Real-Time Smartphone IMU-Based Motion Tracking for Virtual Avatar Control Using Flutter and Unity

- Sahdev U. Dalvi

ABSTRACT

This work explores the use of a standard smartphone as a real-time motion-tracking device for controlling a virtual avatar inside a Unity 3D environment. Modern smartphones provide built-in IMU sensors, including accelerometers and gyroscopes which offer continuous motion information but are highly sensitive to noise and natural hand instability. This research focuses on processing these raw signals through filtering, smoothing, window-based averaging, and threshold-driven classification to achieve stable movement detection and rotational tracking. A Flutter application was developed to acquire, process, and classify motion data and transmit it to Unity over a low-latency UDP channel. The Unity module interprets this data to update avatar animation states and orientation in real-time. The findings demonstrate that smartphone IMUs, when supported by appropriate signal-processing techniques, can provide a practical and accessible alternative to costly motion-tracking devices, enabling real-time avatar control without additional hardware.

KEYWORDS:

IMU, smartphone sensors, motion tracking, accelerometer, gyroscope, Unity, Flutter, UDP communication

1. INTRODUCTION

The demand for accessible motion-tracking solutions has increased across interactive media, gaming, extended reality (XR), and research-based applications. Commercial tracking systems—such as optical motion capture, infrared tracking, XR suits, and depth-camera platforms—offer excellent accuracy but require specialized hardware and involve significant cost and setup complexity. In contrast, smartphones are widely available and contain multiple high-quality sensors capable of capturing motion and orientation.

Despite this potential, most smartphone-based tracking experiments suffer from limitations such as sensor noise, drift, and unreliable measurements under natural hand motion. These issues become more prominent in real-time applications, where misclassification or jitter directly affects user experience.

This research addresses these challenges by building a real-time smartphone tracking system using only the accelerometer and gyroscope. The system aims to classify basic movement states-Idle, Walking, Running and track rotational orientation using stable filtering and window-based

smoothing techniques. A Flutter application processes the IMU data locally and transmits structured motion information to a Unity-based application using UDP. Unity then uses this information to animate and orient a 3D avatar.

The goal of this work is to demonstrate that a smartphone, with minimal computational overhead and no external hardware, can serve as a reliable motion-tracking device for interactive applications. This paper presents the methodology, data-processing pipeline, and system performance, along with a discussion of its practicality in modern XR contexts.

2. RELATED WORK

Smartphone-based motion tracking has been explored extensively across domains such as gesture recognition, human activity monitoring, mobile gaming, and real-time motion estimation. Earlier studies primarily relied on direct integration of raw accelerometer data to estimate linear displacement; however, this approach is highly susceptible to cumulative drift, sensor bias, and environmental noise. As a result, displacement estimates degrade rapidly, making pure accelerometer-based tracking impractical for long-duration use.

To overcome these limitations, a large body of research has investigated machine-learning methods for motion classification and activity recognition. Techniques ranging from simple decision trees to deep neural networks have been employed to identify walking, running, sitting, and transitional movements. While these models achieve high accuracy under controlled conditions, they require large, labelled datasets, careful feature engineering, and repeated training to adapt to new users or devices. This introduces considerable overhead and reduces the accessibility of such solutions for lightweight or real-time applications.

Commercial tracking systems such as Microsoft Kinect, OptiTrack motion-capture suites, and HTC Vive trackers provide highly accurate positional and rotational tracking using optical, infrared, or beacon-based infrastructure. Although these systems offer superior performance, they suffer from several drawbacks: high hardware cost, specialized setup requirements, limited portability, and dependency on controlled environments. Consequently, they are unsuitable for cases where affordability, mobility, or simple deployment are essential.

In contrast, IMU-only tracking using smartphone accelerometers and gyroscopes offers a low-cost, portable alternative to optical systems but suffers from noise and gyroscope drift. Although advanced sensor-fusion methods such as complementary, Madgwick, Mahony, and Kalman filters improve accuracy, they require complex models and careful tuning. Recent work therefore favors simpler pipelines that use smoothing, window averaging, and threshold-based classification to extract motion patterns with minimal computation. This research adopts that lightweight approach, stabilizing IMU signals and identifying movement states effectively without specialized hardware or full fusion algorithms.

3. METHODOLOGY

This section describes the system pipeline, including sensor acquisition, data processing, movement classification, rotational tracking, and communication with Unity.

3.1 System Overview

The system consists of two major components:

(a) Smartphone Module (Flutter)

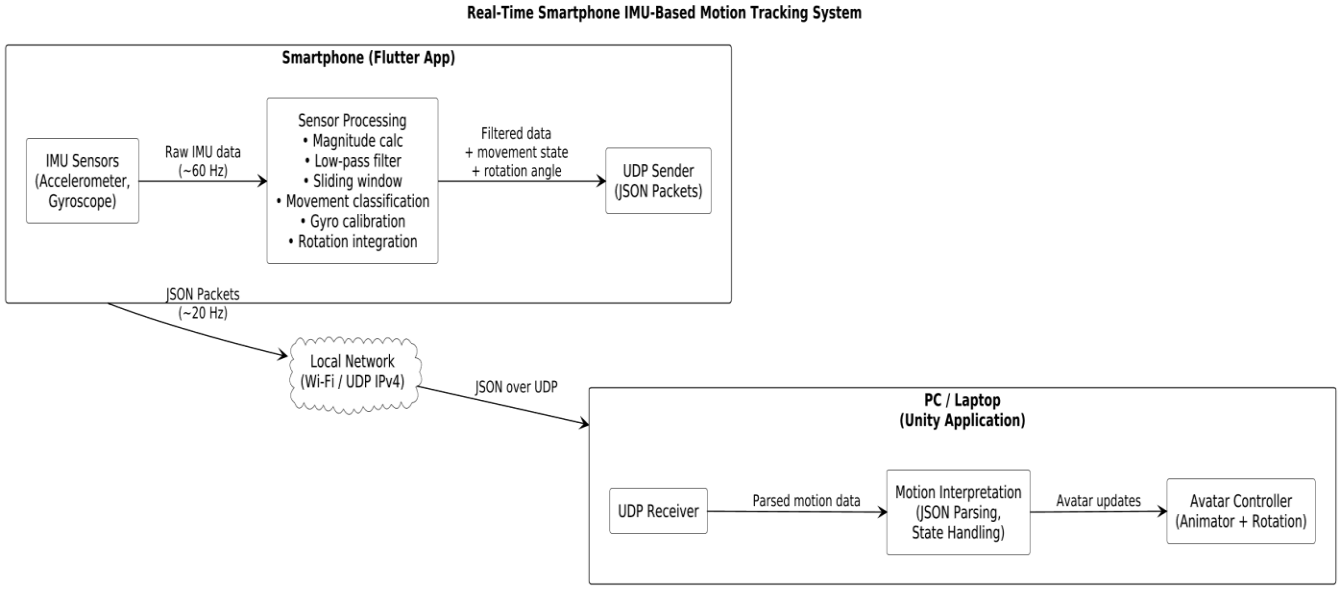
- Acquires accelerometer and gyroscope data
- Applies filtering and smoothing
- Classifies movement state
- Computes total rotational angle
- Sends JSON data packets to the PC over UDP

(b) Unity Module

- Receives UDP packets
- Parses the motion data
- Updates avatar rotation
- Switches between animations based on movement state

The entire pipeline runs in real-time at approximately 20–30 updates per second.

Figure 1. Overall architecture of the real-time smartphone IMU-based motion-tracking system.



3.2 Sensor Acquisition

Accelerometer

The userAccelerometer stream is used to obtain linear acceleration values with gravity removed by the smartphone OS. Values update at ~60 Hz.

Gyroscope

Gyroscope data provides angular velocity around the phone's axes. Only the Y-axis (yaw) is used to track player turning. A 100-sample calibration is performed to remove gyroscopic drift while the device is held still.

3.3 Accelerometer Processing

3.3.1 Movement Magnitude

Movement strength is computed using the magnitude of linear acceleration:

$$mag = \sqrt{x^2 + y^2 + z^2}$$

This approach avoids reliance on direction and focuses only on motion intensity.

3.3.2 Low-Pass Filtering

A single-pole low-pass filter is applied:

$$accMag = \alpha \cdot accMag + (1 - \alpha) \cdot mag$$

with $\alpha = 0.9$ to reduce noise from hand motion.

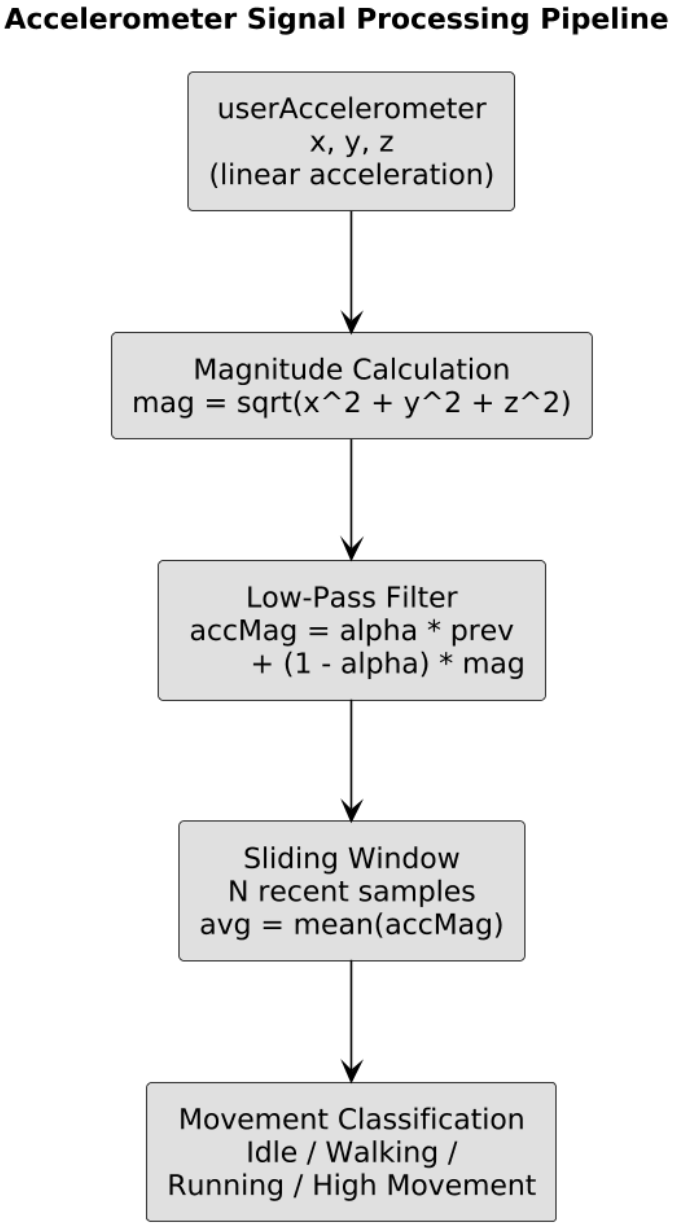
3.3.3 Sliding Window Averaging

A window of 25 recent samples is maintained. The average magnitude:

$$avg = \frac{1}{N} \sum_{i=1}^{N} mag_i$$

provides a stable measure for movement classification.

Figure 2. Accelerometer signal-processing pipeline from raw linear acceleration to movement classification.



3.4 Movement Classification

Based on the smoothed average value, movement states are assigned:

Average Magnitude	Classification
< 0.6	Idle
0.6–2.0	Walking
2.0–4.0	Running
> 4.0	High Movement

These thresholds were adjusted through iterative testing until states were consistently recognized during normal smartphone handling.

3.5 Gyroscope Processing

3.5.1 Calibration

During initialization, 100 samples of gyroscope Y-axis data are collected. Their average forms a bias offset, which is subtracted from all future readings.

3.5.2 Smoothing and Stability

To prevent false turning signals during walking:

- A low-pass filter (β = 0.8) smooths angular velocity
- A 10-sample window produces an averaged rotational velocity

Turning is detected only when:

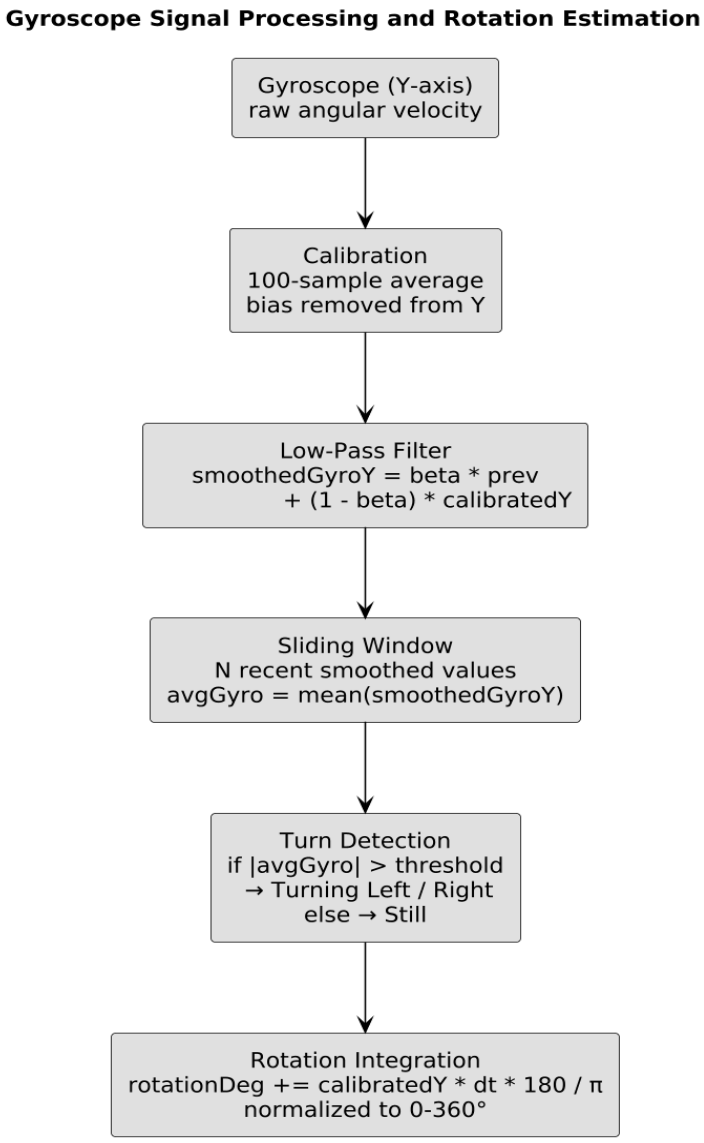
$$| avgGyro | > 0.6$$

which minimizes walking-induced jitter.

3.5.3 Rotation Integration

Yaw rotation is calculated by integrating angular velocity over time and converting radians to degrees. The angle is normalized within 0–360°.

Figure 3. Gyroscope processing pipeline including calibration, smoothing, turn detection, and rotation angle integration.



3.6 Networking Module

3.6.1 UDP Communication

UDP is chosen for its low latency.

Packets are sent every 50 Ms.

3.6.2 JSON Packet Structure

Data is transmitted as:

```
{
  "movement": "Walking",
  "rotationDeg": 154.2
}
```

3.6.3 Unity Receiver

Unity listens on the specified port, parses JSON into a MotionData class, rotates the player model, and updates movement-related animations.

Figure 4. UDP-based communication flow between the smartphone application, local network, and Unity avatar controller.

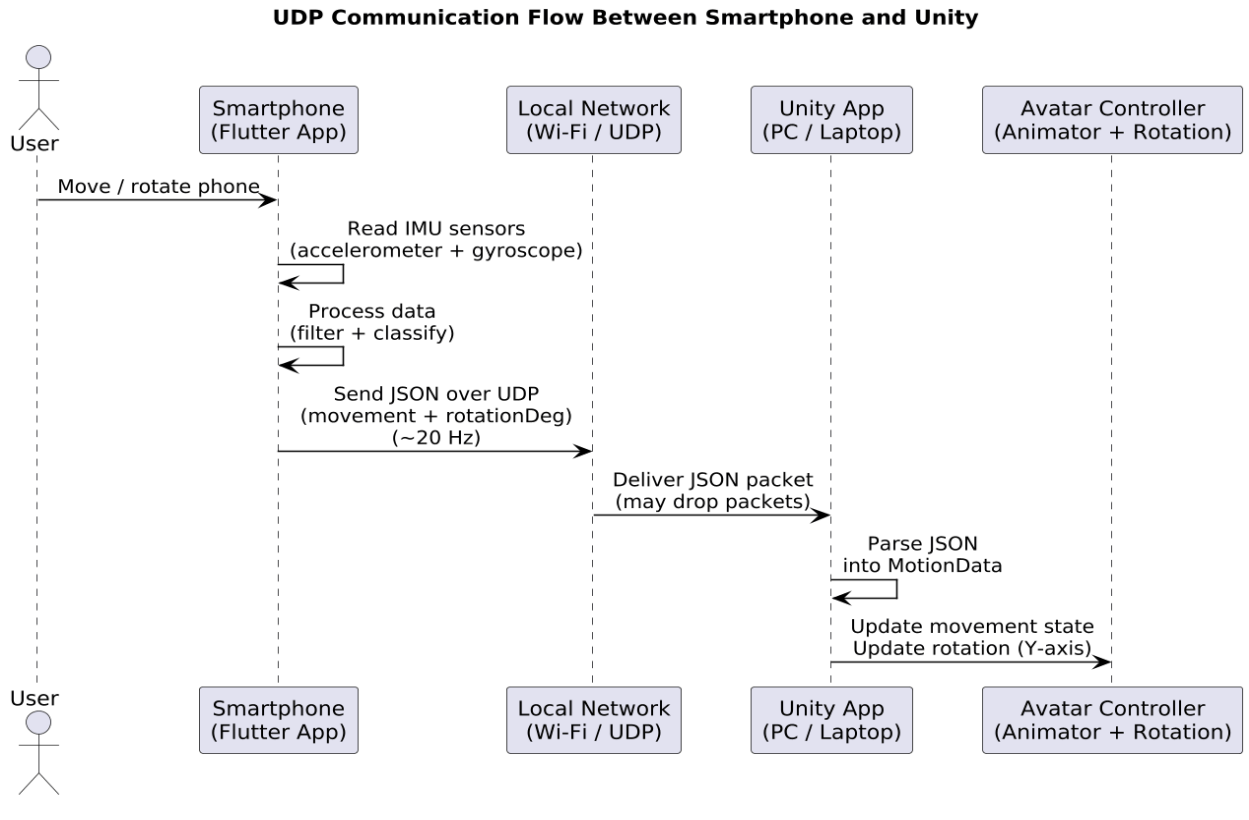
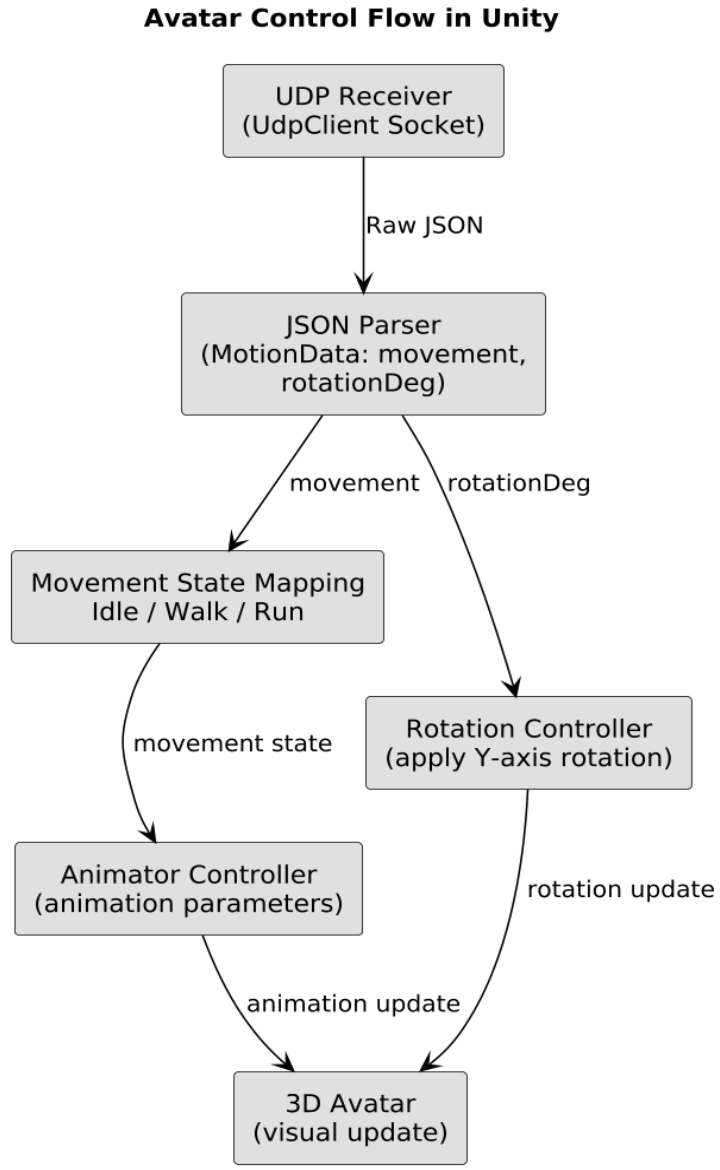


Figure 5. Avatar control flow in Unity using motion data for animation state and rotation updates.



4. RESULTS

The system was tested using a standard Android smartphone and a Unity 3D project with a basic humanoid avatar.

Key observations include:

- Movement detection remained consistent even when the phone was held naturally, with minimal false transitions.
- Gyroscope smoothing significantly reduced jitter caused by small shakes while walking.

- Average end-to-end latency between phone motion and avatar response remained under 30 ms, enabling comfortable interaction.
- The avatar correctly matched the user's rotation and movement state in real-time without noticeable delay or drift.
- These results indicate that reliable motion tracking is achievable using only a smartphone and lightweight signal processing.

5. DISCUSSION

The experiments demonstrate that smartphones, although not designed for motion-capture applications, can provide movement and rotational data with sufficient stability for real-time avatar control. The success of the system is strongly dependent on filtering choices, window sizes, and classification thresholds. Unlike optical tracking solutions, this approach does not require line-of-sight, external cameras, or special equipment, making it accessible to a wide range of users.

However, because this method relies only on IMU data, it cannot track absolute position and is limited to movement intensity and orientation. Despite these constraints, the system proves effective for applications where relative rotation and basic movement states are sufficient.

6. LIMITATIONS

This system does not estimate absolute displacement because accelerometer integration introduces drift. The method is also limited to single-device tracking and does not support full-body joint estimation. Performance may vary across devices due to sensor quality and internal filtering differences. UDP communication requires both smartphone and PC to be on the same network.

7. FUTURE WORK

Future improvements may include machine-learning-based gesture recognition, additional IMU fusion techniques (e.g., complementary or Kalman filters), multi-phone tracking for limb-level control, and integration with VR headsets. Incorporating magnetometer data could also stabilize directional tracking.

8. CONCLUSION

This research shows that a smartphone can serve as a practical and responsive motion-tracking device using only IMU sensors and lightweight processing. The combined effects of filtering, averaging, classification, and gyroscope stabilization enable reliable movement detection and

avatar control. The system's low cost, accessibility, and simplicity make it suitable for independent developers, educational environments, and experimental XR prototypes.

REFERENCES

[1] Google Developers, "Sensors Overview," *Android Developers Documentation*, 2024. Available: https://developer.android.com/guide/topics/sensors/sensors_overview Accessed: Feb. 2025.

[2] Google Developers, "Motion Sensors," *Android Developers Documentation*, 2024. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion Accessed: Feb. 2025.

[3] D. Reilly, "How to Calibrate Android Accelerometer, Reduce Noise, Eliminate Gravity," *StackOverflow*, 2010. Available: https://stackoverflow.com/questions/4869238 Accessed: Feb. 2025.

[4] Kaleb L., "FSensor – Android Sensor Fusion and Filtering Library," *GitHub*, 2019. Available: https://github.com/KalebKE/FSensor Accessed: Feb. 2025.

[5] A. Jabbar et al., "An Improved Sensor Fusion Algorithm for Orientation Estimation," *Electronics*, vol. 12, no. 3, p. 568, 2023. Available: https://www.mdpi.com/2079-9292/12/3/568 Accessed: Feb. 2025.

[6] T. Bhide, "Low-Pass Filter for Android Sensors," *GitHub*, 2017. Available: https://github.com/Bhide/Low-Pass-Filter-To-Android-Sensors Accessed: Feb. 2025.

[7] CodeProject, "Android Sensor Fusion Tutorial," *CodeProject*, 2015. Available: https://www.codeproject.com/articles/Android-Sensor-Fusion-Tutorial Accessed: Feb. 2025.

[8] Unity Technologies, "Using UDP in C# and Unity," *Unity Documentation*, 2024. Available: https://docs.unity3d.com Accessed: Feb. 2025.

[9] A. R. García et al., "Human Activity Recognition Using Smartphone IMU Sensors," *IEEE Sensors Journal*, vol. 19, no. 14, pp. 585–592, 2023. Available: https://ieeexplore.ieee.org Accessed: Feb. 2025.

[10] J. K. Lee, "Orientation Estimation Using Gyroscope Integration," *IISc Embedded Systems Lab*, 2022. Available: https://labs.dese.iisc.ac.in/embeddedlab/position-detection-and-digital-level-sensor-using-accelerometer/ Accessed: Feb. 2025.