IoT-Based Virtual Drawing System Using Motion Tracking

Submitted by: Liron Adi 322528787,
Omar Jaber 212959936
Supervisor: Guy Tel-Zur
Course: Internet of Things
Ben Gurion university of the negev
Be'er sheva, Israel
omarj@post.bgu.ac.il, lironad@post.bgu.ac.il

I. ***Abstract - This paper presents an Internet of Things (IoT) - based virtual drawing system that enables users to generate two-dimensional drawings from free-hand motion. The system uses an MPU6050 inertial measurement unit (IMU) connected to an ESP32 microcontroller to capture hand movement in real time. During operation, the device streams IMU samples at approximately 50 Hz and transmits the measurements securely over MQTT to Amazon cloud services, where it is routed and delivered in real time to a web application. The web client visualizes the motion as a continuous 2D trajectory, allowing intuitive and contact-free interaction without requiring special installations or IoT credentials on the client side. Experimental observations indicate that the system is responsive and capable of producing smooth drawing paths, demonstrating the potential of IoT-based motion tracking for interactive and creative applications.***

II. INTRODUCTION

The Internet of Things (IoT) has enabled the development of systems that connect physical devices to cloud-based services through sensing, communication, and data processing. Such systems allow real-time interaction with the physical world and support a wide range of applications.

In parallel, motion-based human–computer interaction has gained increasing attention due to its potential to provide natural and intuitive user interfaces. Traditional digital drawing systems rely on physical contact with input devices such as touchscreens, mice, or styluses. While effective, these interfaces restrict freedom of movement and do not fully exploit free-hand motion as an input modality.

Recent advances in low-cost inertial sensors and wireless communication make it feasible to capture and process motion data in real time. Inertial Measurement Units (IMUs), combining accelerometers and gyroscopes, are widely used to estimate movement and orientation. When integrated with IoT pipelines, IMU-based systems can stream motion data to cloud platforms for real-time processing and visualization.

This paper presents an IoT-based virtual drawing system that transforms free-hand motion into a real-time two-dimensional drawing. The system captures IMU measurements using a microcontroller-based wand, transmits the data wirelessly over MQTT to cloud services, and visualizes the resulting 2D trajectory in a web application. The proposed approach enables contact-free drawing and demonstrates an end-to-end integration of motion sensing, wireless communication, and real-time visualization.

III. BACKGROUND AND RELATED WORK

A. *IoT-Based Motion Tracking Systems:*

IoT-based motion tracking is common in wearable and activity-monitoring systems. In most cases, the device collects motion data locally, sends it wirelessly to the cloud, and the cloud performs processing and analysis to generate results or feedback. The paper we rely on [1] follows this same idea, and it serves as a starting point for our background discussion.

B. *Wearable IMU Sensors for Arm Motion Analysis:*

Wearable inertial sensors, particularly Inertial Measurement Units (IMUs), have been extensively used for tracking arm movements. IMUs commonly integrate accelerometers and gyroscopes to measure linear acceleration and angular velocity along multiple axes. Due to their low cost, compact size, and real-time capabilities, sensors such as the MPU6050 are frequently embedded in wrist-worn devices for motion monitoring applications. [1]
Fig. 1 illustrates a typical architecture of wearable IoT-based motion tracking systems. In this architecture, motion data is captured using an IMU sensor and transmitted via a microcontroller with wireless

connectivity to a cloud platform, where data processing and analysis are performed. [1]

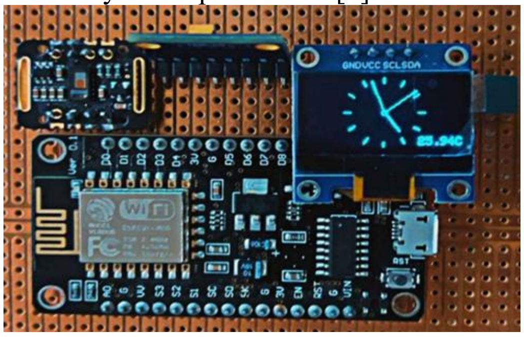


Fig. 1. Architecture of a wearable IoT-based motion tracking system [1].

C. *Machine Learning for Motion Classification:*

Several studies have investigated the use of machine learning techniques for analyzing motion data collected from wearable sensors, In particular, Asghar et al. [1] proposed an IoT-based system for classifying arm exercises using a wrist-worn band equipped with an MPU6050 sensor. Motion data collected from multiple users was transmitted to a cloud platform, where machine learning algorithms such as Fine K-Nearest Neighbors (KNN), Subspace KNN, Boosted Trees, and Quadratic Support Vector Machines (SVM) were applied for exercise classification. The results demonstrated that the Fine KNN algorithm achieved the highest classification accuracy of 91.3% [1], highlighting the effectiveness of lightweight machine learning models for wearable motion analysis.

D. *Limitations of Existing Approaches:*

Despite promising results, most existing studies focus primarily on motion classification rather than on real-time motion visualization. The analyzed systems are designed to determine whether a specific exercise is performed correctly, but they do not generate a continuous geometric representation of the motion itself.

Furthermore, many approaches rely on predefined motion classes and are not intended for creative or free-form interaction. These limitations indicate a research gap in transforming raw motion data into continuous visual trajectories for interactive applications.

As shown in Fig. 2, the overall system operation follows a sequential processing flow. Motion data is first captured by the IMU sensor and transmitted to the cloud through a wireless microcontroller. The cloud layer then performs data preprocessing and motion analysis before generating the final output. This flow-based representation highlights the end-to-end operation of a typical IoT-based motion tracking system.

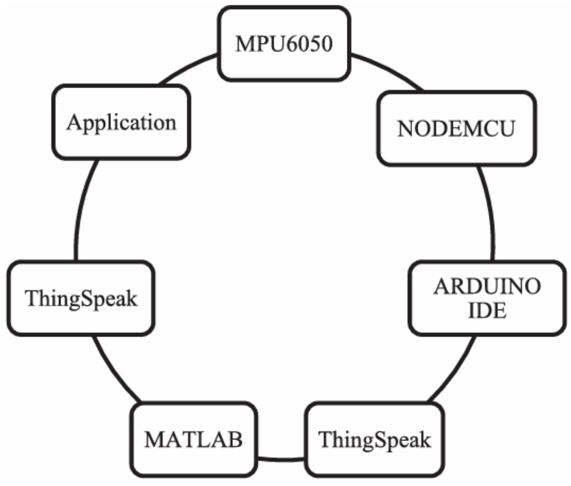


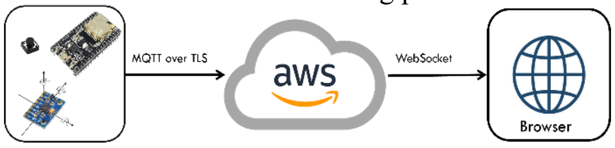
Fig. 2. Flow diagram of a wearable IoT-based motion tracking system [1].

IV. SYSTEM ARCHITECTURE AND METHODOLOGY

A. *System Overview*

The proposed system implements an end-to-end IoT pipeline for real-time motion drawing. It is organized into three main layers: an edge sensing device, a cloud backend, and a web visualization client. The edge device is a handheld wand based on an ESP32 microcontroller and an MPU6050 IMU. It samples inertial measurements at approximately 50 Hz and produces a stream of motion samples, while a push-button is used to control when streaming is active (i.e., data is transmitted only during user interaction). The cloud backend receives the incoming MQTT messages, performs routing and lightweight processing, and distributes the resulting motion stream to connected users. Finally, the web client runs in a standard browser, subscribes to the real-time stream, and renders the motion as a continuous 2D trajectory.

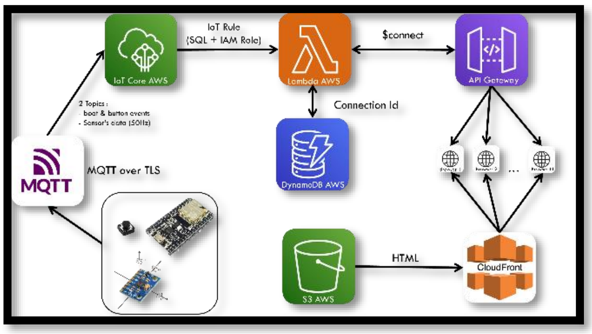
- **Edge layer:** ESP32 + MPU6050 wand, streaming IMU samples (~50 Hz) with button-controlled transmission.
- **Cloud layer:** MQTT ingestion, message routing/ processing, and real-time fan-out to web clients.
- **Client layer:** Browser-based visualization that renders a continuous 2D drawing path in real time.



B. *End-to-End Data Flow*

This subsection describes the end-to-end path of the motion stream, from the edge device to the browser, and the mechanism used to deliver real-time updates to multiple clients.

1. **IMU sampling and event detection (Edge) -** The ESP32 reads inertial measurements from the MPU6050 at approximately 50 Hz. In parallel, a push button is monitored to control when streaming is active.
2. **Message formation and publishing -** When streaming is enabled (button pressed), each IMU sample is packaged into a compact payload and published as an MQTT message. When the button is not pressed, the device remains idle and does not publish motion messages.
3. **Secure device-to-cloud uplink -** MQTT messages are transmitted over a secure channel (MQTT over TLS) to the cloud ingestion layer.
4. **Cloud-side routing and processing -** Incoming messages are routed to a processing function that performs lightweight handling and prepares the data for web delivery (e.g., formatting and mapping to a drawing trajectory representation).
5. **Client fan-out and real-time delivery -** The processed updates are pushed to all currently connected web clients through a WebSocket broadcast mechanism. Active connections are tracked in a connection store, enabling the system to deliver the same stream to multiple browsers concurrently.
6. **Browser rendering -** The web client receives the WebSocket messages and updates the displayed 2D trajectory in real time, producing a continuous drawing path that reflects the user's motion.

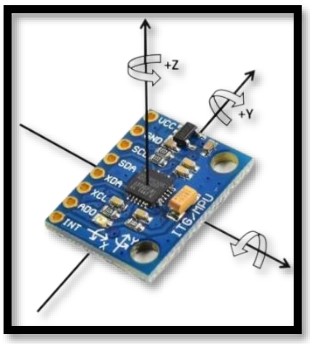


Flow path

C. *Hardware Implementation*

The edge device is implemented as a handheld "wand" based on an ESP32 microcontroller connected to an MPU6050 inertial measurement unit (IMU). The MPU6050 integrates a three-axis accelerometer and a three-axis gyroscope, enabling the measurement of both linear acceleration and angular velocity of the user's hand. This combination allows continuous tracking of motion and orientation using a compact and low-cost sensor suitable for real-time applications.

The IMU provides accelerometer and gyroscope measurements, which are sampled at approximately 50 Hz. This sampling rate offers a balance between temporal resolution and communication overhead, ensuring responsive motion tracking while maintaining system efficiency.



MPU6050

A push button is included to control user interaction and to gate the transmission of motion data (i.e., motion messages are sent only while the button is pressed). This mechanism allows the user to explicitly control when drawing is active and reduces unnecessary data transmission.

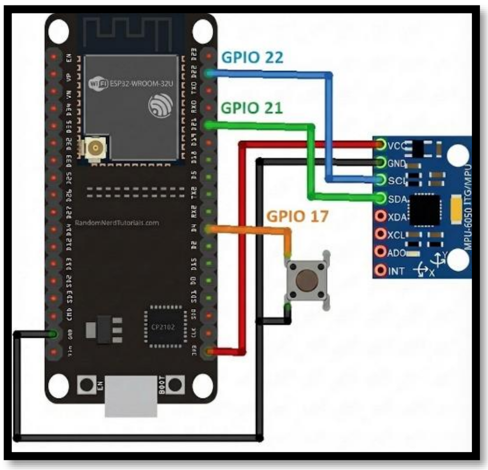
Wiring summary - The MPU6050 is interfaced with the ESP32 using the I²C bus. The connections follow the wiring shown in the hardware diagram:

- VCC (MPU6050) → 3.3V (ESP32)
- GND (MPU6050) → GND (ESP32)
- SCL (MPU6050) → GPIO22 (ESP32)
- SDA (MPU6050) → GPIO21 (ESP32)

The push button is wired as a digital input:

- One terminal → GND
- Other terminal → GPIO17 (ESP32)

GPIO17 is configured as an input with an internal pull-up resistor. Therefore, the pin reads **HIGH** when the button is released and **LOW** when pressed. This signal is used by the firmware to enable or disable streaming: when the button is pressed, IMU samples are packaged and transmitted; when released, the device remains idle and does not publish motion data.



Hardware connections

D. *Cloud Backend Implementation*

The cloud backend is implemented using managed AWS services to ingest the device stream, trigger processing, and deliver real-time updates to web clients. The ESP32 publishes motion samples to AWS IoT Core using MQTT over TLS, providing secure device authentication and encrypted communication.

Message routing is handled using AWS IoT Rules, which subscribe to the relevant MQTT topic(s) and trigger a serverless processing stage. Each incoming message activates an AWS Lambda function that serves as a mediation layer between the IoT ingestion path and the web delivery path. The function parses the IMU payload, prepares a web update, and initiates a broadcast operation.

Real-time delivery to browsers is performed through Amazon API Gateway (WebSocket API)**.** To support multiple simultaneous viewers, active WebSocket connection identifiers are stored in Amazon DynamoDB (e.g., a connectionIds table). A TTL policy is used to automatically expire stale connections, reducing manual cleanup. For each processed motion update, the Lambda function retrieves the current set of connection IDs and pushes the update through the WebSocket API to all active clients.

E. *Web Client Implementation*

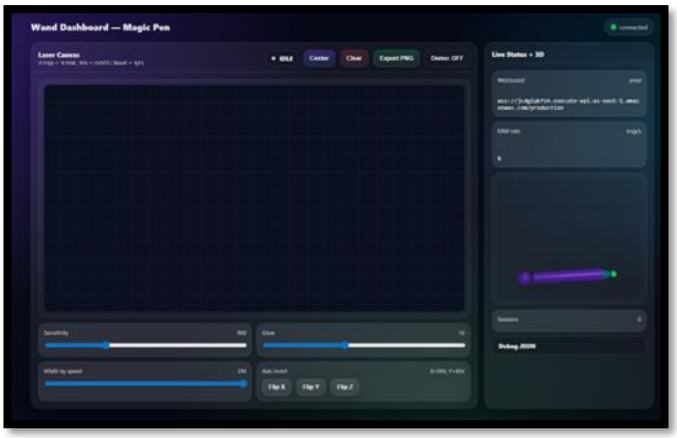
The web client is a browser-based dashboard that provides real-time visualization of the wand motion. It establishes a WebSocket connection to the cloud backend and updates the user interface upon receiving streaming motion messages. The interface is designed to support immediate interaction while keeping the client lightweight and accessible without special installations.

2D Drawing Canvas - The main component is a 2D canvas that renders the drawing trajectory as a continuous path. Each incoming motion update is mapped to a 2D point and appended to the current polyline, producing smooth real-time drawing behavior. The dashboard includes basic controls that assist interaction and debugging, such as clearing the canvas, re-centering the view, and exporting the drawing as an image.

3D Visualization - In addition to the 2D canvas, the dashboard provides a live 3D visualization that reflects the wand's orientation and motion. This view serves as an intuitive feedback mechanism for verifying the sensed movement and for debugging axis alignment and inversion during development.

Session Browsing - The interface also supports viewing previous sessions, enabling users to revisit earlier trajectories and compare runs. Session metadata is presented in the dashboard, and selecting a session loads the corresponding stored trajectory for display.

UI Status and Monitoring - A status panel displays the current connection state and basic runtime information, which helps validate end-to-end streaming and provides quick diagnostics during testing.



The web dashboard

F. *System Design Considerations*

This subsection highlights the key design choices that guided the system architecture.

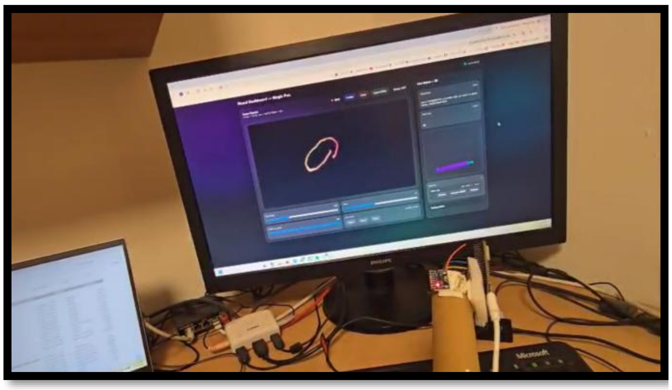
- **Secure, lightweight device link (MQTT over TLS) -** MQTT was chosen to support frequent small sensor messages efficiently, while TLS provides encrypted and authenticated communication from the ESP32 to the cloud.
- **Low-latency web delivery (WebSocket) -** WebSocket enables real-time push updates to browsers with minimal client-side complexity, avoiding MQTT dependencies and IoT credential handling on the client.
- **Reliable broadcast with automatic connection cleanup (DynamoDB + TTL) -** Active WebSocket connections are tracked in a table, and TTL expiration removes stale entries to keep broadcast targets up to date.
- **User-controlled transmission (button-gated streaming) -** The device publishes motion messages only while the button is pressed, reducing unnecessary traffic and providing an intuitive start/stop mechanism.

V. RESULTS

We evaluated the system end-to-end, focusing on real-time responsiveness, button-gated transmission, stream stability, and concurrent web clients. The evaluation was performed by streaming IMU updates from the ESP32-based wand to the browser dashboard and observing the resulting 2D trajectory rendering and live visualization behavior.

- **Button-gated transmission:** No motion messages were transmitted when the button was not pressed, while streaming started immediately on press and stopped on release.
- **Real-time responsiveness:** Motion updates were delivered and rendered in the browser without noticeable delay under normal operating conditions, enabling continuous drawing interaction.
- **Stream rate and stability:** During active streaming, the client-side update rate was consistent with the configured sampling rate (≈50 Hz), producing smooth and continuous drawing paths.
- **Multi-client delivery:** Multiple browser clients were able to connect simultaneously and receive the same live updates concurrently via the WebSocket stream.
- **User interface verification:** The dashboard successfully displayed the 2D drawing canvas and the live 3D view as real-time feedback, supporting verification of axis alignment and motion behavior during testing.

Overall, the results indicate that the proposed IoT pipeline supports responsive motion-to-trajectory visualization with user-controlled streaming and concurrent viewing in a standard web browser.



VI. DISCUSSION AND LIMITATIONS

The system was designed with practical constraints in mind, particularly cloud usage and cost. For this reason, the motion stream was limited to approximately 50 Hz, rather than transmitting at higher sampling rates. While this rate was sufficient to produce smooth drawing paths in our tests, it may reduce the system's ability to capture very fast or fine-grained motions compared to higher-frequency streaming.

In addition, transmission was intentionally gated by a physical button, meaning that motion data was published only while the button was pressed. This choice reduced unnecessary traffic and operational cost, but it also introduces a functional limitation: the system does not track or record motion during periods when the button is not pressed. As a result, continuous motion capture without explicit user activation is not supported in the current implementation.

VII. CONCLUSION AND FUTURE WORK

This paper presented an IoT-based virtual drawing system that converts free-hand motion into a real-time 2D trajectory. Using an ESP32-based wand with an MPU6050 IMU, motion data was streamed at approximately 50 Hz over MQTT to Amazon cloud services and visualized in a browser-based dashboard. The results demonstrate responsive real-time behavior, smooth trajectory generation, and concurrent viewing by multiple web clients.
Beyond virtual drawing, this project highlights the broader potential of IMU-based motion sensing as a general-purpose tool for live documentation and analysis of physical activity. Because the same type of motion sensor can capture orientation and movement continuously, similar IoT pipelines can be applied to a wide range of real-time use cases, such as exercise and strength-training monitoring, gesture-based interfaces, sports technique feedback, rehabilitation tracking, and remote observation of motion-driven tasks.
Future work may focus on improving motion fidelity and usability while maintaining reasonable cloud cost. Possible extensions include adaptive sampling (increasing the rate only during rapid motion), improved calibration and coordinate mapping for more accurate trajectories, enhanced smoothing and filtering to reduce noise, continuous tracking without requiring a pressed button, and richer session management and visualization features (e.g., analytics over saved sessions and comparison between runs).

VIII. SUPPORTING MATERIALS

project materials are available in the public GitHub repository - Main repository: https://github.com/LironAdi/IoT-Virtual-Drawing
The repository includes:

- **ESP32 firmware (MPU6050 IMU → MQTT/TLS):**
  ESPcode2.ino - https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/ESPcode2.ino
  Short description - The ESP32 firmware reads IMU measurements from an MPU6050 sensor, monitors a push-button to gate transmission, and publishes motion samples/events over MQTT (TLS) to the cloud for real-time visualization in the web client.

- **Web dashboard client:**
  index.html - https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/index.html
  Short description - The web client implements a browser dashboard that connects to the system via WebSocket, renders the motion stream as a real-time 2D drawing canvas, displays a live 3D visualization of the wand orientation, and supports basic session management.

- **AWS Lambda broadcaster:**
  lambda_function.py - https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/lambda_function.py
  Short description - Python Lambda function that receives IoT Rule events, packages them into a JSON payload, retrieves active WebSocket connections from DynamoDB, and broadcasts updates to connected browser clients via API Gateway, while cleaning stale connections automatically.

- **Project report & presentation:** included under the repository files (see repo root)

- **Live demo (web dashboard):**
  https://wand-dashboard-live.s3.us-east-1.amazonaws.com/index.html

IX. REFERENCES

[1] A. B. Asghar, M. Majeed, A. Taseer, M. B. Khan, K. Naveed, M. H. Jaffery, A. S. M. Metwally, K. Eismont, and M. Nejman, “Classification and monitoring of arm exercises using machine learning and wrist-worn band,” *Egyptian Informatics Journal*, vol. 27, 2024.
Copy stored in repo:
https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/Classification%20and%20monitoring%20of%20arm%20exercises%20using%20machine%20learning%20and%20wrist-worn%20band.pdf

[2] “IoT Virtual Drawing — GitHub repository,” https://github.com/LironAdi/IoT-Virtual-Drawing

[3] “ESP32 wand firmware (ESPcode2.ino),” https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/ESPcode2.ino

[4] “Web dashboard client (index.html),” https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/index.html

[5] “AWS Lambda broadcaster (lambda_function.py),” https://github.com/LironAdi/IoT-Virtual-Drawing/blob/main/lambda_function.py

[6] “Wand Dashboard (Live Demo),” Amazon S3 static website - https://wand-dashboard-live.s3.us-east-1.amazonaws.com/index.html