Gesture Glove System: Sensor Accuracy, Display Efficiency, and Visualization

Prashant Chotkan, Jamai Esseboom, Jill den Ouden

Abstract—**This research explores the optimization of gesture-based interfaces by evaluating the performance of key components in a gesture glove system, including the MPU6500 motion sensor, flex sensors, and the SSD1331 OLED display. The study focuses on improving gesture recognition accuracy and real-time visualization, addressing common challenges such as sensor inaccuracies and display latency. Experimental tests reveal that partial updates on the SSD1331 OLED display enhance refresh rates, while the MPU6500 sensor provides precise tracking of hand movements. Flex sensors positioned on the back of the finger demonstrate superior accuracy in detecting finger curvature. These findings offer valuable insights for the development of more responsive and accurate gesture-based control systems.**

Index Terms—**gesture recognition, MPU6500, flex sensors, SSD1331 OLED, real-time visualization, gesture glove, human-computer interaction**

I. Introduction

Gesture-based interfaces are increasingly used in various applications, from assistive technologies to virtual reality, yet optimizing their performance remains a challenge. Existing gesture recognition systems often suffer from limitations such as inaccurate sensor data, slow response times, and poor visualization of gestures, which hinder usability and effectiveness.

This research aims to evaluate the key components of a gesture glove system, specifically motion sensors (MPU6500), flex sensors, and the SSD1331 OLED display, to determine their impact on gesture recognition accuracy and real-time visualization. Understanding the limitations and capabilities of these components will help improve gesture-based control systems, making them more reliable and responsive.

This research is valuable to engineers, developers, and researchers working in human-computer interaction, wearable technology, and assistive devices. By providing data-driven insights into component performance, this study contributes to the design of more efficient and accurate gesture-based interfaces, which can benefit applications in healthcare, gaming, robotics, and augmented reality.

II. Assignment Description

A. Problem Definition

Gesture-based interfaces are widely used in applications such as assistive technology, virtual reality, and human-computer interaction. However, existing gesture recognition systems face several challenges that hinder their usability and effectiveness. These challenges include **inaccurate sensor data, slow response times, and poor real-time visualization of gestures**, all of which reduce the reliability of such systems.

One of the main limitations is the **accuracy of motion tracking and finger curvature detection**. Many systems rely on inertial measurement units (IMUs) and flex sensors, but their placement, calibration, and data processing methods significantly impact recognition accuracy. Some systems rely on a camera and ai technology to detect gestures and hand movements which are very prone to instability issues. Additionally, **real-time visualization of gestures remains inefficient**, as current display technologies may introduce latency or fail to represent movements smoothly.

This research addresses these challenges by evaluating the **MPU6500 motion sensor, flex sensors, and the SSD1331 OLED display** in a gesture glove system. The study investigates how sensor placement, data processing strategies, and display update techniques influence gesture recognition accuracy and real-time visualization performance. By identifying the limitations of these components, this research aims to improve gesture-based control systems, making them more responsive, precise, and suitable for applications in **healthcare, gaming, robotics, and augmented reality**.

B. Requirements

- **Finger Curvature Detection** – The system must accurately detect and measure the bending of the fingers.
- **Glove Position and Direction Tracking** – The system must determine the orientation and movement of the glove.
- **Real-Time Data Processing** – The system must process and interpret sensor data with minimal latency for immediate feedback.
- **Wireless Data Transmission over Wi-Fi** – The system must transmit gesture data wirelessly to an external device for further processing or visualization.
- **Simulated Hand Representation** – The system must include a virtual hand model that visually replicates the detected gestures in real time for verification and analysis.

III. Research questions

This research focuses on the question: How can a gesture glove system effectively detect finger curvature, track hand position, and provide real-time visualization? To break this down further, the following subquestions are explored.

1) How do different update strategies and timing intervals affect the responsiveness of the SSD1331 display?
2) How effectively does an MPU6500 track the position and movement of the glove?

3) How does flex sensor placement influence accuracy in measuring finger curvature?
4) How to digitally simulate a hand using real-time collected sensor data?

IV. Methods

A. SSD1331 Display

The SSD1331 is a 96×64 pixel OLED display commonly used in embedded systems. Its refresh rate directly impacts smoothness and responsiveness, making it crucial for real-time applications. This study examines how update strategies (full-screen vs. partial updates) and refresh timing intervals affect display performance.

1) Research Question: How does the refresh rate of the SSD1331 display respond to different update strategies and timing intervals?

We hypothesize that

- Updating fewer pixels results in a faster refresh rate.
- A minimum processing time is required per refresh cycle.
- Excessively frequent refreshes may cause rendering artifacts or frame skipping.

The study is conducted using a TTGO T2 ESP32 board with an SSD1331 OLED display, programmed in C with the Arduino framework. The evaluation consists of two test cases:

- **Full-Screen vs. Partial Updates:** Measuring refresh time for a full-screen update compared to a smaller region (e.g., 10×10 pixels).
- **Effect of Refresh Interval:** Varying update intervals from a few milliseconds to hundreds of milliseconds to assess performance, potential artifacts, and frame skipping.

Results will determine optimal update strategies for improving display performance in real-time applications.

B. MPU6500

1) Relevance of the MPU Sensor: The MPU6500 is essential for tracking hand movement and orientation. It detects shifts along the X, Y, and Z axes (acceleration) and tilting (gyroscope), ensuring reliable hand gesture recognition.

2) Choosing the Right MPU: The MPU9250 was initially selected for its 9 degrees of freedom (gyro, accelerometer, and magnetometer) [2], allowing precise movement and absolute orientation tracking. Its compact size, strong documentation, and wide library support make it ideal. While alternatives like the ICM-20948 exist, the MPU9250 balances performance and availability. For applications not requiring absolute orientation, the MPU6050 is a simpler alternative. However, for stable tracking with minimal drift, a magnetometer-equipped sensor is preferred.

3) MPU Sensor Testing: Testing revealed that the provided sensor was the MPU6500 instead of the expected MPU9250. Despite this, efforts focus on successfully integrating the MPU6500 into the system.

a) Including the Libraries: To interface with the MPU6500 sensor, it is advisable to use a library that facilitates communication over I^{2}C or SPI, as outlined in the sensor's specifications [1]. Several open-source libraries are available that support the MPU6500, simplifying data retrieval from the sensor. The test setup will require the MPU6500 sensor and a compatible micro controller for proper integration.

b) Test Cases: For the test cases you will need a MPU6500 sensor, a ESP32 and a pc/laptop running arduino.ide and python. To run the code for MPU6500 you first need to connect the sensor to the ESP32.

- **GND** – Connect to GND
- **VCC** – Connect to 3.3V
- **SCL** – Connect to GPIO 22
- **SDA** – Connect to GPIO 21

4) IMU Acceleration: This test ensures that the Acceleration meter within the MPU6500 works as expected. **Requirement:** The system must detect various movements along the X, Y, and Z axes. These movements correspond to shifts in hand position.

- **Description:** When the hand is moved, the sensor transmits data to the serial monitor, displaying the X, Y, and Z values.
- **Steps:**
  1) Connect the sensor.
  2) Move the hand.
  3) Verify whether the movements are detected.
- **Acceptance Criteria:** The movements must be correctly detected and printed in the serial monitor.

5) IMU Gyroscope: This test ensures that the Gyroscope meter within the MPU6500 works as expected. **Requirement:** The system must detect various hand movements, specifically tilting.

- **Description:** When the hand is moved, data is transmitted to the serial monitor. The data can be checked by rotating the sensor and hold it to a 90 degrees angle.
- **Steps:**
  1) Connect the sensor.
  2) Move the hand.
  3) Verify whether the movements are detected.
- **Acceptance Criteria:** The movements must be correctly detected and printed in the serial monitor.

6) IMU Gyroscope Accuracy: This test ensures that the gyroscope within the MPU6500 provides reliable readings. **Requirement:** When the sensor is held at different angles, it should return accurate values.

- **Description:** The sensor's output is tested at various angles to verify accuracy.
- **Steps:**
  1) Connect the sensor.
  2) Hold the sensor at a 90-degree angle.
  3) Verify whether the measured angle matches the actual orientation.
- **Acceptance Criteria:** The measured value must not deviate by more than 5 degrees from the actual angle.

7) IMU Acceleration Accuracy: This test ensures that the accelerometer within the MPU6500 provides reliable readings. A Python program is used to visualize sensor data by displaying a moving cube that mimics the sensor's motion. **Requirement:** When moving the sensor, it should return accurate acceleration values corresponding to the movement.

- **Description:** The sensor's accuracy is tested by comparing its output to the motion of a virtual cube on a screen.
- **Steps:**
  1) Connect the sensor.
  2) Close the serial monitor, then start the Python program.
  3) Move the sensor at different speeds and directions.
  4) Verify whether the cube's movement matches the sensor's motion.
- **Acceptance Criteria:** The cube on screen must move in the same direction as the sensor and stop when the sensor stops moving.

C. Flex Sensor

A flex sensor was chosen to measure the bending angle or curvature of the fingers due to its simplicity and effectiveness in detecting finger movements. Compared to other alternatives such as inertial measurement units (IMUs) or strain gauges, flex sensors offer a straightforward integration process, consume low power, and provide analog output that can be easily processed by a microcontroller. While IMUs are not suitable for measuring the curvature of the fingers specifically, they can be used for detecting other movements. Strain gauges can be used to measure finger curvature, but they require precise placement on the glove and are less accurate than a flex sensor while also being relatively more expensive.

To evaluate the performance of the flex sensor in detecting hand gestures, a controlled test setup was designed. The sensor was attached to the index finger at two different locations, as shown in Fig. 1:

1) **Inside the finger (palm-side placement)** – to measure curvature directly as the finger bends inward.
2) **Outside the finger (backhand placement)** – to analyze if it provides a more stable or accurate reading.

The sensor was connected to an ESP32 microcontroller's Analog-to-Digital Converter (ADC) pin to continuously measure the resistance change as the finger moves. A custom testing procedure was followed:

- The hand started in a fully open and straight position.
- The subject gradually curled their fingers into a tight fist over a period of 5 seconds.
- ADC values were recorded in real-time and logged for analysis.
- The test was repeated multiple times to ensure consistency and identify the optimal placement of the sensor.

Fig. 1. Flex Sensor Placement

This experiment aimed to determine the most effective sensor placement by analyzing the signal consistency. By systematically comparing the ADC readings, the goal was to identify the placement that provides the clearest and most reliable data for detecting hand gestures accurately.

D. Simulation

To visualize the sensor data in real-time, we evaluated three simulation programs: Gazebo, Webots, and Unity. The primary goal was to simulate hand movements based on real-time sensor input without requiring complex features such as collision detection. The simulation program simply needed to receive and visualize the sensor data, translating it into hand movements.

1) Simulation Program Evaluation: The programs will each be tested separately by setting them up and trying a simple simulation. We compared the three simulation tools based on their ease of use and ability to visualize hand movements effectively.

- **Gazebo:** Gazebo is a popular robotics simulator that integrates well with ROS and can handle multiple sensors and hardware. While it is highly accurate for simulating physics and controlling robots, its capabilities were found to be more advanced than necessary for this project, where simple movement visualization suffices.
- **Webots:** Webots is another robotics simulation tool with a user-friendly interface, making it easy to simulate and visualize real-time sensor data. While it lacks the deep integration with ROS found in Gazebo, its simplicity and efficiency make it ideal for visualizing basic hand movements based on sensor inputs.
- **Unity:** Unity, typically used for game development, offers high-quality graphics and flexibility for creating custom visualizations. However, its advanced features are better suited for complex simulations, making it less efficient for our needs, which focused on straightforward hand movement visualization.

V. Results

A. SSD1331 Display

The collected data shows the following trends:

- Full-screen updates take significantly longer than partial updates. The function `fillScreen()` required approximately 172 ms to complete, whereas updating a small 10x10 pixel region only took around 3 ms.
- Partial updates are significantly more efficient than full-screen redraws. This suggests that applications should avoid unnecessary full refreshes and instead update only necessary regions to optimize performance.
- No missing frames were observed even at high refresh rates. To verify this, slow-motion recording was used, which confirmed that frames were rendered sequentially without drops or skips.

B. MPU6500

All tests for the MPU6500 sensor were successfully executed, verifying its expected functionality. The results for each test are summarized below:

1) IMU Acceleration Test: The sensor correctly detected motion along the X, Y, and Z axes. Data was displayed in the serial monitor, confirming expected functionality.

2) IMU Gyroscope Test: The gyroscope accurately detected tilting movements. At a 90-degree angle, the measured values corresponded to the expected rotation, meeting the acceptance criteria.

3) IMU Gyroscope Accuracy Test: Measured values deviated by no more than 3 degrees from expected orientations, within the ±5-degree tolerance, confirming accuracy.

4) IMU Acceleration Accuracy Test: A Python program displayed a virtual cube mimicking sensor movement. The cube moved accurately in sync with the sensor, verifying data reliability.

All acceptance criteria were met, demonstrating the MPU6500's capability to measure acceleration and angular velocity reliably in motion detection applications.

C. Flex Sensor

The results of the flex sensor tests demonstrated that both placements provided similar trends in ADC readings, ranging from 0 (completely unbent) to a maximum of 3038 (tight fist). However, the backhand placement showed slightly lower maximum adc value indicating a lower measuring range.

Time (s)	0	1	2	3	4	5
Palm-Side Placement	0	751	1507	2251	2898	3038
Backhand Placement	0	724	1478	2207	2854	3030

TABLE I
ADC READINGS OVER TIME FOR FLEX SENSOR AT DIFFERENT PLACEMENTS.

As seen in Table I, the palm-side placement provided slightly more curvature compared to the backhand placement, which showed minor variations in maximum ADC values. Furthermore bending of the flex sensor when placed palm-side appeared more difficult as bending the fingers decreases the surface on which the flex sensor is mounted. This leads to the flex sensor getting loose from the finger. On the backhand placement the flex sensor is more stable as the contact surface increases a little when bending the fingers. This indicates that the sensor is more stable when positioned on the backhand.

D. Simulation

After comparing and testing the three options, Webots was selected as the most suitable tool for this project due to its balance of simplicity, and ease of use in real-time data visualization. Furthermore this program was easiest to setup overall, making it more accessible for rapid prototyping and iteration. Its user-friendly interface, combined with robust documentation and community support, facilitated quick development cycles, which were crucial for meeting project deadlines

VI. Conclusion

This study successfully integrated key components for a gesture-controlled embedded system, with a focus on performance optimization and accurate hand gesture recognition. The MPU6500 sensor proved reliable for motion and orientation tracking, ensuring precise hand movement detection through both accelerometer and gyroscope data. The flex sensor, placed on the backhand, provided consistent and accurate curvature detection, essential for recognizing gestures like bending. The SSD1331 display showed improved performance when using partial updates, avoiding frame loss and ensuring smoother real-time interactions. Webots was chosen as the simulation tool for its ease of use and effective visualization of real-time sensor data. This work demonstrates the importance of sensor placement and optimization in achieving reliable gesture recognition and provides a foundation for enhancing real-time embedded applications. Further testing could explore the impact of different display controllers, SPI clock speeds, and alternative sensor placements to further improve system performance.

VII. References

1) TDK InvenSense, *MPU-6500 Product Specification*, Revision 1.3, Document Number: PS-MPU-6500A-01, June 2020. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2020/06/PS-MPU-6500A-01-v1.3.pdf
2) InvenSense Inc., "MPU-9250 Product Specification," PS-MPU-9250A-01, Jun. 2016. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
3) LilyGO, "GitHub - LilyGO/TTGO-T2-SSD1331-SD: TTGO-T2 V1.6," GitHub. Available: https://github.com/LilyGO/TTGO-T2-SSD1331-SD