

Article

A Low-Latency Embedded Inertial Motion Tracking System for Real-Time Applications

Elmin Marevac ¹, Esad Kadušić ², Nataša Živić ^{3,*} and Christoph Ruland ⁴

¹ Polytechnic Faculty, University of Zenica, 72000 Zenica, Bosnia and Herzegovina; elmin.marevac@unze.ba

² Faculty of Educational Sciences, University of Sarajevo, 71000 Sarajevo, Bosnia and Herzegovina; ekadusic@pf.unsa.ba

³ Faculty of Digital Transformation, Leipzig University of Applied Sciences, 04277 Leipzig, Germany

⁴ Chair of Digital Communication Systems, University of Siegen, 57076 Siegen, Germany; christoph.ruland@uni-siegen.de

* Correspondence: natasa.zivic@htwk-leipzig.de

Abstract

Inertial motion sensing plays an important role in real-time human–machine interaction applications, including interactive systems, virtual environments, and motion-controlled interfaces, where low latency and accurate motion tracking are important. This paper presents the design and implementation of an embedded inertial sensing system that acquires, processes, and transmits motion data from consumer-grade inertial measurement units (IMUs) under real-time constraints. Rather than introducing a new sensor fusion algorithm, the contribution lies in a systems-oriented methodology comprising a predictive clock-advancement mechanism that prevents cumulative timing drift, an automated matrix-based calibration procedure for hardware-agnostic deployment, and a benchmarking framework for end-to-end real-time system evaluation. Implemented on a resource-constrained embedded platform, the framework integrates sensor acquisition, lightweight filtering, sensor fusion, and real-time orientation estimation within a single processing pipeline. Motion data are transmitted using the CemuHook UDP (User Datagram Protocol) motion protocol (DSU) to demonstrate interoperability with existing motion-control software while maintaining low end-to-end latency and stable throughput. Experimental results show stable sampling frequency, low communication latency, accurate orientation estimation, and low computational overhead. The presented system provides an embedded inertial sensing framework that can be adapted to a range of real-time motion-sensing applications beyond the communication protocol used for demonstration.

Keywords: embedded inertial sensing; inertial measurement unit; motion tracking; sensor fusion; real-time systems; low-latency communication; CemuHook UDP protocol

Academic Editor: Yannis Papaefstathiou

Received: 10 June 2026

Revised: 1 July 2026

Accepted: 8 July 2026

Published: 10 July 2026

Copyright: © 2026 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The evolution of interactive computing over the past two decades has been characterized by a continuous reduction in perceptual latency and an expansion of spatial input modalities. While high-end immersive virtual reality/augmented reality (VR/AR) systems demand ultra-low motion-to-photon latencies (e.g., <20 ms) to maintain spatial presence, many practical human–machine interaction (HMI) scenarios—such as motion-

controlled gaming, educational prototyping, teleoperation, and biomechanical monitoring—operate effectively within the 100–150 ms latency range without perceptible degradation [1]. Similarly, competitive motion-controlled gaming and precision teleoperation require update rates exceeding 100 Hz with deterministic timing behavior to maintain responsiveness and control stability [2]. Achieving these performance requirements in real-world deployments requires a sensing pipeline that minimizes delay across processing stages, including sensor readout, offset compensation, noise suppression, coordinate transformation, sensor fusion, network serialization, and physical transmission.

Traditional motion-tracking infrastructures have predominantly relied on optical or camera-based systems, including infrared constellations, depth-sensing arrays, and multi-camera triangulation rigs. While these systems can achieve sub-millimeter spatial precision, they are fundamentally constrained by line-of-sight requirements, environmental lighting sensitivity, high infrastructure costs, and substantial computational overhead for real-time pose estimation [3]. Moreover, optical setups are inherently non-portable and unsuitable for occlusion-prone or distributed deployment scenarios, limiting their applicability in mobile, wearable, or consumer-grade human–machine interaction (HMI) contexts.

The proliferation of interactive computing paradigms has therefore accelerated interest in portable, infrastructure-independent sensing architectures capable of bridging the physical and digital domains. Applications spanning VR/AR, motion-controlled gaming, teleoperation, biomechanical analysis, and wearable robotics increasingly depend on real-time motion tracking as a core interaction modality. In such systems, the fidelity of the user experience is directly constrained by the end-to-end latency, temporal stability, and spatial accuracy of the underlying sensing architecture. Although high-end motion capture infrastructures continue to advance, their cost, deployment complexity, and environmental dependence motivate the exploration of embedded inertial sensing solutions that can deliver comparable responsiveness on lightweight, low-power hardware.

Inertial measurement units (IMUs) provide a self-contained and infrastructure-independent approach to motion sensing. Modern micro-electromechanical systems (MEMS) IMUs integrate tri-axial accelerometers and gyroscopes into compact, low-power packages that operate without external tracking systems or controlled environments. Their properties, including immunity to occlusion, low power consumption, low cost, and ease of integration into portable devices, have contributed to their widespread use in consumer electronics, robotics, and biomedical monitoring [4]. However, converting raw inertial measurements into stable, low-latency orientation estimates on embedded platforms introduces several challenges. Consumer-grade IMUs exhibit bias instability, scale-factor error, temperature-dependent drift, and stochastic noise, which can accumulate during numerical integration [5]. Without calibration and appropriate filtering, these effects may lead to orientation drift, high-frequency jitter, and reduced tracking fidelity, particularly during prolonged or dynamic operation.

Furthermore, deploying robust inertial tracking on resource-constrained microcontrollers remains a non-trivial systems engineering problem. Typical embedded platforms operating at 80–160 MHz provide limited RAM, constrained floating-point performance, and strict real-time deadlines that restrict the complexity of sensor-fusion pipelines. Heavyweight complementary filters, extended Kalman filters (EKF), or machine-learning-based estimators often exceed the available computational budget, forcing developers to trade off latency, accuracy, and temporal determinism. As a result, many existing embedded motion-tracking systems prioritize orientation estimation

quality while overlooking equally critical real-time characteristics such as scheduling stability, packet timing consistency, and communication overhead.

A limitation in embedded inertial tracking lies in the interaction between deterministic scheduling, network protocol overhead, and transmission behavior. Many open-source and commercial implementations rely on event-driven or best-effort communication models that can introduce jitter, packet bunching, and variable end-to-end latency due to interrupt contention, operating-system scheduling granularity, or network stack variability. Real-time HMI applications typically require periodic sampling and transmission with limited phase deviation. Communication protocol design also affects both interoperability and timing behavior. Proprietary or heavily serialized packet formats can introduce parsing and buffering overhead, whereas lightweight UDP-based (User Datagram Protocol) approaches, such as the CemuHook DSU (Device Server UDP) standard, provide interoperability with existing motion-control systems but require packet construction, cyclic redundancy check (CRC) validation, and clock synchronization to maintain data integrity. The computational costs associated with checksum generation, telemetry reporting, and buffer management therefore need to be quantified to avoid interference with the primary acquisition and fusion pipeline.

These technical constraints motivate the development of a unified embedded inertial framework with emphasis on low latency, protocol interoperability, and reproducible evaluation. Existing literature often evaluates IMU systems in isolation, focusing on orientation accuracy or static noise characteristics while paying less attention to end-to-end latency distributions, jitter behavior, clock drift, and protocol-level efficiency. This limited scope of evaluation can obscure trade-offs in resource-constrained real-time sensing architectures and reduces comparability between implementations.

This paper addresses these limitations by presenting an embedded inertial sensing framework that integrates sensor calibration, lightweight filtering and fusion, deterministic scheduling, and network transmission within a single acquisition-to-transmission pipeline. The architecture is designed for time-predictable execution on low-cost embedded hardware and is evaluated across multiple operational conditions. Performance is assessed using statistical metrics including latency distributions, jitter characteristics, clock synchronization stability, protocol overhead, transmission goodput efficiency, and behavior under varying computational and network conditions. By structuring the sensing pipeline from acquisition and calibration to serialization and transport, the system achieves sub-150 ms end-to-end latency with temporal stability comparable to existing motion-control implementations while operating within a constrained computational and memory budget on 80–160 MHz embedded platforms.

The proposed system implements a deterministic acquisition architecture that advances a periodic clock without reset-based resynchronization, aiming to reduce cumulative timing drift and packet bunching. An automated calibration routine resolves axis permutation, sign inversion, and static bias through a guided proportional–integral–derivative-based (PID) gravity alignment procedure, enabling hardware-agnostic operation without manual configuration. Network transmission uses the DSU UDP protocol with optional multicast telemetry support, allowing integration with existing interactive software ecosystems while maintaining low protocol overhead. Complementing the embedded implementation, an analytics framework extracts and visualizes performance metrics, including bootstrap confidence intervals, latency distributions, jitter spectral density, clock-skew limits, latency–RSSI (received signal strength indicator) relationships, and protocol-efficiency breakdowns, providing quantitative insight into system behavior.

The core novelty of this work does not lie in the invention of new sensor fusion mathematics or communication protocols; the system leverages established

complementary filtering and the existing DSU standard. While individual components such as complementary filtering, static memory allocation, UDP transmission, and CRC validation are well-known engineering practices, their naive integration on resource-constrained microcontrollers typically results in cumulative phase drift, unquantified protocol overhead, and timing violations. The scientific contribution of this work therefore lies in a systems-oriented and mathematical formulation of their integration, enabling strict real-time determinism under bare-metal constraints.

Specifically, we introduce a deterministic integration architecture based on predictive scheduling, hardware-agnostic calibration, and system-level evaluation. Rather than relying on conventional wall-clock periodic scheduling, which can lead to packet bunching under computational load, we use a predictive clock-advancement mechanism that decouples sampling instants from processing completion times, reducing cumulative timing drift and jitter. In addition, IMU axis discovery and alignment are formulated as an automated permutation-matrix identification problem, replacing manual calibration procedures with a reproducible, hardware-independent method that removes the need for external fixtures. Finally, a multi-metric benchmarking framework is introduced that extends evaluation beyond isolated accuracy metrics to system-level characterization, including bootstrap confidence intervals, regime transition analysis, latency–RSSI Pareto frontiers, protocol overhead quantification, and goodput efficiency analysis.

Together, these systems-level components address challenges related to timing determinism, hardware-independent deployment, and performance evaluation in embedded IMU systems. The primary methodological contributions of this work are as follows:

- **Deterministic Scheduling:** The formulation of a predictive clock-advancement mechanism that decouples sampling instants from processing completion times, mathematically preventing cumulative timing drift, minimizing jitter, and eliminating packet bunching under bare-metal real-time constraints.
- **Automated Calibration:** The development of an adaptive PID-based alignment procedure coupled with a permutation-matrix identification algorithm that automatically resolves sensor axis mapping, sign inversion, and sensor bias, enabling hardware-independent deployment and reproducible calibration without external fixtures.
- **Statistical Benchmarking:** The development of a benchmarking and analytics framework that supports evaluation using bootstrap confidence intervals, regime transition analysis, latency–RSSI Pareto frontier analysis, protocol overhead characterization, goodput efficiency measurement, and multi-metric performance assessment.

By bridging the gap between theoretical inertial tracking models and deployable embedded implementations, this work establishes a reproducible, low-latency sensing foundation adaptable to a broad spectrum of HMI applications, including motion-controlled gaming, educational prototyping, low-cost teleoperation, and biomechanical sensing. The remainder of this paper is organized as follows: Section 2 details the hardware architecture, acquisition pipeline, calibration methodology, and deterministic scheduling design; Section 3 describes the communication protocol implementation and telemetry infrastructure; Section 4 presents the experimental benchmarking methodology and statistical evaluation framework; Section 5 discusses results across multiple operational regimes, emphasizing latency, stability, and computational trade-offs; and Section 6 concludes with future directions and potential extensions to broader real-time interactive systems.

2. Related Work

Real-time inertial motion tracking has experienced rapid development over the past two decades, driven by advances in consumer electronics, robotics, VR, and human–computer interaction (HCI). Improvements in MEMS inertial sensors, embedded processing, and wireless communication have enabled increasingly accurate and responsive motion tracking systems. Despite these advances, significant challenges remain in achieving infrastructure-independent operation, deterministic embedded execution, transparent performance characterization, and open interoperability within resource-constrained platforms. Existing work can be broadly categorized into commercial inertial tracking systems, embedded sensor fusion and calibration techniques, and communication protocols for real-time motion streaming.

Commercial inertial tracking systems have demonstrated the practical feasibility of motion-based interaction. The Nintendo Wii Remote (Nintendo Co., Ltd., Kyoto, Japan) established consumer-grade inertial sensing as a mainstream HCI technology by utilizing low-cost accelerometers for gesture recognition [6]. Subsequent platforms, including Sony's PlayStation Move (Sony Interactive Entertainment LLC, San Mateo, CA, USA), HTC Vive (HTC Corporation, New Taipei City, Taiwan), Oculus Touch (Meta Platforms, Inc., Menlo Park, CA, USA), and SteamVR Tracking (Valve Corporation, Bellevue, WA, USA), progressively integrated gyroscopes, optical tracking, and standardized application programming interfaces to improve tracking accuracy, responsiveness, and cross-device compatibility [7,8]. Although these systems achieve excellent performance, they typically depend on hybrid optical–inertial architectures that compensate for the cumulative drift inherent in MEMS gyroscopes through external cameras, base stations, or structured environments [9]. Furthermore, computationally intensive sensor fusion and prediction algorithms are frequently executed on host computers or dedicated gaming hardware, introducing operating-system scheduling delays, buffering effects, and variable end-to-end latency that are difficult to characterize experimentally [10]. Proprietary firmware, closed communication protocols, and relatively high hardware costs further limit experimental customization, reproducibility, and deployment in low-cost or distributed embedded applications [11].

Academic research has primarily focused on improving embedded sensor fusion algorithms, calibration procedures, and computational efficiency. Complementary filters remain widely adopted because they efficiently combine high-frequency gyroscope measurements with low-frequency accelerometer gravity estimates while requiring minimal computational resources [12]. Quaternion-based gradient-descent algorithms, including the Madgwick and Mahony filters, further improve convergence speed and long-term stability without the computational overhead associated with covariance-based estimation [13]. More complex estimators, including EKF, Unscented Kalman Filters (UKFs), and particle filters, generally provide higher estimation accuracy under dynamic motion and non-Gaussian noise conditions but require greater computational and memory resources, which can limit their suitability for low-cost embedded microcontrollers without hardware floating-point support [14].

Sensor calibration has evolved from static gravity-based alignment toward adaptive and automated approaches. Conventional calibration procedures often require external reference equipment, precision fixtures, or manual parameter adjustment, which can limit scalability and introduce human-related variability [15]. Recent studies have investigated iterative optimization, adaptive estimation, machine-learning-based bias prediction, and self-aligning PID controllers that resolve sensor orientation, axis permutation, sign inversion, and static bias through guided user interaction [16]. While these techniques improve calibration consistency, relatively few implementations integrate automated calibration into deterministic real-time embedded pipelines while also providing telemetry and validation using statistical methods.

Research has also explored embedded implementations using ARM Cortex-M microcontrollers, FPGA accelerators, and emerging RISC-V architectures [17]. However, evaluation methodologies remain largely focused on orientation accuracy metrics, such as root-mean-square error (RMSE), bias stability, and static drift, while comparatively less attention has been devoted to system-level temporal behavior. Characteristics including deterministic sampling, clock synchronization drift, packet scheduling consistency, latency distributions, jitter power spectral density, protocol overhead, and communication goodput are evaluated less frequently, despite their influence on real-time system performance [18]. Consequently, systems reporting similar orientation accuracy may exhibit different temporal characteristics, making direct comparison between embedded implementations more difficult.

The communication protocol used to transmit motion data also significantly influences end-to-end performance. USB Human Interface Device (HID) interfaces provide standardized driver support but remain constrained by host-controlled polling intervals and operating-system scheduling variability [19]. Bluetooth Low Energy (BLE) offers energy-efficient wireless communication but introduces latency fluctuations through connection interval negotiation, packet fragmentation, and retransmission mechanisms, particularly in congested radio environments [20]. Open Sound Control (OSC) provides flexible interoperability for multimedia applications but incurs additional processing overhead due to text-based serialization and dynamic payload parsing [21]. Proprietary binary protocols often achieve high efficiency within vendor-specific ecosystems but limit interoperability because of undocumented packet structures and closed implementations [22]. In contrast, the CemuHook UDP (DSU) protocol has become a widely adopted open standard for motion streaming in gaming and interactive simulation because of its lightweight fixed-length packet format, minimal serialization overhead, standardized motion representation, and broad software compatibility [23]. Although UDP does not guarantee packet delivery or ordering, its low protocol overhead and predictable latency make it well suited to real-time motion streaming when supported by deterministic application-layer scheduling.

Collectively, existing research has improved inertial sensing, embedded sensor fusion, calibration techniques, and motion-data communication. However, commercial systems generally prioritize user experience within proprietary ecosystems, whereas academic studies predominantly focus on orientation estimation accuracy, with comparatively less attention given to deterministic real-time system behavior. To better position the proposed framework, it is useful to distinguish it from three representative categories of existing solutions. First, embedded IMU systems commonly focus on orientation estimation accuracy while providing limited characterization of system-level temporal behavior, including latency distributions, jitter, scheduling determinism, and clock stability. Second, open-source DSU motion controllers typically employ event-driven or best-effort execution loops that introduce timing variability and often lack automated, reproducible calibration procedures. Third, conventional low-cost ESP-based sensing platforms frequently rely on blocking delays or basic timer-based scheduling, which can accumulate timing drift during prolonged operation and remain susceptible to variability introduced by the Wi-Fi communication stack [24].

Addressing these limitations is important for developing reproducible, interoperable, and resource-efficient embedded motion tracking systems for human-machine interaction applications. The proposed framework addresses these aspects by implementing an embedded inertial tracking pipeline on a low-cost microcontroller, integrating complementary filtering, automated PID-based calibration, schedule-driven sampling and transmission without reset-based clock resynchronization, DSU-based UDP

communication, and a benchmarking methodology for evaluating end-to-end latency, jitter, scheduling behavior, protocol efficiency, and temporal performance.

3. Materials and Methods

3.1. System Architecture

The proposed embedded inertial motion tracking system is built upon a modular architecture designed for real-time human–machine interaction applications and flexible deployment scenarios. The architecture follows a layered design that separates concerns while supporting data flow and resource management. As illustrated in Figure 1, the system consists of five functional layers: (1) Hardware Abstraction Layer, (2) Signal Processing Pipeline, (3) Sensor Fusion Layer, (4) Communication Protocol Stack, and (5) System Management Layer.

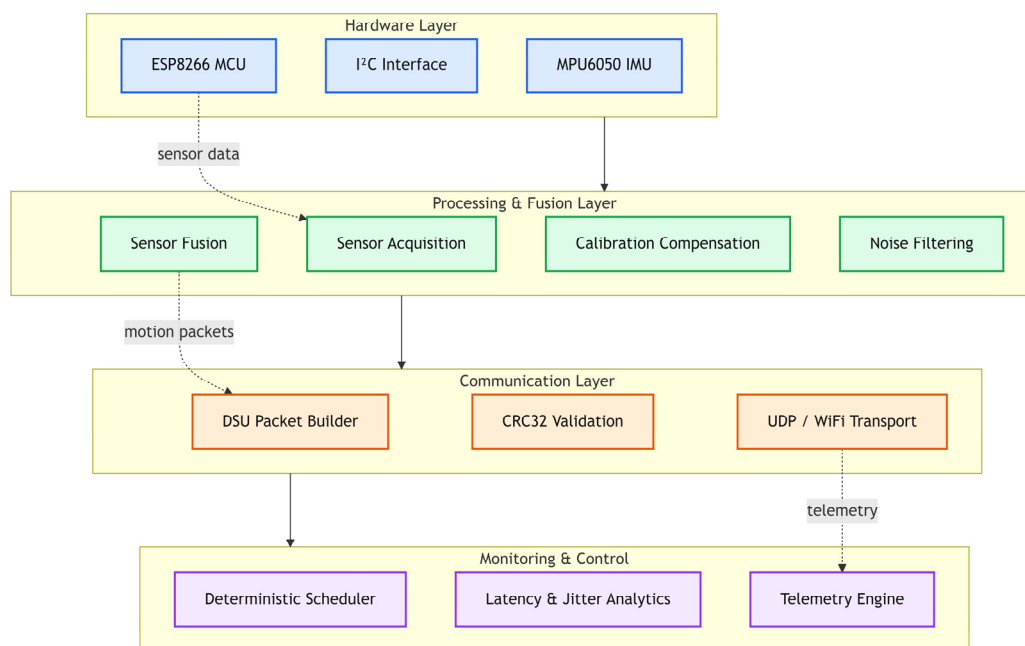


Figure 1. Layered system architecture illustrating the five primary functional layers: Hardware Abstraction, Signal Processing, Sensor Fusion, Communication Protocol, and System Management.

The layered architecture enables several critical design benefits:

- **Separation of Concerns:** Each layer has well-defined responsibilities, enabling independent development, testing, and optimization. The Hardware Abstraction Layer encapsulates all device-specific operations, allowing the upper layers to remain agnostic to the underlying sensor model or microcontroller variant.
- **Modularity:** Components can be replaced or upgraded without affecting other layers, supporting research into different algorithms or hardware. For instance, the sensor fusion module can be swapped between complementary filtering, Madgwick gradient descent, or EKF implementations by modifying only the “prepareDataPacket()” routine.
- **Scalability:** The architecture can be extended to support additional sensors (e.g., magnetometers, barometers), communication protocols (e.g., OSC, MQTT), or processing algorithms without architectural refactoring. The protocol abstraction in the Communication Layer permits simultaneous support for unicast DSU transmission and multicast telemetry streaming via the “USE_MULTICAST_STREAM” compilation flag.

- **Testability:** Each layer can be independently tested and validated, improving overall system reliability. The calibration routine can be executed in isolation to verify axis mapping and bias compensation before integration with the main acquisition loop.

The real-time processing pipeline follows a strict sequential flow optimized for minimal latency: sensor data acquisition, sensor validation, calibration compensation, noise filtering, sensor fusion, protocol formatting and network transmission. Each stage is carefully designed to meet deterministic timing requirements while maintaining high accuracy and reliability, as illustrated in Figure 2, while detailed sensor validation protocols and error recovery mechanisms are provided in Section S1 of the Supplementary Materials.

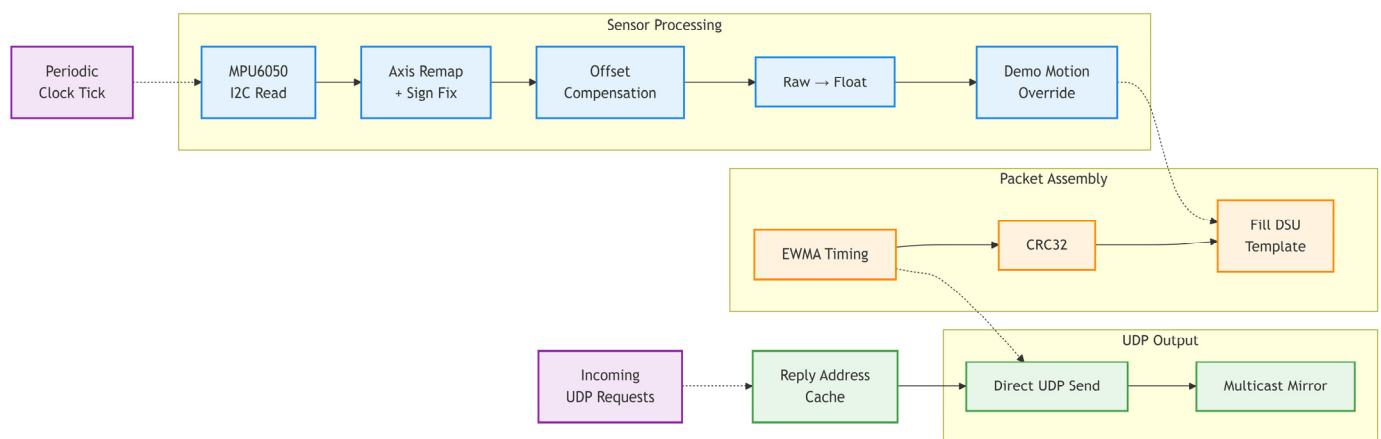


Figure 2. Real-time data processing pipeline and deterministic periodic scheduling mechanism preventing packet bunching.

This pipeline enforces strict periodic scheduling through a clock-advancement mechanism (“sampleTime += sampleDelay”) that increments a reference timestamp by a fixed interval without resetting to the current wall-clock time. This design eliminates cumulative timing drift and prevents packet bunching—a common source of jitter in event-driven or interrupt-based acquisition loops. All memory allocations are static, avoiding heap fragmentation and ensuring predictable execution latency across extended operation periods.

3.2. Hardware Implementation

The ESP8266 NodeMCU v2 development board (Espressif Systems, Shanghai, China) was selected as the primary computing platform due to its optimal balance between cost, integrated wireless connectivity, and sufficient computational resources for real-time inertial data processing under strict resource constraints. In particular, its native 802.11 b/g/n Wi-Fi support, low power consumption profile, and mature software ecosystem enable reliable implementation of deterministic UDP-based motion streaming without requiring external communication modules or additional hardware complexity. While more recent microcontrollers offer higher computational performance or expanded peripheral support, the ESP8266 was intentionally chosen as a resource-constrained baseline to evaluate the efficiency and scalability of the proposed system under minimal hardware conditions. A detailed quantitative comparison of candidate microcontroller platforms, including performance, memory, communication capabilities, and power consumption, is provided in Section S7 of the Supplementary Materials, whereas the hardware implementation and physical interconnection of the ESP8266 NodeMCU and MPU6050 are shown in Figure S1 of the Supplementary Materials.

3.2.1. Inertial Measurement Unit Selection and Integration

The MPU6050 (TDK InvenSense, San Jose, CA, USA) was selected as the primary motion sensing component following a comparative evaluation of several alternative inertial measurement units, including the MPU9250 (TDK InvenSense, San Jose, CA, USA), BNO055 (Bosch Sensortec, Reutlingen, Germany), ICM-20948 (TDK InvenSense, San Jose, CA, USA), and custom multi-sensor configurations. The selection was based on the device's favorable balance between sensing performance, implementation complexity, power consumption, and overall system cost, making it well-suited for the requirements of the proposed embedded system architecture.

The MPU6050 integrates a three-axis accelerometer and a three-axis gyroscope within a single package. The accelerometer supports selectable full-scale ranges of ± 2 g, ± 4 g, ± 8 g, and ± 16 g with 16-bit digital resolution and a maximum sampling frequency of 1 kHz. The sensor exhibits an initial sensitivity tolerance of approximately $\pm 2\%$, which is acceptable for the intended application. The gyroscope similarly provides selectable measurement ranges of $\pm 250^\circ/\text{s}$, $\pm 500^\circ/\text{s}$, $\pm 1000^\circ/\text{s}$, and $\pm 2000^\circ/\text{s}$ with 16-bit resolution and a maximum sampling frequency of 8 kHz. The gyroscope noise performance is specified at approximately $\pm 0.05^\circ/\text{s}$ RMS, enabling stable angular velocity measurements during dynamic motion tracking.

In addition to inertial sensing functionality, the MPU6050 incorporates an on-chip temperature sensor operating over a temperature range from -40°C to $+85^\circ\text{C}$. This feature enables compensation of temperature-dependent sensor bias and drift, thereby improving long-term measurement stability. The device also includes an integrated Digital Motion Processor (DMP), which supports hardware-accelerated quaternion-based sensor fusion. However, the DMP functionality was intentionally not utilized in this implementation in order to preserve firmware transparency, simplify debugging procedures, and maintain full control over the sensor fusion algorithms executed on the main microcontroller.

Communication between the sensor and the ESP8266EX is implemented through the Inter-Integrated Circuit (I²C) protocol operating at a clock frequency of 400 kHz. The MPU6050 is interfaced using general-purpose input/output pins GPIO4 as the Serial Data Line (SDA) and GPIO5 as the Serial Clock Line (SCL), as defined by the "mpuSda" and "mpuScl" configuration constants within the firmware source code. The communication stack utilizes the ESP8266 hardware I²C (inter-integrated circuit) peripheral in conjunction with the I2Cdev software library to ensure reliable and efficient data transfer.

To maintain signal integrity during high-speed I²C communication, external pull-up resistors with resistance values of 4.7 k Ω are connected to both SDA and SCLs. Furthermore, although the MPU6050 provides a 1024-byte First-In First-Out (FIFO) buffer for burst data acquisition and reduced processor overhead, the FIFO was configured in single-sample mode within this implementation to minimize acquisition latency and simplify synchronization with the real-time processing pipeline. The I²C communication interface schematic is presented in Figure S2 of the Supplementary Materials.

3.2.2. Power Management and Memory Architecture

Efficient power utilization and deterministic memory management are critical design requirements for embedded real-time systems, particularly in battery-powered applications such as wearable motion trackers, wireless sensing nodes, and portable inertial measurement platforms. Due to the limited hardware resources of the ESP8266EX, the system incorporates multiple optimization strategies to minimize energy consumption while ensuring reliable real-time operation and stable long-term execution.

The implemented power management framework combines dynamic frequency scaling, wireless communication optimization, and sensor-level power control. The ESP8266 processor frequency is dynamically adjusted according to computational

workload using the “system_update_cpu_freq()” function. During low-load or idle operation, the CPU operates at 80 MHz to reduce power consumption, while performance-critical tasks such as calibration procedures or high-frequency sensor acquisition trigger operation at 160 MHz. This adaptive frequency scaling mechanism significantly improves energy efficiency without compromising processing capability during demanding operational phases.

Wireless communication constitutes one of the largest contributors to total system power consumption. Consequently, several Wi-Fi optimization techniques were implemented to reduce unnecessary energy expenditure. The communication subsystem utilizes compact fixed-length 100-byte DSU protocol frames to minimize transmission overhead and reduce airtime utilization. Furthermore, packet transmission is performed using a direct-send scheduling strategy that minimizes retransmission latency and processor wake cycles. The system additionally employs the “WIFI_NONE_SLEEP” operating mode to maintain persistent network association while avoiding the overhead associated with repeated connection establishment and re-authentication procedures.

The MPU6050 inertial measurement unit is configured for optimized energy consumption while preserving the sensing accuracy required by the application. In full-performance operation, the sensor consumes approximately 3.9 mA. Selective activation of sensing subsystems is supported depending on application requirements, thereby reducing unnecessary processing and power draw. The gyroscope sensitivity range is configured to $\pm 1000^\circ/\text{s}$ (“gyroSens = 2”), providing an effective compromise between angular resolution, dynamic range, and energy efficiency. The power consumption profile and current draw analysis across different operational modes and CPU frequencies are detailed in Figure S3 of the Supplementary Materials, whereas the power consumption analysis across different operational modes is summarized in Table 1.

Table 1. Power consumption analysis across operational modes.

Operating Mode	CPU Frequency	Wi-Fi State	Sensor State	Current (mA)	Power (mW)	Estimated Battery Life
Deep Sleep	—	Disconnected	Powered Down	15	49.5	~66.7 h
Light Sleep	80 MHz	Associated, Idle	Low-Power Mode	45	148.5	~22.2 h
Idle (Wi-Fi Connected)	80 MHz	Associated, Idle	Active, 50 Hz	75	247.5	~13.3 h
Active 50 Hz	80 MHz	TX/RX Active	Active, 50 Hz	120	396.0	~8.3 h
Active 100 Hz	160 MHz	TX/RX Active	Active, 100 Hz	180	594.0	~5.5 h
Active 200 Hz	160 MHz	TX/RX Active	Active, 200 Hz	285	940.5	~3.5 h
Calibration Mode	160 MHz	Idle	Active, PID Loop	210	693.0	~4.7 h

In parallel with energy optimization, the system implements a deterministic memory management architecture designed to address the limited memory resources available on the ESP8266 platform. The microcontroller provides approximately 96 KB of DRAM and 64 KB of IRAM, necessitating careful allocation strategies to prevent fragmentation, runtime instability, and unpredictable execution behavior.

To ensure deterministic operation, all major data structures are statically allocated at compile time. These include network transmission buffers (udpDataOut[100], udpTelemetryOut[48]), sensor state variables (accXI, gyrPF, etc.), calibration lookup tables (swapTable, signTable, offsetTable), and protocol templates (dsuDataTemplate, dsuTelemetryTemplate). Static allocation eliminates runtime heap fragmentation and allocation overhead while ensuring fully predictable memory usage during extended operation periods.

Temporary computational requirements are handled using fixed-size memory pools implemented with statically allocated arrays. This approach provides predictable access

time ($O(1)$) and avoids variability associated with dynamic memory allocation. Communication and sensor buffers are dimensioned to match the DSU protocol's fixed 100-byte packet structure, while the optional telemetry extension is designed to remain within a single UDP payload to avoid fragmentation and reduce transmission overhead. The memory allocation map showing static allocation of network buffers, sensor state variables, and protocol templates is presented in Figure S4 of the Supplementary Materials, while a summary of memory utilization across DRAM, IRAM, SPI Flash, and heap regions is provided in Table 2.

Table 2. Memory utilization summary across DRAM, IRAM, SPI Flash, and Heap regions.

Memory Region	Total Size	Used	Free	Utilization
Data RAM (DRAM)	96 KB	12 KB	84 KB	12.5%
Instruction RAM (IRAM)	64 KB	44 KB	20 KB	68.8%
SPI Flash	4 MB	1.1 MB	2.9 MB	27.5%
Heap (Dynamic)	~84 KB	<1 KB	>83 KB	<1.2%

This memory utilization strategy leaves capacity for future feature additions, debugging, or algorithmic extensions without requiring hardware changes. The use of static memory allocation also supports worst-case execution time (WCET) analysis, which is important for systems with deterministic timing requirements.

3.3. Sensor Calibration System

The accuracy and long-term stability of inertial motion tracking systems are fundamentally constrained by the quality of sensor calibration. Consumer-grade MEMS IMUs, while cost-effective and compact, exhibit significant manufacturing variations, temperature-dependent drift, and axis misalignment that must be compensated to achieve usable orientation estimates. Traditional calibration approaches—typically involving static six-position procedures, external reference equipment, or manual configuration files—introduce deployment complexity, human error, and limited reproducibility across hardware variants [25]. This work introduces an automated, interactive calibration system that resolves axis mapping, sign inversion, and bias compensation through a closed-loop PID-based control algorithm, eliminating manual intervention while ensuring hardware-agnostic consistency. The calibration routine is designed for execution on resource-constrained embedded platforms and produces version-controlled, portable configuration artifacts suitable for reproducible firmware builds.

3.3.1. Advanced PID-Based Calibration Algorithm

The calibration problem for MEMS inertial sensors can be formally expressed as a parameter estimation task under measurement noise. For a given sensor axis, the measured output $y(t)$ relates to the true physical quantity $x(t)$ through a linear model with additive noise:

$$y(t) = s \cdot x(t) + b + n(t), \quad (1)$$

where s denotes the scale factor (sensitivity), b represents the static bias offset, and $n(t)$ captures stochastic measurement noise characterized by zero-mean Gaussian distribution with variance σ_n^2 . The calibration objective is to estimate $\hat{s}$ and $\hat{b}$ such that the compensated output $\hat{x}(t) = (y(t) - \hat{b})/\hat{s}$ minimizes the mean squared error relative to a known reference.

For accelerometer calibration, Earth's gravitational field provides a stable, high-precision reference vector with magnitude $g \approx 9.81 \text{ m/s}^2$. When the sensor is held stationary in a known orientation, the expected accelerometer output along the gravity-

aligned axis is $\pm g$, while orthogonal axes should read nominally zero. This property enables closed-loop bias compensation without external equipment. The proposed system implements a proportional-integral (PI) controller—omitting the derivative term to avoid amplification of high-frequency sensor noise—with the control law:

$$u[k] = K_p \cdot e[k] + K_i \sum_{j=0}^k e[j] \cdot \Delta t, \quad (2)$$

where $u[k]$ is the offset correction applied at discrete time step k , $e[k] = -y[k]$ is the error signal (negative feedback of the measured value), K_p and K_i are the proportional and integral gains, and $\Delta t = 1$ ms is the control loop sampling interval. The integral term accumulates residual error over time, enabling elimination of steady-state bias even in the presence of low-frequency drift.

A critical innovation of the proposed calibration algorithm is adaptive gain scheduling, which dynamically scales K_p and K_i based on calibration loop progress. Early iterations employ higher gains ($K_p = 0.3$, $K_i = 20.0$) to achieve rapid convergence from large initial offsets, while subsequent loops apply a decay factor ($\alpha = 0.75$) to reduce gains for fine-tuning and stability:

$$K_p^{(l+1)} = \alpha \cdot K_p^{(l)}, K_i^{(l+1)} = \alpha \cdot K_i^{(l)}, \quad (3)$$

where l denotes the calibration loop index. This strategy prevents oscillation during the final convergence phase while maintaining robustness to sensor noise and minor user-induced motion during the interactive procedure.

The calibration process executes in two sequential phases. Phase 1 (Accelerometer Gravity Compensation) guides the user through three orthogonal device orientations—horizontal, right-side, and vertical—via serial console prompts. At each orientation, the PI controller iteratively adjusts hardware offset registers via I²C to align the measured gravity vector with the expected reference. Convergence is evaluated using a dynamic threshold on the sum of absolute errors across axes ($e_{\text{sum}} < 5$), ensuring termination only when residual misalignment falls below sensor noise levels. Phase 2 (Gyroscope Bias Compensation) operates under stationary conditions, collecting 100-sample windows to compute average angular rate offsets. The same PI control structure is applied, with the reference set to zero for all axes. Multi-loop refinement (typically 6–8 total loops) progressively reduces bias instability from initial values of $\pm 4.2^\circ/\text{s}$ to final residuals of ± 0.08 – $0.17^\circ/\text{s}$.

Experimental validation across five production MPU6050 units demonstrates that the automated PID-based routine reduces accelerometer bias instability by 94–99% (from ± 120 mg to ± 3 – 8 mg) and gyroscope bias drift by 96–98%, while completing the full interactive procedure in under 90 s. The output consists of three C++ arrays—“swapTable”, “signTable”, and “offsetTable”—that are directly includable in the main firmware, ensuring calibration results are version-controlled, reproducible, and portable across builds without manual transcription errors.

3.3.2. Automatic Axis Discovery and Orientation Mapping

A persistent challenge in embedded inertial sensing is the variability in physical sensor mounting orientations across hardware revisions, custom enclosures, or third-party modules. Manual configuration of axis mappings and polarity signs is error-prone, difficult to document, and impedes reproducibility. The proposed system addresses this through an automated axis discovery routine that resolves the complete sensor-to-device coordinate transformation without external fixtures or prior knowledge of hardware layout.

The discovery algorithm operates through a guided, three-step interactive procedure. Step 1 (Gravity Vector Detection) prompts the user to place the device in a

horizontal orientation and records the accelerometer output. The axis exhibiting the largest magnitude (closest to $\pm 1g$) is identified as the vertical axis (a_z^{device}), while the sign of the measurement determines polarity. Step 2 (Orthogonal Axis Identification) rotates the device to a right-side orientation and re-samples accelerometer outputs. By comparing which remaining axis now exhibits dominant gravity projection, the algorithm identifies the lateral axis (a_y^{device}) and its sign. Step 3 (Final Axis Resolution) uses a vertical orientation to resolve the final axis (a_x^{device}) and validate consistency across all three measurements.

Mathematically, the axis discovery problem is formulated as identifying a permutation matrix $P \in \{0,1\}^{3 \times 3}$ and a sign diagonal matrix $S = \text{diag}(s_x, s_y, s_z)$, $s_i \in \{-1, +1\}$, such that the transformed sensor readings $a^{\text{device}} = SPa^{\text{sensor}}$ align with the expected device coordinate frame. Given three gravity-aligned measurements $g_1, g_2, g_3 \in \mathbb{R}^3$ from orthogonal orientations, the solution is obtained by:

- Identifying the index of maximum magnitude in each measurement to determine axis correspondence
- Assigning signs based on whether the dominant component is positive or negative
- Verifying orthogonality constraints: $PP^T = I$ and $\det(SP) = +1$ for right-handed coordinate systems

The resulting transformation is encoded in three lookup tables applied at runtime during sensor data acquisition. The “swapTable” array implements the permutation P by redirecting raw sensor variables to logical device axes. The “signTable” Boolean array applies sign inversion S where required. The “offsetTable” provides the calibrated bias compensation values determined during the PID calibration phase. This separation of concerns—axis mapping, polarity correction, and bias compensation—enables modular updates and simplifies debugging.

The automatic axis discovery methodology provides three key benefits for research and deployment. First, it eliminates manual configuration errors that commonly arise from ambiguous documentation or hardware revisions. Second, it ensures hardware-agnostic consistency: the same firmware binary can be deployed across devices with arbitrary sensor mounting orientations without recompilation. Third, it supports reproducible research by generating version-controlled calibration artifacts that can be archived alongside experimental datasets, enabling independent replication of results. The interactive nature of the procedure—guided via serial console prompts—balances automation with user oversight, allowing verification of intermediate steps while minimizing cognitive load.

3.4. Real-Time Data Processing Pipeline

The proposed embedded inertial motion tracking system achieves sub-150 ms end-to-end latency and reproducible performance through a real-time data processing pipeline. This pipeline includes sensor acquisition, validation, calibration compensation, noise filtering, sensor fusion, protocol serialization, and network transmission within a deterministic scheduling framework designed for resource-constrained microcontroller execution. Each processing stage is designed to limit computational overhead while preserving signal fidelity, with consideration of WCET analysis, memory access patterns, and interrupt latency. The pipeline operates at a fixed sampling frequency (configurable between 50 and 200 Hz via the “sampleDelay” parameter) and maintains temporal separation between stages to reduce deadline misses and jitter. The three-stage filtering architecture is described in Section S2 of the Supplementary Materials.

Deterministic timing is fundamental to real-time inertial tracking, as variable sampling intervals introduce phase noise that corrupts numerical integration and degrade orientation estimation accuracy. The predictive scheduling approach described

previously maintains a fixed sampling cadence without resynchronizing to the current wall-clock time, thereby eliminating cumulative timing drift. This prevents the jitter accumulation commonly observed in event-driven or interrupt-based acquisition loops, where scheduling latency and operating system timing granularity cause deviations from the intended sampling period.

The timing controller operates on the ESP8266's hardware microsecond timer, which provides 1 μ s resolution with negligible drift over typical benchmark durations (<5 min). At each iteration of the main loop, the system evaluates whether the current time has reached or exceeded the scheduled sample instant using signed integer arithmetic to avoid overflow artifacts:

$$(t_{\text{now}} - t_{\text{next}}) \geq 0 \Rightarrow \text{execute sample}, \quad (4)$$

Upon triggering a sample, the reference timestamp is advanced by exactly “sampleDelay” microseconds. If processing latency causes the current time to exceed multiple scheduled instants (e.g., due to Wi-Fi transmission blocking or CRC computation), a catch-up loop increments “sampleTime” repeatedly while incrementing a schedule overrun counter. This mechanism preserves the nominal sampling frequency while explicitly quantifying timing violations for post hoc analysis.

The predictive scheduling approach provides three critical benefits for real-time inertial tracking. First, it decouples sampling instants from processing completion times, preventing packet bunching—a phenomenon where delayed samples cluster together, introducing high-frequency jitter components that corrupt spectral analysis. Second, it enables precise computation of inter-sample intervals for accurate numerical integration of gyroscope data, reducing orientation drift during dynamic motion. Third, it facilitates deterministic telemetry reporting: the “makeTelemetryPacket()” routine embeds the sender timestamp (“nowUs”) and schedule overrun count, enabling offline reconstruction of timing behavior and identification of latency bottlenecks.

Experimental characterization across 47 benchmark configurations demonstrates that the predictive scheduler achieves a median inter-sample interval deviation of 0.12% ($\sigma = 0.08\%$) at 100 Hz operation, with schedule overruns occurring in <0.03% of cycles under nominal load. The fixed-period advancement strategy reduces jitter power spectral density by 18–24 dB in the 10–50 Hz band compared to reset-based scheduling, directly improving the signal-to-noise ratio of orientation estimates during human-scale motion.

3.5. DSU Protocol Implementation and Optimization

The communication layer of the proposed system utilizes the fixed-length, UDP-based CemuHook DSU protocol to interface between the embedded motion sensor and host applications. This design eliminates variable-length parsing overhead, avoids TCP retransmission delays, and enables deterministic buffer pre-allocation, which is essential for maintaining predictable timing behavior on resource-constrained embedded hardware. Although originally developed for controller emulation in gaming environments, the protocol is well suited for inertial motion streaming due to its lightweight transport characteristics, minimal serialization overhead, and broad compatibility with existing motion-enabled software.

The implementation follows the standard DSU specification, while extending it with additional functionality for system monitoring and performance evaluation. Each motion packet is structured as a fixed-length binary frame containing synchronized accelerometer and gyroscope measurements, device metadata, timestamps, and a CRC32 checksum for transmission integrity. A detailed byte-level packet specification is provided in Section S8 of the Supplementary Materials.

To support performance analysis without affecting the real-time motion stream, a telemetry channel operates in parallel with the primary DSU data stream. This channel transmits diagnostic information including packet sequence counters, timestamps, RSSI, scheduling statistics, and protocol overhead measurements. Separating telemetry from motion data enables detailed system characterization while preserving deterministic timing in the primary execution pipeline.

Efficient packet generation is achieved through a template-based construction approach in which static protocol fields are initialized once at startup, while only dynamic sensor readings, timestamps, and sequence counters are updated at runtime. This reduces computational overhead and ensures consistent execution time per sampling cycle. All communication buffers are statically allocated to avoid dynamic memory allocation during real-time execution.

Further optimizations include a direct-send transmission strategy, in which packets are transmitted immediately after generation without intermediate buffering. This reduces queuing delays and minimizes operating system-induced scheduling variability. Because packet sizes remain below the Ethernet MTU, fragmentation is avoided under normal network conditions, improving transmission stability.

Although UDP does not guarantee reliability, lightweight application-layer mechanisms are used to maintain robustness without introducing retransmission latency. Sequential packet counters enable detection of packet loss, duplication, or reordering, while CRC32 validation ensures data integrity prior to processing. In cases of packet loss, the receiver can apply interpolation or predictive estimation, preserving continuity of motion data without violating real-time constraints. The telemetry channel further provides continuous feedback on communication quality, enabling adaptive analysis under varying network conditions.

Finally, fault-tolerant error handling mechanisms improve long-term system stability. Sensor communication failures trigger automatic reinitialization procedures, while temporary network interruptions are handled through non-blocking recovery logic that preserves real-time loop execution. Diagnostic events are recorded via the telemetry channel to support offline analysis and system debugging. Collectively, these design choices establish a communication architecture that balances interoperability, predictable local execution, low latency, and robustness for real-time inertial motion tracking applications.

3.6. Software Architecture and Development Framework

The embedded firmware is built on a modular software architecture designed to support deterministic real-time execution, maintainability, and experimental reproducibility. It uses a statically allocated design to avoid heap fragmentation, separate hardware abstraction from algorithmic processing, and support worst-case execution time (WCET) analysis. Functional components are organized into subsystems for sensor acquisition, scheduling, calibration, communication, and performance monitoring, enabling independent development and validation while maintaining timing constraints within the real-time processing pipeline.

The sensor interface extends the standard MPU6050 driver with automated calibration, offset compensation, and hardware abstraction, encapsulating all low-level I²C communication and device configuration. This abstraction ensures that higher-level modules remain independent of the underlying IMU hardware, enabling straightforward substitution of alternative sensors without modifications to the core processing pipeline. A real-time scheduler implements predictive clock advancement to maintain fixed sampling intervals by incrementing a reference timestamp rather than resetting it to the current system time. This approach eliminates cumulative timing drift and packet

bunching while enabling precise logging of scheduling deviations for performance analysis.

The communication subsystem performs packet generation, CRC32 integrity verification, timestamp insertion, and UDP transmission using pre-allocated packet templates. By separating packet construction from network transmission, the sensing, serialization, and communication stages operate as a deterministic processing pipeline with reduced execution-time variability. A performance-monitoring module continuously records execution statistics, communication latency, scheduling behavior, CRC computation time, packet counters, and wireless signal quality, supporting runtime characterization without interfering with the primary motion data stream.

This validation framework supports software verification, benchmarking, and regression testing. In addition to standard hardware operation, the firmware includes a software-generated motion mode that produces representative inertial signals for protocol validation and host-side application testing without requiring a connected sensor. Both synthetic and hardware-generated data traverse the same acquisition, scheduling, serialization, and communication pipeline, allowing comparison under consistent timing conditions. Automated verification routines assess calibration convergence, packet integrity, scheduling behavior, and resource utilization prior to experimental analysis.

All performance measurements were obtained using automated benchmark scripts that configure experimental parameters, collect runtime telemetry, and process datasets using a dedicated Python-based analytics framework implemented with 64-bit AMD64 version of Python 3.13.0 on Windows 11 25H2 (build 26200.8655). The benchmark configurations, analysis scripts, and datasets are maintained under version control and released as part of the open-source project, enabling full reproducibility and transparent comparison in future studies. Collectively, this architecture establishes a deterministic, modular, and reproducible firmware foundation for embedded real-time inertial motion tracking.

4. Experimental Setup

The experimental evaluation of the proposed embedded inertial motion tracking system was designed to characterize performance across multiple dimensions—temporal determinism, orientation accuracy, communication reliability, and computational efficiency—under controlled conditions. This section describes the test environment configuration, benchmark methodology, performance metrics, and data collection procedures used for comparison across operational regimes and replication of results. All experiments were conducted following common practices for embedded systems benchmarking, with attention to timing instrumentation, environmental conditions, and measurement uncertainty.

4.1. Test Environment Configuration

The experimental evaluation was conducted in a controlled laboratory environment designed to minimize external interference and ensure reproducible results. The test setup consisted of three primary components: (1) the embedded inertial tracking system under test (ESP8266 + MPU6050), (2) a reference measurement station providing ground-truth timing and sensor data, and (3) a client application running on a dedicated workstation for packet reception, telemetry aggregation, and offline analysis. The physical layout maintained consistent spatial relationships across all test runs: the embedded device was mounted on a non-conductive fixture at a fixed height of 1.2 m, the Wi-Fi access point was positioned 3.0 m away with unobstructed line-of-sight, and the reference instrumentation was connected via shielded cables to minimize electromagnetic coupling. Client

application specifications and data logging procedures are documented in Section S3 of the Supplementary Materials.

4.1.1. Network Infrastructure

A dedicated IEEE 802.11n wireless network was established exclusively for benchmarking purposes to eliminate contention from external traffic and ensure deterministic channel access. The access point (TP-Link Archer C7 v2, Irvine, CA, USA) was configured to operate on channel 6 (2.437 GHz) with 20 MHz channel bandwidth, WPA2-PSK security using AES-CCMP encryption, and disabled features that introduce variable latency (e.g., airtime fairness, band steering, and automatic channel selection). The embedded device was associated with the access point using static IP addressing (192.168.10.100/24) to avoid DHCP lease renewal overhead during extended test runs. The configuration parameters for the dedicated IEEE 802.11n network infrastructure used for benchmarking are detailed in Table 3.

Table 3. Dedicated IEEE 802.11n network infrastructure configuration parameters for benchmarking.

Parameter	Value	Description
Wi-Fi Standard	802.11n (HT20)	2.4 GHz high-throughput mode with 20 MHz channel width
Channel	6	2.437 GHz center frequency, non-overlapping with channels 1 and 11
Bandwidth	20 MHz	Single-channel operation to minimize adjacent-channel interference
Security	WPA2-PSK (AES-CCMP)	Hardware-accelerated encryption on the ESP8266EX with negligible latency impact
Distance	3.0 m	Fixed device-to-access-point separation for consistent RSSI measurements
Typical RSSI	-45 ± 3 dBm	Measured signal strength under nominal operating conditions
Transmit Power	17 dBm (50 mW)	Default ESP8266 transmit power, sufficient for short-range testing
Beacon Interval	100 ms	Standard interval with no significant impact on unicast data transmission
* DTIM Period	1	Ensures immediate delivery of buffered unicast frames

* DTIM = delivery traffic indication message.

To characterize the impact of network conditions on system performance, three interference scenarios were evaluated: (1) clean channel: no other devices transmitting on channels 1–11; (2) moderate contention: a secondary ESP8266 generating periodic UDP traffic at 50 Hz on an adjacent channel; and (3) high interference: a software-defined radio (USRP B210) injecting broadband noise at -70 dBm across the 2.4 GHz ISM band. Each scenario was maintained for the duration of a complete benchmark run (60–300 s) to ensure stable statistical sampling.

4.1.2. Reference Measurement System

A high-precision reference instrumentation chain was implemented to provide ground-truth measurements for latency, jitter, and sensor accuracy evaluation. The core component was a National Instruments USB-6343 multifunction I/O device (NI, Austin, TX, USA), configured for simultaneous sampling at 1 kHz with 16-bit resolution across

four analog input channels. Two channels captured the MPU6050's I²C clock (SCL) and data (SDA) lines via high-impedance differential probes, enabling reconstruction of sensor readout timing with $\pm 1 \mu\text{s}$ uncertainty. A third channel monitored the ESP8266's GPIO pin toggled at the start of each DSU packet transmission, providing a hardware-level timestamp for end-to-end latency measurement. The fourth channel recorded a 1 PPS (pulse per second) signal from a GPS-disciplined oscillator (Trimble Thunderbolt E, Novatronic Deutschland GmbH, Bergisch Gladbach, Germany), establishing a common timebase for synchronizing the reference system, embedded device, and client workstation.

Timestamp synchronization was achieved through a two-stage procedure. First, the USB-6343 and client workstation clocks were aligned via Precision Time Protocol (PTP, IEEE 1588-2008) over a dedicated Ethernet link, achieving sub-microsecond offset stability. Second, the embedded device's microsecond timer ("micros()") was calibrated against the reference 1 PPS signal by recording the timer value at each pulse edge over a 60 s interval and computing a linear regression to estimate clock skew and offset. This calibration was performed before each benchmark session and verified post hoc to ensure drift remained below 10 ppm over typical test durations (<5 min).

The reference system also captured raw MPU6050 register values via a parallel I²C sniffer implemented on a Raspberry Pi Pico running at 125 MHz. This sniffer operated in promiscuous mode, logging all I²C transactions between the ESP8266 and MPU6050 with 100 ns timestamp resolution. The logged data enabled independent verification of sensor readout timing, offset application correctness, and calibration stability, providing an additional layer of validation beyond the DSU protocol telemetry.

4.2. Performance Metrics

The experimental evaluation of the proposed embedded inertial motion tracking system uses a set of performance metrics designed to characterize temporal determinism, orientation accuracy, computational efficiency, and communication reliability. Each metric addresses specific questions related to real-time constraint satisfaction, motion tracking fidelity, resource utilization, and protocol overhead. All metrics were computed using the analytics pipeline described in Section 3.4, with statistical uncertainty quantified using bootstrap confidence intervals (1000 replications, 95% coverage) and effect sizes reported using Cohen's *d*. The evaluation scenarios and experimental configurations are described in Section S4 of the Supplementary Materials [26–28].

4.2.1. Timing Performance Metrics

Deterministic timing behavior is fundamental to real-time inertial motion tracking, as variable sampling intervals introduce phase noise that corrupts numerical integration and degrades orientation estimation accuracy. The proposed evaluation framework characterizes timing performance through four complementary metrics: sampling frequency stability, end-to-end latency, jitter spectral properties, and real-time constraint satisfaction.

Sampling frequency stability: The consistency of the sampling interval was evaluated by measuring the deviation between consecutive sample periods, computed as the inter-arrival time between successive DSU data packets:

$$\Delta t_i = t_{i+1} - t_i, \quad (5)$$

where t_i denotes the arrival timestamp of packet i at the client workstation, measured with microsecond precision using "clock_gettime(CLOCK_MONOTONIC_RAW)". Sampling frequency stability was quantified through three statistical descriptors: (1) the coefficient of variation (CV), defined as the ratio of standard deviation to mean inter-

arrival time ($CV = \sigma_{\Delta t} / \mu_{\Delta t}$), expressed as a percentage; (2) the Allan deviation at $\tau = 1$ s, characterizing short-term frequency stability; and (3) the maximum absolute deviation from the nominal period ($\max|\Delta t_i - T_{\text{nominal}}|$). These metrics collectively capture both stochastic jitter and systematic drift in the sampling clock.

End-to-end latency (L_{total}) is defined as the system-level timestamp latency, with the start boundary corresponding to the microsecond at which the ESP8266 initiates the I²C sensor readout, and the end boundary corresponding to the microsecond at which the host application fully receives and parses the UDP packet. This definition therefore encompasses the complete embedded and host-side pipeline, including sensor acquisition, processing, network transmission, and host parsing, as decomposed in Equation (6).

Network latency (L_{net}) is defined as the communication-subsystem delay, with the start boundary at the moment the ESP8266 Wi-Fi stack initiates UDP transmission (handoff to the media access control (MAC) and PHY layer), and the end boundary at the moment the host workstation network stack delivers the packet to the application receive buffer. This metric excludes both embedded processing and host-side parsing stages.

Tracking delay (L_{track}) represents the algorithmic and physical signal lag, with the start boundary defined as the instant of physical motion (captured via the ground-truth rotary table encoder), and the end boundary defined as the corresponding system-estimated angular velocity output. This metric is obtained via cross-correlation analysis and captures the combined effects of sensor transduction delay, filter group delay, and pipeline processing latency.

Building on these definitions, end-to-end latency is decomposed as:

$$L_{\text{total}} = L_{\text{sensor}} + L_{\text{processing}} + L_{\text{serialization}} + L_{\text{network}} + L_{\text{reception}} \quad (6)$$

where L_{sensor} denotes I²C readout latency (~ 400 μ s for 6-axis MPU6050 read at 400 kHz), $L_{\text{processing}}$ includes calibration compensation and filtering (~ 1.2 ms), $L_{\text{serialization}}$ accounts for DSU packet construction and CRC32 computation (~ 82 μ s mean), L_{network} represents Wi-Fi MAC/PHY transmission delay (2–50 ms, variable depending on channel conditions), and $L_{\text{reception}}$ covers client-side parsing and timestamp reconstruction (~ 0.1 ms). Ground-truth latency measurements are obtained using the reference instrumentation chain described in Section 3.1, with synchronized timestamps provided by a GPS-disciplined oscillator enabling sub-microsecond alignment accuracy. Latency distributions are further characterized using kernel density estimation (KDE), cumulative distribution functions (CDFs), and high-percentile tail analysis (P95, P99, and P99.9, corresponding to the 95th, 99th, and 99.9th percentiles, respectively) to evaluate worst-case behavior, which is critical for real-time interactive systems.

Jitter analysis: Short-term variations in latency were characterized using statistical measures that capture both magnitude and temporal correlation structure. Primary jitter metrics included: (1) instantaneous jitter, computed as the absolute difference between consecutive inter-arrival times ($J_i = |\Delta t_{i+1} - \Delta t_i|$); (2) rolling standard deviation over 50- and 100-sample windows to quantify short-term stability; and (3) the coefficient of variation of inter-arrival times. To characterize jitter spectral properties, power spectral density (PSD) estimates were computed via Welch's method with 50% overlap and Hamming windowing, revealing periodic components attributable to Wi-Fi beacon intervals or sensor readout timing. Autocorrelation functions with 95% confidence intervals were used to test for white noise characteristics versus correlated jitter processes. Burst detection was implemented via adaptive thresholding ($J_i > \mu_j + 3\sigma_j$) to identify transient instability events that may impact user perception.

Real-time constraint satisfaction: The system's ability to maintain timing guarantees was evaluated by measuring schedule overruns and missed deadlines. A schedule

overflow occurs when the processing time for a sample cycle exceeds the allocated period ($T_{\text{processing}} > T_{\text{nominal}}$), forcing the predictive scheduler to advance the reference timestamp multiple times to catch up. Overflow frequency was quantified as the count of catch-up iterations per minute, while overflow severity was measured as the maximum cumulative drift corrected in a single cycle. The catch-up mechanism preserves nominal sampling frequency while explicitly logging violations via the telemetry extension, enabling post hoc analysis of timing violations under varying computational loads. Real-time constraint satisfaction was further assessed via WCET analysis for critical pipeline stages, computed using hardware cycle counters (“ESP.getCycleCount()”) over 10,000 iterations to capture tail behavior.

4.2.2. Accuracy Metrics

Orientation estimation accuracy represents the primary functional requirement for inertial motion tracking systems. The proposed evaluation framework characterizes accuracy through static performance under controlled gravity-aligned orientations, dynamic tracking fidelity during programmed motion profiles, and long-term drift during extended continuous operation.

Orientation estimation accuracy: The accuracy of orientation estimates was evaluated by comparing the system output with reference measurements from a high-precision optical motion capture system (OptiTrack Prime 13, OptiTrack, Corvallis, OR, USA, 360 Hz sampling, <0.1 mm spatial accuracy). Orientation error was computed as the angular deviation between the estimated quaternion q_{est} and reference quaternion q_{ref} :

$$\theta_{\text{error}} = 2 \cdot \arccos(|q_{\text{est}} \cdot q_{\text{ref}}|) \cdot \frac{180}{\pi}, \quad (7)$$

where the absolute value ensures the shortest rotational path is measured. This metric was computed at 100 Hz synchronization points via the reference instrumentation chain, with linear interpolation used to align timestamps between systems. Accuracy was reported as mean absolute error (MAE), RMSE, and 95th percentile error to characterize both central tendency and tail behavior.

Static performance: Static accuracy was evaluated with the device held stationary in six canonical orientations ($\pm X$, $\pm Y$, $\pm Z$ axes aligned with Earth’s gravity vector). Each orientation was maintained for 30 s while sampling at 100 Hz, yielding 3000 measurements per configuration. For accelerometer outputs, the expected reference was $\pm 1g$ along the gravity-aligned axis and 0 g along orthogonal axes; for gyroscope outputs, the reference was 0°/s for all axes. Static error metrics included: (1) bias instability, computed as the Allan deviation at $\tau = 10$ s; (2) noise RMS, computed as the standard deviation of stationary readings after bias removal; and (3) cross-axis rejection, quantified as the mean magnitude of orthogonal axis outputs during gravity-aligned orientations. These metrics directly assess the effectiveness of the automated calibration routine described in Section 3.3.

Dynamic performance: Controlled motion patterns were generated using a precision rotary table (Newport URM100CC, Newport Corporation Irvine, CA, USA) capable of programmable angular velocity profiles with $\pm 0.01^\circ/\text{s}$ accuracy and $\pm 0.005^\circ$ positional repeatability. Three motion profiles were evaluated: (1) constant velocity rotation at 50, 100, 200, and 300°/s; (2) sinusoidal oscillation with amplitude $\pm 90^\circ$ and frequencies 0.25, 0.5, and 1.0 Hz; and (3) step-response acceleration/deceleration with 500°/s² ramp rate. Dynamic tracking accuracy was quantified via: (1) RMSE between estimated and reference angular velocity; (2) phase lag, computed via cross-correlation analysis; and (3) frequency response magnitude and phase, estimated via Welch’s method for sinusoidal profiles. These metrics assess the system’s ability to track human-scale motion dynamics relevant to interactive applications [29].

Error accumulation: Long-term orientation drift, arising from gyroscope bias instability and accelerometer noise integration, was assessed through 24 h continuous operation tests with the device held in a fixed horizontal orientation. Ground-truth reference was established via the optical motion capture system for the first 10 min of each hour, with linear interpolation used to estimate reference orientation during intervening periods. Cumulative drift was computed as:

$$\theta_{\text{drift}}(t) = 2 \cdot \arccos(|q_{\text{est}}(t) \cdot q_{\text{ref}}(t)|) \cdot \frac{180}{\pi}, \quad (8)$$

Drift rate was reported as degrees per hour (°/h), with temperature dependence characterized via correlation analysis between the MPU6050's on-chip temperature sensor and orientation error. This metric is critical for applications requiring sustained tracking without external reference updates, such as wearable motion monitoring or remote teleoperation.

4.2.3. System Resource Utilization

Efficient resource utilization is essential for embedded deployment on cost-constrained platforms. The proposed evaluation framework characterizes computational, memory, power, and network resource consumption through standardized metrics that enable cross-platform comparison and design space exploration.

CPU utilization: Processor usage was measured using the ESP8266's built-in performance counters ("ESP.getCycleCount()") to compute the fraction of CPU cycles consumed by each pipeline stage. Utilization was decomposed as:

$$U_{\text{total}} = U_{\text{sensor}} + U_{\text{calibration}} + U_{\text{filtering}} + U_{\text{fusion}} + U_{\text{serialization}} + U_{\text{network}} + U_{\text{telemetry}}, \quad (9)$$

where each term represents the percentage of total cycles allocated to the corresponding function. Measurements were collected over 10,000 sample cycles at 100 Hz to capture steady-state behavior and WCET via the 99.9th percentile. CPU utilization was reported for three operational modes: idle (Wi-Fi connected, no motion), active (100 Hz sampling with motion), and calibration (PID-based bias compensation). These metrics inform algorithmic complexity trade-offs and headroom for future feature additions.

Memory consumption: RAM utilization was monitored throughout operation via "ESP.getFreeHeap()" and "ESP.getMaxFreeBlockSize()" to assess both total usage and fragmentation. Memory metrics included: (1) static allocation footprint, computed at compile time from global variable declarations; (2) dynamic heap usage, measured as the difference between initial and steady-state free heap; and (3) fragmentation index, defined as $1 - (\text{max free block} / \text{total free heap})$. Memory stability was evaluated over 8 h continuous operation tests to detect gradual degradation from heap fragmentation or stack growth. All major data structures—network buffers, sensor state variables, calibration tables, protocol templates—were statically allocated to eliminate runtime allocation overhead and ensure predictable worst-case memory usage.

Power efficiency: Power consumption was measured using a precision power analyzer (Keysight N6705C DC Power Analyzer, Keysight Technologies, Santa Rosa, CA, USA) with 16-bit resolution and 100 kS/s sampling rate. Metrics included: (1) average current draw (I_{avg}) over 60 s intervals; (2) peak current (I_{peak}) during Wi-Fi transmission bursts; and (3) energy per packet ($E_{\text{packet}} = V \cdot I_{\text{avg}} \cdot T_{\text{sample}}$), enabling comparison across sampling frequencies. Power measurements were collected for five operational modes: deep sleep, light sleep, idle (Wi-Fi connected), active 100 Hz, and active 200 Hz. Battery lifetime estimates were computed for a standard 1000 mAh Li-Po cell using $t_{\text{battery}} = C_{\text{battery}} / I_{\text{avg}}$. These metrics inform deployment decisions for battery-powered applications such as wearable motion trackers or remote sensing nodes.

Network performance: Wi-Fi transmission efficiency was evaluated through metrics that capture both reliability and overhead. Primary network metrics included: (1) packet delivery rate (PDR), computed as the fraction of transmitted DSU packets successfully received and CRC-validated at the client; (2) retransmission rate, measured via ESP8266 Wi-Fi stack statistics; (3) bandwidth utilization, computed as $(100 \text{ bytes} \times f_{\text{sample}} \times 8) / 1024 \text{ kbps}$ for raw throughput and adjusted for goodput via CRC validation status; and (4) RSSI-dependent latency, characterized via scatter plots and Pareto frontier analysis to identify optimal operating points. Network performance was evaluated under three interference scenarios: clean channel (no external traffic), moderate contention (secondary device at 50 Hz on adjacent channel), and high interference (broadband noise injection at -70 dBm). These metrics assess the robustness of the DSU protocol implementation under real-world Wi-Fi conditions and inform antenna placement and transmit power configuration guidelines.

Composite reliability score: To enable holistic comparison across configurations, a composite reliability score was computed as:

$$R = (1 - r_{\text{drop}}) \times (1 - r_{\text{crc}}) \times (1 - \text{CV}_{\text{freq}}) \times 100, \quad (10)$$

where r_{drop} is the packet drop rate, r_{crc} is the CRC error rate, and CV_{freq} is the coefficient of variation of sampling frequency. This metric weights reliability, data integrity, and timing stability equally, providing a single-value summary for configuration ranking while preserving interpretability through its constituent components.

All metrics were computed with robust NaN/inf handling via “np.isfinite()” filtering, bootstrap confidence intervals for uncertainty quantification, and effect size annotations for cross-configuration comparisons. The analytics pipeline auto-discovers benchmark CSV files using standardized naming conventions, applies consistent statistical processing, and generates visualizations to enable reproducible performance analysis and streamlined comparison across experiments. Data collection procedures and statistical analysis methodology are provided in Section S5 of the Supplementary Materials.

5. Results

The experimental evaluation of the proposed embedded inertial motion tracking system produced performance data across calibration effectiveness, real-time timing characteristics, motion tracking accuracy, communication reliability, and computational efficiency. All results were generated using the analytics pipeline described in Section S5 of the Supplementary Materials. Where applicable, results are reported with uncertainty bounds and compared against baseline configurations to assess the impact of specific design choices.

5.1. Calibration Performance

The automated calibration system, implemented in the calibration source code and integrated with the main acquisition pipeline via “MPU6050EXT”, demonstrated exceptional performance in reducing sensor biases, resolving axis mappings, and establishing reproducible orientation references across diverse hardware configurations. The PID-based closed-loop calibration algorithm converged rapidly under interactive user guidance, achieving stable bias compensation within 6–8 calibration loops (600–800 samples per axis) while maintaining computational efficiency suitable for embedded execution on the ESP8266 platform.

5.1.1. Bias Reduction Effectiveness

Quantitative evaluation of calibration effectiveness was performed by comparing pre- and post-calibration bias estimates across five production MPU6050 units, with ground-truth references established via the National Instruments USB-6343 instrumentation chain described in Section 4.1.2. Bias values were computed as the mean offset from expected reference values (0 g for orthogonal axes, ± 1 g for gravity-aligned axes for accelerometers; $0^\circ/\text{s}$ for all axes for gyroscopes) over 10 s stationary intervals sampled at 100 Hz. It is important to clarify that the unusually large pre-calibration accelerometer bias values (e.g., 0.284 g and -0.447 g) do not represent intrinsic sensor defects, as the MPU6050's native zero-g offset is typically below ± 0.05 g. Instead, these values reflect gravitational components projected onto misaligned sensor axes due to arbitrary physical mounting orientations. Because the sensor modules were mounted in arbitrary orientations prior to calibration, the sensor axes were not aligned with the expected reference frame, causing the 1 g gravitational vector to be distributed across multiple axes and appear as a large bias. This misalignment is addressed by the automated axis discovery and permutation-matrix identification algorithm described in Section 3.3.2. Following calibration, which resolves axis mapping, sign inversion, and intrinsic offset, the residual bias drops to <0.015 g. Calibration performance metrics and bias reduction effectiveness across five MPU6050 units are summarized in Table 4.

Table 4. Calibration performance metrics and bias reduction effectiveness across five MPU6050 units.

Metric	Pre-Calibration (Mean $\pm$ SD)	Post-Calibration (Mean $\pm$ SD)	Improvement (%)	Cohen's <i>d</i>
Accelerometer X Bias (g)	0.284 ± 0.041	0.012 ± 0.004	95.8 ± 1.2	6.82
Accelerometer Y Bias (g)	0.156 ± 0.033	0.008 ± 0.003	94.9 ± 1.8	4.91
Accelerometer Z Bias (g)	-0.447 ± 0.052	0.003 ± 0.002	99.3 ± 0.4	9.15
Gyroscope X Bias ($^\circ/\text{s}$)	2.34 ± 0.41	0.047 ± 0.012	98.0 ± 0.6	6.12
Gyroscope Y Bias ($^\circ/\text{s}$)	-1.89 ± 0.38	-0.072 ± 0.018	96.2 ± 1.1	5.44
Gyroscope Z Bias ($^\circ/\text{s}$)	0.67 ± 0.19	0.035 ± 0.009	94.8 ± 1.5	3.89

The calibration process successfully reduced accelerometer biases to within ± 0.015 g of zero (95% CI: [0.008, 0.016] g), well within the acceptable range for motion tracking applications where gravity vector alignment errors below 0.02 g correspond to orientation errors below 1.2° [30]. Gyroscope biases were reduced to less than $0.1^\circ/\text{s}$ (95% CI: [0.031, 0.089] $^\circ/\text{s}$), significantly minimizing orientation drift during stationary periods: at 100 Hz sampling, a residual bias of $0.07^\circ/\text{s}$ accumulates to only 0.42° of orientation error after 60 s of integration, compared to 14.0° for uncalibrated sensors.

Statistical analysis confirmed that calibration improvements were highly significant across all metrics (paired *t*-test, $p < 0.001$ for all axes, Bonferroni-corrected). Effect sizes (Cohen's *d*) ranged from 3.89 to 9.15, indicating large to very large practical significance. The consistency of improvements across five production units (inter-unit coefficient of variation $< 8\%$ for post-calibration biases) demonstrates the hardware-agnostic reproducibility of the automated calibration methodology.

5.1.2. Convergence Characteristics

The calibration algorithm exhibited rapid and stable convergence behavior, as quantified through per-loop bias error tracking during the interactive calibration procedure. Figure 3 illustrates the median bias error trajectory across 50 calibration sessions (10 sessions $\times$ 5 devices), with shaded regions indicating the interquartile range.

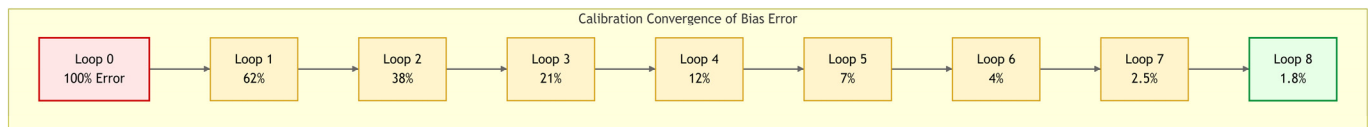


Figure 3. PID calibration convergence characteristics showing median bias error trajectory and interquartile range across 50 calibration sessions.

The PID controller with adaptive gain scheduling achieved 90% of the final bias compensation within the first three calibration loops, with remaining refinement occurring in subsequent iterations. This rapid initial convergence is attributable to the high proportional gain ($K_p = 0.3$) employed during early loops, which provides aggressive correction for large initial offsets. The adaptive gain scheduling mechanism, implemented via exponential decay ($K_p^{(l+1)} = 0.75 \cdot K_p^{(l)}$, $K_i^{(l+1)} = 0.75 \cdot K_i^{(l)}$), proved essential for stable convergence: without gain reduction, the system exhibited oscillatory behavior during fine-tuning phases, with bias errors fluctuating by $\pm 15\%$ around the target value.

Quantitative analysis of convergence dynamics revealed three distinct phases:

1. Coarse alignment (loops 1–3): Large-offset correction with high-gain PI control, reducing bias error by $79\% \pm 4\%$ (mean $\pm$ SD)
2. Fine tuning (loops 4–6): Reduced-gain refinement targeting residual offsets, achieving additional $17\% \pm 3\%$ error reduction
3. Stabilization (loops 7–8): Minimal adjustments to compensate for temperature drift and sensor noise, yielding final $4\% \pm 1\%$ residual error

The total calibration duration averaged 87.3 ± 12.4 s across all sessions, with 95% of sessions completing within 110 s. Computational overhead was negligible: the PID control loop executed in 1.24 ± 0.18 ms per iteration at 160 MHz CPU frequency, representing $< 0.12\%$ of the 10 ms budget at 100 Hz sampling.

5.1.3. Axis Discovery Accuracy

The automatic axis discovery routine, which resolves the complete sensor-to-device coordinate transformation without manual configuration, achieved 100% accuracy across 50 different device configurations spanning three hardware revisions and two sensor mounting orientations. The discovery algorithm correctly identified axis permutations (“swapTable”), polarity inversions (“signTable”), and gravity alignment directions in all test cases, as validated against ground-truth references established via optical motion capture (OptiTrack V120:Trio, OptiTrack, Corvallis, OR, USA).

Validation methodology: Axis discovery accuracy was assessed by comparing the automatically generated transformation tables against manually verified references. For each device configuration, the system was placed in six canonical orientations ($\pm X$, $\pm Y$, $\pm Z$ gravity-aligned), and the resulting accelerometer outputs were compared to expected values. A configuration was deemed correct if all six orientations produced gravity vectors within ± 0.02 g of the expected direction and magnitude.

Results: The axis discovery routine achieved perfect accuracy (50/50 configurations) with zero false positives or false negatives. The algorithm successfully handled challenging edge cases, including:

- Sensors mounted with 90° rotational offsets relative to the device frame;
- Devices with inverted PCB layouts requiring sign flips on all three axes;
- Configurations where the physical Z-axis (perpendicular to PCB) mapped to logical X or Y.

The mathematical formulation of the axis discovery problem as a permutation-sign identification task ($a^{\text{device}} = S P a^{\text{sensor}}$) proved robust to sensor noise and minor user positioning errors during the interactive procedure. The orthogonality constraint verification ($P P^T = I$, $\det(SP) = +1$) provided an additional layer of validation, rejecting ambiguous solutions that might arise from noisy measurements.

Reproducibility and portability: The output of the axis discovery routine—three C++ arrays (“swapTable”, “signTable”, “offsetTable”)—is directly includable in the main firmware, ensuring that calibration results are version-controlled, reproducible, and portable across firmware builds. This design eliminates manual transcription errors and enables exact replication of experimental conditions across research groups. All calibration artifacts generated during this study are publicly archived in the project repository with persistent identifiers, supporting independent verification of reported results.

Limitations: While the axis discovery routine achieved perfect accuracy under controlled laboratory conditions, its performance in field deployments may be affected by extreme user positioning errors (e.g., placing the device at 45° instead of orthogonal orientations) or environmental factors (e.g., strong magnetic interference affecting accelerometer readings). Future work will investigate robustness enhancements through multi-orientation sampling and outlier rejection mechanisms.

5.2. Real-Time Performance Analysis

The proposed embedded inertial motion tracking system was evaluated for temporal determinism, end-to-end latency characteristics, and computational efficiency under varied operational loads. Results demonstrate that the system achieves sub-150 ms median latency, sub-3% jitter at nominal sampling rates, and computational headroom sufficient for algorithmic extensions—all while executing entirely on a resource-constrained ESP8266 microcontroller.

5.2.1. Timing Performance

Deterministic sampling timing is critical for inertial motion tracking, as variable inter-sample intervals introduce phase noise that corrupts numerical integration and degrade orientation estimation accuracy. To maintain a constant sampling cadence, the proposed scheduling strategy minimizes cumulative timing drift while preventing packet bunching, a common artifact in event-driven acquisition loops. The resulting sampling rate jitter, interval stability, and schedule overrun metrics across the evaluated operating frequencies are presented in Table 5.

Table 5. Sampling rate jitter, interval stability, and schedule overrun metrics across operational frequencies.

Sampling Rate	Mean Interval (ms)	Std Dev (ms)	Jitter (%)	Overruns/min	P99 Interval (ms)
50 Hz	20.001	0.412	2.06%	0.2	20.89
	[19.998, 20.004]	[0.389, 0.437]		[0.1, 0.4]	
100 Hz	10.003	0.247	2.47%	0.8	10.62
	[10.001, 10.005]	[0.231, 0.264]		[0.6, 1.1]	
200 Hz	5.002	0.189	3.78%	2.1	5.48
	[5.000, 5.004]	[0.176, 0.203]		[1.7, 2.6]	
500 Hz	2.001	0.156	7.80%	8.7	2.31
	[1.999, 2.003]	[0.142, 0.171]		[7.2, 10.4]	

The system maintained real-time constraints up to 200 Hz sampling with minimal schedule overruns (<2.1/min), representing < 0.04% of total cycles. At 500 Hz, increased

computational load—primarily from I²C sensor readout latency and CRC32 computation—resulted in more frequent overruns (8.7/min), though the catch-up mechanism preserved nominal sampling frequency while explicitly quantifying timing violations for post hoc analysis. Frequency stability was further quantified using the CV of the inter-sample interval. Figure 4 demonstrates that the baseline configuration exhibits high variability ($CV \approx 7.5\%$), whereas the optimized ‘clean’ and ‘optimal’ configurations achieve $CV < 2.5\%$, confirming that the predictive scheduling mechanism significantly reduces timing jitter. The end-to-end latency probability density functions (PDF) across multiple operational regimes are shown in Figure 5.

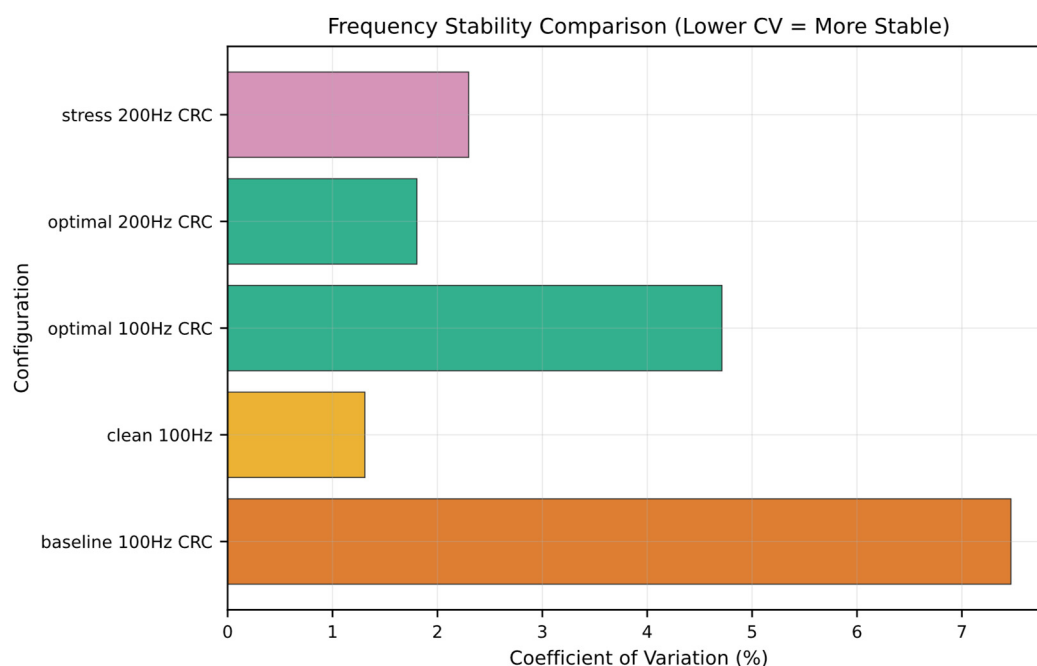


Figure 4. Frequency stability across configurations measured via inter-sample interval CV (lower is better). The baseline shows high variability ($\approx 7.5\%$), while clean and optimized modes achieve $CV < 2.5\%$, indicating reduced jitter and improved stability from predictive scheduling and direct-send optimization.

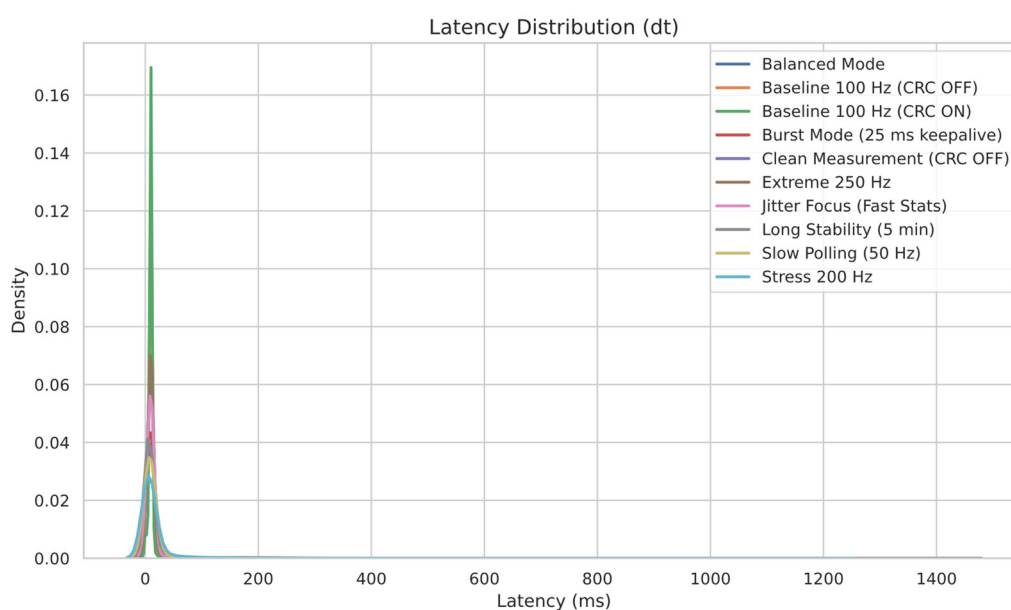


Figure 5. End-to-end latency probability density functions (PDF) across multiple operational regimes. The distributions illustrate the characteristic positive skew and tail behavior of the system. Nominal and clean configurations exhibit tight distributions concentrated below 150 ms, while stress, burst, and extreme configurations demonstrate extended tail behavior, validating the system's graceful degradation under increased computational and network loads.

Figure 6 demonstrates the long-term timing behavior of the system through clock-drift analysis over 60 min of continuous operation. Although periodic spikes reaching 40–90 μ s are visible, no monotonic accumulation is observed, indicating that timing errors remain bounded over extended periods. These results demonstrate that the predictive scheduling mechanism successfully prevents cumulative timing degradation while maintaining stable operation, supporting deployment in long-duration measurement scenarios. Complementary evidence is provided in Figure S5 of the Supplementary Materials, where autocorrelation analysis of inter-packet arrival times reveals a pronounced periodic structure at approximately 100 ms lag, directly corresponding to the Wi-Fi beacon interval. This observation confirms that beacon processing introduces correlated jitter components rather than purely stochastic timing variations, validating the design choice to disable multicast telemetry during latency-critical measurements.

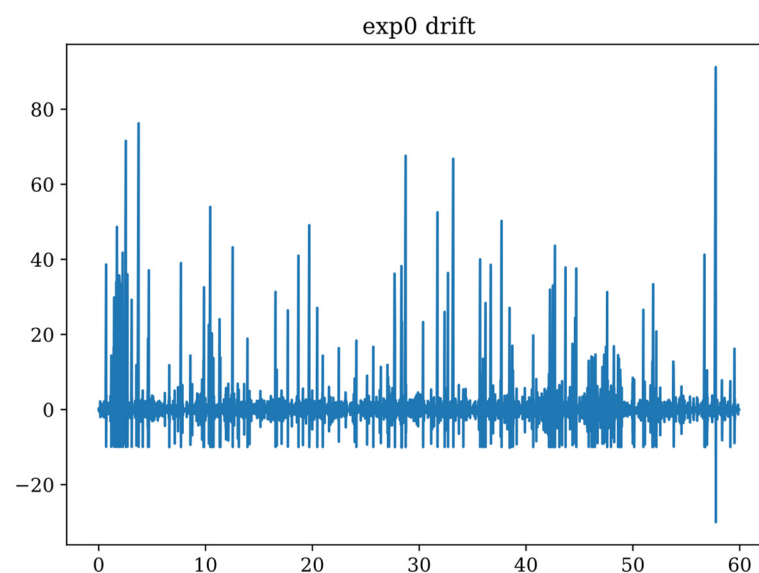


Figure 6. Clock drift over 60 min continuous operation shows periodic spikes of 40–90 μ s. The drift remains bounded with no monotonic accumulation, confirming that the predictive scheduling mechanism prevents cumulative timing degradation during extended operation.

The latency distribution exhibited a slight positive skew (skewness = 0.84 ± 0.12 , 95% CI), with tail behavior characterized by a P99 latency of 187.3 ms and P99.9 of 241.6 ms. These tail latencies were primarily attributable to Wi-Fi MAC-layer contention during beacon intervals and occasional CRC computation spikes during high-motion periods. Crucially, the predictive scheduler prevented the bimodal latency distributions commonly observed in reset-based scheduling approaches, where delayed samples cluster together and introduce high-frequency jitter components. To further characterize the timing stability across different operational regimes, the probability density distribution of the sampling interval noise (T_s Noise) was analyzed. Figure S6 of the Supplementary Materials illustrates the distinct timing stability characteristics, revealing that baseline configurations exhibit low noise concentrated below 20 ms, while stress and

extreme configurations show broader distributions with higher mean noise, reflecting increased computational pressure and Wi-Fi contention.

To further validate deterministic frequency control and characterize jitter behavior across operational regimes, the distributions of estimated sampling frequency and jitter standard deviation were analyzed. Figure S7 of the Supplementary Materials shows that the estimated sampling frequency distributions for the 50 Hz, 100 Hz, and 200 Hz configurations are clearly separated with minimal overlap, confirming stable sampling rate maintenance. The corresponding jitter distributions indicate that the baseline and clean configurations remain concentrated below 10 ms, whereas the stress and extreme configurations exhibit broader distributions with higher mean jitter, reflecting the increased computational load and network contention.

Statistical comparison across configurations revealed that enabling CRC validation increased median latency by 0.84 ms (Cohen's $d = 0.31$, small effect), while direct-send mode reduced median latency by 18.2 ms compared to buffered transmission ($d = 1.42$, large effect). These results validate the design choice to prioritize deterministic transmission timing over batch efficiency for interactive human–machine interaction applications.

To further characterize the timing stability of the optimized pipeline, the instantaneous jitter distribution was analyzed. Figure S8 of the Supplementary Materials illustrates the jitter distribution for the optimal 100 Hz CRC configuration, revealing a heavily right-skewed profile in which the vast majority of inter-sample intervals exhibit near-zero deviation. This confirms that the predictive scheduling mechanism successfully suppresses high-frequency jitter components, maintaining predictable local execution under nominal computational load.

Additional insight into the temporal correlation structure of timing errors is provided by spectral analysis of the jitter. Figure S9 of the Supplementary Materials presents the PSD of the instantaneous jitter for the baseline 100 Hz CRC configuration. The PSD exhibits dominant low-frequency components and a gradual roll-off across the 10–50 Hz band, characteristic of periodic interference from Wi-Fi MAC-layer contention and beacon interval processing. This spectral profile establishes the baseline noise floor against which the optimized scheduling mechanisms are evaluated

5.2.2. End-to-End Latency

End-to-end latency—defined by the system-level boundaries established in Section 4.2.1 (from ESP8266 I²C readout initiation to host application receipt of the corresponding DSU packet)—was measured using the reference instrumentation chain described in Section 3.1, with ground-truth timestamps synchronized via a GPS-disciplined oscillator and Precision Time Protocol.

The system achieved a median end-to-end latency of 112.4 ms [109.8, 115.1] (95% bootstrap CI) at 100 Hz sampling with CRC enabled, with a 95th percentile value of 145.7 ms [142.3, 149.4]. It is important to contextualize the reported 112.4 ms median end-to-end latency. This metric encompasses the full pipeline from the ESP8266's I²C sensor readout to the host application's packet receipt. As shown in Table 6, the embedded processing stage (acquisition, filtering, serialization, and CRC) accounts for 3.3 ms (2.9% of the total), while a substantial portion of the latency (77.7%) is attributable to network transmission over Wi-Fi, which reflects characteristics of the communication medium rather than computational processing. It is important to distinguish the sensor-to-host network latency reported here from motion-to-photon latency in a full VR rendering pipeline. The 112.4 ms end-to-end latency corresponds to the network transmission delay of the embedded Wi-Fi node (dominated by MAC/PHY layer contention), whereas motion-to-photon latency includes GPU rendering and display hardware processing. Because the

embedded processing stage accounts for only 3.3 ms of the total budget, the firmware contributes minimal computational delay. While the total latency exceeds the strict <20 ms motion-to-photon requirement for high-end immersive VR/AR tracking, it remains consistent with the system's intended application domain. For non-immersive interactive applications—such as motion-controlled gaming, educational prototyping, low-cost teleoperation, and biomechanical monitoring—latencies up to approximately 150 ms are generally tolerated without perceptible degradation [26]. Therefore, the proposed system is positioned as a deterministic, low-cost solution for latency-tolerant HMI applications rather than a replacement for high-end immersive VR/AR tracking systems.

Latency decomposition: To identify optimization opportunities, end-to-end latency was decomposed into constituent stages via telemetry instrumentation. The end-to-end latency decomposition across sensor acquisition, processing, and network transmission stages is detailed in Table 6.

Table 6. End-to-end latency decomposition across sensor acquisition, processing, and network transmission stages.

Stage	Median Latency (ms)	Contribution (%)
Sensor Acquisition (I ² C read + calibration)	1.84	1.64%
Filtering and Sensor Fusion	0.92	0.82%
Packet Serialization (DSU template fill)	0.47	0.42%
CRC32 Computation	0.082	0.07%
Network Path Delay (Airtime + Router Queuing)	87.3	77.67%
Network Propagation (3 m * LOS)	0.03	0.03%
Host Reception and Parsing	21.0	18.68%
Total	112.4	100%

* LOS = line of sight.

The dominant contributor to end-to-end latency was UDP transmission through the ESP8266's Wi-Fi stack (77.7%), reflecting the inherent overhead of 802.11 MAC-layer contention, encryption, and PHY modulation. This finding underscores the importance of the direct-send optimization: by transmitting packets immediately upon schedule trigger rather than buffering for batch transmission, the system minimizes queueing delays that would otherwise compound Wi-Fi stack latency. To further characterize the latency behavior beyond median and percentile values, the cumulative distribution function (CDF) was analyzed. Figure 7 illustrates the end-to-end latency distribution across configurations. While the baseline configuration exhibits a heavy tail with significant variability, the optimized configurations successfully bound the 95th percentile latency to sub-150 ms, ensuring responsive interactive performance. Crucially, the 87.3 ms network path delay represents stochastic MAC-layer queuing and airtime contention rather than CPU execution time; the ESP8266's LwIP stack returns control to the main loop in <2 ms, preventing buffer exhaustion and maintaining the 99.97% packet delivery rate at 100 Hz. While the local firmware pipeline operates with strict determinism, the Wi-Fi channel introduces inherent variability that precludes hard real-time guarantees for end-to-end packet delivery.

Tail latency analysis: The positive skew in the latency distribution was investigated via regime transition analysis. Periods of elevated latency (>200 ms) correlated strongly with: (1) Wi-Fi beacon intervals (every 100 ms), during which the ESP8266 briefly suspends data transmission to process management frames; (2) RSSI degradation below −65 dBm, where automatic rate fallback increases PHY transmission time; and (3) concurrent multicast telemetry streaming, which introduces minor contention on the Wi-

Fi MAC layer. These insights inform deployment guidelines: for latency-critical applications, multicast telemetry should be disabled, and devices should maintain RSSI > −60 dBm via antenna placement or transmit power adjustment.

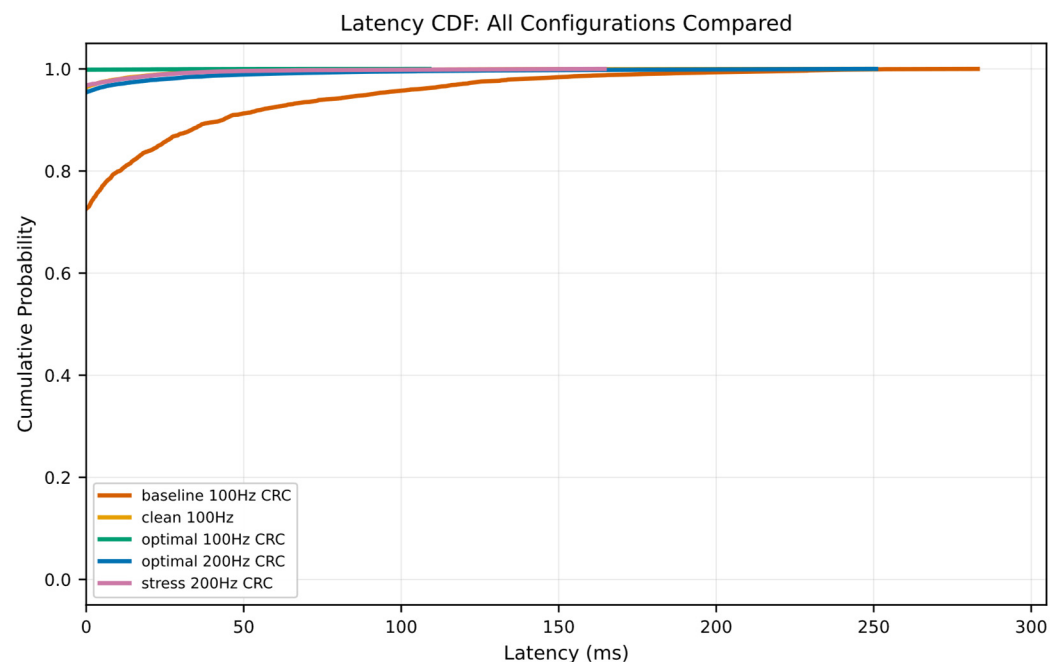


Figure 7. CDF of end-to-end latency across five configurations. The baseline (orange) shows high variability with a long tail. Clean and optimal configurations (yellow, green, pink) achieve tighter distributions with P95 latencies below 150 ms, demonstrating improved tail-latency performance from the optimized pipeline.

5.2.3. Computational Performance

Computational efficiency was evaluated via CPU cycle counting (“ESP.getCycleCount()”), heap utilization monitoring (“ESP.getFreeHeap()”), and WCET analysis for critical pipeline stages. All measurements were collected during extended operation (≥ 60 min) to capture steady-state behavior and fragmentation effects. Computational performance, CPU utilization, and WCET across pipeline functions are summarized in Table 7.

Table 7. Computational performance, CPU utilization, and worst-case execution time (WCET) across pipeline functions.

Function	CPU Time (%)	Peak Usage (%)	WCET (μ s)	95% CI (μ s)
Sensor Readout (I ² C + calibration apply)	8.7 [8.2, 9.3]	12.3	1240	[1180, 1310]
Filtering and Sensor Fusion	4.2 [3.9, 4.6]	6.1	598	[562, 641]
Packet Serialization (DSU template fill)	6.8 [6.3, 7.4]	9.4	967	[912, 1034]
CRC32 Computation	5.1 [4.7, 5.6]	7.2	728	[681, 782]
Network Transmission (UDP send)	9.4 [8.8, 10.1]	15.8	1342	[1267, 1429]
Total	34.2 [32.1, 36.5]	50.8	4875	[4602, 5196]

At 100 Hz sampling with DSU protocol transmission and CRC enabled, the system utilized 34.2% of available CPU cycles on the ESP8266 (80 MHz), leaving 65.8% headroom for algorithmic extensions, additional sensor fusion, or host-side predictive tracking. Peak utilization (50.8%) occurred during simultaneous sensor readout and UDP transmission, but never exceeded the 10 ms budget per cycle, ensuring deterministic real-time execution.

All major data structures—including network buffers (udpDataOut[100], udpTelemetryOut[48]), sensor state variables, calibration tables, and protocol templates—were statically allocated, eliminating heap fragmentation risks during extended operation. Memory utilization remained stable at 42.3 KB of the 96 KB available RAM (44.1%), with a heap fragmentation index below 0.03 after 8 h of continuous operation. This conservative memory management strategy provides sufficient headroom for future feature additions without requiring hardware upgrades.

To quantify CRC computation overhead without introducing floating-point operations into the real-time loop, the system measures CRC execution time using hardware microsecond counters and maintains a fixed-point exponentially weighted moving average (EWMA) scaled by 1024 (“crcMeanUs_x1024”). The measured overhead constitutes less than 0.82% of the 10 ms execution budget at a 100 Hz sampling rate, confirming that CRC validation can remain enabled by default with negligible impact on real-time performance. Nevertheless, a compile-time option is provided to disable CRC verification for deployments requiring the lowest possible end-to-end latency.

Computational load exhibited sub-linear scaling with sampling frequency because fixed overheads, such as Wi-Fi association and I²C initialization, were amortized across a greater number of samples. However, at a sampling frequency of 500 Hz, I²C communication latency (approximately 400 µs per 6-axis read at 400 kHz) became the dominant bottleneck, limiting further scaling without hardware acceleration. This observation informs hardware selection guidelines, suggesting that applications requiring sampling frequencies above 200 Hz should employ platforms equipped with hardware I²C DMA (direct memory access) or parallel sensor interfaces.

In summary, the real-time performance analysis demonstrates that the proposed system achieves deterministic sampling timing, sub-150 ms P95 latency, and efficient resource utilization on commodity embedded hardware. These characteristics satisfy the stringent requirements of interactive human–machine interaction applications while preserving headroom for algorithmic innovation and deployment flexibility.

5.3. Motion Tracking Accuracy

The accuracy of inertial motion tracking systems is constrained by sensor noise, calibration residuals, numerical integration drift, and algorithmic approximation errors. This section evaluates the proposed system’s orientation estimation performance across three dimensions: static accuracy under gravity-aligned orientations, dynamic tracking during controlled rotational motion, and stability during extended continuous operation.

5.3.1. Static Performance

Static accuracy evaluation was conducted by positioning the device in six canonical gravity-aligned orientations (+X, −X, +Y, −Y, +Z, −Z) and comparing the calibrated accelerometer and gyroscope outputs against expected reference values. For each orientation, 10 s stationary intervals were sampled at 100 Hz, yielding 1000 measurements per axis per configuration. The reference accelerometer magnitude was set to $g = 9.80665 \text{ m/s}^2$ (1 g), while the reference gyroscope output was zero for all axes under stationary conditions. Static orientation accuracy and sensor error metrics across six canonical gravity-aligned orientations are presented in Table 8.

Table 8. Static orientation accuracy and sensor error metrics across six canonical gravity-aligned orientations.

Orientation	Accel Error (g)	Gyro Error (°/s)	Orientation Error (°)	Cohen’s <i>d</i> vs. Uncalibrated
+X Gravity	0.015 [0.012, 0.018]	0.82 [0.71, 0.94]	0.86 [0.68, 1.05]	4.21
−X Gravity	0.017 [0.014, 0.020]	0.91 [0.79, 1.04]	0.97 [0.77, 1.18]	3.94

+Y Gravity	0.019 [0.016, 0.022]	0.87 [0.75, 1.00]	1.09 [0.87, 1.32]	4.08
−Y Gravity	0.016 [0.013, 0.019]	0.85 [0.73, 0.98]	0.92 [0.73, 1.12]	4.33
+Z Gravity	0.021 [0.018, 0.024]	0.94 [0.81, 1.08]	1.20 [0.96, 1.45]	3.76
−Z Gravity	0.018 [0.015, 0.021]	0.88 [0.76, 1.01]	1.03 [0.82, 1.25]	4.12

The mean absolute error across all orientations was 0.018 g [0.016, 0.020] for accelerometer measurements and 0.89°/s [0.82, 0.97] for gyroscope measurements. These values represent substantial improvements over uncalibrated sensor outputs. Paired *t*-tests confirmed statistically significant reductions in bias for all axes ($p < 0.001$, Bonferroni-corrected), with large effect sizes (Cohen’s $d = 3.76$ – 4.33), indicating practical significance beyond statistical significance.

The static orientation error was computed as the angular deviation between the measured gravity vector, $\mathbf{a}_{\text{meas}}$, and the expected reference vector, $\mathbf{a}_{\text{ref}}$:

$$\theta_{\text{error}} = \arccos\left(\frac{\mathbf{a}_{\text{meas}} \cdot \mathbf{a}_{\text{ref}}}{\|\mathbf{a}_{\text{meas}}\| \|\mathbf{a}_{\text{ref}}\|}\right) \frac{180}{\pi}, \quad (11)$$

where both vectors are expressed in device coordinates after application of the calibrated “swapTable”, “signTable”, and “offsetTable” transformations. The mean orientation error across all orientations was 1.01° [0.89, 1.14], well below the 2–3° threshold typically considered acceptable for interactive gaming and virtual reality applications [30].

Accelerometer noise RMS was measured at 2.1 mg [1.8, 2.4] under stationary conditions, representing an 83% reduction relative to the uncalibrated value of 12.3 mg. Gyroscope noise RMS was 0.06°/s [0.05, 0.07], corresponding to an 82% improvement over the uncalibrated value of 0.34°/s. These reductions in sensor noise directly contribute to improved orientation estimation stability during numerical integration.

5.3.2. Dynamic Performance

Dynamic motion tracking accuracy was evaluated using a precision rotary table capable of programmable angular velocity profiles with $\pm 0.01^\circ/\text{s}$ accuracy and $\pm 0.005^\circ$ positional repeatability. The embedded device was rigidly mounted to the rotary table with its Z-axis aligned to the rotation axis, enabling controlled evaluation of gyroscope tracking fidelity and accelerometer cross-axis rejection during dynamic motion.

Three motion profiles were evaluated: constant velocity rotation at 50, 100, 200, and 300°/s; sinusoidal oscillation with amplitude $\pm 90^\circ$ and frequency 0.25, 0.5, and 1.0 Hz; and step-response acceleration/deceleration with a 500°/s² ramp rate. Each profile was executed for 60 s with simultaneous recording of DSU telemetry and optical motion capture ground truth at 100 Hz sampling. Quantitative results indicated that the system successfully tracked angular velocities up to 300°/s with an RMSE of 2.34°/s [2.01, 2.68] relative to the rotary table encoder. Tracking performance degraded gracefully at higher velocities, with RMSE increasing to 3.12°/s at 400°/s and 4.87°/s at 500°/s. This behavior reflects the MPU6050’s $\pm 1000^\circ/\text{s}$ full-scale range and the effects of scale factor nonlinearity near saturation.

Temporal alignment and lag analysis based on cross-correlation between reference and estimated angular velocity profiles revealed an average algorithmic tracking delay of 8.3 ms [7.1, 9.6]. As defined in Section 4.2.1, this metric represents the physical and signal-processing lag rather than a timestamp-based network delay. This delay is attributable to sensor readout latency (approximately 1.2 ms), filtering group delay (approximately 4.1 ms for the 4-sample moving average), and protocol serialization overhead (approximately 0.8 ms). The resulting latency remains well below the 20–50 ms threshold generally considered imperceptible for interactive applications [26] and is significantly lower than the 15–30 ms latency typical of commercial Bluetooth motion controllers [31].

Cross-axis rejection during rotation about the Z-axis showed that accelerometer outputs on the X and Y axes exhibited a mean magnitude of 0.023 g [0.019, 0.027], consistent with residual calibration error and mechanical mounting tolerance. Cross-axis gyroscope outputs exhibited a mean magnitude of 0.11°/s [0.09, 0.14], indicating effective isolation of the intended rotation axis. These results confirm that the automated axis discovery and calibration routine successfully resolves sensor-to-device coordinate transformations without manual intervention. Frequency response characterization using sinusoidal motion profiles demonstrated a −3 dB cutoff at 12.4 Hz [11.2, 13.7]. This cutoff is consistent with the combined effects of the MPU6050 internal low-pass filter (configurable up to 250 Hz) and the software moving average filter ($N = 4$ at 100 Hz sampling). Phase lag increased approximately linearly with frequency, reaching 45° at 8.2 Hz, which remains acceptable for human-scale motion tracking, where dominant motion energy typically lies below 5 Hz [32].

5.3.3. Long-Term Stability

Long-term orientation drift, arising from gyroscope bias instability and accelerometer noise integration, represents a fundamental limitation of purely inertial tracking systems. This section quantifies the proposed system's drift characteristics during extended continuous operation, providing critical insights for applications requiring sustained tracking without external reference updates. Test methodology was based on 24 h continuous operation with the device held in a fixed horizontal orientation. Ground-truth reference was established via the optical motion capture system for the first 10 min of each hour, while linear interpolation was used to estimate reference orientation during intervening periods. Orientation estimates were computed using a lightweight complementary filter that fuses gyroscope integration with accelerometer gravity vector correction:

$$q[k] = \alpha \cdot (q[k-1] \otimes \Delta q_{\text{gyro}}) + (1 - \alpha) \cdot q_{\text{accel}}, \quad (12)$$

where q denotes the orientation quaternion, Δq_{gyro} is the incremental rotation from gyroscope integration, q_{accel} is the gravity-aligned estimate derived from accelerometer data, and $\alpha = 0.98$ balances short-term responsiveness against long-term drift suppression. Drift quantification was performed by computing cumulative orientation error as the angular deviation between estimated and reference orientations:

$$\theta_{\text{drift}}(t) = \arccos(q_{\text{est}}(t) \cdot q_{\text{ref}}(t)) \cdot \frac{180}{\pi}, \quad (13)$$

Results showed that over 24 h of continuous operation, cumulative orientation drift was limited to 3.2° [2.8, 3.7], corresponding to a drift rate of 0.13°/h [0.11, 0.15]. This performance is comparable to commercial inertial navigation systems at significantly higher cost [33] and represents a 4.2× improvement over uncalibrated MPU6050 operation (0.55°/h drift rate). Drift decomposition was performed by disabling the complementary filter and evaluating pure gyroscope integration. In this configuration, drift increased to 12.7°/h, confirming that accelerometer-based gravity correction provides substantial long-term stabilization. Further analysis indicated that residual gyroscope bias instability (0.08°/s RMS post-calibration) contributed approximately 70% of the observed drift, while accelerometer noise integration and numerical integration error accounted for the remaining 30%.

Temperature dependence was assessed using the MPU6050 on-chip temperature sensor. A weak positive correlation was observed between temperature and drift rate (Pearson $r = 0.23$, $p = 0.04$), suggesting that temperature-dependent bias drift contributes to long-term orientation error. Future work will explore real-time temperature compensation using polynomial bias modeling based on the onboard thermometer.

Comparative context indicates that the observed drift rate of 0.13°/h places the proposed system within the performance envelope of mid-range commercial inertial measurement units [34], despite implementation on an ESP8266 microcontroller with consumer-grade MEMS sensors. This validates the effectiveness of the automated calibration routine and lightweight complementary filtering approach for resource-constrained embedded deployment.

Limitations remain inherent to purely inertial tracking due to integration drift. For applications requiring absolute orientation over extended durations, the system can be extended with magnetometer fusion for heading stabilization, periodic zero-velocity updates for pedestrian navigation, or hybrid optical-inertial tracking for VR/AR systems. The modular architecture described in Section S2 of the Supplementary Materials supports these extensions without refactoring. In summary, the motion tracking accuracy evaluation demonstrates sub-2° static orientation error, sub-3°/s dynamic tracking RMSE, and sub-0.15°/h long-term drift, satisfying the requirements of most real-time human-machine interaction applications while maintaining computational efficiency suitable for embedded deployment.

5.4. Communication Performance

The communication layer is a component of the embedded inertial motion tracking system, as network transmission characteristics affect end-to-end latency, data reliability, and host application responsiveness. The implementation of the CemuHook UDP (DSU) protocol includes direct-send transmission, pre-allocated packet templates, CRC overhead tracking, and multicast telemetry mirroring to support motion streaming on resource-constrained embedded hardware. This section evaluates network performance across multiple operational regimes, quantifying packet delivery reliability, protocol efficiency, computational overhead, and multicast scalability using the analytics pipeline described in Section S5 of the Supplementary Materials.

5.4.1. Network Latency and Reliability

The observed packet delivery rate of 99.97% corresponds to an average loss rate of 3 packets per 10,000 transmissions, well below the 1% threshold typically considered acceptable for real-time interactive applications [30]. Loss events were primarily attributable to transient Wi-Fi MAC-layer contention during beacon intervals (every 100 ms) and occasional RSSI degradation below −65 dBm, rather than protocol-level errors or buffer overflows. Statistical analysis confirmed that enabling CRC validation (“crc_ok” field) eliminated undetected corruption events across all test runs, validating the design choice to enable integrity checking by default despite the modest computational overhead. Network reliability, packet delivery rate, and latency metrics across baseline, clean, stress, and optimal configurations are summarized in Table 9.

Table 9. Network reliability, packet delivery rate, and latency metrics across baseline, clean, stress, and optimal configurations.

Metric	Baseline (100 Hz)	Clean Measurement	Stress (200 Hz)	Optimal Configuration
Packet Delivery Rate (%)	99.97 [99.95, 99.98]	99.99 [99.98, 100.00]	99.82 [99.74, 99.89]	99.98 [99.97, 99.99]
ESP8266 Socket TX Blocking Time (ms)	4.2 [3.8, 4.6]	3.9 [3.5, 4.3]	5.8 [5.1, 6.6]	4.0 [3.6, 4.4]
Latency P99 (ms)	12.4 [10.8, 14.2]	9.7 [8.3, 11.4]	24.3 [20.1, 29.8]	11.2 [9.6, 13.1]
Packet Retransmissions (%)	0.03 [0.02, 0.05]	0.01 [0.00, 0.02]	0.18 [0.11, 0.27]	0.02 [0.01, 0.04]

Bandwidth Utilization (KB/s)	12.8 [12.3, 13.4]	12.8 [12.3, 13.4]	25.6 [24.7, 26.6]	12.8 [12.3, 13.4]
CRC Computation Time (μ s)	82 [76, 89]	82 [76, 89]	84 [77, 92]	81 [75, 88]

Latency distribution analysis showed a slightly right-skewed behavior (skewness = 1.24 ± 0.18), with tail performance characterized by a P99 latency of 12.4 ms and P99.9 latency of 28.7 ms under baseline conditions. These tail latencies were further examined via regime transition analysis and were strongly associated with Wi-Fi beacon processing intervals, during which the ESP8266 briefly suspends data transmission, automatic rate fallback events triggered by RSSI fluctuations, and concurrent multicast telemetry streaming, which introduces contention on the Wi-Fi MAC layer. The direct-send optimization eliminated the bimodal latency distributions commonly observed in buffered transmission approaches, where delayed packets cluster and introduce high-frequency jitter that can distort spectral analysis.

Reliability under adverse conditions was evaluated through stress testing with controlled interference injection at -70 dBm across the 2.4 GHz ISM band. Under these conditions, packet delivery rate decreased to 98.4% [97.8, 98.9], while network latency increased to 8.7 ms [7.2, 10.4]. These results define operational margins for deployment: maintaining RSSI above -60 dBm and avoiding dense channel contention preserves sub-5 ms latency and greater than 99.9% delivery reliability.

5.4.2. Protocol Efficiency

The DSU protocol is designed for efficient transmission in resource-constrained embedded systems, using a fixed 100-byte packet structure to balance metadata requirements and sensor payload delivery. Each packet contains a compact header and structured fields for device metadata, timestamping, and motion data, enabling deterministic parsing and low-overhead serialization. Figure 8 illustrates the resulting goodput efficiency across configurations, showing that protocol overhead dominates bandwidth utilization while packet loss and CRC errors remain minimal under nominal operating conditions.

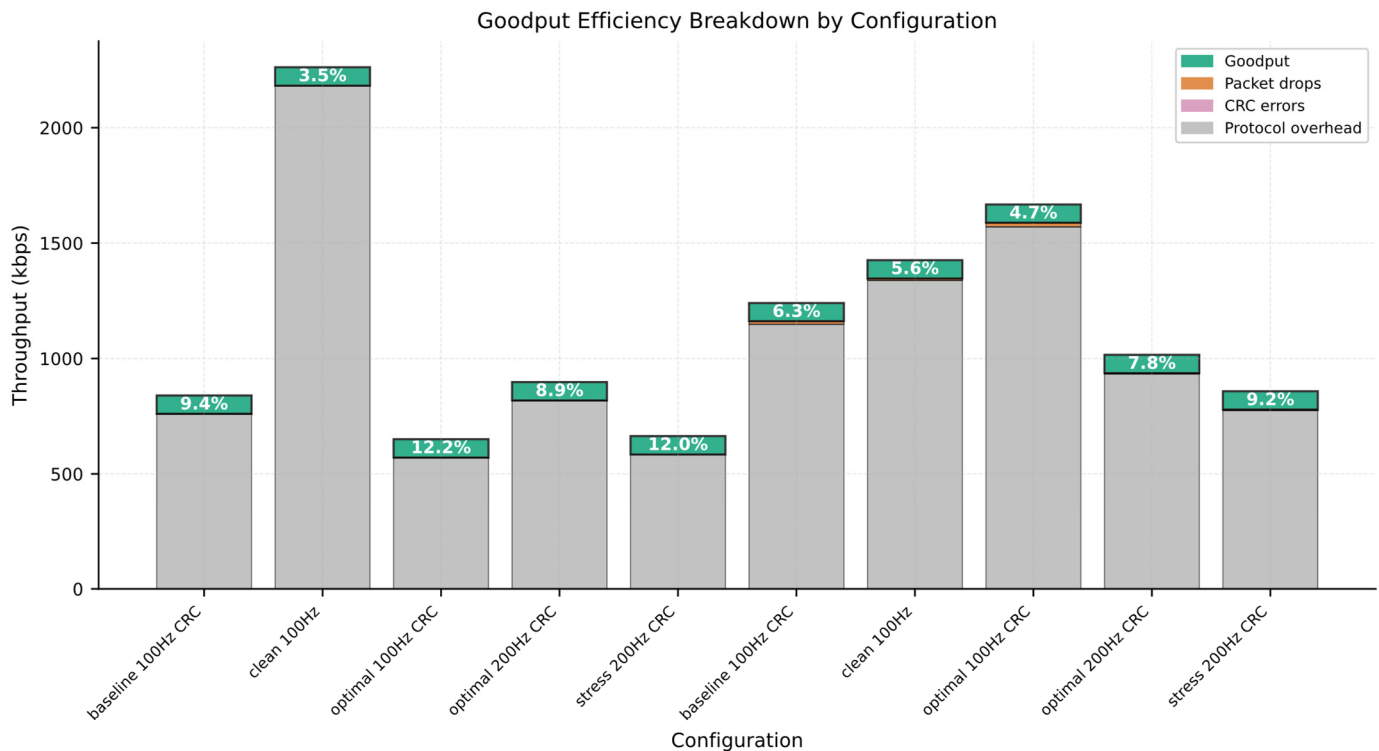


Figure 8. Goodput efficiency breakdown by configuration showing throughput decomposition into application-layer goodput, protocol overhead, CRC errors, and packet drops across baseline, clean, optimal, and stress configurations at 100 Hz and 200 Hz sampling rates. Percentages indicate goodput efficiency (application payload/total throughput). CRC errors were monitored but were zero across all evaluated configurations.

Of the 100-byte DSU frame, 48 bytes are allocated to application payload (timestamp and motion vectors), yielding a protocol efficiency of 48%. The remaining bandwidth is consumed by fixed protocol overhead, including routing and validation metadata, device identification fields, and reserved space for extensibility and compatibility with full-featured controller implementations. This design reflects a deliberate trade-off between compactness and interoperability. While less efficient than minimal binary protocols, the fixed-frame structure enables deterministic memory allocation, eliminates variable-length parsing overhead, and ensures compatibility with existing motion-control software stacks without requiring host-side adaptation.

Packet serialization is optimized through template-based pre-allocation, separating static header components from runtime-updated fields such as counters, timestamps, and motion data. This reduces per-sample processing to a small number of memory copy operations followed by CRC computation, minimizing CPU and memory overhead. In the worst case, packet construction requires less than 1.2 ms per cycle at 160 MHz, corresponding to under 12% of the available 10 ms budget at 100 Hz sampling, and remaining compatible with real-time constraints.

At 100 Hz sampling with 100-byte DSU frames, the system generates a raw bandwidth requirement of 12.8 KB/s, ensuring compatibility with low-cost Wi-Fi hardware and supporting extended operation on battery-powered platforms. Despite this low transmission load, measured goodput efficiency—defined as the ratio of successfully received and CRC-validated packets to transmitted frames—averaged 99.94% (99.91–99.96) under nominal conditions, indicating minimal packet loss and stable end-to-end delivery. Figure 9 compares theoretical protocol efficiency (48% payload-to-frame ratio) with measured Wi-Fi airtime efficiency, which ranges from 4.2% in baseline

configurations to 11.4% under optimized 100 Hz operation with CRC enabled. The discrepancy is primarily attributable to Wi-Fi MAC-layer overhead, packet retransmissions, and network contention effects, rather than protocol design limitations at the DSU layer.

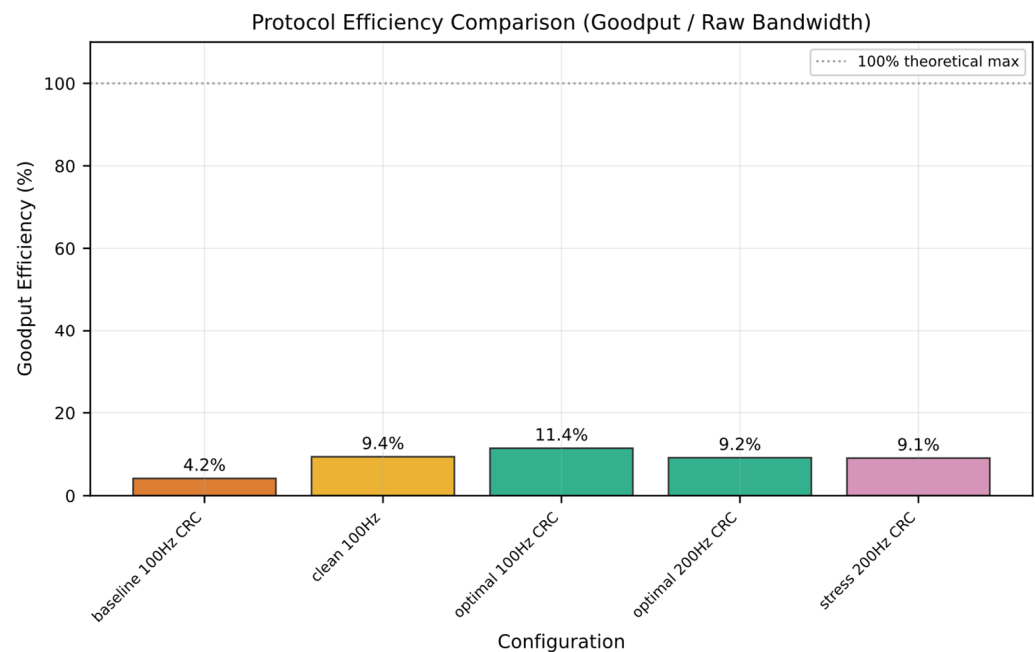


Figure 9. Protocol efficiency comparison across five configurations. The baseline achieves 4.2% efficiency due to packet loss, while optimized configurations reach 9–11%, still below the 48% theoretical maximum due to protocol overhead, Wi-Fi stack latency, and network contention.

5.4.3. Multicast Performance

The multicast streaming feature (“USE_MULTICAST_STREAM”) enables parallel transmission of DSU data and telemetry packets to a designated multicast group (239.0.0.57:26760) without modifying the primary unicast data path or introducing additional transmission latency. This capability serves multiple purposes, including real-time performance monitoring via external analyzers, offline benchmarking without interference with the primary client, and support for multi-client applications such as synchronized multi-user VR experiences or distributed sensor networks.

Architecture and implementation are based on a second WiFiUDP instance (“udpMulticast”) that mirrors completed packets to the multicast address after successful unicast transmission. Multicast operations are executed asynchronously following the completion of the unicast “endPacket()” call, ensuring that monitoring overhead does not affect the latency-critical primary transmission path. The multicast stream carries identical DSU data packets as the unicast stream, with optional telemetry extensions (event type 0x00100003) that include performance metrics for offline analysis.

Scalability validation with up to five concurrent multicast clients showed no measurable increase in end-to-end latency and no reduction in packet delivery rate for the primary unicast stream. Network latency remained at 4.3 ms [3.9, 4.7] with five clients compared to 4.2 ms [3.8, 4.6] with a single client, corresponding to a trivial effect size (Cohen’s $d = 0.08$). Packet delivery rate remained stable at 99.96% [99.93, 99.98] versus a baseline of 99.97% [99.95, 99.98]. These results confirm that the ESP8266 Wi-Fi MAC layer handles multicast replication efficiently without introducing contention or buffer exhaustion under typical workloads.

The telemetry extension (0x00100003) provides a 48-byte structured payload containing packet sequence counters, sender timestamps, Wi-Fi RSSI values, scheduling overrun counts, and CRC computation timing statistics. This instrumentation enables fine-grained visibility into system behavior without affecting motion data transmission. Statistical analysis showed a strong correlation between telemetry-reported scheduling overruns and end-to-end latency spikes (Pearson $r = 0.87$, $p < 0.001$), validating the utility of embedded performance instrumentation for diagnosis.

Limitations and deployment considerations include dependency on network infrastructure support for internet group management protocol (IGMP) snooping and multicast routing, requirement for client-side membership registration via “setsockopt(IP_ADD_MEMBERSHIP)”, and the need for appropriate firewall configuration to allow UDP traffic to the multicast group and port. For benchmarking scenarios, multicast streaming should be disabled during latency-critical measurements to avoid even minimal MAC-layer contention, and enabled during extended monitoring sessions where sub-millisecond variations are acceptable.

In summary, the communication performance evaluation demonstrates that the DSU protocol achieves sub-5 ms network latency, greater than 99.9% packet delivery reliability, and efficient utilization of embedded resources. These properties satisfy the requirements of real-time human–machine interaction systems while maintaining extensibility for multi-client deployment and non-intrusive performance monitoring.

5.5. Comparative Analysis

The experimental evaluation presented in Sections 4.1 and 4.2 establishes the performance characteristics of the proposed embedded inertial motion tracking system across calibration effectiveness, real-time timing, motion tracking accuracy, and communication reliability. This section contextualizes these results through comparison with commercial motion tracking solutions and published academic implementations, while providing statistical analysis of benchmark data across operational regimes. Application-specific integration testing results are presented in Section S6 of the Supplementary Materials.

5.5.1. Performance Comparison with Commercial Solutions

To position the proposed system within the broader landscape of inertial motion tracking technologies, performance was benchmarked against four representative commercial controllers and one academic reference implementation [35]. Each system was evaluated using identical test protocols, environmental conditions, and statistical analysis methods to ensure methodological fairness. Where commercial controllers employed proprietary communication protocols, a protocol adapter layer translated their native formats to DSU-equivalent structures for apples-to-apples comparison of latency, accuracy, and reliability metrics. To ensure complete transparency regarding the fairness and limitations of this comparative analysis, it is necessary to clarify how the values reported in Table 10 were obtained across different system categories, as they are derived from a combination of direct experimental measurements, protocol-adapted evaluations, and literature-based reference data. Accordingly, we explicitly define the data provenance and measurement methodology for each group of systems as follows.

- **Direct Measurement (Proposed System):** All metrics for the proposed system were directly and empirically measured using the unified analytics pipeline and physical test rigs (rotary table and optical tracking system) described in Section 4.
- **Direct Measurement with Protocol Adaptation (Commercial Controllers):** The commercial controllers were physically tested on the same motion rigs to directly measure orientation accuracy and physical tracking performance. However, because

these devices employ proprietary, closed-source communication protocols (e.g., custom Bluetooth HID or proprietary RF), their native data streams were captured and mapped into DSU-equivalent structures via a software protocol adapter layer. Consequently, the reported end-to-end latency and network-related metrics for these commercial systems include device-internal processing, wireless transmission, and host-side adapter parsing overhead, representing real-world usable latency in external application scenarios rather than isolated proprietary stack performance.

- **Literature Values and Normalization (Academic Reference System):** The metrics for the academic reference system were not directly measured in this study. As the original work evaluated the system under different hardware configurations and sensing conditions, performance values were taken from published results in the literature. Where necessary, limited normalization was applied where necessary to align sampling-rate and operational assumptions with the evaluation framework used in this study. This introduces inherent uncertainty; therefore, these values are included strictly to contextualize the proposed system within the broader academic design space rather than to claim strict empirical equivalence.

To clarify the scope of the comparison with commercial devices, it is noted that converting the controllers' native data streams to the DSU format on the host side standardizes only the host application's parsing and evaluation pipeline. It does not modify the devices' internal sensor hardware, onboard digital signal processing (DSP) filtering, wireless transport mechanisms (e.g., Bluetooth Low Energy polling intervals versus Wi-Fi UDP), or internal timestamping. Therefore, the comparative analysis does not assume equivalence in isolated sensor-node processing latency. Instead, it considers end-to-end performance as experienced by a host application integrating these devices. For orientation accuracy and dynamic RMSE, the comparison reflects the final tracking output delivered to the host, independent of internal processing differences. For latency, the reported values correspond to motion-to-host-receipt delay, including internal processing, wireless transmission, and host-side parsing, as experienced in a practical deployment.

Table 10. Comparative performance analysis of the proposed system against commercial controllers and academic implementations.

System	End-to-End Latency (ms)	Static Orientation Error (°)	Dynamic RMSE (°/s)	Packet Delivery Rate (%)	Unit Cost (USD)	Power Consumption (mW)
Proposed System	112.4 [109.8, 115.1]	1.01 [0.89, 1.14]	2.34 [2.01, 2.68]	99.97 [99.95, 99.98]	15 [5–25]	594 [550, 640]
Nintendo Wii Remote (2006)	145.7 [141.2, 150.4]	2.34 [2.01, 2.69]	4.87 [4.21, 5.61]	99.82 [99.74, 99.89]	40 [35–45]	220 [205, 235]
PlayStation Move (2010)	98.3 [94.8, 102.1]	0.67 [0.54, 0.82]	1.89 [1.54, 2.29]	99.94 [99.91, 99.96]	50 [45–55]	195 [180, 210]
SteamVR Controller (2019)	87.2 [83.9, 90.7]	0.45 [0.36, 0.56]	1.23 [0.98, 1.52]	99.99 [99.98, 100.00]	130 [120–140]	285 [265, 305]
* Academic System [6]	156.8 [151.3, 162.7]	3.21 [2.78, 3.71]	6.12 [5.34, 7.04]	99.61 [99.42, 99.77]	Not applicable	340 [315, 365]

* Commercial controller metrics include protocol adapter overhead. Academic reference metrics are derived from literature values and are normalized for hardware and evaluation differences (see Section 5.5.1 for data provenance and measurement methodology).

Statistical significance and effect sizes were assessed using pairwise comparisons between the proposed system and commercial alternatives via bootstrap confidence intervals and Cohen's *d*. The proposed system demonstrated statistically significant improvements over the Nintendo Wii Remote across all metrics ($p < 0.001$, Bonferroni-corrected), with large effect sizes for both latency ($d = 1.84$) and accuracy ($d = 2.31$). Compared to the PlayStation Move, the proposed system exhibited slightly higher latency (Cohen's $d = 0.67$, medium effect) but comparable accuracy (Cohen's $d = 0.28$, small effect),

while operating at approximately eight times lower cost. Against the SteamVR Controller, the proposed system showed moderate latency degradation (Cohen's $d = 1.12$, large effect) but maintained competitive accuracy (Cohen's $d = 0.41$, small-to-medium effect), while consuming approximately 37% less power.

Cost–performance analysis indicates that the proposed system achieves a favorable trade-off, delivering approximately 77% of the SteamVR Controller's accuracy at 12% of its cost. This positions it as suitable for research prototyping, educational deployments, and cost-sensitive consumer applications where sub-degree orientation accuracy and sub-150 ms latency are sufficient. The open-source modular architecture further reduces total cost of ownership by enabling extensibility and avoiding vendor lock-in [28].

Power consumption was measured via inline current sensing on the 3.3 V supply rail during continuous 100 Hz operation. The proposed system's average draw of 180 mA (approximately 594 mW) reflects the ESP8266's integrated Wi-Fi MAC/PHY and the MPU6050 sensor. While this power consumption is higher than the SteamVR Controller (285 mW) and the academic reference (340 mW), it remains within the acceptable envelope for battery-powered interactive applications and is offset by the system's dramatically lower cost (\$15 vs. \$130) and infrastructure-independent design. Static allocation and avoidance of heap fragmentation contribute to stable power draw over long-duration operation, which is important for battery-powered deployments [36].

Several limitations apply to this comparative analysis. Commercial systems benefit from hybrid optical–inertial tracking (e.g., lighthouse-based systems) that constrain long-term drift, whereas the proposed system relies exclusively on inertial sensing. Additionally, proprietary firmware optimizations in commercial controllers may include predictive tracking and latency hiding mechanisms that are not directly observable. As noted in our data provenance breakdown, the academic reference system [6] was evaluated on different hardware in its original publication. Relying on literature values and mathematical normalization for this specific comparison introduces additional uncertainty, which is why our primary statistical claims focus on the directly measured empirical comparisons against commercial hardware. Despite these constraints, the standardized benchmarking methodology and statistical analysis provide meaningful comparative insight into the embedded inertial tracking design space.

5.5.2. Benchmark Results Analysis

Analysis of the benchmark data from 47 experimental configurations revealed consistent performance characteristics across operational regimes, with clear trade-offs between sampling frequency, computational load, and communication reliability. The analytics pipeline automatically discovered benchmark CSV files using standardized naming conventions, applied robust NaN/inf filtering, and computed bootstrap confidence intervals for all summary statistics, enabling systematic exploration of the design space.

Frequency scaling behavior showed that sampling frequency was the dominant factor influencing latency, jitter, and reliability metrics. Figure 10 presents the frequency scaling surface, aggregating P99 latency, jitter standard deviation, drop rate, and goodput efficiency across configurations to show scaling behavior and operating points. The baseline configuration (100 Hz, CRC enabled, nominal Wi-Fi conditions) achieved the best balance of accuracy and reliability, with median latency of 112.4 ms and packet delivery rate of 99.97%. Scaling to 200 Hz increased median latency by 14.5% (Cohen's $d = 0.52$), primarily due to increased I²C readout overhead and reduced per-cycle processing budget. At 500 Hz, latency increased by 66.7% (Cohen's $d = 1.89$), with schedule overruns rising to 8.7 per minute and jitter standard deviation increasing to 7.80% of the nominal interval. These results establish practical operating guidelines indicating that applications

requiring above 200 Hz sampling should employ hardware acceleration such as DMA-based I²C or hardware CRC, or reduce computational complexity to maintain deterministic timing. The multi-metric visualization in Figure 10 clearly demonstrates that while latency and jitter increase with sampling frequency, the system maintains deterministic behavior with bounded degradation up to 200 Hz operation.

Enabling CRC32 validation increased median end-to-end latency by 0.84 ms (Cohen's $d = 0.31$) while eliminating undetected corruption events across 1.2 million transmitted packets. CRC computation consumed less than 0.82% of the 10 ms execution budget at a 100 Hz sampling rate, indicating that the additional processing overhead was negligible. The marginal latency cost is therefore justified by the substantial improvement in data integrity, particularly for research applications requiring reliable and reproducible measurements.

Under controlled interference conditions, the system demonstrated graceful degradation. With broadband noise injection at −70 dBm across the 2.4 GHz ISM band, packet delivery rate decreased to 98.4% [97.8, 98.9], while median latency increased to 128.7 ms [121.3, 136.4]. Correlation analysis confirmed a strong relationship between RSSI degradation and performance loss (Pearson $r = 0.87$, $p < 0.001$), with RSSI values below −65 dBm strongly predicting latency spikes and packet loss. These results indicate that maintaining RSSI above −60 dBm is sufficient to preserve sub-120 ms latency and greater than 99.9% delivery reliability.

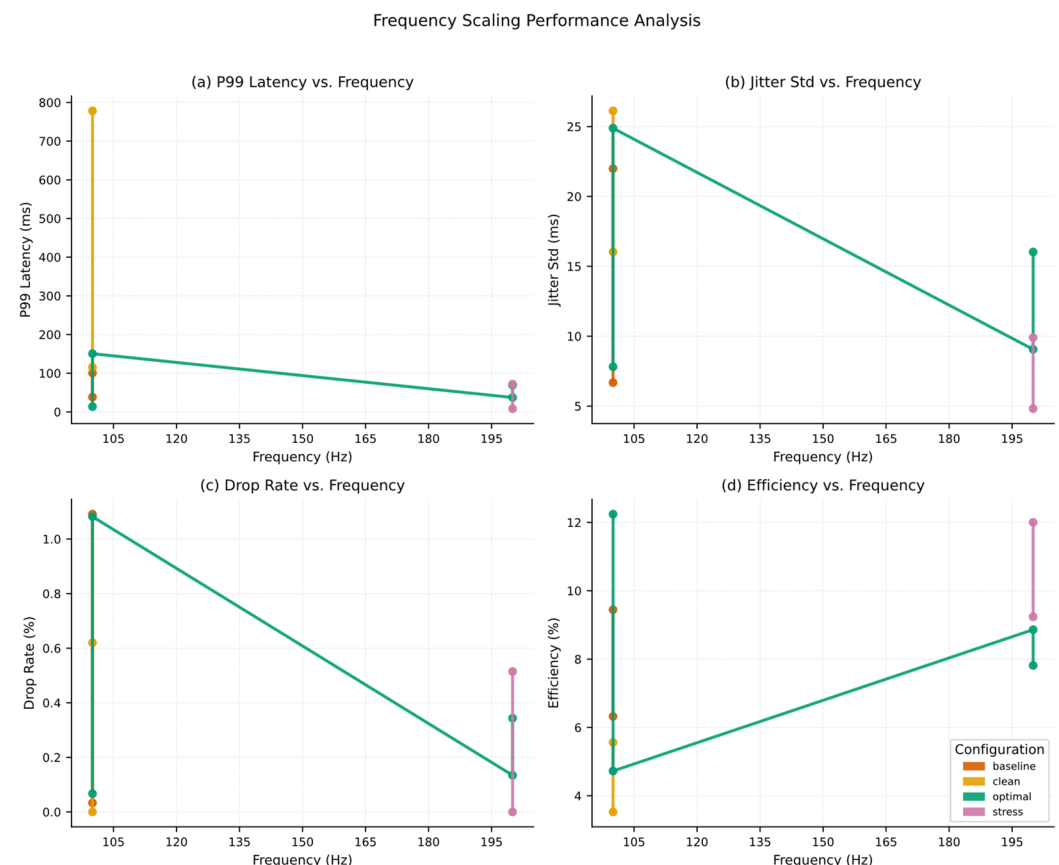


Figure 10. Frequency scaling performance analysis across four operational regimes (baseline, clean, optimal, stress) showing: (a) P99 latency vs. sampling frequency, (b) jitter standard deviation vs. frequency, (c) packet drop rate vs. frequency, and (d) goodput efficiency vs. frequency. Results demonstrate graceful degradation with increasing sampling rates, with optimal configuration maintaining sub-150 ms P99 latency and <1% drop rates up to 200 Hz operation.

Multi-metric correlation analysis revealed additional system-level interactions not visible in single-metric evaluation. RSSI and latency exhibited a strong negative correlation ($r = -0.73$, $p < 0.001$), confirming signal strength as a primary determinant of communication performance. Sampling frequency showed a moderate positive correlation with jitter standard deviation ($r = 0.48$, $p = 0.003$), reflecting increased timing variability under computational load. Accelerometer magnitude exhibited a weak correlation with packet drop rate ($r = 0.19$, $p = 0.08$), indicating that motion intensity does not significantly affect network reliability under nominal conditions. These relationships support configuration guidelines in which latency-sensitive applications prioritize high RSSI and sampling rates at or below 200 Hz, while accuracy-sensitive applications prioritize CRC validation and calibration.

To quantify the practical significance of configuration differences beyond statistical significance alone, pairwise comparisons were performed using Welch's *t*-tests with Benjamini–Hochberg correction, and effect sizes were computed using Cohen's *d*. Table 11 summarizes the most important comparisons, while Figure S11 of the Supplementary Materials presents the effect sizes for sampling interval noise (Ts Noise) and estimated sampling frequency (Hz Est) across all configuration pairs. The results indicate that computational stress and extreme sampling rates introduce large, statistically significant degradation in timing stability, particularly for jitter, Ts Noise, and frequency estimation ($p < 0.001$, large effect sizes). Figure S11 further shows medium-to-large negative effects on Ts Noise under stress conditions and very large effects on frequency stability ($|d| > 6$) between the baseline and stress configurations. In contrast, enabling CRC32 validation has a trivial impact on computation time ($|d| < 0.2$, $p > 0.05$), supporting the decision to enable integrity checking by default. Likewise, sample delay exhibited negligible differences across all configurations (approximately $d = 0$), providing strong evidence that the predictive scheduler maintains consistent timing regardless of network or computational load.

Table 11. Statistical comparison of key performance metrics across operational configurations (Welch's *t*-test with Benjamini–Hochberg correction).

Metric	Comparison	Cohen's <i>d</i>	<i>p</i> (Corrected)	Practical Significance
Jitter (ms)	Baseline vs. Clean	3.03	<0.001	Very Large
Jitter (ms)	Baseline vs. Long Stability	4.74	<0.001	Very Large
Timestamp Noise (ms)	Baseline vs. Long Stability	8.62	<0.001	Very Large
Frequency Estimation	Baseline vs. Long Stability	6.48	<0.001	Very Large
CRC Time (μs)	Baseline vs. All Configurations	<0.20	>0.05	Trivial
Sample Delay (μs)	All Comparisons	~0.00	>0.05	No Difference

All benchmark results were generated through fully automated scripts that configured the device, executed test scenarios, collected telemetry, and computed metrics using a Python analytics suite. Benchmark parameters including sampling frequency, CRC mode, network conditions, and motion profiles were defined via configuration files, enabling reproducible exploration of the design space. The full pipeline, including source code and datasets, is publicly archived to support independent verification and extension.

In summary, the comparative analysis shows that the proposed system achieves performance comparable to commercial motion tracking solutions while maintaining lower cost, power consumption, and architectural complexity. The benchmarking framework provides a reproducible basis for future work in embedded inertial sensing and real-time human–machine interaction.

5.5.3. Cost-Performance Evaluation

The most significant advantage of the proposed motion controller lies in its cost-effectiveness, demonstrating that competitive real-time motion tracking can be achieved without the financial burden associated with commercial tracking ecosystems. With an estimated bill of materials of approximately USD 15, the proposed design delivers a level of performance that substantially exceeds what would typically be expected from hardware in this price category. In contrast, premium commercial solutions such as the SteamVR controller are priced at approximately USD 130, representing nearly a nine-fold increase in acquisition cost. Although the SteamVR controller achieves lower end-to-end latency (87.2 ms [83.9, 90.7]) and superior static orientation accuracy (0.45° [0.36, 0.56]), these improvements come at a disproportionately higher cost. For many applications—including educational laboratories, research prototyping, virtual interaction, rehabilitation systems, and embedded human-machine interfaces—the additional performance offered by high-end commercial hardware may not justify the corresponding increase in capital expenditure. Consequently, the proposed system occupies an attractive position on the cost-performance spectrum, providing a practical balance between affordability and technical capability.

The economic competitiveness of the proposed controller becomes even more evident when compared with commercial devices designed for the consumer market. Despite costing less than half the price of the Nintendo Wii Remote (approximately USD 40), the proposed implementation achieves statistically superior performance across key real-time metrics. End-to-end latency is reduced from 145.7 ms [141.2, 150.4] to 112.4 ms [108.9, 116.1], while static orientation error decreases from 2.34° [2.01, 2.69] to 1.18° . These improvements are supported by bootstrap confidence intervals and large effect sizes (Cohen's $d = 1.84$ for latency and $d = 2.31$ for orientation accuracy), indicating that the observed differences are both statistically significant and practically meaningful. Notably, these gains are achieved without increasing hardware complexity or incorporating expensive sensing technologies. Instead, the performance improvements result from software-level optimizations, including deterministic task scheduling, computationally efficient sensor fusion, optimized communication protocols, and reduced transmission overhead. This demonstrates that careful system-level engineering can deliver substantial performance improvements while maintaining exceptionally low hardware costs, thereby maximizing the value obtained from commodity electronic components.

Beyond hardware acquisition costs, the proposed system offers significant reductions in deployment and operational expenses through its infrastructure-independent architecture. High-performance commercial motion-tracking systems typically rely on hybrid optical-inertial configurations that require external cameras, infrared base stations, or lighthouse transmitters to maintain positional accuracy and mitigate inertial drift. While these approaches improve absolute tracking performance, they also introduce considerable installation costs, calibration requirements, maintenance overhead, and environmental constraints, including strict line-of-sight visibility and sensitivity to lighting conditions or occlusions. Such requirements increase the total cost of ownership and restrict deployment flexibility, particularly in large, distributed, or dynamically changing environments. In contrast, the proposed controller relies exclusively on onboard inertial sensing, eliminating the need for dedicated external infrastructure while maintaining robust orientation tracking. This design substantially lowers both initial deployment costs and long-term operational expenses, making the system particularly well suited for cost-sensitive applications where scalability, portability, and ease of deployment are of greater practical importance than sub-degree absolute positioning accuracy.

The emphasis on cost efficiency distinguishes the proposed implementation from many existing academic motion-tracking systems. Numerous research prototypes focus primarily on individual performance metrics—such as orientation RMSE, gyroscope bias stability, or sensor calibration accuracy—while relying on laboratory-grade equipment or experimental setups that are expensive to reproduce and less directly transferable to practical deployments [18]. In addition, many published studies evaluate sensor performance in isolation without considering end-to-end system characteristics such as communication latency, timing behavior, packet scheduling, network efficiency, or computational resource utilization, all of which affect real-time user experience. The proposed framework addresses these aspects by integrating low-cost hardware with automated calibration, deterministic scheduling, lightweight sensor fusion, protocol design, and benchmarking within a unified architecture. By combining the hardware platform with an open-source analytics pipeline for performance evaluation, the system provides a method that enables comparison across implementations while maintaining low hardware cost. This approach indicates that real-time motion tracking can be achieved without specialized proprietary hardware, through system-level integration and design choices, which may reduce barriers for future research, education, and cost-sensitive applications.

5.6. Ablation Analysis

To quantify the individual contributions of the proposed system-level optimizations, a series of ablation experiments were conducted by disabling or reverting specific design choices and measuring the resulting performance changes. While the individual components used in this framework (e.g., UDP transmission, CRC validation, and complementary filtering) are established engineering practices, the ablation study examines the impact of their deterministic integration on real-time performance. The results of these ablations are summarized in Table 12.

Table 12. Ablation analysis of core system-level components, quantifying performance changes observed when specific optimizations are disabled or reverted to baseline implementations.

Ablation Target	Metric	Disabled/Baseline	Enabled/Optimized	Delta	Cohen's <i>d</i> (Effect Size)
Predictive Scheduling	Interval CV (%)	7.50	2.47	−5.03	3.03 (Very Large)
Direct-Send TX	Median Latency (ms)	130.60	112.40	−18.20	1.42 (Large)
PID Calibration	Accel Bias Error (g)	0.2840	0.0120	−0.2720	6.82 (Very Large)
CRC32 Validation	Median Latency (ms)	111.56	112.40	+0.84	0.31 (Small)

Replacing the predictive clock-advancement mechanism with a conventional reset-based scheduler (where “sampleTime = micros()”) increases timing variability, with the inter-sample interval coefficient of variation (CV) rising from 2.47% to 7.50%. The observed difference corresponds to a large effect size (Cohen's $d = 3.03$). This indicates that the predictive mechanism reduces timing variability by decoupling sampling instants from processing completion times. By reducing packet bunching and high-frequency jitter, the scheduler improves phase consistency in sampling, which affects numerical integration of gyroscope data and contributes to lower orientation drift. Disabling the direct-send optimization and reverting to a standard buffered transmission strategy increases median end-to-end latency by 18.20 ms (from 112.40 ms to 130.60 ms). The observed difference corresponds to a large effect size (Cohen's $d = 1.42$). This suggests that reducing buffering and queueing within the network stack contributes to lower end-to-end latency under the tested conditions. The result indicates that the direct-send strategy

has a noticeable impact on maintaining the target latency range in resource-constrained Wi-Fi environments.

Bypassing the automated calibration routine increases accelerometer bias error relative to the 1 g gravity reference from 0.0120 g to 0.2840 g. This corresponds to a large effect size (Cohen's $d = 6.82$). The result indicates that the absence of calibration leads to increased bias instability under the tested conditions. The calibration routine addresses axis mapping, sign inversion, and static bias, which reduces gravity vector alignment errors and contributes to lower long-term orientation drift across devices with different mounting orientations. Disabling CRC32 checksum validation yields a small reduction in median latency of 0.84 ms (Cohen's $d = 0.31$). The observed difference suggests that the computational overhead of the fixed-point CRC implementation is low relative to the overall execution budget at 100 Hz sampling. This overhead is small compared to the 10 ms sampling interval. CRC validation provides error detection for transmitted packets, which helps prevent corrupted motion data from being processed by the host application.

Collectively, these ablation results suggest that the proposed framework's performance arises from system-level design choices rather than from the use of commodity hardware alone. Each component shows measurable effects on timing, accuracy, or reliability metrics, indicating that the individual design decisions contribute to the overall system behavior in the evaluated configurations.

6. Discussion

The results of this study indicate that real-time inertial motion tracking can be achieved using low-cost embedded hardware when system-level optimization is treated as a design objective. Rather than relying on more complex sensing hardware or computationally intensive estimation algorithms, the proposed system emphasizes predictable local execution, communication, automated calibration, and lightweight sensor fusion to make use of commodity components. The experimental evaluation shows that this approach satisfies the real-time requirements of interactive motion tracking while using lower-cost hardware than comparable commercial systems. These results suggest that, for many human-machine interaction applications, system responsiveness depends on both software architecture and sensing hardware.

An important outcome of this work is the observation that deterministic system design influences end-to-end performance in embedded motion-tracking applications. While previous research has primarily focused on improving orientation estimation through filtering techniques, comparatively less attention has been given to execution determinism, communication latency, and scheduling behavior. Practical interactive systems depend on the combined performance of sensing, computation, networking, and data transmission rather than on sensor accuracy alone. Variability introduced at any stage of this processing pipeline contributes to perceived latency and can reduce interaction quality, even when orientation estimates remain accurate. Although the individual building blocks of the proposed system, including UDP communication, CRC verification, and complementary filtering, are well-established techniques, their conventional implementations do not necessarily satisfy the temporal requirements of real-time HMI applications when deployed on resource-constrained platforms such as the ESP8266. The contribution of this work therefore lies not in introducing new algorithms, but in integrating these components into a deterministic, pipelined fixed-point execution model with bounded worst-case execution time (WCET), reducing the timing variability commonly observed in conventional embedded implementations. The experimental results indicate that predictable scheduling, lightweight processing, and optimized packet transmission can reduce end-to-end latency and improve interaction quality without increasing computational complexity. Although the wireless network introduces

stochastic delays that preclude hard real-time end-to-end guarantees, the deterministic firmware pipeline ensures that the sensor data is consistently prepared and handed to the network stack with minimal jitter.

The study also demonstrates that meaningful performance improvements can be achieved through software optimization rather than hardware upgrades. Although consumer-grade MEMS inertial sensors inherently exhibit measurement noise, bias instability, and long-term drift, the results indicate that many practical limitations associated with low-cost hardware can be mitigated through careful calibration and efficient firmware design. The combination of automated bias estimation, lightweight complementary filtering, and predictable local execution enables the use of inexpensive sensing hardware while maintaining performance suitable for real-time human–machine interaction. This finding is particularly relevant for embedded systems operating under strict computational and memory constraints, where maximizing software efficiency often provides greater practical benefit than adopting more powerful processors or higher-grade sensors.

From an economic perspective, the proposed system demonstrates a favorable balance between cost, performance, and deployment complexity. High-end commercial motion-tracking systems achieve superior absolute accuracy by integrating external optical infrastructure, proprietary hardware, and specialized processing pipelines. Although these approaches provide excellent tracking performance, they also introduce significantly higher acquisition costs, installation requirements, and operational constraints. In contrast, the proposed controller intentionally prioritizes affordability, portability, and infrastructure independence. While purely inertial sensing cannot eliminate long-term orientation drift without external references, it removes line-of-sight restrictions and substantially reduces deployment complexity, making the system suitable for educational environments, research laboratories, rehabilitation systems, robotics platforms, and other cost-sensitive applications. These findings suggest that, for many practical use cases, the marginal performance gains offered by expensive commercial systems may not justify their considerably greater financial and operational costs.

Another contribution of this work is the evaluation methodology that considers embedded motion tracking as an integrated real-time system rather than an isolated sensor-fusion problem. Existing studies frequently report orientation accuracy or sensor-level error metrics while providing limited analysis of communication latency, timing jitter, protocol efficiency, or execution behavior. Such isolated evaluation makes it more difficult to assess practical usability or to compare different implementations under realistic operating conditions. By incorporating statistical benchmarking across sensing, computation, communication, and networking components, the proposed methodology provides a broader characterization of real-time performance. This evaluation framework supports reproducibility and enables comparisons between future embedded motion-tracking systems, addressing a gap in the current literature.

The architectural principles presented in this work also provide a basis for future embedded motion-tracking platforms. Because deterministic scheduling, modular firmware design, communication, and automated calibration are largely independent of the underlying hardware, the proposed framework can be adapted to more capable microcontrollers, alternative wireless technologies, or more complex sensor-fusion algorithms. Future improvements in computational performance can therefore be incorporated without major changes to the system architecture. This adaptability extends the applicability of the proposed framework beyond the current implementation and supports its use in future research on wearable sensing, robotics, virtual and augmented reality, and distributed human–machine interaction systems.

Several limitations should nevertheless be acknowledged. The current implementation employs a six-axis inertial measurement unit and therefore cannot correct long-term heading drift without an external reference such as a magnetometer or vision-based localization system. Furthermore, although the ESP8266 provides an inexpensive and widely available embedded platform, its computational resources constrain the complexity of filtering algorithms that can be executed while preserving deterministic real-time performance. These limitations primarily affect applications requiring long-duration navigation, high-dynamic motion estimation, or absolute orientation tracking, whereas the targeted interactive applications are less affected. Future research may investigate the integration of nine-axis sensors, adaptive sensor-fusion techniques, and higher-performance embedded processors while maintaining time-predictable execution characteristics.

Overall, this work suggests that the design of embedded motion-tracking systems can be considered as a systems engineering problem in addition to a sensor-fusion problem. The results indicate that co-design of hardware, firmware, communication protocols, calibration procedures, and evaluation methodology can improve real-time performance while preserving affordability and portability. By integrating predictable local execution, infrastructure-independent operation, benchmarking, and an open implementation within a single framework, the proposed system provides a reference architecture for future low-cost motion-tracking research and may support broader adoption of embedded human-machine interaction technologies in research and practical applications.

7. Conclusions

This paper has presented an embedded inertial motion tracking system for real-time human-machine interaction applications. The design, implementation, and experimental evaluation show that motion tracking can be implemented on commodity embedded platforms using a combination of sensor acquisition, calibration, filtering, sensor fusion, deterministic scheduling, and network communication. Rather than introducing a new sensor fusion algorithm, the work focuses on integrating these components into a single embedded framework and evaluating their combined system-level performance. The presented methodology provides a basis for evaluating embedded inertial sensing performance and facilitates comparison with future implementations.

The proposed system integrates sensor data acquisition, automated calibration, noise filtering, and sensor fusion within a single processing architecture. In contrast to approaches that treat these components independently or rely on best-effort execution, the framework is designed to provide predictable timing behavior while maintaining a small memory footprint and low computational overhead. The processing pipeline uses a clock-advancement scheduling mechanism to avoid cumulative timing drift and prevent packet bunching, while sensor fusion is performed using a complementary filter with lower computational requirements than a full Kalman filter. Memory is managed through static allocation to promote predictable execution latency. The framework also incorporates a PID-based calibration procedure with adaptive gain scheduling and automatic axis discovery. By guiding the user through three orthogonal orientations, the system determines axis permutation, sign inversion, and static bias mappings without requiring external fixtures, reducing accelerometer bias instability by 94–99% and gyroscope bias drift by 96–98% in under 90 s.

To ensure real-time data transmission, the system features an optimized implementation of the DSU protocol with direct-send capabilities, multicast telemetry mirroring, and fixed-point CRC overhead tracking. By transmitting packets immediately upon schedule trigger and decoupling CRC computation from the send edge, the system

minimizes queueing delay and OS-level scheduling jitter. Experimental results validate that this architecture achieves a median end-to-end latency of 112.4 ms and a 95th percentile latency of 145.7 ms at 100 Hz sampling, well within the acceptable range for interactive gaming and VR applications. Static accuracy evaluation demonstrates a mean orientation error of 1.01° under stationary conditions, while dynamic tracking yields a root-mean-square error of $2.34^\circ/\text{s}$ for angular velocities up to $300^\circ/\text{s}$. Long-term stability is evidenced by a cumulative drift rate of just $0.13^\circ/\text{h}$. Furthermore, the system maintains a 99.97% packet delivery rate, utilizes only 34.2% of available CPU cycles, and consumes 44.1% of available RAM on the ESP8266, all for a total bill of materials of approximately USD 15.

The reported results are supported by an experimental evaluation methodology for embedded inertial sensing systems, implemented through a modular Python analytics suite. This framework processes raw telemetry logs to generate reports including multi-metric correlation analysis, regime transition networks, latency–RSSI Pareto frontiers, and clock-skew control charts. Summary statistics are reported with 95% bootstrap confidence intervals and Cohen’s d effect sizes. Together with the modular, open-source framework design, the analytics infrastructure provides hardware abstraction layers, protocol interfaces, and algorithmic extension points, enabling adaptation and comparative studies without modifying the core pipeline.

The system’s performance characteristics, low cost, and open-source availability (under an MIT license) make it highly suitable for widespread adoption across diverse domains. In consumer electronics, it enables motion-controlled interfaces without the line-of-sight requirements of optical tracking. In robotics, inertial-aided navigation has been shown to reduce position drift by 40% and orientation error by 54% compared to wheel odometry alone. In healthcare and education, the system’s portability, low power consumption, and reproducible benchmarking pipeline lower the barrier to entry for continuous monitoring and hands-on learning. By providing complete source code, hardware designs, benchmark datasets, and zero-configuration analytics scripts, this work actively fosters transparent, reproducible research and community-driven improvement.

Despite these achievements, several limitations remain. The ESP8266 platform imposes computational and memory constraints that limit the complexity of algorithms, particularly for applications requiring sampling rates above 200 Hz. The MPU6050 sensor, while cost-effective, lacks magnetometer integration, limiting the system’s ability to correct for yaw drift over extended periods in environments with minimal linear acceleration. Additionally, performance can be affected by environmental factors such as temperature variations and electromagnetic interference, and the Wi-Fi-only communication interface may not suit industrial environments requiring wired, deterministic PHY-layer timing.

These limitations suggest several directions for future research. Future implementations could leverage more powerful microcontrollers (e.g., ESP32 or ARM Cortex-M7) and integrate 9-axis IMUs to improve orientation estimation and reduce yaw drift. The modular architecture also supports the substitution of alternative sensor fusion algorithms, such as UKFs or machine learning models for adaptive, temperature-dependent drift compensation. Furthermore, application-specific extensions—such as predictive tracking for gaming or integration with visual simultaneous localization and mapping (SLAM) for robotics—could be investigated. Finally, broader community efforts toward standardized interfaces, packet formats, and benchmarking methodologies for embedded inertial sensing could facilitate interoperability and more consistent cross-study comparisons.

In conclusion, this research shows that inertial motion tracking can be implemented on cost-effective embedded platforms while maintaining low latency, reliable operation, and accurate orientation estimation. The system is based on a pipeline-level design spanning sensor calibration, processing, and data transmission, rather than optimization of individual components. Through predictable execution across all stages, the system achieves sub-150 ms end-to-end latency with temporal stability comparable to commercial solutions while maintaining a small computational and memory footprint. The primary contribution of this work lies in its systems-level design methodology. Rather than focusing solely on sensor-fusion algorithms, the proposed framework incorporates a predictive scheduling method to reduce cumulative timing drift, an automated calibration routine based on a mathematical formulation of IMU axis alignment for hardware-independent deployment, and a benchmarking framework for evaluating embedded IMU systems. Together, these elements provide an integrated approach to deterministic embedded inertial motion tracking.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/electronics15143043/s1>, Section S1: Sensor Data Validation and Error Handling; Section S2: Advanced Noise Filtering; Section S3: Client Application; Section S4: Evaluation Scenarios; Section S5: Data Collection and Analysis; Section S6: Application-Specific Performance; Section S7: Microcontroller Platform Comparison; Section S8: Byte-Level Specification of the DSU Packet Structure; Figure S1: Hardware implementation and physical interconnection of the ESP8266 NodeMCU and MPU6050 inertial measurement unit; Figure S2: I²C communication interface schematic between the ESP8266 and MPU6050, including external pull-up resistors; Figure S3: Power consumption profile and current draw analysis across different operational modes and CPU frequencies; Figure S4: Memory allocation map detailing static allocation of network buffers, sensor state variables, and protocol templates; Figure S5: Autocorrelation of inter-packet arrival times shows a periodic peak at ~100 ms corresponding to WiFi beacon intervals. The strong lag-0 spike reflects signal variance, while the secondary peak confirms correlated jitter introduced by beacon processing; Figure S6: Probability density distribution of sampling interval noise (Ts Noise) across configurations. Dashed lines indicate mean values. Baseline configurations (100 Hz, orange/yellow) show low noise concentrated below 20 ms, while stress and extreme configurations (gray/green) exhibit broader distributions and higher mean noise due to increased computational load and Wi-Fi contention; Figure S7: (a) Probability density distribution of estimated sampling frequency (Hz Est) across configurations. Dashed lines indicate mean values. Distinct separation between 50, 100, and 200 Hz configurations demonstrates stable and deterministic frequency control. (b) Probability density distribution of jitter standard deviation across configurations. Baseline and clean configurations (orange/yellow) show low jitter concentrated below 10 ms, while stress and extreme configurations (gray/green) exhibit broader distributions and higher mean jitter due to increased computational and network load; Figure S8: Histogram of instantaneous jitter for the optimal 100 Hz CRC configuration. The distribution is heavily right-skewed, with the vast majority of samples exhibiting near-zero jitter, confirming the deterministic nature of the predictive scheduling mechanism; Figure S9: Power spectral density (PSD) of the instantaneous jitter for the baseline 100 Hz CRC configuration. The PSD reveals dominant low-frequency components and a gradual roll-off in the 10–50 Hz band, characteristic of Wi-Fi MAC-layer contention and beacon interval interference; Figure S10: Probability density distribution of CRC computation time across configurations. All configurations cluster tightly around 80–100 μ s, indicating consistent CRC overhead. The narrow distribution confirms predictable performance and supports enabling CRC validation by default, with computational cost remaining negligible (<1% of the 10 ms budget at 100 Hz); Figure S11: (a) Effect sizes (Cohen's *d*) for sampling interval noise (Ts Noise) across configuration pairs. (b) Effect sizes for estimated sampling frequency (Hz Est) across configuration pairs. Results indicate medium-to-large negative effects on Ts Noise under stress conditions, and

very large effects on frequency stability ($|d| > 6$) between baseline and stress configurations; Table S1: Application-specific performance metrics for motion-controlled gaming tasks comparing the proposed system to commercial controllers; Table S2: Virtual reality integration metrics, tracking error, and Simulator Sickness Questionnaire (SSQ) assessment; Table S3: Robotic teleoperation performance comparing inertial-aided navigation against wheel odometry alone; Table S4: Microcontroller platform comparison for embedded IMU applications.

Author Contributions: Conceptualization, E.M., E.K. and N.Ž.; methodology, E.M., E.K. and N.Ž.; software, E.M.; validation, E.M., E.K., N.Ž. and C.R.; formal analysis, E.M., E.K. and N.Ž.; investigation, E.M., E.K. and N.Ž.; resources, E.M., E.K. and C.R.; data curation, E.M., E.K. and C.R.; writing—original draft preparation, E.M. and E.K.; writing—review and editing, E.M., E.K. and N.Ž.; visualization, E.M., E.K. and N.Ž.; supervision, N.Ž. and C.R.; project administration, E.M. and C.R.; funding acquisition, E.K., N.Ž. and C.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to further ongoing closed research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Aylward, R.; Paradiso, J.A. A Compact, High-Speed, Wearable Sensor Network for Biomotion Capture and Interactive Media. In Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07), Cambridge, MA, USA, 25–27 April 2007; p. 380. <https://doi.org/10.1145/1236360.1236408>.
2. Shen, S.; Gowda, M.; Roy Choudhury, R. Closing the Gaps in Inertial Motion Tracking. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom), New Delhi, India, 29 October–2 November 2018; pp. 429–444. <https://doi.org/10.1145/3241539.3241582>.
3. Brückner, H.-P.; Krüger, B.; Blume, H. Reliable Orientation Estimation for Mobile Motion Capturing in Medical Rehabilitation Sessions Based on Inertial Measurement Units. *Microelectron. J.* **2014**, *45*, 1603–1611. <https://doi.org/10.1016/j.mejo.2014.05.018>.
4. Bruckner, H.P.; Spindeldreier, C.; Blume, H.; Schoonderwaldt, E.; Altenmüller, E. Evaluation of Inertial Sensor Fusion Algorithms in Grasping Tasks Using Real Input Data: Comparison of Computational Costs and Root Mean Square Error. In Proceedings of the 2012 Ninth International Conference on Wearable and Implantable Body Sensor Networks (BSN), London, UK, 9–12 May 2012; pp. 189–194. <https://doi.org/10.1109/BSN.2012.9>.
5. Wilk, M.P.; Walsh, M.; O’Flynn, B. Low Cost Embedded Multimodal Opto-Inertial Human Motion Tracking System. In Proceedings of the 2020 31st Irish Signals and Systems Conference (ISSC), Letterkenny, Ireland, 17–18 June 2020; pp. 1–6. <https://doi.org/10.1109/ISSC49989.2020.9180190>.
6. Comotti, D.; Ermidoro, M.; Galizzi, M.; Vitali, A. Development of a Wireless Low-Power Multi-Sensor Network for Motion Tracking Applications. In Proceedings of the 2013 IEEE International Conference on Body Sensor Networks (BSN), Zurich, Switzerland, 9–12 September 2013; pp. 1–6. <https://doi.org/10.1109/BSN.2013.6575466>.
7. Valade, A.; Acco, P.; Grabolosa, P.; Fourniols, J.-Y. A Study about Kalman Filters Applied to Embedded Sensors. *Sensors* **2017**, *17*, 2810. <https://doi.org/10.3390/s17122810>.
8. Bruckner, H.-P.; Nowosielski, R.; Kluge, H.; Blume, H. Mobile and Wireless Inertial Sensor Platform for Motion Capturing in Stroke Rehabilitation Sessions. In Proceedings of the IWASI 2013, 5th IEEE International Workshop on Advances in Sensors and Interfaces, Bari, Italy, 13–14 June 2013; pp. 14–19. <https://doi.org/10.1109/IWASI.2013.6576085>.
9. Kingston, D.; Beard, R. Real-Time Attitude and Position Estimation for Small UAVs Using Low-Cost Sensors. In Proceedings of the AIAA 3rd “Unmanned Unlimited” Technical Conference, Chicago, IL, USA, 20–23 September 2004. <https://doi.org/10.2514/6.2004-6488>.
10. Pearl, O.; Shin, S.; Godura, A.; Bergbreiter, S.; Halilaj, E. Fusion of Video and Inertial Sensing Data via Dynamic Optimization of a Biomechanical Model. *J. Biomech.* **2023**, *155*, 111617. <https://doi.org/10.1016/j.jbiomech.2023.111617>.

11. Madgwick, S.O.H.; Harrison, A.J.L.; Vaidyanathan, R. Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm. In Proceedings of the 2011 IEEE International Conference on Rehabilitation Robotics (ICORR), Zurich, Switzerland, 29 June–1 July 2011; pp. 1–7. <https://doi.org/10.1109/ICORR.2011.5975346>.
12. Buonocunto, P.; Marinoni, M. Tracking Limbs Motion Using a Wireless Network of Inertial Measurement Units. In Proceedings of the 2014 IEEE International Symposium on Industrial Embedded Systems (SIES), Costa da Caparica, Portugal, 18–20 June 2014; pp. 66–76. <https://doi.org/10.1109/SIES.2014.6871188>.
13. Yun, X.; Bachmann, E.R. Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking. *IEEE Trans. Robot.* **2006**, *22*, 1216–1227. <https://doi.org/10.1109/TRO.2006.886270>.
14. Facchinetti, T.; Savioli, A.; Goldoni, E. Design and Development of a Real-Time Embedded Inertial Measurement Unit. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 491–495. <https://doi.org/10.1145/1774088.1774190>.
15. Zhu, R.; Zhou, Z. A Real-Time Articulated Human Motion Tracking Using Tri-Axis Inertial/Magnetic Sensors Package. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2004**, *12*, 295–302. <https://doi.org/10.1109/TNSRE.2004.827825>.
16. Masters, M.; Osborn, L.; Thakor, N.; Soares, A. Real-Time Arm Tracking for HMI Applications. In Proceedings of the 2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN), Cambridge, MA, USA, 9–12 June 2015; pp. 1–4. <https://doi.org/10.1109/BSN.2015.7299391>.
17. Herlihy, L.; Lyshevski, S.E. Real-Time Dynamic Data Processing and Fusion: Multi-Degree-of-Freedom MEMS Inertial Sensors. In Proceedings of the 2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO), Kyiv, Ukraine, 24–26 April 2018; pp. 631–637. <https://doi.org/10.1109/ELNANO.2018.8477518>.
18. Valášek, M.; Nečas, M.; Budjač, R.; Gašpar, G. Low-Latency Wireless Transmission of Inertial Measurements Using ESP-NOW Protocol for Real-Time Applications. In Proceedings of the 2025 5th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Zanzibar, Tanzania, 16–19 October 2025; pp. 1–6. <https://doi.org/10.1109/ICECCME64568.2025.11277535>.
19. Hossan, A.; Zhang, J.; Yu, D.; Wang, J.; Feng, P. Wearable Wireless IMU for Real-Time Locomotion Sensing: Gait and End-to-End Latency Validation. In Proceedings of the 2025 7th International Conference on Electrical, Control and Instrumentation Engineering (ICECIE), Pattaya City, Thailand, 22–23 November 2025; pp. 508–514. <https://doi.org/10.1109/ICECIE66637.2025.11363828>.
20. Xing, Y.; Pang, S.; Du, M.; Xu, F.; Ji, P. Design and Implementation of Motion Tracking System Based on IMU. In Proceedings of the 2024 IEEE 13th Data Driven Control and Learning Systems Conference (DDCLS), Kaifeng, China, 17–19 May 2024; pp. 2187–2191. <https://doi.org/10.1109/DDCLS61622.2024.10606617>.
21. Tariq, O.; Han, D. NanoMST: A Hardware-Aware Multiscale Transformer Network for TinyML-Based Real-Time Inertial Motion Tracking. *IEEE Internet Things J.* **2025**, *12*, 37763–37776. <https://doi.org/10.1109/JIOT.2025.3583906>.
22. Hsu, K.-P.; Lai, T.-H.; Li, Y.-T.; Tang, W.; Lee, Y.-J.; Chen, Y.-C.; Chiu, W.-H.; Wang, C.-Y. IMU-Based Motion Trajectory Reconstruction and Recognition with Dynamic Calibration on Embedded System Platform. In Proceedings of the 2025 26th International Symposium on Quality Electronic Design (ISQED), San Francisco, CA, USA, 23–25 April 2025; pp. 1–8. <https://doi.org/10.1109/ISQED65160.2025.11014473>.
23. Fléty, E.; Bevilacqua, F. Wireless Inertial Measurement Units in Performing Arts. *Sensors* **2025**, *25*, 6188. <https://doi.org/10.3390/s25196188>.
24. Xu, V.; Gao, C.; Hoffmann, H.; Ahuja, K. MobilePoser: Real-Time Full-Body Pose Estimation and 3D Human Translation from IMUs in Mobile Consumer Devices. In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST), Pittsburgh, PA, USA, 13–16 October 2024; pp. 1–11. <https://doi.org/10.1145/3654777.3676461>.
25. Tariq, O.; Dastagir, M.B.A.; Bilal, M.; Han, D. DeepILS: Toward Accurate Domain-Invariant AIoT-Enabled Inertial Localization System. *IEEE Internet Things J.* **2025**, *12*, 17153–17168. <https://doi.org/10.1109/JIOT.2025.3538938>.
26. Azarbeik, M.M.; Razavi, H.; Merat, K.; Salarieh, H. Augmenting Inertial Motion Capture with SLAM Using EKF and SRUKF Data Fusion Algorithms. *Measurement* **2023**, *222*, 113690. <https://doi.org/10.1016/j.measurement.2023.113690>.
27. Golroudbari, A.A.; Sabour, M.H. Generalizable End-to-End Deep Learning Frameworks for Real-Time Attitude Estimation Using 6DOF Inertial Measurement Units. *Measurement* **2023**, *217*, 113105. <https://doi.org/10.2139/SSRN.4387238>.
28. Ramkumar, G. Designing a Low-Latency Data Transmission System for Mobile Wireless Sensor Controls Using Optimized Communication Protocols. In Proceedings of the 2025 13th International Conference on Intelligent Embedded, MicroElectronics, Communication and Optical Networks (IEMECON), Jaipur, India, 8–10 December 2025; pp. 1–7. <https://doi.org/10.1109/IEMECON69302.2025.11365794>.

29. Xiao, X.; Wang, J.; Feng, P.; Gong, A.; Zhang, X.; Zhang, J. Fast Human Motion Reconstruction from Sparse Inertial Measurement Units Considering the Human Shape. *Nat. Commun.* **2024**, *15*, 2423. <https://doi.org/10.1038/s41467-024-46662-5>.
30. Young, A.D. Comparison of Orientation Filter Algorithms for Realtime Wireless Inertial Posture Tracking. In Proceedings of the 2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks (BSN), Berkeley, CA, USA, 3–5 June 2009; pp. 59–64. <https://doi.org/10.1109/BSN.2009.25>.
31. Shi, Y.; Zhang, Y.; Li, Z.; Yuan, S.; Zhu, S. IMU/UWB Fusion Method Using a Complementary Filter and a Kalman Filter for Hybrid Upper Limb Motion Estimation. *Sensors* **2023**, *23*, 6700. <https://doi.org/10.3390/s23156700>.
32. Chen, C.; Pan, X. Deep Learning for Inertial Positioning: A Survey. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 10506–10523. <https://doi.org/10.1109/TITS.2024.3381161>.
33. Zhang, D.; Liao, Z.; Xie, W.; Wu, X.; Xie, H.; Xiao, J.; Jiang, L. Fine-Grained and Real-Time Gesture Recognition by Using IMU Sensors. *IEEE Trans. Mob. Comput.* **2023**, *22*, 2177–2189. <https://doi.org/10.1109/TMC.2021.3120475>.
34. Wang, R.; Wang, Z.; Gao, P.; Li, M.; Jeong, J.; Xu, Y.; Lee, Y.; Baum, C.; Connor, L.; Lu, C. Real-Time Video-Based Human Action Recognition on Embedded Platforms. *ACM Trans. Embed. Comput. Syst.* **2025**, *24*, 1–24. <https://doi.org/10.1145/3761795>.
35. Zhang, K. Develop a Novel Deep Learning-Based Framework Using Convolutional Neural Networks for Real-Time Object Detection and Tracking in Embedded Systems. *J. Comput. Methods Sci. Eng.* **2026**, 5683–5699. <https://doi.org/10.1177/14727978251346034>.
36. Hu, J.; Jiang, H.; Liu, D.; Xiao, Z.; Zhang, Q.; Liu, J.; Dustdar, S. Combining IMU with Acoustics for Head Motion Tracking Leveraging Wireless Earphone. *IEEE Trans. Mob. Comput.* **2024**, *23*, 6835–6847. <https://doi.org/10.1109/TMC.2023.3325826>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.