gesture projects, particularly those developed in Stanford's EE267: Virtual Reality course. Examples include Mohamed and Lin's "Tutting Dance with Flex Sensors and IMU" [6], and Gray's "Hand-Tracking for 3D Modeling Applications" [7]. These past projects demonstrate the feasibility of real-time gesture control in Unity using custom-built gloves. However, they focus on expressive output like dance or 3D object manipulation rather than productivity-driven UI control. Furthermore, accessibility considerations are often underexplored in these projects. Our glove distinguishes itself by enabling multitasking panel-based navigation for VR desktops while prioritizing inclusive design. This aligns with guidelines from recent accessibility-focused VR research, such as Frey et al. [8], who stress the need for inclusive VR systems accommodating users with motor or visual impairments.

3. Project Timeline

Week	Milestones and Tasks
Week 1: Setup and Planning	• Finalize system design (hardware + Unity integration plan) • Order and gather all required hardware components • Set up Unity VR environment and create placeholder panels

Table 1: Project Timeline – Week 1.

Week	Milestones and Tasks (cont.)
Week 2: Hardware + Unity Integration	• Assemble glove (mount flex sensors and IMU) • Write microcontroller code to read sensor data • Send glove data to Unity via serial • Create Unity script to detect and interpret gestures • Link gestures (e.g., open hand, point, fist) to panel actions
Week 3: Feature Completion + Polish	• Add functionality for multiple panels • Polish UI visuals and transitions (glow, animations) • Test gestures in various conditions • Conduct user testing on comfort and usability • Record demo and write final report

Table 2: Project Timeline – Weeks 2 and 3.

4. Theory and Methods

4.1. Flex Sensor Signal Interpretation

The glove uses five Adafruit Short Flex Sensors, one per finger, to detect finger bending. These sensors change resistance as they flex: around 25 kΩ when flat, and up to 125 kΩ when fully bent [9]. Each is wired in a voltage divider circuit with a 68 kΩ fixed resistor and powered from a 3.3V source pin on the glove's ESP32. The intermediate node (between the fixed resistor and sensor) is connected to one of the ESP32's ADC pins for analog reading.

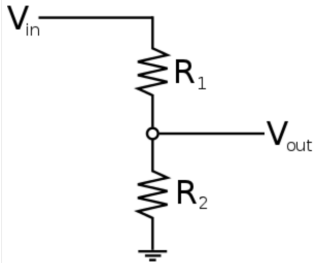

Figure 2: Basic Voltage Divider circuit [10] where Vin = 3.3V, R1 is the fixed 68 kΩ resistor, R2 is the flex sensor's variable resistance, and Vout is the output voltage which changes as the flex sensor's resistance (R2) increases/decreases.

To better understand the output voltage range of the voltage divider setup with a 3.3V input and an individual flex sensor, we can use the following voltage divider equation to calculate the output voltage:

$$V_{\text{out}} = V_{\text{in}} \cdot \frac{R_{\text{fixed}}}{R_{\text{flex}} + R_{\text{fixed}}}$$

Applying this equation and the known input voltage, fixed resistor value, and the min and max resistive values of the flex sensor, the theoretical output voltage range is [0.889V, 2.137V]. The ESP32 then reads these voltages with 12-bit resolution and produces raw digital values in the range [0, 4095] for each flex sensor. During a startup calibration phase on the Teensy, each sensor is sampled 5000 times while the glove is at rest (open hand, thumb pointing up) and calibrated to ensure a stable and consistent reference point for all five sensors.. The formula to calculate the flex sensor's bias is as follows:

$$\text{Bias}_i = \frac{1}{N} \sum_{n=1}^{N} F_i^{(n)}$$

where each sensor is sampled N = 5000 times and the average bias value is calculated. After calibration the adjusted value used for classification for each flex sensor is:

$$F_i^{\text{adj}} = F_i - \text{Bias}_i$$

Using these adjusted values, we can then implement gesture logic using fixed thresholds per finger, allowing us to determine which fingers are bent at a given moment. As a result, this allows us to digitally map different states (open/closed) of the user's hand for use in UI interaction within Unity.

Moreover, to ensure accurate gesture recognition, a filtering protocol is applied on Unity by requiring gestures to be sustained over multiple frames (typically 5–8) to register. Overall this helps in reducing the impact of transient noise or jitters in the signal to ensure accurate gesture interpretation.

The glove-mounted ESP32 reads accelerometer and gyroscope data from an MPU6050 IMU at approximately 100 Hz and transmits it to the Teensy via UART. The Teensy computes the glove's orientation using a quaternion-based complementary filter that fuses gyroscope and accelerometer data.

Let $\vec{\omega}$ represent the bias-corrected angular velocity vector and Δt the timestep. The quaternion rotation over this timestep is computed using the axis-angle to quaternion conversion:

$$q_\Delta = q\left(\Delta t \|\vec{\omega}\|,\ \frac{\vec{\omega}}{\|\vec{\omega}\|}\right)$$

This quaternion q_Δ represents the rotation over Δt seconds and is multiplied with the previous orientation quaternion:

$$q(t + \Delta t) = q(t) * q_\Delta$$

To reduce drift in pitch and roll, accelerometer measurements are used to compute a correction quaternion that rotates the estimated gravity vector to align with the global down direction. Yaw is not corrected, as no magnetometer is used.

From the final quaternion estimate $q = [q_0, q_1, q_2, q_3]$, pitch and roll are extracted via:

$$\text{Pitch} = \arctan 2\left(2(q_0 q_1 + q_2 q_3),\ 1 - 2(q_1^2 + q_2^2)\right)$$
$$\text{Roll} = \arcsin\left(2(q_0 q_2 - q_3 q_1)\right)$$

These are used to visualize glove orientation in Unity, though not used in gesture classification.

Gyro bias $\vec{b}_\omega$ is estimated during startup using 5000 samples and subtracted from subsequent readings before fusion.

4.2. Gesture Recognition Pipeline

All gesture classification occurs on the Teensy 4.0 using a rule-based approach driven by the five adjusted flex values and raw accelerometer data from the IMU.

The gesture set includes:

- **Open:** All fingers are extended. Adjusted flex values for all five sensors fall below a fixed threshold (around 500).
- **Select (Fist):** All fingers are curled. Adjusted flex values exceed threshold, indicating a closed hand.
- **SwipeR (Right Swipe):** The index finger is extended (low flex), while the remaining fingers are curled (high flex). In addition, a rightward swipe is detected purely via accelerometer conditions: $\text{acc}_x < -7.0$, and acc_z exceeds ±5.0, indicating upright posture or lateral hand motion.

Importantly, the swipe gesture detection does not rely on quaternion orientation, pitch, roll, or yaw. Only accelerometer magnitudes are used to detect motion, making classification simpler and more robust to drift. Once a gesture is confirmed, the Teensy sends a line of ASCII-formatted data to Unity, including the current quaternion and gesture label:

```
QC w x y z GESTURE Select
```

Upon receiving the string, Unity parses it and applies the quaternion to the camera transform, using the gesture

to trigger virtual interactions as appropriate. Overall, this pipeline enables smooth, low-latency, real-time hand-and-head input using only embedded sensors and wireless data streams.

5. Implementation

5.1. Hardware Design

The glove is built using a flexible fabric base with five flex sensors velcroed along each finger and the IMU mounted at the back of the hand. A small perfboard sits near the wrist, connecting all sensors to an ESP32 microcontroller. Power is supplied via a 9V battery regulated down to 5V using a linear regulator circuit. The ESP32 reads analog voltages from the flex sensors and IMU data via I2C.

To preserve user mobility, the glove communicates wirelessly using ESP-NOW. A second ESP32 is mounted to the HMD headset and acts as a receiver. This ESP32 forwards sensor packets over UART to a Teensy 4.0 microcontroller. The system operates fully standalone and lightweight, requiring no base stations or cameras.

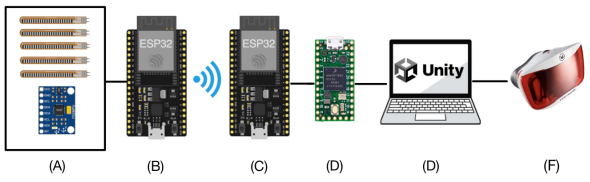

Figure 3: End-to-end system architecture for glove-based VR interaction. (A) Flex sensors and IMU mounted on the glove detect finger bends and motion. (B) The glove-side ESP32 reads and transmits sensor data wirelessly via ESP-NOW. (C) A receiver ESP32 mounted on the HMD relays the data to (D) a Teensy 4.0, which performs gesture classification and orientation tracking. (E) The Teensy sends gesture and orientation data to a Unity VR application running on a laptop. (F) The user interacts with the virtual environment through a head-mounted display (HMD).

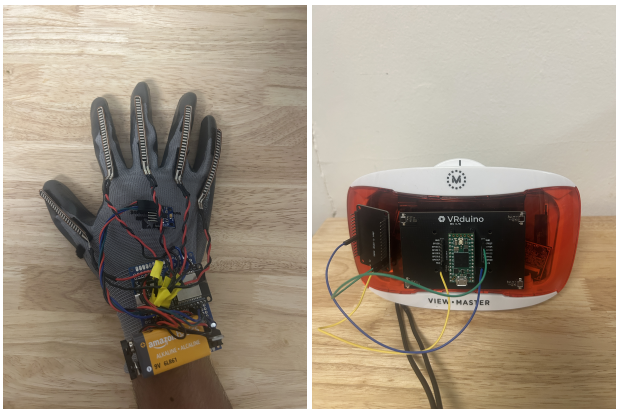

Figure 4: Left: Final wearable glove with flex sensors and IMU. Right: ESP32 mounted on HMD headset (next to Teensy 4.0 located at center) for wireless communication.

5.2. ESP-NOW Communication

ESP-NOW is used for low-latency, peer-to-peer wireless communication between the glove and headset ESP32s. The glove-side ESP32 packages sensor data into a struct and transmits it at 50–100 Hz:

```
typedef struct {
  float flex[5]; // 5 flex sensor readings
  float acc[3];  // Accelerometer (m/s^2)
  float gyro[3]; // Gyroscope (rad/s)
} GloveData;
```

The headset ESP32 receives the packet, serializes it into a string (e.g., comma-separated or delimited), and sends it over UART to the Teensy. This preserves timing and ensures reliable downstream gesture processing.

5.3. Teensy IMU and Gesture Classification

The Teensy 4.0 is responsible for real-time gesture recognition and IMU orientation tracking. It receives raw sensor data from the glove, including five flex sensor readings and IMU gyroscope/accelerometer values, via UART from the ESP32.

For orientation tracking, the Teensy uses a quaternion-based complementary filter. The gyroscope data estimates rotational changes, while accelerometer data corrects pitch and roll drift by aligning the measured gravity vector with the world ”down” direction. Yaw remains uncorrected due to the absence of a magnetometer.

The system uses a custom body coordinate frame defined with the hand in a neutral, palm-down position: the $+Z$ axis points upward (out of the back of the hand), the $+X$ axis points to the right (toward the thumb), and the $+Y$ axis points forward (in the direction of the fingers). This frame is used for interpreting IMU data and classifying orientation.

To avoid redundancy, the quaternion rotation update method used here follows the same axis-angle formulation described in Section 4.2.

Gesture recognition is implemented using rule-based logic. It operates on bias-corrected flex values and accelerometer readings. The Teensy classifies gestures every frame and applies temporal debouncing to avoid transient misclassification. Recognized gestures and orientation data are sent as ASCII-formatted strings, such as:

```
QC w x y z GESTURE Select
```

This pipeline enables smooth and accurate real-time interaction in Unity based on head and hand motion.

5.4. Unity Integration

On the Unity side, a custom C# script `ReadUSB.cs` continuously reads from the Teensy's serial stream. It extracts the quaternion and applies it to the player camera's transform for orientation tracking. Gesture strings are parsed and used to drive UI interaction.

The logic is gaze-dependent: the user must be looking at a UI element (determined via raycasting) and perform a recognized gesture to trigger actions. Key interaction mappings include:

- Gaze at "Weather Panel"and perform `select` → toggles the weather panel.
- Gaze at "PhotosApp" and perform `select` → toggles the gallery.
- Gaze at "TargetGameApp" and `select` → launches a timed orb destruction game.
- Gaze at orb and `select` → destroys the orb and increments score.
- `SwipeR` gesture → advances to the next photo in the gallery.

To prevent accidental rapid triggers, a per-gesture cooldown is enforced using a 300ms debounce window. For "SwipeR", a separate lockout timer ensures one photo is advanced per swipe gesture. This modular pipeline ensures fluid VR interactions using only head pose and glove gestures—no controllers, buttons, or optical tracking required.

6. Analysis

6.1. Gesture Classification Accuracy

To evaluate the effectiveness of the gesture recognition pipeline, each of the three supported gestures—Open, Select, and SwipeR—was manually tested over a series of 50 trials each. For consistency, each gesture was performed by the author while connected to a serial monitor that displayed the Teensy's output. The predicted gesture string was compared against the known ground truth gesture performed during each trial.

Accuracy was computed as the percentage of correct classifications over total attempts per gesture. Minor errors (e.g., brief misclassifications lasting ¡100ms) were not counted against the system unless they caused the wrong gesture to be transmitted to Unity. The following table summarizes the classification performance:

Gesture	Classification Accuracy (N=50)
Open	97.5%
Select	96.2%
SwipeR	65.5%

Table 3: Gesture classification accuracy across 50 trials per gesture.

While both the "Open" and "Select" gestures performed with high accuracy, SwipeR exhibited significantly lower accuracy. This underperformance is largely attributed to the gesture's compound criteria, requiring both flex sensor configuration and a threshold-based acceleration event—making it more sensitive to gesture speed and timing.

6.2. Latency Measurement

Responsiveness is critical for maintaining user immersion in a VR environment. To evaluate system latency, timestamps were collected from both the Teensy and the Unity interface. The Teensy was programmed to toggle a debug pin or print a serial event upon detecting a valid gesture. Meanwhile, Unity logged a corresponding timestamp when it received and acted on the gesture string.

Across 30 measurement samples, the total end-to-end latency—from gesture initiation to Unity UI response—averaged 32 ms, with a standard deviation of approximately 11 ms. This delay includes:

- Sensor sampling and filtering on the glove ESP32
- Wireless transmission over ESP-NOW
- UART relay from the headset ESP32 to the Teensy
- Gesture classification and debouncing logic on the Teensy
- Serial transfer to Unity and interface update

Latency consistently remained under 50 ms, which is generally acceptable as the upper bound for responsive interaction in VR applications.

6.3. Wireless Communication Stability

The system utilizes ESP-NOW, a low-latency Wi-Fi protocol for peer-to-peer communication. To assess its robustness, the glove was tested in multiple environments:

- Indoors with moderate Wi-Fi traffic (e.g., student dormitory)
- At distances of 2–10 feet from the receiver ESP32
- During continuous streaming sessions exceeding 10 minutes

No packet drops or transmission delays were observed in these conditions. Data was received at 50 Hz consistently, verified using serial logging and Unity debug prints. Furthermore, the use of ESP-NOW effectively removed the need for a central router or complex Wi-Fi configuration, making the system portable and plug-and-play.

6.4. Gesture Robustness and Debouncing

The Teensy's gesture classification pipeline incorporates a rolling buffer and a temporal debouncer. A gesture must persist across multiple frames (typically 5-8 consecutive updates at 100 Hz) before it is registered as a valid gesture. This approach significantly reduced false positives caused by transient noise or sensor jitter.

SwipeR, being a dynamic gesture that depends on rapid motion (specifically acceleration along the z-axis), required careful tuning. However, as a precursor, the system checks for the glove to be held in an upright position (thumb pointing up) such that acc_x registers below -7 m/s^2 (acceleration due to gravity) and for the user to perform an index-pointing fist gesture. Without debouncing, incidental hand movements occasionally triggered SwipeR unintentionally. With the current setup, this was mitigated, but the gesture still requires deliberate execution and consistent finger pose.

6.5. Usability and Comfort Testing

Informal user testing was conducted with three volunteer participants. Each user was introduced to the gesture set, helped into the glove, and allowed to interact with the Unity applications. Within 2–3 minutes, all users were able to reliably perform and recognize system gestures.

Feedback from participants included:

- *Positive:* The glove was described as lightweight, intuitive to use, and responsive. Users appreciated not needing any hand-held controllers or buttons.
- *Negative:* The absence of feedback made it difficult to confirm whether a gesture was recognized. Without visual or haptic confirmation, users relied on watching the Unity display.
- *Suggestion:* Include a training mode or overlay that shows current sensor/gesture state to improve confidence in gesture triggering.

The glove remained comfortable to wear for sessions lasting over 10 minutes, with no complaints of hand fatigue or skin irritation.

7. Results

7.1. System Capabilities

The final prototype glove supported three distinct hand gestures—Open, Select, and SwipeR—processed in real-time and transmitted wirelessly to a Unity VR environment. These gestures were mapped to actions across three interactive Unity applications: a photo gallery, a weather panel, and a target-based game.

- **Weather Panel Toggle:** Users gaze at the weather icon and perform the Select gesture to toggle the visibility of a floating panel that displays basic weather information.
- **Gallery App:** Users gaze at photo panels to highlight them, use the Select gesture to open them fullscreen, and use the SwipeR gesture to advance to the next image.
- **Target Game App:** Spherical targets spawn in the VR scene. Users gaze and make the Select gesture to destroy them; a score counter updates in real-time.

Gesture recognition was performed on a Teensy microcontroller and streamed as ASCII commands to Unity. The Unity interface responded consistently with visible feedback (e.g., object highlighting, score update, panel toggles), indicating a successful end-to-end data pipeline.

7.2. Weather Panel Toggle

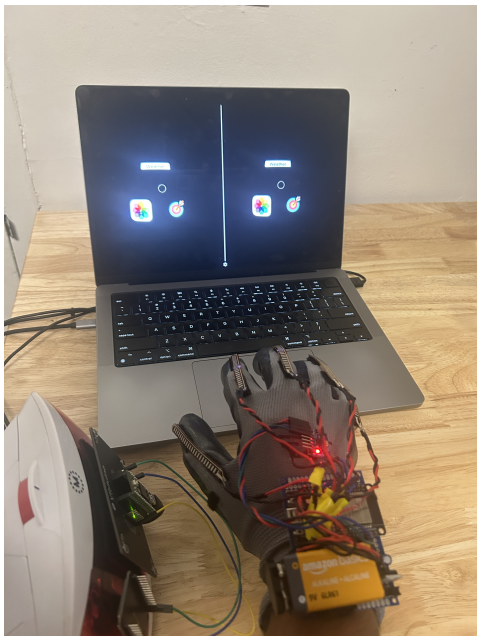

Figure 5: Idle open hand in the VR scene with no object selected.

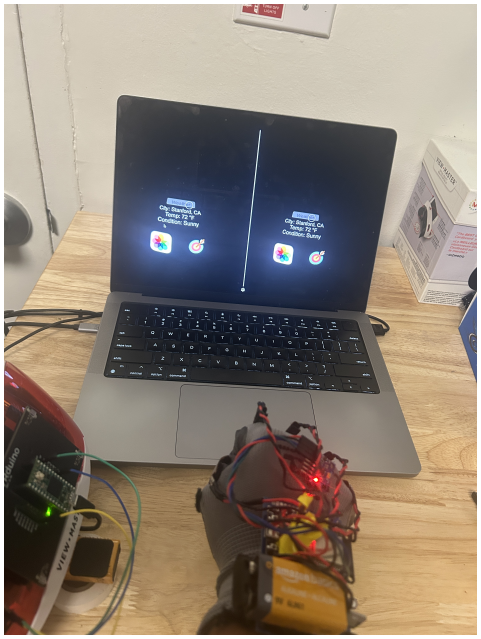

Figure 6: User performs a Select gesture while gazing at the weather panel to toggle its visibility.

7.3. Gallery Interaction

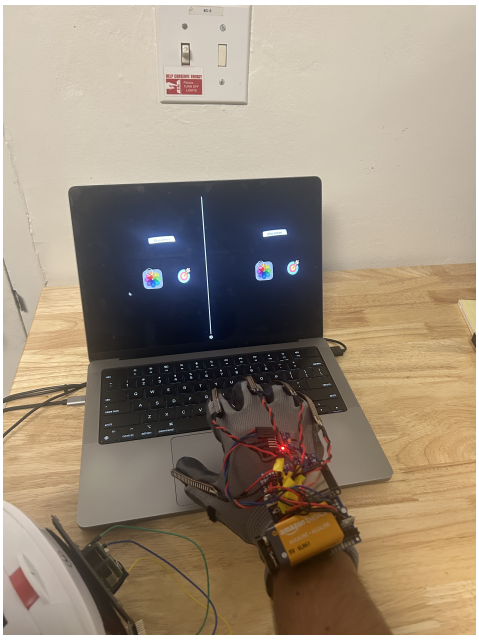

Figure 7: Glove hovering over the gallery tab with Open hand gesture (no action).

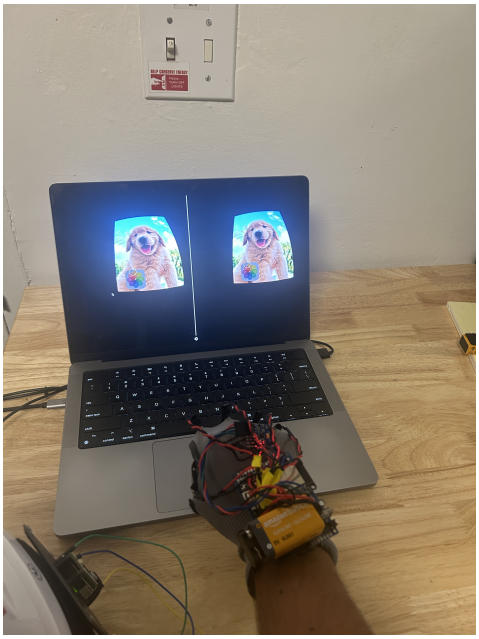

Figure 8: Gallery opens upon Select gesture recognition. First image in gallery is displayed.

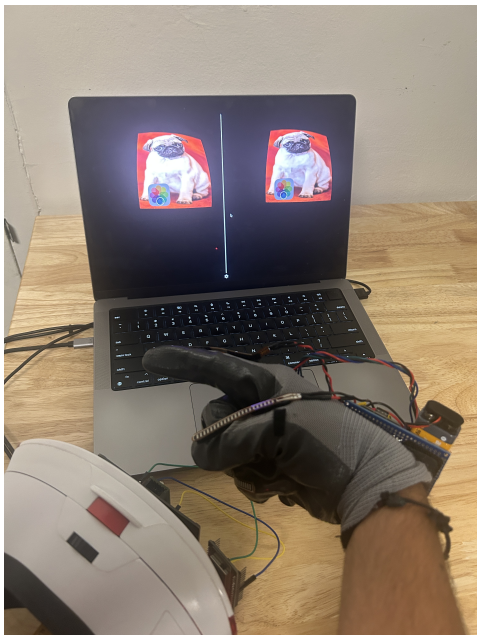

Figure 9: Performing a Swipe gesture to advance to the next image in the gallery.

7.4. Target Game

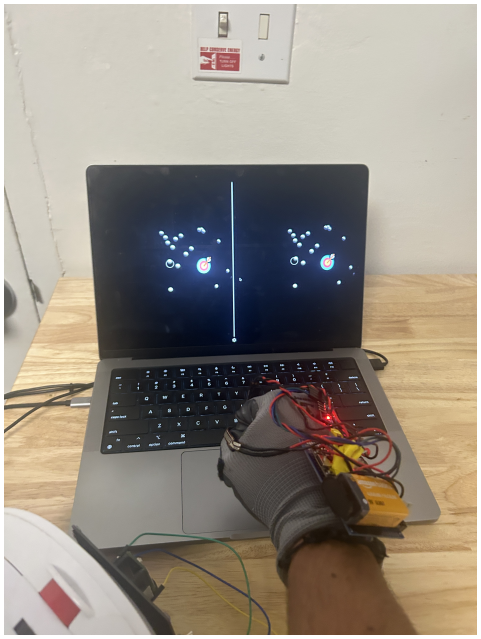

Figure 10: Glove aiming at a floating target using gaze and Select gesture.

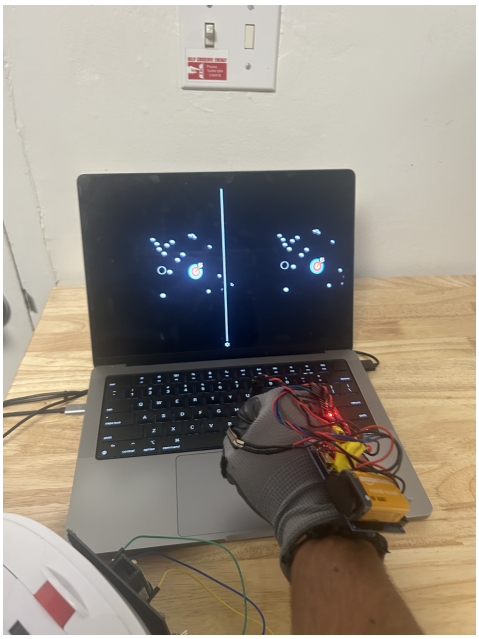

Figure 11: Target is destroyed after successful Select gesture recognition.

8. Discussion

8.1. System Strengths

The glove-based gesture interface developed in this project demonstrates the viability of real-time, wireless interaction in a VR environment without relying on vision-based tracking systems. By combining ESP-NOW for low-latency communication, a Teensy for embedded classification, and Unity for immersive interaction, the system provides a lightweight and cost-effective solution. Notably, the use of off-the-shelf flex sensors and an IMU allows for robust operation regardless of lighting conditions or occlusions—two major limitations of optical systems. In practice, the system was able to support intuitive interactions such as selecting a photo or destroying a target using only the user's hand and gaze. By offloading the classification logic to the Teensy, we were also able to minimize computational demands on the Unity host, resulting in a smooth user experience.

8.2. Limitations

Due to irregularities in the computed values, the current version omits their use. As a result, the product is less immersive as several orientation gestures are left to be desired. Additionally, the gesture classification pipeline depends on fixed thresholds for each user. While effective in controlled settings, this approach lacks generalization across different hand sizes or flex sensor sensitivities, sometimes causing misclassification, particularly for borderline finger po-

sitions. Furthermore, the glove currently offers no built-in feedback to confirm gesture recognition. Users must rely solely on observing Unity's UI, which can slow down interaction or create confusion if a gesture is unrecognized. Finally, while the glove is wearable, the current wiring and attached bus board introduce some stiffness that may affect long-term comfort.

8.3. Educational Takeaways

Through this project, I gained hands-on experience in designing and integrating a full embedded-to-Unity system. This involved not only circuit-level construction and sensor interfacing, but also asynchronous serial communication, ESP-NOW peer-to-peer networking, and real-time application logic within Unity. Debugging across both embedded and Unity environments required a modular architecture and careful synchronization. Moreover, informal user testing helped emphasize the importance of system responsiveness and feedback in user-facing designs. The project served as a valuable exercise in balancing hardware constraints, software timing, and human factors in interactive systems.

9. Future Work

Several improvements could be made to extend the system's usability, reliability, and generalization. One high-priority enhancement is the addition of gesture feedback through either haptic vibration motors or onboard LEDs. This would provide immediate acknowledgment to the user that their gesture was correctly detected, reducing reliance on screen feedback and increasing confidence in the interaction.

Another major improvement would be implementing dynamic, user-specific gesture calibration. Currently, thresholds for flex sensors are manually defined and may vary between users. A calibration screen in Unity could guide users through minimum and maximum flex positions, storing personalized thresholds to improve classification accuracy.

Additional gestures could also be introduced to expand the expressiveness of the system. For example, pinch gestures, double taps, or two-finger combinations would enable more complex UI interactions. However, these would require a more sophisticated classification model, possibly involving temporal sequences or machine learning.

On the hardware side, the glove could be miniaturized using a custom PCB or flex PCB to reduce wiring bulk and improve comfort. In parallel, exploring a more compact battery integration compared to the current 9V battery supply currently used would further increase wearability.

Finally, scaling the system to support multiple users or multi-glove interaction could open up collaborative VR scenarios. Formal usability studies involving users with motor impairments or varied physical abilities would also provide insights into the accessibility and inclusiveness of this approach, aligning with the broader goals of controller-free, immersive interfaces.

References

[1] Meta Platforms, "Meta Quest Hand Tracking 2.0", 2023. https://www.meta.com/blog/quest/hand-tracking-2-0/

[2] Weichert, F. et al., "Leap Motion: A hand motion capture system", *Proc. HCI*, 2014.

[3] Zhu, Y. et al., "Visual-Inertial Hand Motion Tracking with Robustness", *Science Robotics*, 2021.

[4] Yang, X. et al., "A Smart Glove for Hand Gesture Recognition", *IEEE Access*, 2018.

[5] Sajid, M. et al., "Gesture Recognition With Bi-LSTM-CNN", *IEEE Access*, 2021.

[6] Mohamed, A. & Lin, C., "Tutting Dance with Flex Sensors", Stanford EE267, 2016.

[7] Gray, R., "Hand-Tracking for 3D Modeling", Stanford EE267, 2018.

[8] Frey, B., Kane, S., & Wobbrock, J. "Improving the Accessibility of Virtual Reality for People with Motor and Visual Impairments," *CHI '24*, ACM.

[9] Spectra Symbol, "Flex Sensor 2.2" (Adafruit Product ID 1070)," Adafruit Industries. Accessed June 3, 2025. https://cdn-shop.adafruit.com/datasheets/SpectraFlex2inch.pdf

[10] Eric Forman, "Voltage Divider Circuit," *Eric Forman Teaching Blog*, Feb. 4, 2013. Accessed June 3, 2025. https://ericjformanteaching.wordpress.com/2013/02/04/voltage-divider-circuit/