

Politecnico di Torino

Master's Degree in Mechatronic Engineering

Academic Year 2025/2026

Graduation Session September 2026

Model-Based Design of Sensor Fusion Algorithms: Implementation and Experimental Evaluation on an Embedded Platform for Wearables

Supervisor:
Alessandro Rizzo
Saverio Milo

Candidate:
Rosario Arancio Febbo

Abstract

The rapid advancement of MEMS technology has enabled widespread adoption of wearable devices for human motion tracking, yet robust 3D attitude estimation remains a significant mathematical and physical challenge. Raw data from Inertial Measurement Units (IMUs) suffer from gyroscope drift bias, high-frequency accelerometer noise, and magnetic distortions. The Attitude and Heading Reference Systems (AHRS) developed to fuse these signals present critical trade-offs between dynamic accuracy and embedded system computational limits.

This thesis presents a comparative analysis of four prominent AHRS architectures: the Complementary Filter, Mahony, Madgwick, and the Extended Kalman Filter (EKF). The algorithms are implemented and experimentally evaluated on a resource-constrained Raspberry Pi Pico W microcontroller interfaced with an ICM-20948 MARG sensor. Through a hybrid software approach combining automatic C-code generation with manual optimization, each filter is evaluated on two primary metrics: architectural efficiency (execution time and memory footprint) and dynamic robustness (disturbance rejection and transient response recovery).

Experimental results show computational cost scales with algorithmic complexity: the Complementary filter completes its full acquisition-and-estimation cycle in approximately 2.5 ms with the smallest static RAM allocation (10,856 B for the complete build), while the EKF requires roughly five times longer, forcing a reduction to 50 Hz against the 100 Hz achievable by the other filters, reaching 19,480 B of static RAM (including 6,112 B for its internal state). At rest, all four algorithms exhibit sub-degree noise on the Roll and Pitch axes; targeted sensitivity experiments indicate part of the residual yaw drift stems from uncorrected gyroscope bias rather than external disturbances. Under fast, large-angle rotations, three of the four filters return to within 0.6° of their starting orientation, while the EKF alone shows a measurable yaw closure error and settling time. Finally, a severe magnetic disturbance test reveals the Complementary filter's structural immunity to magnetic anomalies alongside a persistent shift in the magnetometer-derived heading (consistent with a modified hard-iron offset or remanent magnetization) that no magnetometer-based filter can compensate for algorithmically; this shift is consistently observed through three filter implementations and a raw compass calculation based on the same measurements.

Ultimately, this research demonstrates that no single algorithm is universally optimal: the ideal choice depends on computational headroom, environmental noise, and the dynamic stress of the target application. This work provides the analytical and empirical framework needed to make that selection for real-time, resource-constrained wearable applications.

Table of Contents

List of Tables VI

List of Figures VII

1 Introduction 1
- 1.1 Goal . . . 2
- 1.2 Thesis Structure . . . 3

2 Fundamentals of Attitude Estimation 4
- 2.1 Coordinate Frames . . . 4
  - 2.1.1 Global Frame . . . 5
  - 2.1.2 Body Frame . . . 5
- 2.2 Euler Angles . . . 6
  - 2.2.1 Rotation Matrices . . . 7
  - 2.2.2 Euler Angles Kinematics . . . 7
  - 2.2.3 The Gimbal Lock Problem . . . 8
- 2.3 Quaternions . . . 9
  - 2.3.1 Mathematical Definition and Properties . . . 9
  - 2.3.2 Fundamental Operations . . . 9
  - 2.3.3 Quaternion Rotations . . . 11
  - 2.3.4 Quaternion Kinematics . . . 12

3 Attitude Estimation and Sensor Fusion Algorithms 13
- 3.1 Complementary Filter . . . 14
  - 3.1.1 Mathematical Model . . . 14
- 3.2 Mahony Filter . . . 16
  - 3.2.1 Mathematical Model . . . 16
- 3.3 Madgwick Filter . . . 17
  - 3.3.1 Mathematical Model and Gradient Descent Optimization . . . 18
  - 3.3.2 Magnetic Distortion Compensation . . . 19
- 3.4 Extended Kalman Filter (Error-State Formulation) . . . 19

3.4.1 The Quaternion Normalization Problem 20
3.4.2 The Error-State Architecture 20
3.4.3 Mathematical Model and Predict-Update Cycle 21
3.4.4 Simulink Architecture Extension: 12-State Framework . . . 23

4 Hardware Architecture and Sensor Calibration 24
4.1 Hardware Setup and Component Specifications 24
4.1.1 Microcontroller Unit: Raspberry Pi Pico W 24
4.1.2 The MARG Sensor: ICM-20948 25
4.1.3 System Integration and Communication 26
4.2 Intrinsic Sensor Errors and Noise Characteristics 26
4.3 On-Board Calibration Procedures 27
4.3.1 Accelerometer and Gyroscope Static Calibration 27
4.3.2 Magnetometer Full Calibration 28
4.3.3 Non-Volatile Storage of Calibration Parameters 30

5 Firmware Implementation and Hybrid Model-Based Design 31
5.1 Model-Based Design Setup and Embedded Code Generation 32
5.2 Manual Coding vs. Automated Generation: The Complementary Filter 34
5.3 Firmware Architecture and Task Scheduling 36
5.3.1 Real-Time Scheduling and Hardware Timers 36
5.3.2 Sensor Data Pipeline and Pre-processing 36
5.4 Bidirectional Serial Telemetry and System Diagnostics 38

6 Experimental Results and Performance Analysis 40
6.1 Experimental Setup and Methodology 40
6.2 Offline Algorithmic Validation 42
6.2.1 Long-Term Static Noise and Drift Analysis 42
6.2.2 Dynamic Response and Kinematic Tracking 43
6.2.3 Magnetic Disturbance and Anomaly Rejection 50
6.2.4 Residual Heading Shift After Magnetic Exposure 51
6.3 Online Hardware Profiling (RP2040) 53
6.3.1 Execution Time Analysis 53
6.3.2 Memory Footprint 54

7 Conclusions and Future Perspectives 55
7.1 Summary of Experimental Findings 55
7.2 Engineering Trade-offs and Selection Criteria 56
7.3 Limitations and Threats to Validity 57
7.4 Future Perspectives 58
7.5 Concluding Remarks 59

A Embedded Implementation Highlights 60
A.1 Algebraic Ellipsoid Fitting 60
A.2 Soft-Iron Compensation Matrix Calculation 61
A.3 Hard-Iron Offset Extraction and Final Assembly 62
A.4 Physical Scale Correction 63

B Mahony and Madgwick Filter MATLAB Implementation 66

Bibliography 70

List of Tables

5.1 Summary of the embedded configurations for the implemented sensor fusion algorithms. 34

6.1 Standard deviation and angular drift across the four evaluated algorithms during the 15-minute static dataset. 44
6.2 Closure error, peak deviation, and settling time for the four algorithms during the slow dynamic return test (Dataset 2). 45
6.3 Closure error, peak deviation, and settling time for the four algorithms during the fast dynamic return test (Dataset 3). 47
6.4 Peak deviation during the acute magnetic disturbance and recovery time for the four algorithms (Dataset 4). 51
6.5 Permanent yaw offset relative to the pre-disturbance baseline, measured independently by four different computational paths (Dataset 5). 53
6.6 Per-cycle execution time and time budget utilization measured on the RP2040 microcontroller. 54
6.7 Static RAM allocation extracted from the compiled firmware linker map. 54

List of Figures

2.1 Representation of the fixed Global Frame adopting the NWU convention (left) and the corresponding mobile Body Frame attached to the IMU (right). 5
2.2 Euler angles (ψ, θ, ϕ) according to the standard aeronautical Z-Y-X convention. Reproduced from [16]. 6

3.1 Continuous-time complementary filter block diagram applied to attitude estimation. 14

4.1 Hardware setup featuring the Raspberry Pi Pico W connected to the ICM-20948 MARG sensor. 25
4.2 Conceptual representation of magnetic distortions on a 2D plane: (a) ideal measurements with no distortions, (b) Hard-Iron effect causing a translation (offset), and (c) combined Hard and Soft-Iron effects causing both translation and deformation. 27
4.3 Magnetometer point cloud before and after the on-board calibration procedure, shown after the physical-scale correction described below. **Left:** Raw uncalibrated measurements expressed in μT, exhibiting a significant translation from the origin due to Hard-Iron distortions. **Right:** Calibrated data, structurally corrected for Soft-Iron effects and expressed at the sensor's true physical scale (in μT) rather than mapped to an arbitrary unit sphere. 29

5.1 Simulink architecture for the Mahony filter, implemented via a custom MATLAB Function block to ensure procedural C-code generation. The inputs include the PI tuning parameters (K_p, K_i) and the hardware sample period. 33
5.2 Simulink architecture for the Madgwick filter, featuring the gradient descent tuning parameter β. 33
5.3 Stochastic estimation pipeline in Simulink featuring the native *ahrs-filter* block and the cascaded quaternion-to-Euler conversion subsystem. 34

5.4 Real-time execution flowchart representing the hardware timer callback and the three-state FSM governing sensor calibration and data filtering. 37

6.1 Direct comparison of the estimated Euler angles during the long-term static dataset. The raw output clearly highlights the higher baseline noise of the Madgwick algorithm and the absence of yaw estimation in the 6-DoF Complementary filter. 44
6.2 Estimated Euler angles during the slow dynamic return test (Dataset 2). The three full 9-DoF filters track the same physical rotation closely, while the Complementary filter's transient divergence is clearly visible during the motion itself. 46
6.3 Pairwise 3D angular distance over time between the three full 9-DoF algorithms during the slow dynamic return test, quantifying the maximum disagreement of 3.62° reported in the text. 46
6.4 Estimated Euler angles during the fast dynamic return test (Dataset 3). All four independently reach near-inversion; the EKF's slower, more oscillatory settling after the motion concludes is visible relative to the other three. 48
6.5 Pairwise 3D angular distance over time between the three full 9-DoF algorithms during the fast return test. The disagreement visibly peaks after the detected motion window ends, during the EKF's extended settling period, contributing to the inverse behavior described in the text. 48
6.6 Per-filter Euler angles during the magnetic disturbance test (individual scales). The plot highlights the Complementary filter's immunity ($\pm 0.1^\circ$) alongside the distinct divergence paths of the magnetometer-based algorithms. 52
6.7 Pairwise 3D angular distance over time during the magnetic disturbance test, quantifying the largest inter-filter disagreement recorded anywhere in this study among filters with a directly comparable 3-axis orientation (Mahony vs. EKF, up to 128.28°). 52

Chapter 1

Introduction

The rapid advancement of Micro-Electro-Mechanical Systems (MEMS) has driven a revolution in wearable devices, enabling the creation of miniaturized, low-power, and cost-effective sensors for large-scale applications ranging from human motion tracking [1] and fall detection in the elderly [2] to posture monitoring [3] and upper limb motion analysis [4]. At the core of this revolution lies a fundamental challenge: robustly and accurately estimating the spatial orientation, or attitude, of a rigid body moving in three-dimensional space in real-time.

Attitude estimation is typically performed by fusing raw data from an Inertial Measurement Unit (IMU). Modern IMUs are typically configured as 6-axis devices (comprising a triaxial accelerometer and a triaxial gyroscope) or 9-axis MARG devices (Magnetic, Angular Rate, and Gravity sensors) that also include a triaxial magnetometer. However, deriving an accurate orientation from these raw signals is inherently complex, primarily due to the physical and thermal imperfections of MEMS sensors. While gyroscopes provide precise, high-frequency angular velocity measurements in the short term, they suffer from drift bias caused by the numerical integration of low-frequency noise [5, 6]. Accelerometers, conversely, offer a stable and reliable long-term reference for gravity, but their signals are highly sensitive to high-frequency noise and are distorted by external linear and centrifugal accelerations, which add vectorially to gravity [7]. Magnetometers, essential for an absolute yaw reference relative to magnetic North, are easily disrupted by local electromagnetic interference, historically classified into hard-iron distortions (generated by permanent magnetic fields near the sensor) and soft-iron distortions (caused by ferromagnetic materials that warp the surrounding field) [8, 9].

To overcome these issues and achieve a reliable estimate, the scientific community has developed various sensor fusion algorithms, commonly known as Attitude and Heading Reference Systems (AHRS), which mathematically balance the contributions of the different sensors by exploiting their complementary frequency characteristics [10]. The most widely adopted architectures include Complementary

Filters in both linear and nonlinear variants, geometric filters based on optimization or proportional-integral control such as the Mahony and Madgwick algorithms [11, 12], and stochastic estimation methods based on the Extended Kalman Filter (EKF). Each of these approaches involves distinct trade-offs regarding static and dynamic accuracy and robustness to external disturbances and, most importantly for embedded systems, computational complexity and memory footprint [13].

Similar comparative studies have been conducted on other embedded platforms: Cavallo et al. [14] compared the EKF, Madgwick, and Mahony filters on an STM32F3Discovery board validated against a KUKA robotic manipulator, while Simon et al. [15] compared the complementary, EKF, and Mahony filters on an STM32G491 microcontroller using simulated IMU data.

1.1 Goal

The primary objective of this thesis is to implement, test, and evaluate four sensor fusion architectures (Complementary Filter, Mahony, Madgwick, and Extended Kalman Filter). This evaluation is conducted on a resource-constrained embedded platform, specifically a Raspberry Pi Pico W microcontroller interfaced with an ICM-20948 inertial sensor. The 6-DoF, Euler-domain Complementary filter is included primarily as a lightweight embedded baseline: since it estimates roll and pitch only, its transient response during large-angle rotations is not directly comparable to that of the three quaternion-based 9-DoF estimators.

The study provides a rigorous comparative analysis to assess the performance and suitability of each algorithm, focusing on two key aspects:

- **Architectural and Computational Efficiency:** The execution time and memory footprint of each filter are quantified directly on the target hardware. This approach highlights the physical limitations of the microcontroller when handling algorithms of varying complexity, ranging from the highly efficient Complementary Filter to the heavy computational load imposed by the matrix operations of the EKF.

- **Robustness and Estimation Dynamics:** The accuracy and stability of the filters are evaluated under real-world physical scenarios. Specifically, the analysis measures each algorithm's disturbance rejection capabilities. This includes observing how quickly they compensate for drift, their transient response, and the settling time required to return to equilibrium after impulsive movements, continuous shaking, or severe external perturbations.

Ultimately, this research aims to analytically outline the intrinsic advantages and drawbacks of each architecture, determining the trade-offs between spatial

estimation accuracy and computational cost. Rather than identifying a single, universally optimal algorithm, the findings are intended to provide the analytical and empirical basis needed to select the most appropriate AHRS architecture for a given operational scenario, within the constraints of a low-cost board intended for wearable applications.

1.2 Thesis Structure

The present work is organized into seven chapters, structured as follows:

- **Chapter 2: Fundamentals of Attitude Estimation.** It provides the theoretical background for 3D kinematics, detailing the transition from Euler angles to the more robust Quaternion algebra to avoid singularities.
- **Chapter 3: Attitude Estimation and Sensor Fusion Algorithms.** This chapter examines the mathematical architecture and control theory of the four selected AHRS algorithms: Complementary Filter, Mahony, Madgwick, and Extended Kalman Filter (EKF).
- **Chapter 4: Hardware Architecture and Sensor Calibration.** It describes the physical setup, including the Raspberry Pi Pico W and the ICM-20948 MARG sensor. It also details intrinsic sensor errors, such as drift and magnetic distortions, alongside the implemented calibration procedures.
- **Chapter 5: Firmware Implementation and Hybrid Model-Based Design.** This chapter focuses on the software methodology, describing the hybrid approach that combines MATLAB/Simulink modeling and automatic C-code generation with the manual implementation of the Complementary Filter to optimize the porting process.
- **Chapter 6: Experimental Results and Performance Analysis.** It presents the experimental evaluation, providing a comparative analysis of the four filters concerning execution time, memory footprint, and dynamic robustness against external perturbations, including the discovery of a persistent shift in the magnetometer-derived heading following acute magnetic exposure, consistently observed through three filter implementations and a raw compass calculation on the same magnetometer data.
- **Chapter 7: Conclusions and Future Perspectives.** The final chapter summarizes the findings, discusses the trade-offs that guide algorithm selection for the target hardware depending on the operational scenario, and suggests future developments.

Chapter 2

Fundamentals of Attitude Estimation

The spatial orientation of a 3D body can be represented in three main ways: Euler angles, defining orientation via three parameters (ϕ, θ, ψ); quaternions, requiring four (q_0, q_1, q_2, q_3); and the Direction Cosine Matrix (DCM), based on nine parameters. This chapter will focus on the mathematical foundations of Euler angles and quaternions, as they are the methods directly implemented in this work.

2.1 Coordinate Frames

The spatial orientation of a rigid body is never an absolute property; rather, it is inherently relative, meaning it can only be defined and understood in comparison to another established reference system. Because physical space lacks a universal baseline, determining how an object is tilted or rotated requires a constant standard of measurement. Therefore, to measure and track attitude consistently across applications such as robotics, aerospace, or biomechanics, it is strictly essential to define the mathematical relationship between two distinct coordinate frames:

- **A mobile frame (Body Frame):** A localized coordinate system rigidly attached to the object in motion. It typically originates at the body's center of mass and its axes align with the object's geometry.

- **A fixed global frame (Inertial or Earth Frame):** A stationary coordinate system, often attached to the Earth or a specific operational point of origin. Unlike the body frame, it remains immutable regardless of the object's movements.

2.1.1 Global Frame

The global frame, often referred to as the navigation frame, serves as the absolute motionless baseline for our system. While the most widely used reference systems, especially in aeronautics and robotics, are the North-East-Down (NED) and East-North-Up (ENU) conventions, this project adopts the North-West-Up (NWU) convention.

This specific choice was made to strictly align the global frame with the native data output of the IMU sensor. Maintaining the native coordinate system minimizes axes remapping, thereby reducing computational overhead and the risk of software artifacts during the filtering process:

- The X-axis points towards the magnetic North.
- The Y-axis points West.
- The Z-axis points Upward, opposing the direction of gravity.

2.1.2 Body Frame

The body frame is a mobile Cartesian coordinate system securely attached to the device. Its origin is typically located at the center of the Inertial Measurement Unit (IMU). The axes, usually denoted as X_b, Y_b, and Z_b, are aligned with the physical dimensions of the sensor board and translate and rotate along with it.

Therefore, defining the spatial orientation of the device essentially means determining the rotation that relates this mobile body frame to the fixed global frame.

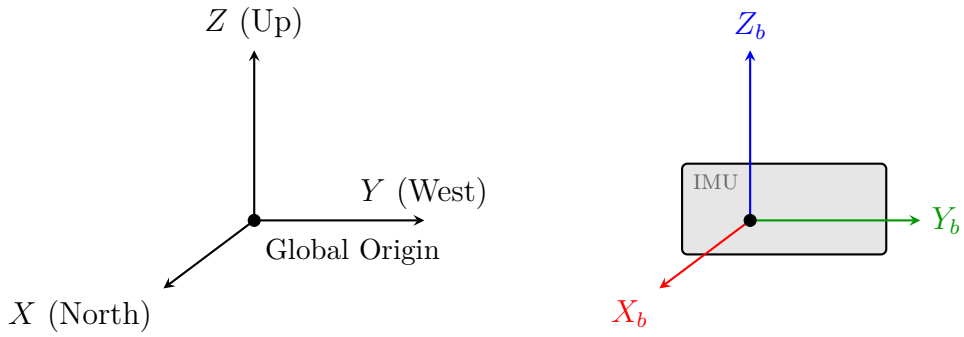


Figure 2.1: Representation of the fixed Global Frame adopting the NWU convention (left) and the corresponding mobile Body Frame attached to the IMU (right).

2.2 Euler Angles

To describe the rotation between two reference frames, one of the most intuitive methods is the use of Euler angles, which consist of three successive elemental rotations. The 12 possible rotation sequences fall into two primary classifications:

- **Proper Euler Angles:** where the first and third rotations occur about the same axis.
- **Tait-Bryan Angles:** where each rotation takes place about a different, distinct axis.

The implementation in this project relies on the Tait-Bryan convention, specifically utilizing the standard aeronautical terminology:

- **Roll** (ϕ): Rotation about the X-axis.
- **Pitch** (θ): Rotation about the Y-axis.
- **Yaw** (ψ): Rotation about the Z-axis.

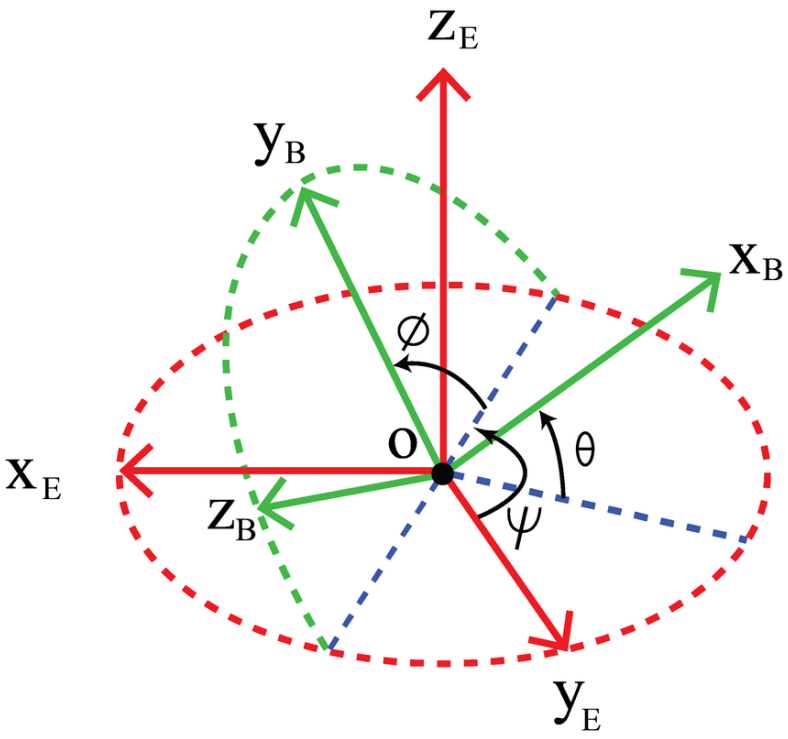


Figure 2.2: Euler angles (ψ, θ, ϕ) according to the standard aeronautical Z-Y-X convention. Reproduced from [16].

2.2.1 Rotation Matrices

The rotation of a rigid body in three-dimensional space is described by an orthogonal 3×3 matrix, whose general form is expressed as:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.1}$$

Based on the Tait-Bryan convention previously introduced, any complex orientation can be decomposed into three elemental rotations. The fundamental rotation matrices corresponding to Roll (ϕ), Pitch (θ), and Yaw (ψ) are defined as follows:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.2}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.3}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The combined spatial orientation is achieved by multiplying these elemental rotation matrices. For the standard Tait-Bryan Z-Y-X sequence, the final composite rotation matrix R is computed as:

$$R = R_z(\psi)R_y(\theta)R_x(\phi) \tag{2.5}$$

It is crucial to emphasize that matrix multiplication is not commutative (i.e., $R_z R_y R_x \neq R_x R_y R_z$). Consequently, the order in which these rotations are applied defines the final physical orientation of the body.

2.2.2 Euler Angles Kinematics

Real-time attitude estimation using inertial data requires establishing a relationship between the angular velocity vector measured by the gyroscope in the body frame, denoted as $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$, and the time derivatives of the Euler angles, $\dot{\eta} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$.

The body angular velocity does not directly correspond to the time derivative of the Euler angles. Instead, it is the vector sum of the individual angular rate variations, each projected into the body frame. Since Roll is the last elemental

rotation applied, the roll rate $\dot{\phi}$ is already expressed about the body's own x-axis and needs no further transformation. The pitch rate $\dot{\theta}$, however, is originally defined about the y-axis of the intermediate frame that exists *before* the roll rotation is applied, and must therefore be rotated into the body frame by undoing that roll, i.e. by $R_x(\phi)^{-1} = R_x(-\phi)$. Similarly, the yaw rate $\dot{\psi}$ is originally defined about the z-axis of the original global frame, before either the pitch or the roll rotation, and must be rotated into the body frame by undoing both, applied in the same order in which they were originally introduced: $R_x(-\phi)R_y(-\theta)$. Adopting the standard Z-Y-X sequence, this kinematic relationship is expressed as:

$$\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_x(-\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_x(-\phi)R_y(-\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{2.6}$$

By expanding the matrix products, we obtain the Jacobian matrix that maps the Euler angle rates to the angular velocities measured in the local frame:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.7}$$

To express the rate of change of the Euler angles as a function of the measured body angular velocity, the inverse relationship is required. By inverting the Jacobian matrix, the kinematic differential equation is obtained:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.8}$$

2.2.3 The Gimbal Lock Problem

While Euler angles provide an intuitive representation of a body's attitude, their formulation suffers from a severe limitation known as *Gimbal Lock*.

Beyond its geometric nature, this issue emerges as a numerical instability in the inverse kinematic matrix. Due to the coefficients containing the terms $\tan\theta$ and $\frac{1}{\cos\theta}$, the differential equation presents a mathematical singularity when the pitch angle (θ) reaches exactly $\pm 90°$ ($\pm\pi/2$ rad).

Here, the cosines approach zero, driving the matrix terms to infinity. Consequently, one degree of freedom is lost as the roll and yaw axes align. This instability makes Euler angles unsuitable for continuous orientation tracking within the core control logic.

2.3 Quaternions

To overcome the singularities caused by the Gimbal Lock and guarantee continuous, reliable orientation tracking within the control system, a fundamentally different mathematical approach is necessary: quaternions.

2.3.1 Mathematical Definition and Properties

Quaternions are defined as an extension of complex numbers to four dimensions. A quaternion q consists of a real (scalar) part, q_0, and three imaginary components that form a vector part, $\mathbf{q} = (q_1, q_2, q_3)$. The fundamental imaginary units i, j, and k correspond to the standard orthonormal basis vectors for $\mathbb{R}^3$, defined as $i = (1, 0, 0)$, $j = (0, 1, 0)$, and $k = (0, 0, 1)$. Accordingly, a quaternion is expressed as:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{2.9}$$

In the context of spatial orientation, attitude estimation algorithms employ *unit quaternions* to represent pure rotations without altering the scale of the rotated object. The norm (or magnitude) of a quaternion is defined as:

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{2.10}$$

A unit quaternion is one that satisfies the condition $\|q\| = 1$.

2.3.2 Fundamental Operations

To practically employ quaternions in attitude estimation algorithms, it is necessary to define the core algebraic operations used to manipulate spatial orientations. Specifically, three fundamental mathematical tools form the basis of the control logic: addition, which is used to accumulate error terms and integrate angular variations; multiplication, which allows for the composition of successive spatial rotations; and inversion, which is essential to reverse a coordinate transformation between the global and the body frame.

Addition and Multiplication

The addition of two quaternions is performed component-wise. Considering two quaternions $p = p_0 + p_1 i + p_2 j + p_3 k$ and $q = q_0 + q_1 i + q_2 j + q_3 k$, their sum is calculated as:

$$p + q = (p_0 + q_0) + (p_1 + q_1)i + (p_2 + q_2)j + (p_3 + q_3)k \tag{2.11}$$

This operation is straightforward, associative, and commutative.

Unlike addition, quaternion multiplication is governed by Hamilton's fundamental rules for the imaginary units:

$$i^2 = j^2 = k^2 = ijk = -1, \tag{2.12}$$
$$ij = k = -ji,$$
$$jk = i = -kj,$$
$$ki = j = -ik. \tag{2.13}$$

Given two quaternions p and q, their product can be compactly expressed by applying the distributive law alongside the rules above. Splitting the quaternions into their scalar parts (p_0, q_0) and their three-dimensional vector parts $(\mathbf{p}, \mathbf{q})$, the product is formulated as:

$$pq = (p_0 q_0 - \mathbf{p} \cdot \mathbf{q}) + (p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q}) \tag{2.14}$$

where $\cdot$ denotes the standard dot product and $\times$ represents the cross product in $\mathbb{R}^3$.

The mutually orthogonal cross-products mathematically demonstrate the strict non-commutative nature of quaternion multiplication ($pq \neq qp$). This is a deliberate and essential feature, as it accurately mimics the physical behavior of three-dimensional space: the sequence in which consecutive rotations are applied strictly dictates the final physical orientation of a body.

Conjugate and Inverse

The complex conjugate of a quaternion q, denoted as q^*, is obtained by reversing the sign of its imaginary vector components:

$$q^* = q_0 - q_1 i - q_2 j - q_3 k = q_0 - \mathbf{q} \tag{2.15}$$

In standard quaternion algebra, the inverse q^{-1}, which mathematically represents the reversed rotation, is defined by dividing the conjugate by the squared norm of the quaternion:

$$q^{-1} = \frac{q^*}{\|q\|^2} \tag{2.16}$$

However, since attitude tracking algorithms inherently rely exclusively on unit quaternions ($\|q\| = 1$), the denominator always evaluates to one. As a result, the inverse spatial rotation can be calculated directly by taking the conjugate:

$$q^{-1} = q^* \tag{2.17}$$

Geometrically, this means that if a given unit quaternion represents a rotation from a fixed global frame to a mobile body frame, its conjugate effortlessly provides the exact opposite transformation, moving from the body frame back to the global one.

2.3.3 Quaternion Rotations

From a geometric perspective, the purely algebraic definition of a quaternion lacks an immediate intuitive link to three-dimensional orientations. However, any unit quaternion q (satisfying $\|q\| = 1$) can be expressed in a trigonometric form that directly maps to spatial geometry.

It is standard practice to parameterize this relationship using the axis-angle representation with a half-angle convention:

$$q = \cos\left(\frac{\theta}{2}\right) + \mathbf{u}\sin\left(\frac{\theta}{2}\right) \tag{2.18}$$

where $\mathbf{u}$ is a unit vector representing the invariant axis of rotation, and θ is the total rotation angle around that axis. This bidirectional mapping ensures that every unit quaternion corresponds to a specific spatial rotation, providing a compact and singularity-free mathematical framework for orientation tracking.

The three elementary rotations around the respective principal axes correspond to the following quaternions:

$$q_x(\alpha) = \left(\cos\frac{\alpha}{2}, \sin\frac{\alpha}{2}, 0, 0\right) \tag{2.19}$$

$$q_y(\beta) = \left(\cos\frac{\beta}{2}, 0, \sin\frac{\beta}{2}, 0\right) \tag{2.20}$$

$$q_z(\gamma) = \left(\cos\frac{\gamma}{2}, 0, 0, \sin\frac{\gamma}{2}\right) \tag{2.21}$$

Once the quaternion representation of a rotation is established, this operator can be applied to three-dimensional vectors. For attitude estimation algorithms, this step is required to transform reference vectors, such as gravity and the Earth's magnetic field, between the fixed global frame and the mobile body frame.

To rotate a standard 3D vector $\mathbf{v} = [v_x, v_y, v_z]^T$, it must first be embedded into the four-dimensional quaternion space. This is achieved by defining it as a pure quaternion, which is characterized by a scalar part equal to zero:

$$v = (0, v_x, v_y, v_z) \tag{2.22}$$

The rotation of the embedded vector is performed using the quaternion rotation operator. Given a unit attitude quaternion q, the transformed vector v' is computed as:

$$v' = qvq^* \tag{2.23}$$

By the algebraic properties of the Hamilton product and the unit norm constraint on q, the scalar part of the resulting quaternion v' evaluates to zero. Its vector part thus contains the exact coordinates of the rotated vector in the target frame, preserving the original magnitude.

2.3.4 Quaternion Kinematics

To track the time-varying orientation of a rigid body, the temporal evolution of the attitude quaternion must be mathematically related to the angular velocity vector. Let $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$ represent the angular velocity measured in the body-fixed frame. To operate within the quaternion algebra framework, the angular velocity vector is expressed as a pure quaternion $\omega_q = (0, \omega_x, \omega_y, \omega_z)$, similarly to the spatial vectors discussed in Section 2.3.3.

The continuous-time derivative of the attitude quaternion q is governed by the following linear differential equation:

$$\dot{q} = \frac{1}{2} q \otimes \omega_q \tag{2.24}$$

where $\otimes$ denotes the Hamilton product. Unlike kinematic formulations based on Euler angles, this expression is entirely free from trigonometric functions and does not suffer from geometric singularities or gimbal lock.

In sampled-data systems with a fixed sampling period Δt, the continuous kinematic equation must be evaluated at discrete time intervals. Assuming the angular velocity remains constant within the sampling window, a first-order forward Euler integration yields the discrete-time attitude update:

$$q_t = q_{t-1} + \dot{q}\Delta t \tag{2.25}$$

Due to truncation errors and sensor noise inherent to digital numerical integration, the norm of the integrated quaternion q_t will gradually drift away from unity. To ensure that q_t remains a valid rotation operator, it is necessary to normalize the quaternion after each discrete update step:

$$q_{\text{norm}} = \frac{q_t}{\|q_t\|} \tag{2.26}$$

Chapter 3

Attitude Estimation and Sensor Fusion Algorithms

This chapter details the mathematical models of the filtering algorithms implemented for attitude estimation. Standard MEMS sensors, like the ICM-20948 [17] used in this project, combine a gyroscope, an accelerometer, and a magnetometer. However, relying on a single sensor is not enough to accurately estimate attitude due to their distinct dynamic and frequency-domain behaviors. The gyroscope is highly precise in the short term (high-frequency dynamics) but suffers from low-frequency integration drift. Conversely, the accelerometer and magnetometer provide a stable absolute reference over time (low-frequency), but are highly sensitive to high-frequency noise and unreliable during fast movements [10].

The main goal of the sensor fusion algorithms implemented here is to mathematically combine these measurements to obtain a robust attitude estimate, exploiting the complementary frequency characteristics of each sensor.

To validate their performance under real-time constraints, all algorithms must adhere to specific architectural guidelines dictated by the resource limitations of the target embedded hardware (the RP2040 microcontroller). Before detailing the specific architectures of the Complementary Filter, the Mahony Filter, the Madgwick Algorithm, and the Extended Kalman Filter (EKF), it is important to note that they all share the digital implementation principles introduced at the end of the previous chapter. Specifically, their continuous-time models are discretized using the fixed sampling period Δt and, for all quaternion-based variants, the normalization step ($\mathbf{q} \leftarrow \mathbf{q}/\|\mathbf{q}\|$) is systematically enforced at the end of every execution loop to counteract numerical drift.

3.1 Complementary Filter

The Complementary Filter is the simplest sensor fusion technique. It works by combining a high-pass and a low-pass filter to directly exploit the frequency-domain characteristics of the sensors established previously: mitigating the gyroscope's drift while reducing the accelerometer's noise [10]. In this project, the filter is implemented in its classic 6-DoF configuration (using only the gyroscope and accelerometer) to serve as a lightweight, highly efficient computational baseline. To preserve this foundational simplicity, the magnetometer is deliberately excluded. Consequently, the absolute yaw angle remains unobservable in this architecture, leaving full 9-DoF spatial tracking to the more advanced estimators detailed in the following sections.

3.1.1 Mathematical Model

The basic complementary filter architecture, as described by Higgins [18], is shown in Figure 3.1, assuming that $G(s)$ is a low-pass filter and $1 - G(s)$ is a high-pass filter. In the context of attitude estimation, the input $\boldsymbol{\theta}_{acc}$ represents the absolute angles calculated from the accelerometer, while $\boldsymbol{\theta}_{gyro}$ represents the angles obtained by integrating the gyroscope's angular velocity ($\boldsymbol{\omega}_{meas}$). The low-pass filter transfer function is typically defined as a first-order system:

$$G(s) = \frac{1}{\tau s + 1} \tag{3.1}$$

where τ is the time constant of the filter. This parameter defines the crossover threshold of the system. Consequently, the high-pass filter takes the form:

$$1 - G(s) = \frac{\tau s}{\tau s + 1} \tag{3.2}$$

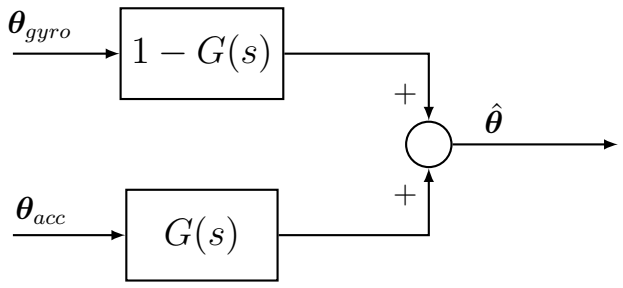


Figure 3.1: Continuous-time complementary filter block diagram applied to attitude estimation.

To map this block diagram architecture to physical inputs, the absolute orientation angles $\boldsymbol{\theta}_{acc}$ are derived from the gravity vector $\mathbf{a} = [a_x, a_y, a_z]^T$ measured by the accelerometer. When the sensor is in a quasi-static condition, the orientation can be computed using the following trigonometric relations:

$$\boldsymbol{\theta}_{acc} = \begin{bmatrix} \phi_{acc} \\ \theta_{acc} \end{bmatrix} = \begin{bmatrix} \text{atan2}(a_y, a_z) \\ \text{atan2}(-a_x, \sqrt{a_y^2 + a_z^2}) \end{bmatrix} \tag{3.3}$$

Conversely, the gyroscope measures the raw angular velocity vector $\boldsymbol{\omega}_{meas} = [\omega_x, \omega_y, \omega_z]^T$ in [rad/s]. As demonstrated in Section 2.2.2, the body angular velocities measured by the gyroscope do not directly correspond to the Euler angle rates. The exact kinematic relationship, established in Equation 2.8, is non-linear and cross-coupled.

However, for the specific implementation of the baseline Complementary Filter, it is common practice to assume that the system operates near a quasi-static condition with small angular variations. Under the small-angle approximation ($\phi \approx 0$ and $\theta \approx 0$), the trigonometric terms evaluate to $\sin(\cdot) \approx 0$, $\cos(\cdot) \approx 1$, and $\tan(\cdot) \approx 0$. Substituting these limits into the inverse Jacobian matrix simplifies the kinematic differential equation as follows:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \implies \begin{cases} \dot{\phi} \approx \omega_x \\ \dot{\theta} \approx \omega_y \\ \dot{\psi} \approx \omega_z \end{cases} \tag{3.4}$$

Under this specific assumption, the rate of change of each Euler angle can be directly approximated by its corresponding measured angular velocity. Consequently, the gyroscope's contribution is obtained through numerical integration over the sampling period Δt:

$$\boldsymbol{\theta}_{gyro,t} = \begin{bmatrix} \phi_{gyro,t} \\ \theta_{gyro,t} \end{bmatrix} = \begin{bmatrix} \phi_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \end{bmatrix} \Delta t \tag{3.5}$$

The digital execution of this logic yields a computationally efficient difference equation, parameterized by a single tuning factor α:

$$\begin{bmatrix} \phi_t \\ \theta_t \end{bmatrix} = \alpha \left(\begin{bmatrix} \phi_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \end{bmatrix} \Delta t \right) + (1 - \alpha) \begin{bmatrix} \phi_{acc,t} \\ \theta_{acc,t} \end{bmatrix} \tag{3.6}$$

Here, ϕ_t and θ_t represent the current estimated roll and pitch angles, and α is derived strictly from the continuous-time constant τ:

$$\alpha = \frac{\tau}{\tau + \Delta t} \tag{3.7}$$

This parameter α dictates the cutoff frequency of the complementary filter. A value close to 1 places more trust in the gyroscope, effectively filtering out high-frequency translational noise from the accelerometer, while a lower value increases responsiveness to the accelerometer's absolute gravity reference.

3.2 Mahony Filter

As discussed in Section 2.2.3, relying on Euler angles is incompatible with robust and continuous attitude tracking due to kinematic singularities. Consequently, transitioning from the baseline complementary filter to more sophisticated algorithms based on quaternions becomes essential for full 3D tracking.

The Mahony filter [11] addresses the sensor fusion problem through a non-linear feedback control architecture based entirely on unit quaternions. Instead of relying on simple frequency-domain filtering, this algorithm evaluates the orientation error geometrically and employs a Proportional-Integral (PI) controller to continuously correct the gyroscope's angular velocity.

3.2.1 Mathematical Model

While the theoretical foundation of the Mahony filter relies on complex non-linear stability analysis, the focus of this work lies in its practical embedded realization.

The core logic relies on a predictor-corrector architecture. The filter maintains an estimation of the current orientation represented by the unit quaternion $\hat{\mathbf{q}}$. If this estimation were perfectly accurate, the filter could easily predict the direction of the Earth's gravitational and magnetic fields in the sensor's body frame.

Let $\hat{\mathbf{v}}_{acc}$ and $\hat{\mathbf{v}}_{mag}$ be the estimated directional vectors in the body frame. As introduced in Section 2.3.3, these are calculated by applying the inverse quaternion rotation operator to the inertial reference vectors $\mathbf{v}_{0,acc}$ and $\mathbf{v}_{0,mag}$ which represent the known global directions of gravity and the Earth's local magnetic field, respectively:

$$\hat{\mathbf{v}}_{acc} = \hat{\mathbf{q}}^* \otimes \mathbf{v}_{0,acc} \otimes \hat{\mathbf{q}} \tag{3.8}$$

$$\hat{\mathbf{v}}_{mag} = \hat{\mathbf{q}}^* \otimes \mathbf{v}_{0,mag} \otimes \hat{\mathbf{q}} \tag{3.9}$$

The orientation error $\mathbf{e}$ is computed as the sum of the cross products between the measured directions and the estimated ones:

$$\mathbf{e} = k_a(\mathbf{v}_{acc} \times \hat{\mathbf{v}}_{acc}) + k_m(\mathbf{v}_{mag} \times \hat{\mathbf{v}}_{mag}) \tag{3.10}$$

where $\mathbf{v}_{acc}$ and $\mathbf{v}_{mag}$ are the physical fields measured by the accelerometer and the magnetometer, respectively. The coefficients k_a and k_m represent the weights assigned to each sensor measurement. For the scope of this implementation, these gains are set to unity ($k_a = k_m = 1$), assuming equal reliability for both sensors under the considered operating conditions.

The cross product is a geometrical measure of misalignment: if the estimated orientation matches the real one perfectly, the vectors are parallel and the error is zero. The resulting error vector $\mathbf{e}$ represents the axis and magnitude of the rotation required to align the estimated frame with the measured physical fields.

This error is fed into a PI controller to correct the raw angular velocity measured by the gyroscope, $\boldsymbol{\omega}_{meas}$. The continuous-time corrected angular velocity $\boldsymbol{\omega}_c$ is calculated as:

$$\boldsymbol{\omega}_c = \boldsymbol{\omega}_{meas} + K_p\mathbf{e} + K_i \int \mathbf{e}\, dt \tag{3.11}$$

Finally, this corrected angular velocity is used to update the quaternion derivative using the standard kinematic relationship for quaternions:

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \otimes \boldsymbol{\omega}_c = \frac{1}{2}\hat{\mathbf{q}} \otimes \begin{bmatrix} 0 \\ \omega_{cx} \\ \omega_{cy} \\ \omega_{cz} \end{bmatrix} \tag{3.12}$$

Discretizing the observer equations for the microcontroller execution results in the following sequential updates:

$$\mathbf{e}_{int,t} = \mathbf{e}_{int,t-1} + \mathbf{e}_t \Delta t \tag{3.13}$$

$$\boldsymbol{\omega}_{c,t} = \boldsymbol{\omega}_{meas,t} + K_p\mathbf{e}_t + K_i\mathbf{e}_{int,t} \tag{3.14}$$

$$\hat{\mathbf{q}}_t = \hat{\mathbf{q}}_{t-1} + \left(\frac{1}{2}\hat{\mathbf{q}}_{t-1} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{c,t} \end{bmatrix}\right) \Delta t \tag{3.15}$$

The iteration then concludes with the standard algebraic normalization established in the introduction to maintain the quaternion's unit magnitude.

In this architecture, the tuning parameters K_p and K_i dictate the filter's dynamic response. The proportional gain K_p defines the crossover frequency, determining how heavily the filter trusts the accelerometer and magnetometer to pull the estimation towards the absolute reference. The integral gain K_i dictates the rate at which the gyroscope bias is estimated and removed.

3.3 Madgwick Filter

Introduced by Sebastian Madgwick [12], this algorithm represents a highly efficient alternative to traditional stochastic estimators, such as the Extended Kalman Filter (EKF), for Inertial Measurement Units (IMUs) and Magnetic, Angular Rate, and Gravity (MARG) sensor arrays. While the Mahony filter relies on a non-linear PI control loop to compensate for errors geometrically, the Madgwick algorithm formulates the attitude estimation task as an optimization problem solved via an analytical gradient descent method.

By analytically deriving the Jacobian matrices, the algorithm minimizes the mathematical operations required per cycle. Furthermore, the filter introduces a magnetic distortion compensation mechanism and defines its dynamic response through a single tuning parameter, β, which possesses a direct physical interpretation.

3.3.1 Mathematical Model and Gradient Descent Optimization

The orientation of the sensor is represented by the unit quaternion $\hat{\mathbf{q}}$. The optimization process begins by establishing an objective function f that quantifies the alignment error between a known reference field in the Earth's inertial frame, $\mathbf{d}$, and the corresponding physical measurement in the sensor's body frame, $\mathbf{s}$. Given the previous orientation estimate $\hat{\mathbf{q}}_{t-1}$, the objective function is formulated as:

$$f(\hat{\mathbf{q}}_{t-1}, \mathbf{d}, \mathbf{s}) = \hat{\mathbf{q}}_{t-1}^{*} \otimes \mathbf{d} \otimes \hat{\mathbf{q}}_{t-1} - \mathbf{s} \tag{3.16}$$

To compute the orientation that minimizes this field misalignment error, a gradient descent step is executed. The algorithm computes the gradient of the objective function, ∇f, by evaluating the product of the analytical Jacobian matrix $\mathbf{J}$ and the objective function itself:

$$\nabla f = \mathbf{J}^{T}(\hat{\mathbf{q}}_{t-1}, \mathbf{d})\, f(\hat{\mathbf{q}}_{t-1}, \mathbf{d}, \mathbf{s}) \tag{3.17}$$

The computed gradient ∇f defines the direction of the steepest descent toward the minimum error configuration. For a full MARG implementation, the complete objective function $\mathbf{F}$ and its cumulative Jacobian $\mathbf{J}$ are constructed by compounding the individual components of the accelerometer ($\mathbf{a}$) and the magnetometer ($\mathbf{m}$). The resulting normalized gradient descent step represents an estimated rate of change of orientation based purely on the reference fields:

$$\xi_t = \frac{\nabla \mathbf{F}}{\|\nabla \mathbf{F}\|} \tag{3.18}$$

Simultaneously, the gyroscope measures the body angular velocity vector $\boldsymbol{\omega}_{meas}$. The kinematic rate of change of the orientation quaternion derived from these angular rates is expressed as:

$$\dot{\mathbf{q}}_{\omega,t} = \frac{1}{2}\hat{\mathbf{q}}_{t-1} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{meas,t} \end{bmatrix} \tag{3.19}$$

The core innovation of Madgwick's formulation lies in the fusion of these two separate orientation rates. The algorithm corrects the high-frequency kinematic update by subtracting the normalized gradient descent step, weighted by the filter gain β:

$$\dot{\hat{\mathbf{q}}}_t = \dot{\mathbf{q}}_{\omega,t} - \beta \xi_t \tag{3.20}$$

The tuning parameter β represents the divergence rate of $\hat{\mathbf{q}}$ expressed as the magnitude of a quaternion derivative. Physically, it is directly related to the maximum gyroscope measurement error and gyroscope noise.

As established in the introduction to this chapter, the iteration then concludes with the explicit algebraic normalization to ensure $\|\hat{\mathbf{q}}_t\| = 1$.

3.3.2 Magnetic Distortion Compensation

In practical applications, local magnetic anomalies and hard/soft iron distortions heavily corrupt magnetometer data. If left uncompensated, these errors degrade the estimation of all attitude angles. To prevent this, Madgwick proposed a dynamic compensation mechanism that ensures magnetic perturbations only affect the yaw (heading) angle, leaving the pitch and roll tracking decoupled and secure.

At each time step, the raw magnetometer measurement in the body frame, $\mathbf{m} = [0, m_x, m_y, m_z]^T$, is rotated into the Earth's inertial frame using the current orientation estimate $\hat{\mathbf{q}}_{t-1}$:

$$\mathbf{h} = [0, h_x, h_y, h_z]^T = \hat{\mathbf{q}}_{t-1} \otimes \mathbf{m} \otimes \hat{\mathbf{q}}_{t-1}^* \tag{3.21}$$

In an ideal environment free of distortion, the inertial magnetic vector points strictly toward the magnetic North within the horizontal plane (x-axis) and down into the Earth (z-axis), meaning its y-component should evaluate to zero. To filter out local anomalies, the algorithm computes a corrected Earth reference vector $\mathbf{b}$ by consolidating the magnitude of the measured horizontal components into the x-axis:

$$\mathbf{b} = \begin{bmatrix} 0 \\ b_x \\ 0 \\ b_z \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{h_x^2 + h_y^2} \\ 0 \\ h_z \end{bmatrix} \tag{3.22}$$

By forcing the reference vector's y-component to zero, the algorithm ensures that any measured magnetic deviation in the transverse plane is treated as a heading error rather than a physical inclination. This vector $\mathbf{b}$ is then fed back as the reference vector $\mathbf{d}$ inside the magnetometer portion of the objective function f in Equation 3.16.

3.4 Extended Kalman Filter (Error-State Formulation)

While the Mahony and Madgwick algorithms address the attitude estimation problem through deterministic, geometric error-correction models [11, 12], the Extended Kalman Filter (EKF) introduces a rigorous stochastic approach. Instead of relying on static tuning parameters, the Kalman Filter dynamically tracks not only the system's state but also the uncertainty of its estimation (via the error covariance matrix) and the statistical noise properties of the sensors. This allows the filter to continuously calculate an optimal blend of forward kinematic predictions and vector observations.

The specific attitude estimator deployed in this project utilizes the automated code generation features of the Simulink `ahrsfilter` System object, which implements a variant of the traditional EKF known in literature as the *Error-State Kalman Filter* (ESKF) [19].

3.4.1 The Quaternion Normalization Problem

In a standard, direct EKF formulation, the true state vector would include the four components of the attitude quaternion $\mathbf{q}$. The traditional Kalman Filter inherently assumes that the state space is a flat, linear vector space, meaning it applies correction updates via standard vector addition [19]:

$$\hat{\mathbf{q}}_{new} = \hat{\mathbf{q}}_{old} + \Delta\mathbf{q} \tag{3.23}$$

Adding a linear update vector $\Delta\mathbf{q}$ to a unit quaternion inevitably pushes the resulting vector off the unit hypersphere, violating the fundamental algebraic constraints of 3D rotations. Over successive iterations, this constraint violation causes the 4×4 error covariance matrix to lose rank, become singular, and lead to the estimation collapse [20]. While heuristic brute-force normalization can be applied post-update, it disrupts the mathematical optimality and stochastic consistency of the Kalman Filter framework [20].

3.4.2 The Error-State Architecture

To bypass this structural singularity, the Error-State Kalman Filter separates the system into two distinct parallel states: the *Nominal State* and the *Error State* [19].

The nominal state vector $\mathbf{x}$ tracks the high-frequency, non-linear physical quantities. It contains the full unit quaternion $\mathbf{q}$ and the estimated gyroscope bias $\mathbf{b}_g$:

$$\mathbf{x} = \begin{bmatrix}\mathbf{q}^T & \mathbf{b}_g^T\end{bmatrix}^T = \begin{bmatrix}q_0 & q_1 & q_2 & q_3 & b_{gx} & b_{gy} & b_{gz}\end{bmatrix}^T \tag{3.24}$$

This state is maintained entirely outside the primary Kalman filter loop. It is updated using non-linear kinematic differential equations driven purely by the raw gyroscope measurements $\boldsymbol{\omega}_{meas}$, without considering sensor noise or external corrections [19].

Conversely, the error state vector $\delta\mathbf{x}$ describes the small stochastically driven deviations between the nominal state and the true physical state. Instead of tracking a redundant 4D quaternion error, the filter tracks a minimal 3D angular error vector $\delta\boldsymbol{\theta}$, representing the local rotational perturbation, alongside the bias error vector $\delta\mathbf{b}_g$:

$$\delta\mathbf{x} = \begin{bmatrix}\delta\boldsymbol{\theta}^T & \delta\mathbf{b}_g^T\end{bmatrix}^T = \begin{bmatrix}\delta\theta_x & \delta\theta_y & \delta\theta_z & \delta b_{gx} & \delta b_{gy} & \delta b_{gz}\end{bmatrix}^T \tag{3.25}$$

Because these error terms are kept infinitesimally small through a continuous reset mechanism, their behavior is practically linear [19]. Consequently, they are perfectly suited for the traditional linear Kalman Filter machinery without suffering from gimbal lock or trigonometric non-linearities.

3.4.3 Mathematical Model and Predict-Update Cycle

The ESKF algorithm operates in a discrete-time continuous loop composed of three main phases: Nominal Prediction, Error Update, and State Correction. The implementation described in this section closely follows the kinematic formulations and structural conventions outlined by Solà [19].

Nominal State Prediction

During the prediction phase, the raw angular velocity from the gyroscope is compensated with the current nominal bias estimation to obtain the corrected angular rate vector $\boldsymbol{\omega} = \boldsymbol{\omega}_{meas} - \mathbf{b}_g$. The continuous-time kinematic differential equation for the nominal orientation is given by:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{3.26}$$

Assuming the angular velocity remains constant over the sampling interval Δt, the nominal quaternion is updated using the exact trigonometric exponential map. This approach also preserves the unit magnitude of the quaternion during the integration:

$$\mathbf{q}_{t|t-1} = \mathbf{q}_{t-1|t-1} \otimes \begin{bmatrix} \cos\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right) \\ \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right) \end{bmatrix} \tag{3.27}$$

Simultaneously, the filter propagates the error covariance matrix $\mathbf{P}$ using the system's linearized state transition Jacobian matrix $\mathbf{F}_t$ and the process noise covariance matrix $\mathbf{Q}$, which models the stochastic white noise and random walk of the gyroscope:

$$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1|t-1} \mathbf{F}_t^T + \mathbf{Q} \tag{3.28}$$

Error State Update

When new absolute reference vector measurements $\mathbf{z}_t$ become available from the accelerometer and the magnetometer, the filter computes the expected measurements $h(\mathbf{x}_{t|t-1})$ based purely on the predicted nominal orientation. The difference between the actual and expected measurements yields the innovation vector.

The Kalman Gain matrix $\mathbf{K}_t$ is calculated using the measurement Jacobian $\mathbf{H}_t$ (evaluated with respect to the error state), the propagated covariance $\mathbf{P}_{t|t-1}$, and the sensor measurement noise covariance matrix $\mathbf{R}$:

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T \left(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}\right)^{-1} \tag{3.29}$$

The error state vector $\delta\mathbf{x}_t$ is then updated via the linear Kalman update equation:

$$\delta\mathbf{x}_t = \mathbf{K}_t \left(\mathbf{z}_t - h(\mathbf{x}_{t|t-1})\right) \tag{3.30}$$

State Correction (Error Injection and Normalization)

In the final step, the stochastically estimated error is injected back into the nominal state to correct it. The bias error is integrated via standard vector addition, whereas the angular error vector $\delta\boldsymbol{\theta}_t$ is converted into an error quaternion $\delta\mathbf{q}_t$ and applied via quaternion multiplication.

To apply this rotation correctly, the 3D error vector is transformed using exact trigonometry:

$$\delta\mathbf{q}_t = \begin{bmatrix} \cos\left(\frac{\|\delta\boldsymbol{\theta}_t\|}{2}\right) \\ \frac{\delta\boldsymbol{\theta}_t}{\|\delta\boldsymbol{\theta}_t\|}\sin\left(\frac{\|\delta\boldsymbol{\theta}_t\|}{2}\right) \end{bmatrix} \tag{3.31}$$

The nominal states are then corrected as follows:

$$\mathbf{q}_t \leftarrow \mathbf{q}_{t|t-1} \otimes \delta\mathbf{q}_t \tag{3.32}$$

$$\mathbf{b}_{g,t} \leftarrow \mathbf{b}_{g,t|t-1} + \delta\mathbf{b}_{g,t} \tag{3.33}$$

A common simplification in literature involves reducing Equation 3.31 to a first-order linear approximation ($\delta\mathbf{q}_t \approx \begin{bmatrix} 1 & \frac{1}{2}\delta\boldsymbol{\theta}_t^T \end{bmatrix}^T$) [19]. However, because this linearized vector inherently possesses a norm strictly greater than unity, injecting it directly would cause the nominal quaternion to artificially inflate over consecutive iterations.

To prevent this mathematical inconsistency, the generated Simulink C code avoids the linear approximation entirely, utilizing the exact trigonometric transformation for the error injection as well. Furthermore, the standard explicit algebraic normalization is enforced immediately after the update to guarantee long-term numerical stability.

Finally, the internal error vector is reset to zero ($\delta\mathbf{x} = 0$) to prepare the Kalman Filter for the next discrete time step.

3.4.4 Simulink Architecture Extension: 12-State Framework

While the 6-state ESKF detailed in Eqs. 3.24 and 3.25 represents the theoretical baseline, physical sensors rarely operate in perfectly ideal conditions. Accelerometers inevitably measure transient linear movements in addition to gravity, and magnetometers are frequently affected by local magnetic interference.

To improve the accuracy of the estimation under these realistic conditions, the deployed Simulink `ahrsfilter` block extends the core architecture into a 12-state framework. The state vector is augmented to track two additional 3D variables: the linear acceleration and the magnetic disturbance. Although this expands the error covariance matrix $\mathbf{P}$ to a 12×12 dimension and increases the overall computational load, the fundamental mechanics of the filter remain identical to the foundational theory described above.

Chapter 4

Hardware Architecture and Sensor Calibration

This chapter describes the hardware ecosystem selected for the wearable device, detailing the computational capabilities required to execute the auto-generated Simulink code in real-time. Furthermore, it addresses the intrinsic errors of the inertial and magnetic sensors, outlining the on-board calibration procedures performed to ensure reliable data acquisition. The transition from raw, noisy signals to calibrated, stable inputs is a fundamental prerequisite for any robust attitude estimation system.

4.1 Hardware Setup and Component Specifications

The physical architecture of the system is designed around two main components: a microcontroller unit capable of handling complex matrix operations and a high-resolution 9-DoF (Degrees of Freedom) sensor.

4.1.1 Microcontroller Unit: Raspberry Pi Pico W

The core of the wearable system is the Raspberry Pi Pico W, built around the RP2040 microcontroller [21]. It features a dual-core ARM Cortex-M0+ processor clocked at 133 MHz. While foundational filters, such as the Complementary or Madgwick algorithms, require moderate processing power, the Extended Kalman Filter (EKF) involves heavy floating-point matrix operations (e.g., matrix inversions and Jacobian calculations). The 133 MHz clock speed proved sufficient to execute software-emulated floating-point arithmetic within the sampling period of each

filter's final configuration, once the sampling rate of each algorithm was matched to its computational cost, as detailed in Section 4.1.3.

Additionally, the board provides 264 KB of internal SRAM and 2 MB of onboard Flash memory. This footprint is highly suitable for a Model-Based Design approach, comfortably accommodating the memory overhead typically associated with C-code automatically generated by MATLAB/Simulink, alongside the necessary hardware drivers.

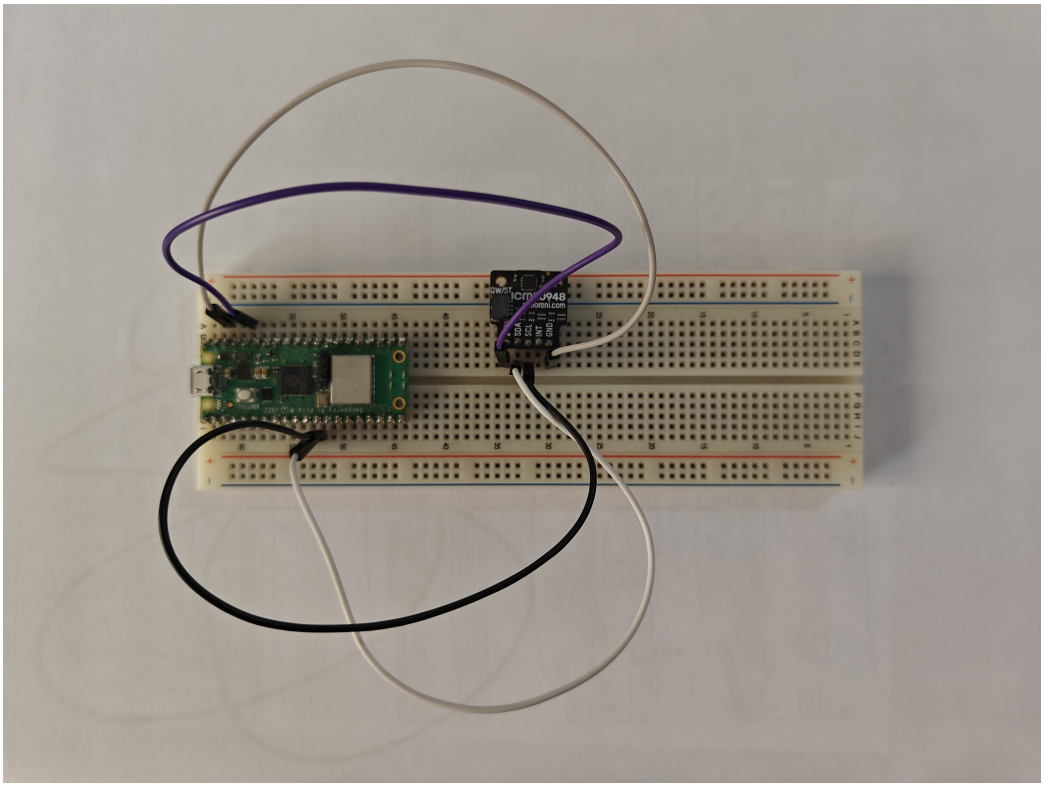


Figure 4.1: Hardware setup featuring the Raspberry Pi Pico W connected to the ICM-20948 MARG sensor.

4.1.2 The MARG Sensor: ICM-20948

Inertial and magnetic measurements are acquired using the TDK InvenSense ICM-20948 [17], a highly integrated MARG (Magnetic, Angular Rate, and Gravity) sensor. It houses a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis AK09916 magnetometer [22], utilizing independent 16-bit Analog-to-Digital Converters (ADCs) for high-resolution data sampling.

A key design choice during the sensor initialization phase was the configuration of the Full-Scale Ranges (FSR). The accelerometer FSR was explicitly set to its maximum value of $\pm 16g$, and the gyroscope FSR to $\pm 2000°/s$. This conservative configuration was deliberately chosen to maximize the dynamic range and definitively prevent sensor saturation during sudden or vigorous human movements. Empirical testing revealed that sensor clipping (saturation) severely degraded the performance of the implemented algorithms, particularly causing rapid divergence in the Extended Kalman Filter (EKF), which is highly sensitive to discrepancies between its predictive model and truncated measurement updates.

4.1.3 System Integration and Communication

To minimize latency and guarantee strict real-time execution, the microcontroller and the ICM-20948 communicate via the I2C protocol. The Pico W acts as the I2C Controller, driving the bus at 400 kHz (Fast-mode).

An important aspect of the hardware integration involves the magnetometer, addressed directly via the *I2C Bypass Mode*. Initially, the system was configured to sample data at 100 Hz. While this rate was perfectly sustainable for computationally lighter algorithms (such as the Madgwick, Mahony, and Complementary filters), empirical validation revealed a critical bottleneck during the execution of the Extended Kalman Filter. The heavy floating-point matrix inversions and Jacobian calculations required by the EKF exceeded the available timing budget at 100 Hz, leading to task overruns. To guarantee deterministic real-time execution, the sampling rate for the EKF implementation was purposefully reduced to 50 Hz. This engineering trade-off successfully prevented CPU overruns while still maintaining a sufficient data rate for human motion tracking.

4.2 Intrinsic Sensor Errors and Noise Characteristics

Real-world MEMS (Micro-Electro-Mechanical Systems) sensors and magnetometers are subject to intrinsic errors that, if left uncompensated, severely degrade the performance of sensor fusion algorithms. These errors can be broadly categorized into deterministic (systematic) and stochastic (random) errors.

For the accelerometer and gyroscope, the primary deterministic error source is the static **Bias** (or zero-offset). Bias represents the non-zero output of the sensor when it is subjected to zero input (i.e., when perfectly still). In the gyroscope, uncompensated bias is particularly critical: because angular velocity is integrated to estimate orientation, a constant bias leads to an unbound, linearly growing angular error over time, commonly known as *drift* [6].

For the magnetometer, errors are heavily influenced by the surrounding environment and the physical composition of the board itself [9]:

- **Hard-Iron Distortions:** Caused by permanent ferromagnetic materials on the wearable board (e.g., magnetized components, batteries). These magnetic fields superimpose on the Earth's magnetic field, causing a constant additive shift in the sensor readings. Visually, this shifts the center of the 3D magnetic measurement sphere away from the origin (0,0,0).
- **Soft-Iron Distortions:** Caused by materials that interact with and deform the Earth's magnetic field (e.g., metallic shields, iron traces). These distortions scale and rotate the measurements differently along each axis, visually deforming the ideal spherical locus into an ellipsoid.

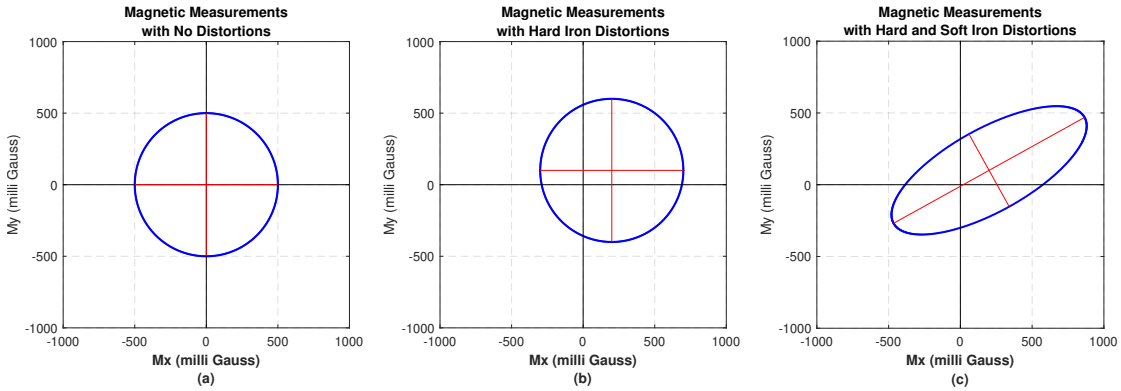


Figure 4.2: Conceptual representation of magnetic distortions on a 2D plane: (a) ideal measurements with no distortions, (b) Hard-Iron effect causing a translation (offset), and (c) combined Hard and Soft-Iron effects causing both translation and deformation.

4.3 On-Board Calibration Procedures

To ensure the wearable device could operate independently without relying on an external PC for continuous data processing, all calibration procedures were implemented directly on the Raspberry Pi Pico W.

4.3.1 Accelerometer and Gyroscope Static Calibration

A straightforward yet effective static calibration approach was adopted for the inertial sensors. Upon initialization, with the wearable device placed on a flat, stationary surface, the microcontroller gathers a predefined window of 500 samples

from the raw registers. The static bias is estimated by computing the arithmetic mean of these 500 samples. Assuming the device is stationary, the expected output of the gyroscope is zero, and the expected output of the accelerometer is $1g$ along the vertical axis. Once the bias vectors are calculated, they are subtracted from every subsequent raw reading in real-time to obtain the calibrated data:

$$y_{cal} = y_{raw} - b$$

where y_{raw} is the raw measurement and b is the estimated bias vector.

4.3.2 Magnetometer Full Calibration

Magnetometer calibration proved to be significantly more complex. Initial implementations relied on a simplified algorithm that only evaluated the minimum and maximum values along each axis. While computationally cheap, this approach failed to fully correct Soft-Iron distortions, leaving the measurement locus as an unaligned ellipsoid.

To overcome this, a comprehensive calibration algorithm, adapted from an open-source pure C implementation [23], was integrated into the system. This approach avoided heavy external mathematical libraries to maintain a low memory footprint on the RP2040. By moving the sensor in a dynamic "figure-8" pattern, the algorithm collects a cloud of spatial points, storing a fixed dataset of 600 samples.

The system then solves a least-squares fitting problem to map the deformed ellipsoid back into a perfect sphere centered at the origin [9]. Specifically, the algorithm computes the parameters V and W that minimize the error between the measured points and an ideal sphere of radius B (representing the local Earth's magnetic field strength), optimizing the following cost function:

$$J(V, W) = \sum_{i=1}^{N} \left(\|W(h_{raw,i} - V)\|^2 - B^2 \right)^2$$

By minimizing this algebraic distance dynamically on the RP2040, the system calculates the final correction matrices. The calibration is then applied in real-time via the linear transformation:

$$h_{cal} = W(h_{raw} - V)$$

where h_{raw} is the uncalibrated magnetic reading, V is the 3×1 Hard-Iron offset vector, and W is the 3×3 Soft-Iron compensation matrix.

Although the cost function $J(V, W)$ is formulated with respect to a sphere of physical radius B, the specific ellipsoid-fitting library adopted for this implementation was empirically found, during validation, to normalize its output to an arbitrary radius close to unity, regardless of the true local field strength implied by

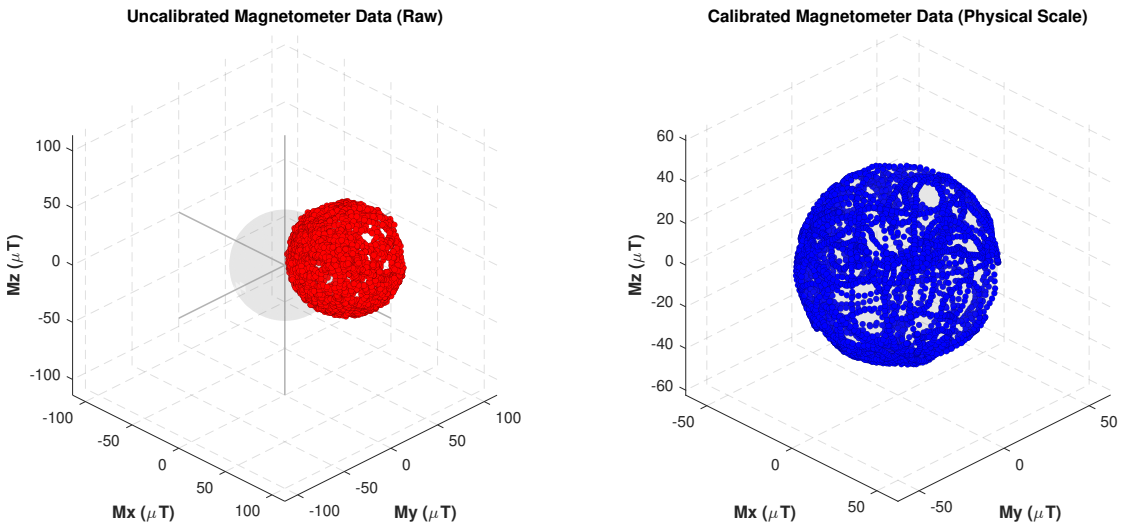


Figure 4.3: Magnetometer point cloud before and after the on-board calibration procedure, shown after the physical-scale correction described below. **Left:** Raw uncalibrated measurements expressed in μT, exhibiting a significant translation from the origin due to Hard-Iron distortions. **Right:** Calibrated data, structurally corrected for Soft-Iron effects and expressed at the sensor's true physical scale (in μT) rather than mapped to an arbitrary unit sphere.

the raw data. Left uncorrected, this behavior discards the physical magnitude of the magnetic field entirely, which is a problem for any downstream component that relies on it, most notably the EKF's expected-field-strength parameter and the absolute-threshold magnetic-disturbance detection used in the offline validation of Chapter 6.

To correct this, an additional scale-correction step is applied immediately after the fit. The mean norm of the hard-iron-corrected calibration samples (i.e., $h_{raw,i} - V$, before the soft-iron correction W is applied) is compared against the mean norm that the fitted matrix W produces on those same samples. The entire matrix W is then rescaled by the ratio of the two quantities. Because scaling a linear transformation by a scalar factor scales its output by the same factor, this restores h_{cal} to the sensor's physical scale, in μT, without altering the shape correction already achieved by the fit. Strictly speaking, the scale being restored is the mean field magnitude as measured by the sensor rather than the true local field strength B: on an ellipsoidal locus, the mean norm of the hard-iron-corrected samples only approximates B, with an error that grows with the eccentricity of the ellipsoid and with the non-uniformity of the sampling pattern over the sphere. For the nearly spherical raw locus observed on this hardware (Figure 4.3, left), this approximation was considered negligible. The same figure reflects the calibration output after

this correction is applied. For a detailed view of the embedded C implementation, including the algebraic ellipsoid fitting and the derivation of the compensation matrices, the reader is referred to Appendix A.

4.3.3 Non-Volatile Storage of Calibration Parameters

Requiring the user to perform a complex "figure-8" calibration sequence at every device boot is impractical for a plug-and-play wearable application. Therefore, a data persistence mechanism was required. Once the Hard-Iron vector V and the Soft-Iron matrix W (alongside the inertial biases) are successfully computed, the Pico W writes these parameters into a dedicated 256 KB *LittleFS* partition [24] within the Flash memory.

At every subsequent power cycle, the system automatically mounts the file system, retrieves the calibration matrices, and applies them directly before feeding the data to the Simulink-generated control logic. This mechanism drastically improves the usability of the device, confining the calibration procedure only to the initial setup phase or when environmental magnetic conditions change drastically.

Chapter 5

Firmware Implementation and Hybrid Model-Based Design

To implement the sensor fusion algorithms detailed in Chapter 3, this work adopts a Hybrid Model-Based Design (MBD) approach. This methodology strategically combines the automated C-code generation capabilities of MATLAB/Simulink with the precise control of manual software implementation.

For the more advanced filtering architectures, the Model-Based Design paradigm was utilized to speed up development and minimize transcription errors. However, the implementation strategy varied depending on the algorithm. The Mahony and Madgwick filters were integrated using *MATLAB Function* blocks within Simulink. This required refactoring existing Object-Oriented MATLAB scripts into procedural code compatible with Simulink's automated code generation tools. Conversely, the Extended Kalman Filter (EKF) was implemented by using the native Simulink *ahrsfilter* block. This block processed the full 9-DoF sensor data (accelerometer, gyroscope, and magnetometer), and its output quaternions were subsequently routed to a dedicated function block to extract the corresponding Euler angles (Roll, Pitch, and Yaw).

While MBD significantly accelerates the prototyping of complex algorithms, relying exclusively on standard Simulink libraries can sometimes introduce unnecessary overhead or architectural deviations from the required theoretical baseline. This was particularly evident with the Complementary Filter. The standard Complementary Filter block provided by MATLAB implements a quaternion-based nonlinear complementary architecture, regulated by separate accelerometer and magnetometer gains, rather than the classic single-parameter formulation described in Chapter 3. Since the objective for this specific filter was to establish the simplest

purely inertial (6-DoF) baseline utilizing only the accelerometer and gyroscope, the built-in block was deemed unsuitable. Consequently, the Complementary Filter was deliberately excluded from the automated generation pipeline and implemented manually in pure C.

5.1 Model-Based Design Setup and Embedded Code Generation

To bridge the gap between theoretical models and hardware deployment on the Raspberry Pi Pico W, an automated code generation workflow was established using MATLAB/Simulink and the *Embedded Coder* tool. The integration with the target hardware was facilitated by the *Arduino Support Package*, which provides a compatible hardware abstraction layer for the RP2040 microcontroller.

Before initiating the code generation process, the algorithmic logic was rigorously tested and validated offline within the MATLAB desktop environment. This preliminary validation was performed using a standard open-source IMU and AHRS dataset provided by x-io Technologies [25]. Testing the algorithms against this validated dataset ensured that the mathematical foundations of the filters were sound prior to any hardware deployment.

Within the Simulink environment, the implementation strategy was tailored to the specific nature of each algorithm:

- **Mahony and Madgwick Filters:** The baseline code for these geometric filters was sourced from publicly available MATLAB repositories [25]. However, these original implementations were structured using Object-Oriented Programming (OOP) paradigms, specifically relying on MATLAB classes. Because *Embedded Coder* requires procedural code to generate deployable C/C++ files, a structural adaptation was necessary.

  The OOP-based algorithms were manually refactored into procedural routines and encapsulated within Simulink *MATLAB Function* blocks, as illustrated in Figures 5.1 and 5.2. A key aspect of this refactoring was replacing class properties with `persistent` variables to allocate static memory for the filter states (such as the estimated quaternion) across simulation steps. The complete refactored procedural MATLAB code for these filters is provided in Appendix B.

- **Extended Kalman Filter (EKF):** The stochastic estimation was implemented using the native Simulink *ahrsfilter* block. This pre-built system object was configured to accept the complete 9-DoF sensor data (accelerometer, gyroscope, and magnetometer) as inputs. Since the *ahrsfilter* inherently outputs

the estimated attitude in the form of quaternions, a secondary function block was cascaded to its output to transform the unit quaternions into human-interpretable Euler angles (Roll, Pitch, and Yaw), completing the execution lifecycle shown in Figure 5.3.

Finally, to ensure deterministic real-time execution on the microcontroller, the Simulink model solver was configured to a *Fixed-step discrete* mode. This setting explicitly enforced the sampling intervals analyzed in Chapter 4, specifically $\Delta t = 0.01\,\text{s}$ (100 Hz) for the Mahony and Madgwick filters, and a reduced rate of $\Delta t = 0.02\,\text{s}$ (50 Hz) for the heavier EKF algorithm.

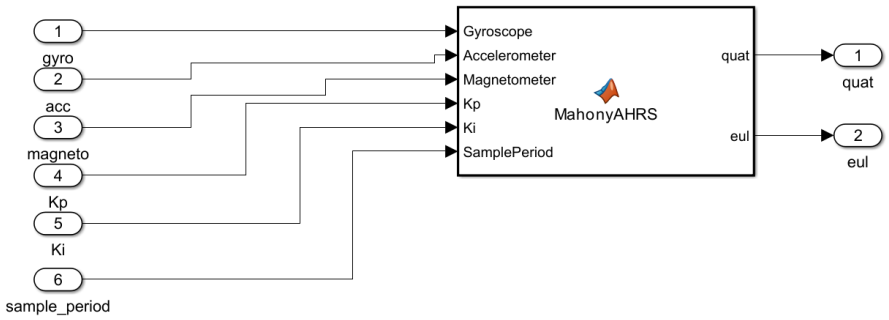


Figure 5.1: Simulink architecture for the Mahony filter, implemented via a custom MATLAB Function block to ensure procedural C-code generation. The inputs include the PI tuning parameters (K_p, K_i) and the hardware sample period.

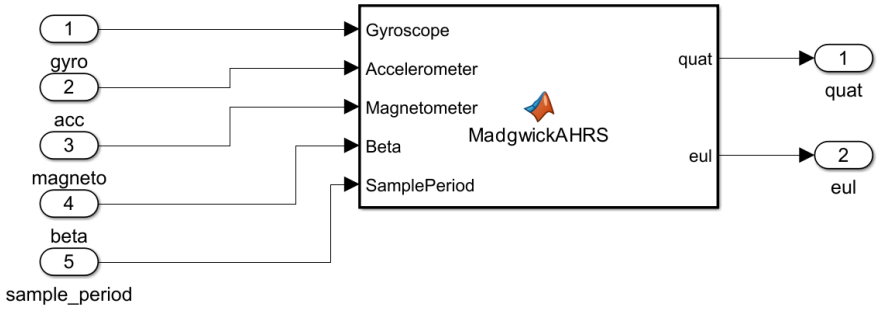


Figure 5.2: Simulink architecture for the Madgwick filter, featuring the gradient descent tuning parameter β.

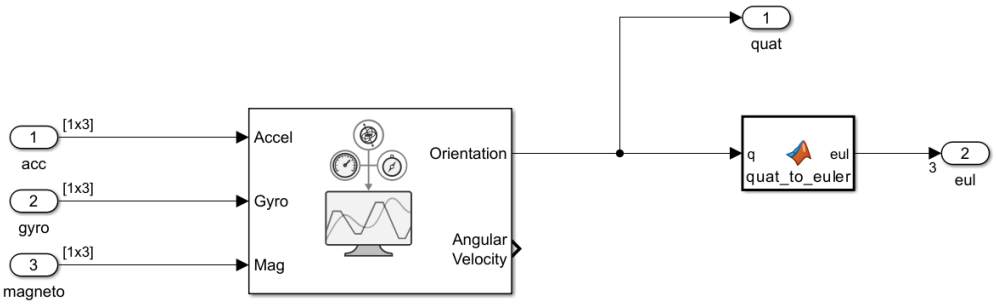


Figure 5.3: Stochastic estimation pipeline in Simulink featuring the native *ahrsfilter* block and the cascaded quaternion-to-Euler conversion subsystem.

Algorithm	Implementation	Sample Rate	Inputs (DoF)
Complementary Filter	Manual C-Code	100 Hz	6 (Acc, Gyr)
Mahony Filter	MBD (MATLAB Function)	100 Hz	9 (Acc, Gyr, Mag)
Madgwick Filter	MBD (MATLAB Function)	100 Hz	9 (Acc, Gyr, Mag)
Extended Kalman Filter	MBD (ahrsfilter Block)	50 Hz	9 (Acc, Gyr, Mag)

Table 5.1: Summary of the embedded configurations for the implemented sensor fusion algorithms.

5.2 Manual Coding vs. Automated Generation: The Complementary Filter

While the Model-Based Design (MBD) methodology proved highly effective for deploying mathematically dense algorithms like the EKF and the Madgwick filter, its application to simpler control architectures can sometimes introduce unjustifiable software overhead. This section details the reasoning behind the decision, already introduced above, to implement the Complementary Filter manually instead.

The decision to bypass the Simulink toolchain for this specific filter was driven by two primary engineering considerations. First, the native *complementaryFilter* provided by MATLAB, available both as a System object and as a Simulink block, does not implement the classic complementary architecture adopted as the baseline of this study: it implements the quaternion-based nonlinear complementary filter of Valenti et al. [26], regulated by separate accelerometer and magnetometer gains. Although the block can be restricted to a 6-DoF configuration by disabling its magnetometer input, the underlying algorithm would still differ structurally from the intended reference, namely the single-parameter (α), Euler-domain, first-order

filter described in Chapter 3, whose very simplicity is the reason for its inclusion in this comparative study. Furthermore, the block outputs the estimated attitude exclusively as quaternions or rotation matrices, which would have required cascading an additional conversion stage to extract the Roll and Pitch angles.

Second, the mathematical core of a 6-DoF Complementary Filter is incredibly lightweight, consisting of a straightforward first-order difference equation. As detailed in Chapter 3, the fusion of the high-pass filtered gyroscope data and the low-pass filtered accelerometer data is governed by a single tuning parameter, α. Generating C code for such a simple algebraic relation through *Embedded Coder* would have introduced unnecessary structural wrappers, state structures, and initialization routines, artificially inflating the memory footprint and execution time.

Consequently, the filter was manually coded directly into the firmware's C/C++ architecture. The implementation dynamically updates the Roll and Pitch angles by integrating the angular velocity over the hardware sample period Δt and applying the discrete complementary gain against the accelerometer's absolute gravity reference. The modular C function designed for this task is shown in Listing 5.1.

```
1  void update_complementary_filter(const IMUData *sensor_data, float
       *roll, float *pitch, float dt, float alpha)
2  {
3      // 1. Calculate absolute angles from accelerometer (returns
       radians)
4      float acc_roll = atan2f(sensor_data->acc.y, sensor_data->acc.z
       );
5
6      float acc_pitch = atan2f(-(sensor_data->acc.x),
7                               sqrtf(sensor_data->acc.y *
       sensor_data->acc.y +
8                                     sensor_data->acc.z *
       sensor_data->acc.z));
9
10     // 2. Convert gyroscope data from degrees/s to radians/s
11     float gyro_rad_x = sensor_data->gyro.x * DPS_TO_RADS;
12     float gyro_rad_y = sensor_data->gyro.y * DPS_TO_RADS;
13
14     // 3. Complementary Filter fusion
15     *roll = alpha * (*roll + gyro_rad_x * dt) + (1.0f - alpha) *
       acc_roll;
16     *pitch = alpha * (*pitch + gyro_rad_y * dt) + (1.0f - alpha) *
        acc_pitch;
17 }
```

Listing 5.1: Manual C implementation of the 6-DoF Complementary Filter core logic.

5.3 Firmware Architecture and Task Scheduling

The final step in the software methodology involved designing a robust real-time execution environment on the Raspberry Pi Pico W to coordinate data acquisition, sensor calibration, and the MBD auto-generated filter algorithms.

To maintain a scalable and maintainable codebase, the C/C++ project was structured using a highly modular architecture. Hardware macros, I2C pin mappings, and global tuning parameters (such as the sampling rate and IMU register map configuration) were centralized within a dedicated `config.h` header. Simultaneously, auxiliary hardware routines, including I2C bus initialization, LED status signaling, and custom data structures, were abstracted into a separate `functions.c` module. This strict separation of concerns allows the `main.c` file to act purely as a high-level scheduler, cleanly managing the auto-generated Simulink headers (e.g., `mahony_c.h`) without being cluttered by low-level hardware directives.

To ensure reliable attitude tracking, this high-level scheduler was structured around a deterministic time-triggered architecture governed by a Finite State Machine (FSM), whose complete execution lifecycle and state transitions are illustrated in Figure 5.4.

5.3.1 Real-Time Scheduling and Hardware Timers

Inertial estimation algorithms strictly rely on numerical integration; therefore, guaranteeing a constant and precise sampling period (Δt) is critical. Relying on simple delay functions within a standard infinite loop (`while(1)`) is inadequate, as the loop execution time naturally fluctuates depending on conditional branching and I/O operations.

To achieve hard real-time execution, the firmware utilizes the RP2040 hardware timer API. A repeating timer was configured using the `add_repeating_timer_ms()` function. By passing a negative value for the sampling interval (e.g., `-SAMPLE_RATE_MS`), the hardware timer is instructed to calculate the next trigger precisely from the *start* of the previous callback, effectively neutralizing any jitter caused by the code execution duration. The interrupt callback simply sets a volatile flag (`timer_fired = true`), which dictates the execution of the main algorithmic block. The total elapsed time for each cycle is constantly monitored via `time_us_32()` to strictly verify that the computational load never exceeds the hardware timing budget.

5.3.2 Sensor Data Pipeline and Pre-processing

When the timer flag is asserted, the firmware initiates a burst read over the I2C bus operating in Fast-mode. To minimize communication overhead, the 6-DoF inertial

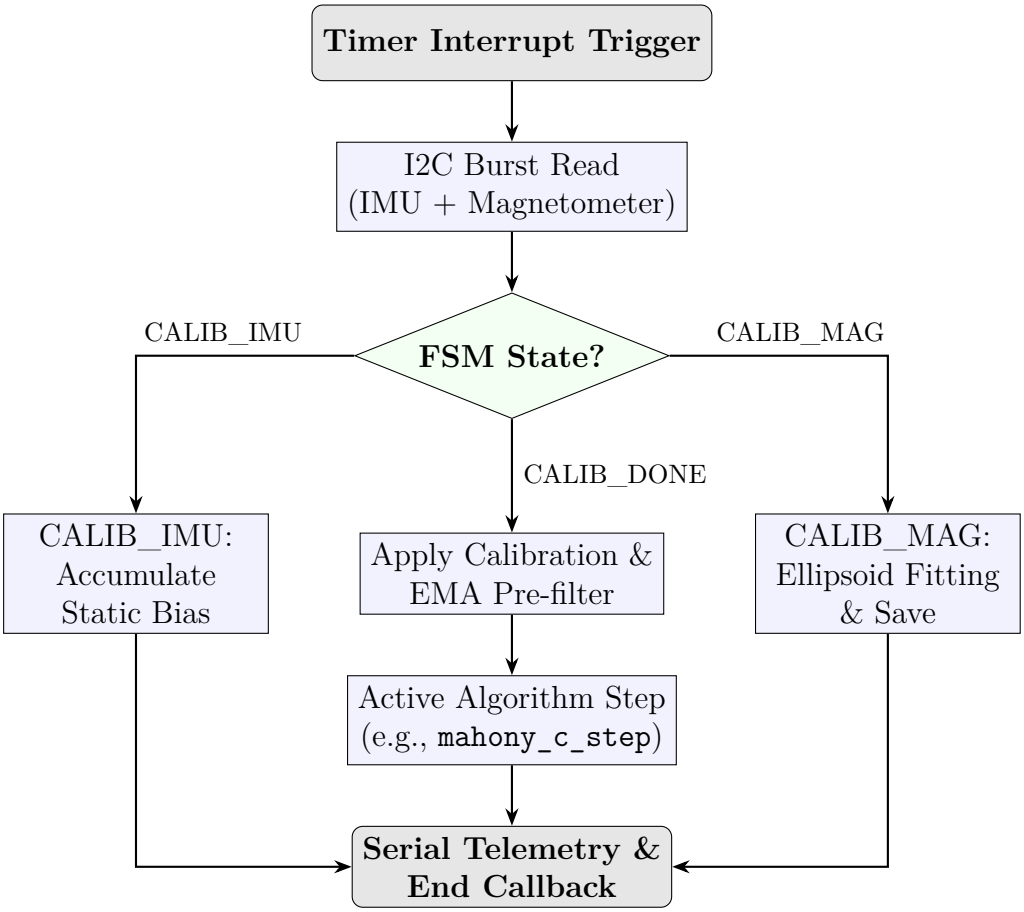


Figure 5.4: Real-time execution flowchart representing the hardware timer callback and the three-state FSM governing sensor calibration and data filtering.

data (accelerometer and gyroscope) are fetched in a single contiguous 12-byte block read, followed by an 8-byte block read for the magnetometer.

The acquired raw data flows through a Finite State Machine defining three operational phases:

1. **CALIB_IMU:** Upon startup, the system captures a predefined number of static samples to compute the gyroscope zero-offset and the accelerometer bias (compensating for the 1g gravity vector on the Z-axis).

2. **CALIB_MAG:** If the non-volatile memory (LittleFS) does not contain valid magnetic calibration parameters, the firmware prompts the user to rotate the device. It dynamically collects a spatial point cloud and allocates deterministic `double`-precision memory vectors to solve the algebraic ellipsoid fitting problem. Once the Soft-Iron matrix and Hard-Iron vector are computed,

the memory is explicitly freed (`vec_free`) to prevent heap fragmentation, and the parameters are saved to the Flash memory.

3. **CALIB_DONE:** The core operational state. The static and magnetic calibrations are applied to the raw data.

Before passing the calibrated data to the active algorithm's execution step (whether it is an MBD auto-generated `_step()` function or the manual C routine), the firmware applies an Exponential Moving Average (EMA) pre-filter. This lightweight filtering stage reduces high-frequency mechanical noise without introducing significant phase delay. The same calibrated, EMA-filtered stream feeds all four algorithms uniformly, including the EKF, and is also the stream recorded for the offline validation of Chapter 6, ensuring that every filter, both on-board and in the offline analysis, processes an identical input sequence.

5.4 Bidirectional Serial Telemetry and System Diagnostics

To evaluate the dynamic performance of the implemented algorithms and manage the embedded application, a bidirectional real-time telemetry link was established between the Raspberry Pi Pico W and a host PC via a USB Virtual COM port.

During nominal operation, at the conclusion of each hardware timer callback, the firmware streams the estimated attitude data (Roll, Pitch, and Yaw) as comma-separated values (CSV). On the host side, a custom MATLAB script intercepts this stream to drive a 3D graphical user interface. A virtual parallelepiped, mirroring the physical hardware, is rendered and continuously updated to provide intuitive visual feedback on the sensor fusion accuracy. Furthermore, this telemetry architecture doubles as a data logger: for dataset acquisition, a dedicated firmware configuration replaces the attitude stream with the calibrated and EMA-filtered sensor axes themselves, in the ten-field format described at the end of this section, which MATLAB records and subsequently processes offline to generate the time-series plots used for the experimental validation in Chapter 6.

Beyond simple data streaming, the firmware serial interface implements a lightweight Command Line Interface (CLI) to interact with the Finite State Machine at runtime, avoiding the need for continuous firmware recompilations. The system asynchronously polls the serial RX buffer for user inputs:

- **Diagnostic Mode (`'i'` command):** Triggers a comprehensive human-readable system report. This diagnostic snapshot outputs key runtime metrics, including the exact execution time of the previous algorithmic cycle (benchmarked in microseconds via `time_us_32()`), the active tuning parameters of

the selected filter (e.g., the proportional/integral gains or the complementary α factor), the computed IMU offsets, and the complete Soft/Hard-Iron magnetic calibration matrices. It also provides a snapshot of the data pipeline, displaying side-by-side the raw, calibrated, and EMA-filtered values for all 9-DoF axes.

- **Calibration Hot-Reset (`'c'` command):** Initiates a dynamic reset of the system. It physically erases the stored calibration file from the LittleFS Flash memory, resets the active filter states, clears the EMA buffers, and dynamically forces the FSM back to the `CALIB_IMU` state. This allows the user to perform a fresh hardware calibration on the fly without rebooting the microcontroller.

This interactive telemetry framework significantly accelerated the debugging phase, enabling instantaneous profiling of the execution time and straightforward validation of the sensor data integrity before proceeding with the experimental tests.

An earlier firmware iteration streamed a wider set of diagnostic columns over serial on every cycle (raw and uncalibrated magnetometer values, vector norms), in addition to the data required for offline analysis. This overhead measurably increased the real loop period to approximately 15.3 ms, exceeding the nominal 10 ms sampling period. Once identified, these diagnostic columns were removed from the per-cycle telemetry, retaining only the ten fields required by the offline pipeline (timestamp plus the nine calibrated and filtered sensor axes). To verify that this fix restored the intended sampling behavior, the native hardware timestamp embedded in every transmitted line was inspected across a 15-minute continuous recording taken with the corrected telemetry format: across all 89,999 consecutive intervals, the maximum deviation from the nominal 10 ms period was 24 μs, occurring within the very first samples of the recording, confirming that no sampling cycle is silently skipped or delayed.

Chapter 6

Experimental Results and Performance Analysis

This chapter presents the experimental evaluation and performance analysis of the implemented Attitude and Heading Reference System (AHRS). Since no external ground-truth system was available, the campaign assesses stability, return-to-start closure, inter-filter consistency, disturbance response, and embedded resource usage, rather than absolute orientation accuracy. To comprehensively evaluate the proposed firmware and the generated filter algorithms, the testing protocol was divided into two distinct operational phases. The first phase consists of an offline algorithmic validation, utilizing the calibrated, EMA-filtered sensor data recorded on the hardware and subsequently processed within the MATLAB environment. This offline approach guarantees exact repeatability, ensuring that all evaluated algorithms are exposed to the identical sensor noise, vibration profiles, and dynamic movements. The second phase shifts to online hardware profiling, executing the embedded code directly on the target microcontroller to measure the real-world computational footprint.

6.1 Experimental Setup and Methodology

The methodology adopted for evaluating the orientation algorithms relies on a comparative cross-validation across a diverse set of physical scenarios. Since an absolute ground truth from a high-precision optical tracking system was not used, the algorithms are evaluated based on the expected physical behavior of the sensor during each specific test. To achieve this, the offline testing phase utilizes five distinct datasets, carefully recorded to stress-test the algorithms under varying dynamic and environmental conditions:

- **Dataset 1 (Long-Term Static):** A 15-minute static recording designed to evaluate the baseline noise floor and characterize the long-term angular drift of each algorithm.
- **Dataset 2 (Slow Dynamic Return):** A slow, large-angle rotation returning to the exact starting position. This isolates the algorithms' tracking consistency and closure error without introducing high-frequency kinematic stress.
- **Dataset 3 (Fast Dynamic Return):** A rapid, near-inversion rotation that also returns to the initial starting position, aimed at testing the transient response and robustness against sudden linear accelerations.
- **Dataset 4 (Magnetic Disturbance):** A static test where a strong external magnet is deliberately brought near the sensor to evaluate each algorithm's ability to reject acute magnetic anomalies.
- **Dataset 5 (Post-Disturbance Static):** A final static recording taken immediately after the disturbance to analyze any residual hardware magnetization and observe how the filters process a permanently altered magnetic field.

By design, a static test inherently possesses a constant true orientation, whereas a return-to-start dynamic test requires the final computed attitude to identically match the initial state. To maintain a rigorous and unbiased comparison, all configuration parameters were kept strictly constant across these scenarios. In particular, the entire evaluation was processed using a single, fixed configuration for the Extended Kalman Filter (EKF), explicitly avoiding any per-dataset re-tuning to ensure the comparison remains on an equal footing.

For completeness and reproducibility, the offline processing chain deserves explicit mention. The recorded telemetry is first resampled onto an ideal, uniformly spaced 100 Hz time grid via linear interpolation, removing the residual timestamp jitter quantified in Chapter 5. The Madgwick, Mahony, and Complementary filters process this 100 Hz stream directly, while for the EKF the stream is decimated by a factor of two to its native 50 Hz rate; the resulting quaternion output is then brought back onto the 100 Hz grid by linear interpolation of the quaternion components (after enforcing sign continuity between consecutive samples) followed by re-normalization, a scheme known as normalized linear interpolation (nlerp), numerically indistinguishable from spherical interpolation for the small inter-sample rotations involved, so that all pairwise metrics operate on time-aligned sequences. For the two dynamic datasets, the motion window is detected automatically from the gyroscope magnitude, using a $3\,^\circ/\mathrm{s}$ threshold with a 0.3 s debounce on the motion onset, while baselines and closure errors are computed as means over 2 s static windows. For the tilt-only comparisons involving the Complementary filter, samples where either filter's pitch estimate

approaches the gimbal-lock singularity ($|\theta| \geq 80°$) are excluded, since Euler-angle differences become degenerate in that region. The fixed EKF configuration used throughout is: `AccelerometerNoise` $= 4.2983 \times 10^{-4}$, `GyroscopeNoise` $= 1 \times 10^{-3}$ (deliberately raised above its default value to shift weight from the gyroscope toward the vector observations), `MagnetometerNoise` $= 1$, `GyroscopeDriftNoise` $= 1 \times 10^{-12}$, `LinearAccelerationNoise` $= 0.1$, `LinearAccelerationDecayFactor` $= 0.1$, `MagneticDisturbanceNoise` $= 10$, and `MagneticDisturbanceDecayFactor` $= 0.1$, all expressed in the `ahrsfilter` block's native units. The only per-dataset quantity is `ExpectedMagneticFieldStrength`, which is not hand-tuned but initialized automatically from the mean magnetometer norm over the first 50 samples of each recording.

Following the offline algorithmic validation, the performance analysis transitions to the target embedded hardware. For this phase, firmware resource usage, specifically execution time per cycle and Random Access Memory (RAM) footprint, was measured directly on the RP2040 microcontroller. This on-device profiling strategy ensures that the stated resource costs reflect the true operational metrics of the embedded system, rather than relying on theoretical execution times derived from simulation environments.

6.2 Offline Algorithmic Validation

This section details the results obtained from processing the predefined datasets. The analysis isolates specific performance metrics, such as baseline noise, kinematic tracking consistency, and magnetic anomaly rejection, comparing the sensor fusion algorithms side-by-side.

6.2.1 Long-Term Static Noise and Drift Analysis

The first dataset consists of a 15-minute static recording, with 885 seconds of data analyzed after discarding the initial setup transient. This extended duration is mathematically crucial for accurately characterizing the angular drift (measured in °/h). Compared to a short 15-second window, which would require an extremely noise-sensitive 240× multiplier to extrapolate the hourly drift, the 885-second window reduces that multiplier to a much more reliable 4.07×. The test was deliberately conducted in a magnetically quiet environment to evaluate the baseline noise floor of the algorithms.

At rest, the Complementary filter ($\alpha = 0.98$) and the EKF demonstrated the highest stability on the Roll and Pitch axes, exhibiting standard deviations of approximately 0.03°. Furthermore, these two filters agreed with each other almost perfectly, presenting a mean pairwise disagreement of only 0.03°. Conversely, the Madgwick filter, evaluated at the widely used default value of the reference

open-source implementation ($\beta = 0.1$), was the noisiest of the magnetometer-based algorithms. It exhibited standard deviations roughly two to four times higher than the Mahony and EKF algorithms across all axes (e.g., 0.118° on Roll and 0.337° on Yaw). In fact, the computed drift on the Roll and Pitch axes for all evaluated filters fell completely within their respective noise floors, rendering any theoretical drift indistinguishable from baseline measurement noise on those two axes.

On the Yaw axis, however, the long-term static dataset explicitly highlighted the effectiveness of integral action in correcting slow gyroscope bias. For the Mahony algorithm, setting the integral gain to $K_i = 0.005$ reduced the yaw drift by more than half (from $-0.87°$/h to $-0.36°$/h) compared to a baseline pass with no integral term. This behavior confirms that uncorrected gyroscope bias accumulates over long, quiet windows, and the dedicated integral term successfully mitigates this effect without noticeably degrading the standard deviation on the Roll and Pitch axes.

To further investigate this bias-correction mechanism, a sensitivity check was performed on the EKF by varying its `GyroscopeDriftNoise` parameter. Raising this value from the near-frozen baseline of 1×10^{-12} to 1×10^{-6} allowed the filter to track and correct the slow bias more aggressively, successfully reducing the yaw drift by approximately 26% (from $-1.07°$/h to $-0.79°$/h). However, this improvement came at a direct cost to noise sensitivity, worsening the standard deviation across all axes by 18% to 29% (e.g., Roll noise increased from 0.028° to 0.033°). This experiment confirms a fundamental speed-versus-noise trade-off inherent to bias-correction parameters across different sensor fusion architectures: allowing a filter to track slow drift more rapidly inevitably makes it more reactive to ordinary measurement noise. Consequently, the conservative value of 1×10^{-12} was retained for the EKF to prioritize peak stability on the structurally important Roll and Pitch axes.

Strikingly, this speed-versus-noise trade-off was observed independently on two mathematically unrelated filter architectures. In both the Mahony and EKF cases, enabling bias correction on exactly one filter reduced the long-term drift of that filter alone, while the other, uncorrected algorithms remained unaffected. This consistent, isolated response on two unrelated architectures strengthens the interpretation that the observed yaw drift is attributable, at least in part, to uncorrected gyroscope bias rather than to a purely external environmental or thermal effect.

6.2.2 Dynamic Response and Kinematic Tracking

The dynamic performance of the algorithms was evaluated using two return-to-start datasets, designed to isolate tracking consistency and closure error under varying levels of kinematic stress. In these tests, the physical requirement that the sensor

Table 6.1: Standard deviation and angular drift across the four evaluated algorithms during the 15-minute static dataset.

Filter	**Std Dev (°)**			**Drift (°/h)**		
	Roll	**Pitch**	**Yaw**	**Roll**	**Pitch**	**Yaw**
Madgwick ($\beta = 0.1$)	0.118	0.101	0.337	0.10	0.10	-0.83
Mahony ($K_p = 2$, $K_i = 0.005$)	0.051	0.040	0.143	0.01	-0.02	-0.36
Complementary ($\alpha = 0.98$)	0.026	0.024	N/A	0.05	0.10	N/A
EKF (`GyroDrift` 1e-12)	0.028	0.026	0.144	0.11	0.11	-1.07

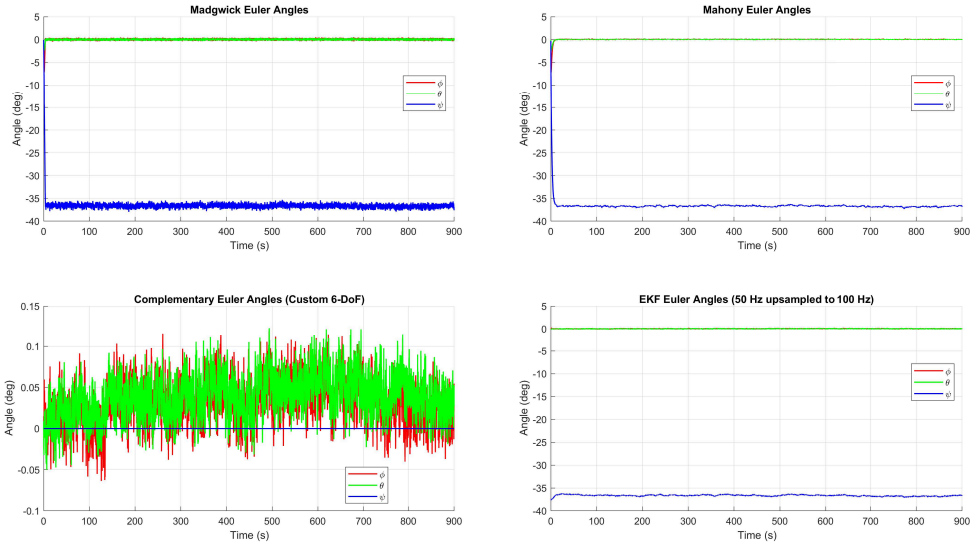


Figure 6.1: Direct comparison of the estimated Euler angles during the long-term static dataset. The raw output clearly highlights the higher baseline noise of the Madgwick algorithm and the absence of yaw estimation in the 6-DoF Complementary filter.

returns exactly to its initial orientation serves as the implicit reference for evaluating accumulated drift and transient recovery.

Dataset 2: Slow Dynamic Return

During the slow return test, the gyroscope detected a continuous motion phase lasting 42.1 s. Across this window, all four algorithms demonstrated excellent closure properties. Regardless of their distinct mathematical architectures, every

filter returned to its starting orientation within 0.09° of error across all active axes (e.g., Mahony closed with +0.08° on Roll, +0.02° on Pitch, and −0.09° on Yaw). Furthermore, no measurable settling time (0.00 s) was required by any of the implementations. Once the initial convergence transient had elapsed ($t > 15$ s, cf. Figure 6.3), the three full 9-DoF algorithms (Madgwick, Mahony, and EKF) maintained strong pairwise agreement for the remainder of the recording, with peak 3D deviations consistently tracking the identical physical event between 97.69° and 98.60°. Beyond that transient, the maximum 3D distance among these three algorithms never exceeded 3.62° (recorded between Madgwick and EKF).

Table 6.2: Closure error, peak deviation, and settling time for the four algorithms during the slow dynamic return test (Dataset 2).

Filter	**Closure Error (°)**			**Peak 3D (°)**	**Settling (s)**[†]
	Roll	**Pitch**	**Yaw**		
Madgwick	+0.06	+0.02	-0.04	98.60	0.00
Mahony	+0.08	+0.02	-0.09	98.30	0.00
Complementary	+0.08	+0.02	N/A	140.75[*]	0.00
EKF	-0.09	+0.01	+0.08	97.69	0.00

[*]Tilt-only (Roll+Pitch) peak deviation, since the Complementary filter does not estimate Yaw; not directly comparable to the true 3D peak reported for the other three algorithms.

[†]Settling time is measured from the detected end of motion until the filter's output remains within 2° of its final value for the remainder of the recording; 0.00 s indicates the output was already within tolerance as soon as motion ended.

However, the 6-DoF Complementary filter exhibited a significant transient divergence during the movement itself, reaching a tilt-only peak deviation of 140.75° relative to its own starting orientation. This is the same kind of self-referential quantity (a comparison against the filter's own start, not against another algorithm) as the 179.61° reported for this filter in the fast return test below (Table 6.3). Unlike that case, however, where the self-referential peak closely tracked the nominal ∼180° physical rotation, here it substantially overshoots the ∼98° motion consistently reported by the other three filters (Table 6.2), a further, independent symptom of the same structural limitation, visible even without comparing the Complementary filter against any other algorithm. This resulted in large pairwise disagreements with the other filters (up to ∼ 64.6° against the EKF) during the maneuver. Because the test was deliberately slow, this severe divergence is not caused by uncompensated high-frequency linear acceleration. Instead, it is directly attributable to the known structural limitation of the filter's simplified kinematics, specifically the absence of proper yaw-rate coupling when computing

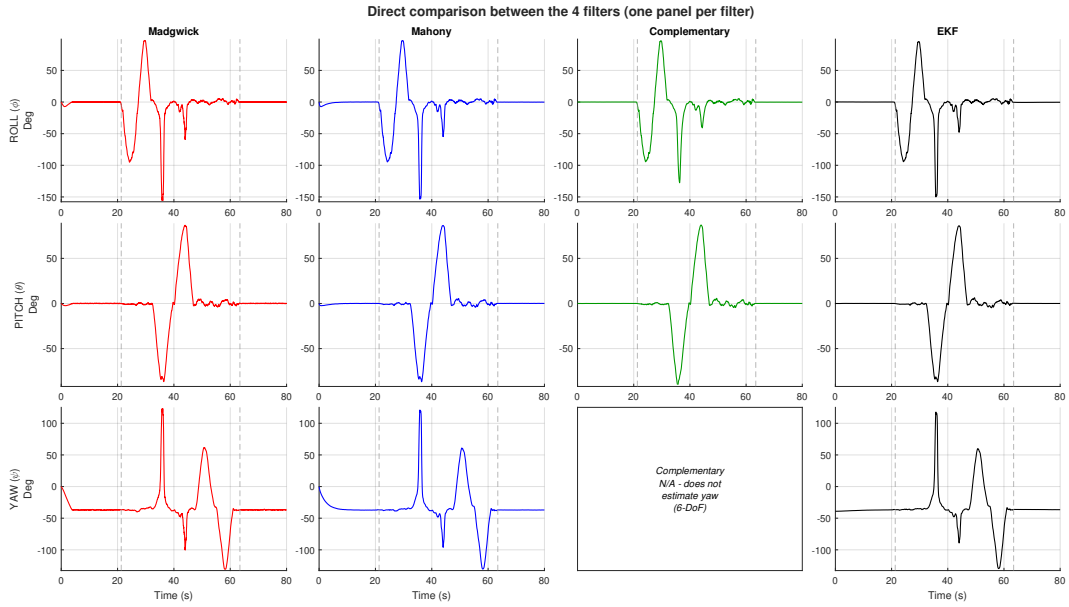


Figure 6.2: Estimated Euler angles during the slow dynamic return test (Dataset 2). The three full 9-DoF filters track the same physical rotation closely, while the Complementary filter's transient divergence is clearly visible during the motion itself.

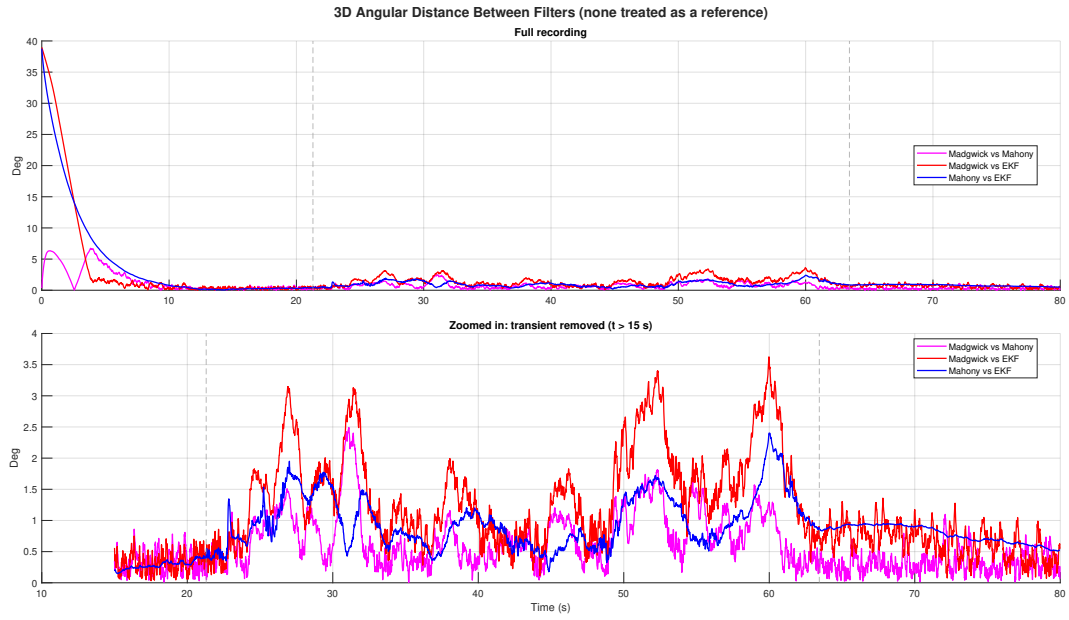


Figure 6.3: Pairwise 3D angular distance over time between the three full 9-DoF algorithms during the slow dynamic return test, quantifying the maximum disagreement of 3.62° reported in the text.

Roll and Pitch updates during large spatial rotations. Once the motion ceased, the Complementary filter smoothly re-converged, confirming no permanent error was introduced.

Dataset 3: Fast Dynamic Return

The fast return test subjected the algorithms to a much more aggressive, near-inversion rotation, with the active motion window spanning 31.7 s. All four algorithms independently registered a peak spatial deviation clustered tightly between 179.61° and 179.99°. Since the Complementary filter utilizes a fundamentally different computation path lacking magnetometer data, this unified peak provides a strong independent confirmation that the physical sensor was indeed rotated to near-inversion.

Table 6.3: Closure error, peak deviation, and settling time for the four algorithms during the fast dynamic return test (Dataset 3).

Filter	**Closure Error (°)**			**Peak 3D (°)**	**Settling (s)†**
	Roll	**Pitch**	**Yaw**		
Madgwick	-0.48	-0.03	+0.60	179.85	0.00
Mahony	-0.49	0.00	+0.53	179.99	0.00
Complementary	-0.49	-0.04	N/A	179.61*	0.00
EKF	-0.18	+0.47	-6.08	179.77	3.22

*Tilt-only (Roll+Pitch) peak deviation, since the Complementary filter does not estimate Yaw; not directly comparable to the true 3D peak reported for the other three algorithms.

†Settling time is measured from the detected end of motion until the filter's output remains within 2° of its final value for the remainder of the recording; 0.00 s indicates the output was already within tolerance as soon as motion ended.

This is a marked departure from the EKF's clean performance in the slow-return test above, where its closure error and settling time were indistinguishable from the other two full filters. The contrast suggests the degradation is specifically triggered by the higher angular rates and abrupt deceleration of this more aggressive maneuver, rather than a general weakness of the algorithm under this tuning.

While the Madgwick, Mahony, and Complementary algorithms maintained small final closure errors ($\leq 0.60°$ on all axes) and settled instantly, the EKF exhibited a noticeable degradation in tracking performance under high dynamic stress. Specifically, the EKF recorded a substantial yaw closure error of $-6.08°$ and required a measurable settling time of 3.22 s after the motion physically concluded. This slower, more oscillatory recovery also manifested in an unusual statistical

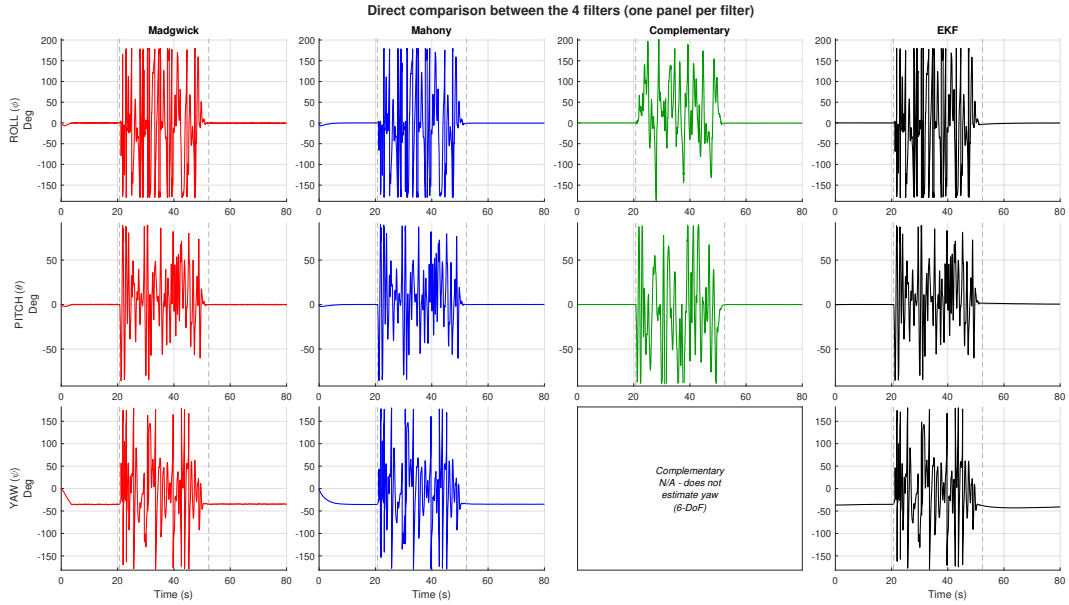


Figure 6.4: Estimated Euler angles during the fast dynamic return test (Dataset 3). All four independently reach near-inversion; the EKF's slower, more oscillatory settling after the motion concludes is visible relative to the other three.

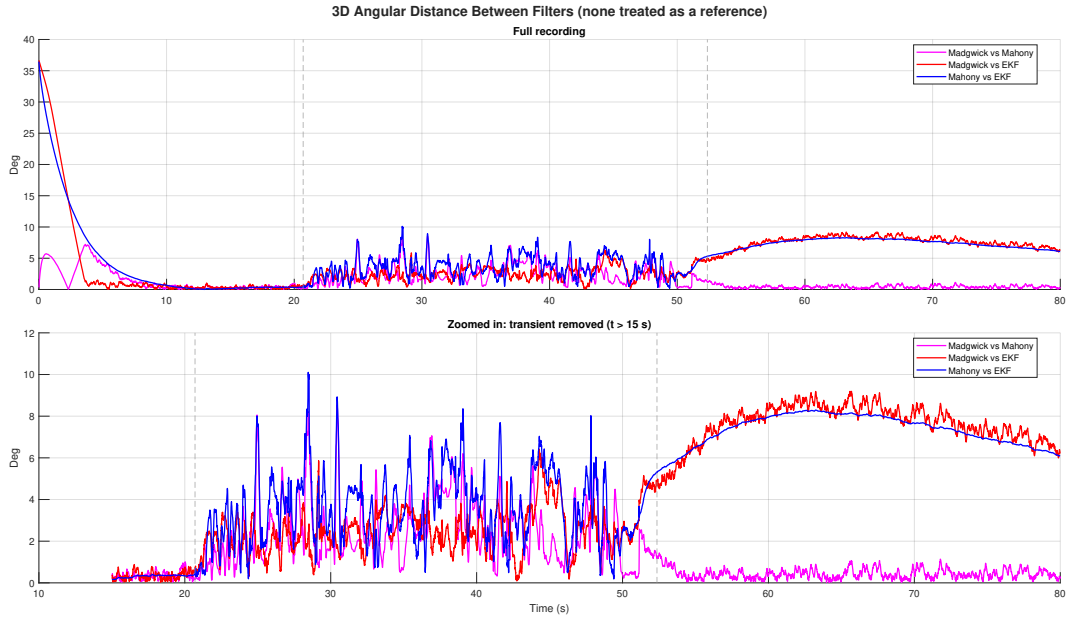


Figure 6.5: Pairwise 3D angular distance over time between the three full 9-DoF algorithms during the fast return test. The disagreement visibly peaks after the detected motion window ends, during the EKF's extended settling period, contributing to the inverse behavior described in the text.

signature: the mean pairwise disagreement between the EKF and the other full filters was actually lower during the detected motion window (e.g., 3.60° against Mahony) than over the full recording (4.90°), a statistic that, consistently with the criterion adopted throughout this analysis, already excludes the initial 15 s convergence transient. This inverse behavior occurred because a large share of the EKF's angular disagreement took place during its extended post-motion settling period, which fell outside the actively detected motion window.

Two observations are relevant to interpreting this degradation. First, unlike Madgwick, Mahony, and the Complementary filter, which weight the accelerometer through a single fixed gain regardless of the sensor's recent history, the EKF explicitly models linear acceleration as a separate state with its own decay dynamics (`LinearAccelerationDecayFactor`, Section 3.4.4). This structural difference means the EKF is the only one of the four filters that attempts to reason about how transient, non-gravitational acceleration should evolve over time, and is therefore also the only one that can be misled if that evolution does not match its underlying decay assumption, for instance in the presence of an elastic rebound of the mount or cabling following the abrupt deceleration at the end of this maneuver. Second, this same sensitivity is documented independently by MathWorks for the underlying Simulink `ahrsfilter` block: the worked example *Track Shaking 9-Axis IMU* shows the filter, at its default tuning, converging slowly to the true resting orientation after a shaking event, and resolves this specifically by adjusting `GyroscopeDriftNoise` and `LinearAccelerationNoise`, the same two parameters that govern the EKF's linear-acceleration state. This work retained a near-frozen `GyroscopeDriftNoise` (1×10^{-12}, of comparable magnitude to the default 3.0462×10^{-13}) to prioritize resting-noise stability, as established by the sensitivity analysis in Section 6.2.1; this is the same conservative choice that MathWorks' own example identifies as slow to converge after a sudden, transient motion event [27].

Furthermore, during this aggressive rotation, the Complementary filter's tilt-only pairwise comparisons against the other algorithms mathematically exceeded 180°, reaching up to 206.95°. Note that this is a distinct quantity from the 179.61° peak reported in Table 6.3: the table value measures the Complementary filter's deviation from *its own* starting orientation, whereas this pairwise figure measures the instantaneous disagreement between the Complementary filter and *another* algorithm at the same moment in time: two different comparisons that need not, and here do not, respect the same bound. This specific anomaly is a mathematical artifact of computing the Euclidean norm of two independently wrapped angular differences (Roll and Pitch), which is not naturally bounded to 180° like a true 3D spatial distance. This artifact confirms that a 2D tilt-only comparative metric becomes fundamentally unreliable during severe rotations where the Complementary filter's simplified representation diverges drastically. For a trustworthy evaluation of the true spatial disagreement during this rapid maneuver,

the peak pairwise 3D angular distances among the three complete 9-DoF algorithms remained appropriately bounded, falling between 8.5° and 10.1°.

6.2.3 Magnetic Disturbance and Anomaly Rejection

The fourth dataset evaluates the algorithms' robustness against a severe, localized magnetic anomaly. A strong external magnet was deliberately brought near the resting sensor for approximately 10 seconds. To accurately isolate this event, the detection window was strictly defined using a magnetometer norm threshold of 100 μT above the baseline.

A clear result from this test is the complete immunity of the 6-DoF Complementary filter. By structurally excluding magnetometer data, the filter maintained its Roll and Pitch estimates within a $\pm 0.1°$ margin (specifically peaking at 0.065° and 0.071°) for the entire duration of the test, up to two orders of magnitude below the deviations recorded by the two most affected algorithms (Madgwick and Mahony), and roughly twenty times below even the EKF's tilt deviations. Figure 6.6 shows this immunity at its true scale: on a shared axis with the other three algorithms (whose deviations reach tens of degrees) the Complementary trace would be compressed to an indistinguishable flat line near zero, which is why each filter is plotted on its own scale here.

For the three magnetometer-based algorithms, the disturbance triggered three distinctly different dynamic responses, summarized in Table 6.4 and visible in Figure 6.6:

- **Madgwick:** The filter responded with a slow, inertia-like linear ramp over the entire 10-second disturbance window, eventually reaching a peak yaw deviation of 92.40° and a tilt deviation of up to 14.73°. This smooth integration of the corrupted data is consistent with its fixed, low gain ($\beta = 0.1$).

- **Mahony:** Utilizing a more aggressive proportional gain ($K_p = 2$), the filter tracked the corrupted magnetic vector almost immediately. Its yaw estimate exhibited a large step-like jump within the first couple of seconds, followed by a second, smaller step near the end of the disturbance window, ultimately reaching the highest peak deviation recorded in this test at 127.68°. This rapid divergence also caused the largest pairwise disagreement recorded in the entire study among filters that share a directly comparable 3-axis orientation, with Mahony differing from the EKF by up to 128.28° (Figure 6.7); this excludes the tilt-only 2D artifact discussed above for the fast dynamic return test (Table 6.3), which is not bounded to 180° and is not a like-for-like comparison.

- **EKF:** During the true 10-second acute disturbance window, the EKF demonstrated highly effective anomaly rejection. Owing to its explicit `Magnetic`

`DisturbanceNoise` shielding term, the EKF's peak deviation remained near the noise floor across all axes (Roll 1.25°, Pitch 1.35°, Yaw 0.69°). However, a nuanced behavior emerged immediately after the acute event ended: as the magnetic field stabilized at a new corrupted baseline, the filter began trusting the compass again. Consequently, its yaw estimate drifted through a slow, S-shaped curve toward a new incorrect heading, eventually reaching a similar offset (+69.01°) as the other filters, albeit via a delayed and smoother path.

Crucially, none of the three magnetometer-based filters recovered to their original baseline within the remainder of the recording. Even 29 seconds after the physical magnet was removed, all three filters reported a residual yaw offset of approximately 67° to 69°, yielding a `RecoveryTime` of `NaN` (Not a Number).

Table 6.4: Peak deviation during the acute magnetic disturbance and recovery time for the four algorithms (Dataset 4).

Filter	**Max Roll (°)**	**Max Pitch (°)**	**Max Yaw (°)**	**Recovery (s)**
Madgwick	14.73	12.79	92.40	NaN
Mahony	10.09	12.02	127.68	NaN
Complementary	0.065	0.071	N/A	0*
EKF	1.25	1.35	0.69	NaN

*A recovery time of 0 s reflects that the Complementary filter never leaves tolerance in the first place, immune by design, not by recovering from a disturbance. `NaN` indicates the filter never returns within 2° of baseline for the remainder of the recording (see discussion below).

6.2.4 Residual Heading Shift After Magnetic Exposure

To investigate the failure to recover observed in Dataset 4, a fifth dataset was recorded immediately afterward, with the sensor perfectly static. In this post-disturbance recording, the pairwise agreement among all algorithms tightened drastically (mean disagreements $\leq 0.24°$, maximums $\leq 0.71°$), confirming that the sensor was genuinely at rest and the acute external disturbance was no longer present.

Despite being physically stationary, the baseline heading was permanently shifted. All four computational paths, sharing the same magnetometer measurements, reported this shift, summarized in Table 6.5: the Madgwick algorithm reported a permanent yaw offset of +67.32°, the Mahony filter +67.29°, the EKF +67.09°, and a filter-independent, raw tilt-compensated compass calculation yielded +67.37° against the pre-disturbance baseline.

These four measurements, agreeing within a tight 0.28° spread, localize the shift in the magnetometer data itself: the lack of recovery was not an algorithmic failure

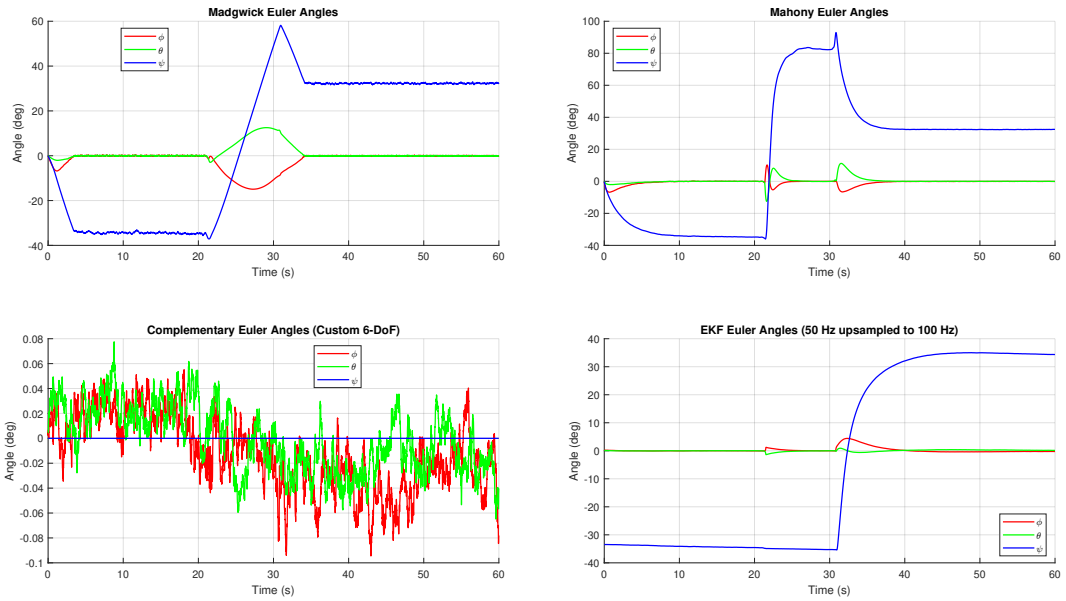


Figure 6.6: Per-filter Euler angles during the magnetic disturbance test (individual scales). The plot highlights the Complementary filter's immunity ($\pm 0.1°$) alongside the distinct divergence paths of the magnetometer-based algorithms.

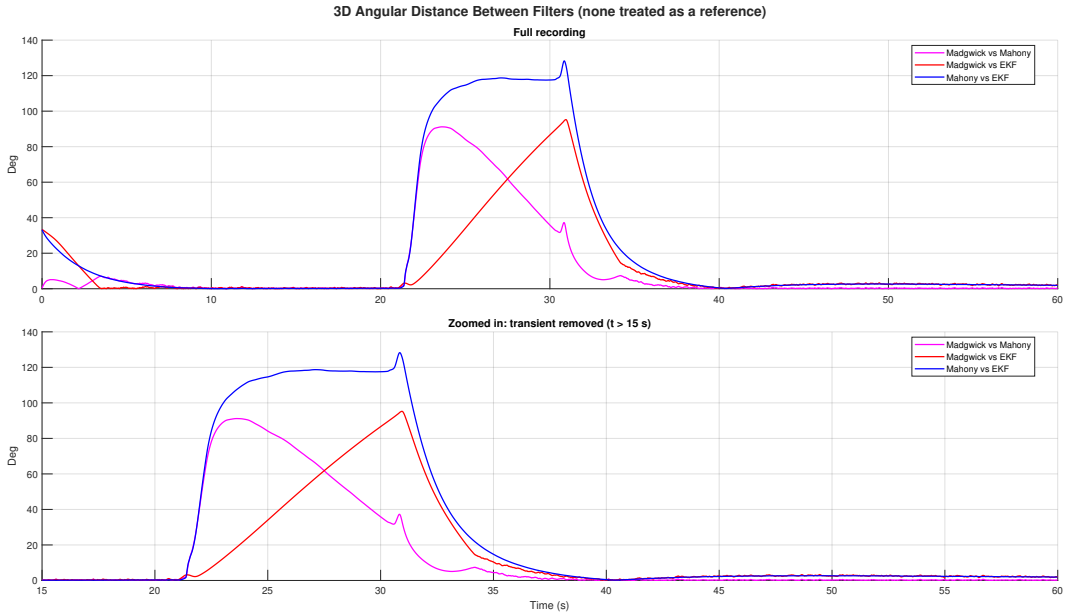


Figure 6.7: Pairwise 3D angular distance over time during the magnetic disturbance test, quantifying the largest inter-filter disagreement recorded anywhere in this study among filters with a directly comparable 3-axis orientation (Mahony vs. EKF, up to 128.28°).

Table 6.5: Permanent yaw offset relative to the pre-disturbance baseline, measured independently by four different computational paths (Dataset 5).

Method	Permanent Yaw Offset (°)
Madgwick	+67.32
Mahony	+67.29
EKF	+67.09
Raw magnetometer (filter-independent)	+67.37

or a filter artifact. Instead, the intense exposure in Dataset 4 persistently altered the magnetic field perceived by the sensor, consistent with a modified hard-iron condition or remanent magnetization of the sensor assembly, shifting the perceived North by roughly 67° until a new calibration (dynamic ellipsoid fitting) could be performed. The available measurements do not uniquely identify the underlying physical mechanism.

6.3 Online Hardware Profiling (RP2040)

While the offline algorithmic validation establishes the mathematical tracking capabilities of the algorithms, deploying these sensor fusion techniques on an embedded system requires a strict evaluation of their computational demands. To capture the true real-world resource cost, the execution time and Random Access Memory (RAM) footprint were measured directly on the target Raspberry Pi Pico W hardware, rather than relying on software-based simulations.

6.3.1 Execution Time Analysis

The execution time was profiled on a per-cycle basis. The measured duration encompasses the entire active data pipeline: the I2C sensor burst read, algebraic calibration application, EMA pre-filtering, the core algorithm's `_step()` execution, and data logging overhead.

As shown in Table 6.6, the Complementary, Mahony, and Madgwick filters operated at 100 Hz (10 ms period), consuming roughly 25% to 38% of their available cycle time. The Extended Kalman Filter (EKF), instead, required an average of 13.3 ms per cycle: as a deliberate design choice mirrored in the offline analysis, it was run at half the sample rate (50 Hz, 20 ms period) to accommodate its computational load. Even with this larger budget, the EKF consumed about 66% of the available cycle time, emerging as the most computationally expensive architecture by a significant margin.

Table 6.6: Per-cycle execution time and time budget utilization measured on the RP2040 microcontroller.

Algorithm	Sample Period	Execution Time	Time Budget Used
Complementary	10 ms (100 Hz)	$\sim$ 2.5 ms	$\sim$ 25%
Mahony	10 ms (100 Hz)	$\sim$ 3.7 ms	$\sim$ 37%
Madgwick	10 ms (100 Hz)	$\sim$ 3.8 ms	$\sim$ 38%
EKF	20 ms (50 Hz)	$\sim$ 13.3 ms	$\sim$ 66%

6.3.2 Memory Footprint

The static RAM allocation was extracted from the compiled firmware's linker map (`.data` and `.bss` sections), isolating the algorithm's internal state from the shared infrastructure overhead, near-identical across the magnetometer-based implementations.

Table 6.7: Static RAM allocation extracted from the compiled firmware linker map.

Algorithm	Total Static RAM	Algorithm's Own State
Complementary	10,856 B	0 B (Stack-only)
Mahony	13,736 B	376 B
Madgwick	13,904 B	544 B
EKF	19,480 B	6,112 B

The memory analysis in Table 6.7 reveals distinct architectural characteristics. The manually implemented Complementary filter carries zero static RAM footprint for its own state, as its roll and pitch estimates reside entirely in local stack variables. Its lower build total, however, is not a property of the algorithm alone: net of each filter's state memory, the three Simulink-generated builds share a near-identical infrastructure of approximately 13,360 B, against 10,856 B for the Complementary build, a gap due to the absence of the magnetometer calibration structures and of the Simulink runtime support. Conversely, the EKF demands 6,112 B for its internal state memory, over 11 times larger than Madgwick's state and 16 times larger than Mahony's. This increase is consistent with a Kalman filter's covariance matrices, which scale with the square of the state dimension. Finally, the slight discrepancy between Madgwick (544 B) and Mahony (376 B) is attributable to the Simulink-generated block-state storage rather than to any inherent complexity of the Madgwick algorithm.

Chapter 7

Conclusions and Future Perspectives

This thesis has investigated the performance, computational efficiency, and robustness of four distinct sensor fusion algorithms (Complementary, Madgwick, Mahony, and the Extended Kalman Filter, or EKF), targeted for deployment on resource-constrained embedded systems. By conducting an extensive experimental campaign on the Raspberry Pi Pico W (RP2040) and performing cross-algorithm comparative analysis, this research has provided a clear mapping of the trade-offs inherent in modern AHRS design, without relying on an external ground truth: each test's own physical structure (a constant orientation, a return to the starting position, a controlled disturbance) served as the implicit reference against which the four algorithms were compared to one another.

7.1 Summary of Experimental Findings

The experimental results have yielded four primary insights into the behavior of orientation estimation algorithms:

- **Algorithmic and Hardware Response to Magnetic Anomalies:** The study demonstrated that the Complementary filter's structural exclusion of the magnetometer keeps its deviations under a localized magnetic disturbance up to two orders of magnitude smaller than those of the two most affected algorithms (Madgwick and Mahony). However, cross-referencing four independent computational paths (three filters plus a raw, filter-independent compass calculation) revealed that the failure to recover baseline heading after severe magnetic exposure was a physical property of the sensor hardware, affecting

all four estimation paths essentially identically, rather than an algorithmic artifact specific to any one of them.

- **Kinematic Tracking Under Dynamic Stress:** In the slow return-to-start test, all four algorithms closed to within 0.09° of their starting orientation; during the motion itself, however, the 6-DoF Complementary filter, lacking yaw-rate coupling in its simplified kinematics, exhibited a much larger transient divergence than the three full 9-DoF filters (Madgwick, Mahony, EKF), consistent with its known structural limitation. Under a much faster, near-inversion rotation, Madgwick, Mahony, and Complementary all closed to within 0.60°, but the EKF alone exhibited a substantial residual yaw closure error (−6.08°) and a measurable settling time, a degradation not observed in the slower maneuver.
- **Isolating Gyroscope Bias via Controlled Parameter Variation:** A further methodological contribution of this work was the use of targeted, single-parameter sensitivity experiments to help distinguish the source of the small long-term yaw drift observed at rest. By enabling bias-correction mechanisms on exactly one filter at a time (Mahony's integral gain K_i, and separately the EKF's `GyroscopeDriftNoise`), the study observed that only the filter under test showed a corresponding reduction in its own drift, while the other, unmodified algorithms remained unaffected. This consistent, isolated response, reproduced on two mathematically unrelated architectures, is evidence that the drift originates at least in part from uncorrected gyroscope bias rather than a purely external environmental effect, a distinction that a single-filter analysis could not have established.
- **Computational Efficiency and Real-World Constraints:** The hardware profiling on the RP2040 revealed the practical limitations of sophisticated estimation. While the EKF provides advanced state estimation, its filter-state static RAM allocation (over 11 times larger than Madgwick's) and computational cost (requiring a 50 Hz sampling rate versus the 100 Hz possible for lighter filters) represent a significant design penalty.

7.2 Engineering Trade-offs and Selection Criteria

The data collected in this research supports a multi-criteria approach to selecting an orientation algorithm for embedded deployment. Engineering selection should not be based solely on mathematical accuracy, but rather on the operational environment:

- For applications prioritizing **minimal computational load and RAM usage** in electromagnetically noisy environments, the Complementary filter

remains the superior choice, provided that the system does not require yaw estimation.

- For applications requiring **full 9-DoF heading estimation with a lean computational profile**, Madgwick and Mahony are both viable at 100 Hz on the RP2040, but they are not interchangeable: Madgwick's fixed low gain gives a smoother, more predictable response but roughly twice the resting noise of Mahony; Mahony's long-term yaw drift at rest can be more than halved with the addition of a small integral term ($K_i = 0.005$: $-0.87°$/h down to $-0.36°$/h), but it also tracks a corrupted magnetic reading faster and more aggressively during a disturbance, reaching the largest peak deviation recorded in the magnetic disturbance test specifically. The choice between them is a choice between resting-noise stability and disturbance restraint, not a simple ranking of one over the other.

- For **high-precision systems with sufficient computational headroom**, the EKF offers the best rejection of *acute* magnetic disturbances by a wide margin, remaining near its own noise floor throughout an event that pushed the other filters tens of degrees off course. This advantage is specific to transient anomalies, however: once the sensor's own magnetic calibration was permanently altered by the disturbance, the EKF converged to the same incorrect heading as the other algorithms, just later and via a smoother path. None of the three magnetometer-based filters was immune to that persistent shift, which originates in the magnetometer data itself rather than in any individual algorithm. A second caveat is specific to fast, large-angle rotations: in the near-inversion return test, the EKF, in the specific configuration evaluated in this work (conservative parameter tuning), was the only algorithm that failed to fully close its yaw estimate back to the starting orientation (a $-6.08°$ residual, against $\leq 0.60°$ for the other three filters), together with a measurable settling time absent in the others. With the tuning used in this study, the EKF's acute-disturbance rejection did not extend to fast kinematic transients, an important consideration for applications involving abrupt, wide-range movements.

7.3 Limitations and Threats to Validity

The conclusions of this work should be read within the boundaries of its experimental protocol. First, no external ground-truth system (such as an optical tracker or a calibrated mechanical rig) was available: the campaign therefore assesses stability, return-to-start closure, inter-filter consistency, disturbance response, and embedded resource usage, rather than absolute orientation accuracy along the trajectory.

Second, all measurements were collected on a single specimen of the ICM-20948 sensor, so unit-to-unit variability is not captured. Third, each test condition corresponds to a single recording; without repeated acquisitions, the dispersion of the reported metrics cannot be quantified. Fourth, the EKF executes at 50 Hz while the other algorithms run at 100 Hz, and its output is brought back onto the common grid by nlerp interpolation: its dynamic results therefore reflect the interaction of algorithm, tuning, execution rate, and post-processing, rather than the algorithm in isolation. Finally, the filter parameters were selected through targeted single-parameter experiments rather than a common optimization procedure against reference data, as would be required for a formal accuracy claim. These limitations chart the natural extensions of this work, addressed in the next section.

7.4 Future Perspectives

While this work has established a baseline for AHRS performance on the RP2040, several avenues remain open for future development:

1. **Dynamic Online Calibration:** Future iterations of this firmware should integrate dynamic ellipsoid fitting, re-triggered periodically or on demand rather than only once at startup. As demonstrated by the residual hardware magnetization observed after Dataset 4, a static, one-time calibration is insufficient for systems that may be exposed to fluctuating magnetic environments over their operating lifetime.

2. **Targeted Gyroscope Bias Compensation:** Since this work found evidence that the static yaw drift is linked, at least in part, to uncorrected gyroscope bias rather than to the magnetometer or the environment, a natural next step is to characterize that bias more directly, for instance through temperature-dependent calibration or a longer, dedicated Allan-variance-style characterization of the ICM-20948, rather than relying solely on each filter's own generic bias-correction gain.

3. **Optimization for Micro-Architecture:** The implementation of the EKF could be further optimized using fixed-point arithmetic (or Q-format numbers) to reduce the burden of software-emulated floating-point arithmetic on the RP2040's Cortex-M0+ cores, which lack a hardware floating-point unit, potentially allowing for 100 Hz operation while maintaining current accuracy.

4. **Validation on Real Human Motion:** The experimental campaign in this work relies on controlled, repeatable bench-top stimuli (static holds, prescribed return-to-start rotations, an isolated magnetic disturbance), a deliberate choice that enabled a fair, reproducible comparison across the four algorithms under

identical conditions. Confirming that the trade-offs identified here transfer to the continuous, less structured motion patterns of actual wearable use, such as walking, arm swing, or sudden gestures, is a natural and necessary extension of this work.

7.5 Concluding Remarks

The results of this study confirm that there is no "one-size-fits-all" solution in sensor fusion. The optimal algorithm is defined by the intersection of computational constraints, hardware limitations, and environmental noise profiles, and in at least one respect (recovery from a persistent magnetometer heading shift, compatible with a modified hard-iron condition or a remanent magnetization of the sensor assembly) no amount of tuning could help the three magnetometer-based filters compensate on their own; only a hardware-aware recalibration strategy, or forgoing the magnetometer entirely as the Complementary filter does, could address it. This thesis has provided both the analytical framework and the empirical evidence required to make informed decisions when deploying orientation estimation in real-world embedded applications.

Appendix A

Embedded Implementation Highlights

This appendix provides key excerpts from the embedded C codebase implemented on the Raspberry Pi Pico W. These snippets highlight the mathematical complexity of the on-board magnetometer calibration, which was executed without the use of high-level mathematical libraries to minimize the memory footprint and optimize execution time.

A.1 Algebraic Ellipsoid Fitting

The `ellipsoid_fit` function is the core of the calibration algorithm. It solves the least-squares fitting problem by constructing the scatter matrix $\mathbf{S}$, partitioning it, and computing the eigenvalues to find the optimal algebraic parameters of the deformed ellipsoid.

```
1  Ellipsoid_t ellipsoid_fit(Vector x, Vector y, Vector z) {
2      Matrix dataM = ellipsoid_generate_data_mat(x, y, z);
3      Matrix invC = ellipsoid_generate_inverse_restrain_mat();
4      Matrix S = mat_multiply_MMT(dataM);
5      mat_free(dataM);
6
7      // Matrix partitioning for the least-squares problem
8      Matrix S11 = mat_copy_submat(S, 0, 0, 6, 6);
9      Matrix S12 = mat_copy_submat(S, 0, 6, 6, 4);
10     Matrix S21 = mat_copy_submat(S, 6, 0, 4, 6);
11     Matrix invS22 = mat_copy_submat(S, 6, 6, 4, 4);
12     mat_inv_4x4(invS22);
13     mat_free(S);
14
15     // Compute M = invC * (S11 - S12 * invS22 * S21)
```

```
    Matrix invS22_S21 = mat_multiply(invS22, S21);
    mat_free(S21);
    mat_free(invS22);
    Matrix S12_invS22_S21 = mat_multiply(S12, invS22_S21);
    mat_free(S12);
    mat_sub(S11, S12_invS22_S21);
    mat_free(S12_invS22_S21);
    Matrix M = mat_multiply(invC, S11);
    mat_free(S11);
    mat_from_array_free(invC);

    // Solve for eigenvalues to extract ellipsoid coefficients
    Eigen_t eig = eig_solve(M);
    mat_free(M);
    Ellipsoid_t res;
    res.coefA = eig_eigvec_of_largest_eigval(eig);
    eig_free(eig);
    res.coefB = mat_multiply_vec(invS22_S21, res.coefA);
    vec_negate(res.coefB);
    mat_free(invS22_S21);

    return res;
}
```

Listing A.1: Core algebraic ellipsoid fitting via matrix partitioning and eigenvalue decomposition.

A.2 Soft-Iron Compensation Matrix Calculation

Once the algebraic coefficients are found, they must be converted into the Soft-Iron compensation matrix (W). The `calib_calibrate_rotation` function applies an eigenvalue decomposition on the 3×3 ellipsoid matrix to align the principal axes and scale them to a perfect sphere.

```
Matrix calib_calibrate_rotation(Matrix ellipMat, Vector prq,
    double d) {
    Eigen_t eig = eig_solve(ellipMat);
    mat_orthogonalize(eig.eigenvectors);  // Compensating
    inaccuracy
    Matrix rotation = mat_transpose(eig.eigenvectors);
    Matrix rotatedEllipMat = mat_multiply(rotation, ellipMat);
    Matrix diagonizedEllipMat = mat_multiply(rotatedEllipMat, eig.
    eigenvectors);
    mat_free(rotatedEllipMat);

    Vector rotPrq = vec_multiply_mat(prq, eig.eigenvectors);

```

```
11      // Scaling gains derivation along principal axes
12      double a_ = MAT_ELEM(diagonizedEllipMat, 0, 0);
13      double b_ = MAT_ELEM(diagonizedEllipMat, 1, 1);
14      double c_ = MAT_ELEM(diagonizedEllipMat, 2, 2);
15      mat_free(diagonizedEllipMat);
16
17      double p_ = VEC_ELEM(rotPrq, 0);
18      double q_ = VEC_ELEM(rotPrq, 1);
19      double r_ = VEC_ELEM(rotPrq, 2);
20      vec_free(rotPrq);
21
22      Matrix gain = mat_new(3, 3);
23      MAT_ELEM(gain , 0, 0) = 1/sqrt((p_*p_)/(a_*a_) + (q_*q_)/(a_*
        b_) + (r_*r_)/(a_*c_) - d/a_);
24      MAT_ELEM(gain , 1, 1) = 1/sqrt((p_*p_)/(a_*b_) + (q_*q_)/(b_*
        b_) + (r_*r_)/(b_*c_) - d/b_);
25      MAT_ELEM(gain , 2, 2) = 1/sqrt((p_*p_)/(a_*c_) + (q_*q_)/(b_*
        c_) + (r_*r_)/(c_*c_) - d/c_);
26
27      // Re-apply rotation to form the final W matrix
28      Matrix gain_Rotation = mat_multiply(gain, rotation);
29      Matrix res = mat_multiply(eig.eigenvectors, gain_Rotation);
30
31      eig_free(eig);
32      mat_free(gain_Rotation);
33      mat_free(gain);
34      mat_free(rotation);
35
36      return res;
37  }
```

Listing A.2: Derivation of the Soft-Iron compensation matrix using orthogonalization and eigenvalue scaling.

A.3 Hard-Iron Offset Extraction and Final Assembly

While the Soft-Iron distortion is represented by the deformation of the sphere, the Hard-Iron distortion is represented by its spatial translation. The `calib_calibrate_offset` function computes this translation vector (V) by finding the geometric center of the fitted ellipsoid. Finally, the `calib_calibrate_sensor` wrapper function orchestrates the entire process, yielding the final calibration structure ready to be stored in the non-volatile memory.

```
1   Vector calib_calibrate_offset(Matrix ellipMat, Vector prq) {
```

```
    // The center of the ellipsoid is found by inverting the 3x3 matrix
    // and multiplying it by the linear coefficients vector.
    mat_inv_3x3(ellipMat);
    Vector res = mat_multiply_vec(ellipMat, prq);
    vec_negate(res);
    return res; // Returns the Hard-Iron offset vector (V)
}

Callibration_t calib_calibrate_sensor(Vector x, Vector y, Vector z) {
    Callibration_t res;
    res.offset = (Vector)0;
    res.transform = (Matrix)0;

    if (x->size < 6) return res; // Minimum points required for an ellipsoid

    // Step 1: Fit the algebraic ellipsoid
    Ellipsoid_t ellip = ellipsoid_fit(x, y, z);
    Matrix M = calib_ellipsoid_matrix(ellip.coefA);
    Vector prq = vec_copy_subvec(ellip.coefB, 0, 3);
    double d = ellip.coefB->data[3];
    ellipsoid_free(ellip);

    // Step 2: Compute Soft-Iron Matrix (W) and Hard-Iron Vector (V)
    res.transform = calib_calibrate_rotation(M, prq, d);
    res.offset = calib_calibrate_offset(M, prq);

    mat_free(M);
    vec_free(prq);
    return res; // Final calibration structure
}
```

Listing A.3: Computation of the Hard-Iron offset vector and calibration structure assembly.

A.4 Physical Scale Correction

As discussed in Chapter 4, the ellipsoid-fitting library adopted for `calib_calibrate_rotation` was empirically found, during validation, to normalize its output to an arbitrary radius close to unity, rather than to the true local magnetic field strength implied by the raw data. Left uncorrected, this discards the physical magnitude of the calibrated field entirely.

Unlike the routines above, which belong to the standalone calibration library, the following snippet is not a separate function: it is executed directly inside the firmware's `main.c`, immediately after `calib _calibrate_sensor` returns and still within the calibration state of the main loop. It measures the discrepancy directly on the same samples used for the fit, by comparing the mean norm of the hard-iron-corrected samples (before the Soft-Iron matrix is applied) against the mean norm the fitted matrix actually produces on those samples, and rescales the entire Soft-Iron matrix by the ratio of the two. Because matrix-vector multiplication is linear, scaling the matrix by a scalar factor scales its output by the same factor, restoring the calibrated data to the sensor's true physical scale (in μT) without altering the shape correction already achieved by the fit.

```
1  float fit_native_radius = 0.0f;
2  float target_radius = 0.0f;
3
4  for (int i = 0; i < MAG_SAMPLE; i++)
5  {
6      float rx = mag_sample_calibration[i].x - calibration_data.
       magneto_hard.x;
7      float ry = mag_sample_calibration[i].y - calibration_data.
       magneto_hard.y;
8      float rz = mag_sample_calibration[i].z - calibration_data.
       magneto_hard.z;
9
10     // Norm of the hard-iron-corrected sample, before the Soft-
       Iron fit
11     target_radius += sqrtf(rx * rx + ry * ry + rz * rz);
12
13     // Norm the fitted Soft-Iron matrix produces on this same
       sample
14     float cx = calibration_data.magneto_soft[0][0] * rx
15              + calibration_data.magneto_soft[0][1] * ry
16              + calibration_data.magneto_soft[0][2] * rz;
17     float cy = calibration_data.magneto_soft[1][0] * rx
18              + calibration_data.magneto_soft[1][1] * ry
19              + calibration_data.magneto_soft[1][2] * rz;
20     float cz = calibration_data.magneto_soft[2][0] * rx
21              + calibration_data.magneto_soft[2][1] * ry
22              + calibration_data.magneto_soft[2][2] * rz;
23     fit_native_radius += sqrtf(cx * cx + cy * cy + cz * cz);
24 }
25 fit_native_radius /= (float)MAG_SAMPLE;
26 target_radius /= (float)MAG_SAMPLE;
27
28 // Rescale the entire Soft-Iron matrix by the ratio of the two
       radii
29 if (fit_native_radius > 1e-6f)
30 {
```

```
31      float rescale_factor = target_radius / fit_native_radius;
32      for (int i = 0; i < 3; i++)
33          for (int j = 0; j < 3; j++)
34              calibration_data.magneto_soft[i][j] *= rescale_factor;
35  }
```

Listing A.4: Physical scale correction applied to the fitted Soft-Iron matrix (snippet from `main.c`, not a standalone function), measured empirically against the hard-iron-corrected calibration samples.

Appendix B

Mahony and Madgwick Filter MATLAB Implementation

This appendix contains the procedural MATLAB code implemented within the Simulink *MATLAB Function* block for the Mahony and Madgwick filters. The algorithm was refactored from class-based structures to a procedural flow to ensure compatibility with the *Embedded Coder* generation pipeline. The helper routines `quaternProd`, `quaternConj`, and `quatern2euler` used in both listings are the standard quaternion utilities distributed alongside the original open-source implementations from which the filters were refactored.

```
1  function [quat, eul]  = MahonyAHRS(Gyroscope, Accelerometer,
       Magnetometer, Kp, Ki, SamplePeriod)
2  Gyroscope = (Gyroscope(:)') * (pi/180);
3  Accelerometer = Accelerometer(:)';
4  Magnetometer  = Magnetometer(:)';
5
6  persistent q;
7  persistent eInt;
8
9  if isempty(q)
10     q = [1 0 0 0];
11 end
12 if isempty(eInt)
13     eInt = [0 0 0];
14 end
15
16 quat = q;
17 eul = quatern2euler(quaternConj(q)) * (180/pi);
18
```

```
19 % Normalise accelerometer measurement
20 if(norm(Accelerometer) == 0), return; end   % handle NaN
21 Accelerometer = Accelerometer / norm(Accelerometer);    %
       normalise magnitude
22
23 % Normalise magnetometer measurement
24 if(norm(Magnetometer) == 0), return; end    % handle NaN
25 Magnetometer = Magnetometer / norm(Magnetometer);   % normalise
       magnitude
26
27 % Reference direction of Earth's magnetic field
28 h = quaternProd(q, quaternProd([0 Magnetometer], quaternConj(q)));
29 b = [0 norm([h(2) h(3)]) 0 h(4)];
30
31 % Estimated direction of gravity and magnetic field
32 v = [2*(q(2)*q(4) - q(1)*q(3))
33     2*(q(1)*q(2) + q(3)*q(4))
34     q(1)^2 - q(2)^2 - q(3)^2 + q(4)^2];
35
36 w = [2*b(2)*(0.5 - q(3)^2 - q(4)^2) + 2*b(4)*(q(2)*q(4) - q(1)*q
       (3))
37     2*b(2)*(q(2)*q(3) - q(1)*q(4)) + 2*b(4)*(q(1)*q(2) + q(3)*q(4)
       )
38     2*b(2)*(q(1)*q(3) + q(2)*q(4)) + 2*b(4)*(0.5 - q(2)^2 - q(3)
       ^2)];
39
40 % Error is sum of cross product between estimated direction and
       measured direction of fields
41 e = cross(Accelerometer, v) + cross(Magnetometer, w);
42
43 if(Ki > 0)
44     eInt = eInt + e * SamplePeriod;
45 else
46     eInt = [0 0 0];
47 end
48
49 % Apply feedback terms
50 Gyroscope = Gyroscope + Kp * e + Ki * eInt;
51
52 % Compute rate of change of quaternion
53 qDot = 0.5 * quaternProd(q, [0 Gyroscope(1) Gyroscope(2) Gyroscope
       (3)]);
54
55 % Integrate to yield quaternion
56 q = q + qDot * SamplePeriod;
57 q = q / norm(q); % normalise quaternion
58
59 quat = q;
```

```
60 eul = quatern2euler(quaternConj(q)) * (180/pi); % use conjugate
       for sensor frame relative to Earth and convert to degrees.
61 end
```

Listing B.1: Procedural MATLAB implementation of the Mahony filter for Simulink.

```
1  function [quat, eul]  = MadgwickAHRS(Gyroscope, Accelerometer,
       Magnetometer, Beta, SamplePeriod)
2
3  Gyroscope = (Gyroscope(:)') * (pi/180);
4  Accelerometer = Accelerometer(:)';
5  Magnetometer  = Magnetometer(:)';
6
7  persistent q;
8
9  if isempty(q)
10     q = [1 0 0 0];
11 end
12
13 quat = q;
14 eul = quatern2euler(quaternConj(q)) * (180/pi);
15
16 % Normalise accelerometer measurement
17 if(norm(Accelerometer) == 0), return; end % handle NaN
18 Accelerometer = Accelerometer / norm(Accelerometer);  % normalise
       magnitude
19
20 % Normalise magnetometer measurement
21 if(norm(Magnetometer) == 0), return; end  % handle NaN
22 Magnetometer = Magnetometer / norm(Magnetometer); % normalise
       magnitude
23
24 % Reference direction of Earth's magnetic field
25 h = quaternProd(q, quaternProd([0 Magnetometer], quaternConj(q)));
26 b = [0 norm([h(2) h(3)]) 0 h(4)];
27
28 % Gradient descent algorithm corrective step
29 F = [2*(q(2)*q(4) - q(1)*q(3)) - Accelerometer(1)
30      2*(q(1)*q(2) + q(3)*q(4)) - Accelerometer(2)
31      2*(0.5 - q(2)^2 - q(3)^2) - Accelerometer(3)
32      2*b(2)*(0.5 - q(3)^2 - q(4)^2) + 2*b(4)*(q(2)*q(4) - q(1)*q(3)
       ) - Magnetometer(1)
33      2*b(2)*(q(2)*q(3) - q(1)*q(4)) + 2*b(4)*(q(1)*q(2) + q(3)*q(4)
       ) - Magnetometer(2)
34      2*b(2)*(q(1)*q(3) + q(2)*q(4)) + 2*b(4)*(0.5 - q(2)^2 - q(3)
       ^2) - Magnetometer(3)];
35 J = [-2*q(3),                    2*q(4),                    -2*q(1)
       ,                          2*q(2)
```

```
36      2*q(2),                    2*q(1),                    2*q(4),
                                2*q(3)
37      0,                          -4*q(2),                    -4*q(3)
        ,                         0
38      -2*b(4)*q(3),               2*b(4)*q(4),               -4*b(2)
        *q(3)-2*b(4)*q(1),        -4*b(2)*q(4)+2*b(4)*q(2)
39      -2*b(2)*q(4)+2*b(4)*q(2), 2*b(2)*q(3)+2*b(4)*q(1),  2*b(2)*q
        (2)+2*b(4)*q(4),       -2*b(2)*q(1)+2*b(4)*q(3)
40      2*b(2)*q(3),               2*b(2)*q(4)-4*b(4)*q(2),  2*b(2)*q
        (1)-4*b(4)*q(3),        2*b(2)*q(2)];
41  step = (J'*F);
42  step = step / norm(step); % normalise step magnitude
43
44  % Compute rate of change of quaternion
45  qDot = 0.5 * quaternProd(q, [0 Gyroscope(1) Gyroscope(2) Gyroscope
        (3)]) - Beta * step';
46
47  % Integrate to yield quaternion
48  q = q + qDot * SamplePeriod;
49  q = q / norm(q); % normalise quaternion
50
51  quat = q;
52  eul = quatern2euler(quaternConj(q)) * (180/pi); % use conjugate
        for sensor frame relative to Earth and convert to degrees.
53  end
```

Listing B.2: Procedural MATLAB implementation of the Madgwick filter for Simulink.

Bibliography

[1] Hsien-Ting Chang and Jen-Yuan Chang. «Sensor Glove Based on Novel Inertial Sensor Fusion Control Algorithm for 3-D Real-Time Hand Gestures Measurements». In: *IEEE Transactions on Industrial Electronics* 67.1 (Jan. 2020), pp. 658–666. ISSN: 1557-9948. DOI: 10.1109/tie.2019.2912765. URL: http://dx.doi.org/10.1109/TIE.2019.2912765 (cit. on p. 1).

[2] Paola Pierleoni, Alberto Belli, Lorenzo Maurizi, Lorenzo Palma, Luca Pernini, Michele Paniccia, and Simone Valenti. «A Wearable Fall Detector for Elderly People Based on AHRS and Barometric Sensor». In: *IEEE Sensors Journal* 16.17 (2016), pp. 6733–6744. ISSN: 2379-9153. DOI: 10.1109/jsen.2016.2585667. URL: http://dx.doi.org/10.1109/JSEN.2016.2585667 (cit. on p. 1).

[3] Paola Pierleoni, Alberto Belli, Lorenzo Palma, Michele Paoletti, Sara Raggiunto, and Federica Pinti. «Postural stability evaluation using wearable wireless sensor». In: *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*. IEEE, 2019, pp. 256–260. DOI: 10.1109/isce.2019.8901040. URL: http://dx.doi.org/10.1109/ISCE.2019.8901040 (cit. on p. 1).

[4] S. Gawsalyan, T. S. Janarthanan, N. Thiruthanikan, R. Shahintha, and Pujitha Silva. «Upper limb analysis using wearable sensors for cricket». In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, Feb. 2017, pp. 1–6. DOI: 10.1109/icecct.2017.8118010. URL: http://dx.doi.org/10.1109/ICECCT.2017.8118010 (cit. on p. 1).

[5] Yutong Shi, Yongbo Zhang, Zhonghan Li, Shangwu Yuan, and Shihao Zhu. «IMU/UWB Fusion Method Using a Complementary Filter and a Kalman Filter for Hybrid Upper Limb Motion Estimation». In: *Sensors* 23.15 (2023), p. 6700. ISSN: 1424-8220. DOI: 10.3390/s23156700. URL: http://dx.doi.org/10.3390/s23156700 (cit. on p. 1).

[6] Manon Kok, Jeroen D. Hol, and Thomas B. Schön. «Using Inertial Sensors for Position and Orientation Estimation». In: *Foundations and Trends® in Signal Processing* 11.1-2 (Nov. 2017), pp. 1–153. ISSN: 1932-8354. DOI: 10.1561/2000000094. URL: http://dx.doi.org/10.1561/2000000094 (cit. on pp. 1, 26).

[7] A.M. Sabatini. «Quaternion-Based Extended Kalman Filter for Determining Orientation by Inertial and Magnetic Sensing». In: *IEEE Transactions on Biomedical Engineering* 53.7 (2006), pp. 1346–1356. ISSN: 0018-9294. DOI: 10.1109/tbme.2006.875664. URL: http://dx.doi.org/10.1109/TBME.2006.875664 (cit. on p. 1).

[8] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira. «Geometric Approach to Strapdown Magnetometer Calibration in Sensor Frame». In: *IEEE Transactions on Aerospace and Electronic Systems* 47.2 (Apr. 2011), pp. 1293–1306. ISSN: 0018-9251. DOI: 10.1109/taes.2011.5751259. URL: http://dx.doi.org/10.1109/TAES.2011.5751259 (cit. on p. 1).

[9] Valérie Renaudin, Muhammad Haris Afzal, and Gérard Lachapelle. «Complete Triaxis Magnetometer Calibration in the Magnetic Domain». In: *Journal of Sensors* 2010 (2010), pp. 1–10. ISSN: 1687-7268. DOI: 10.1155/2010/967245. URL: http://dx.doi.org/10.1155/2010/967245 (cit. on pp. 1, 27, 28).

[10] Parag Narkhede, Shashi Poddar, Rahee Walambe, George Ghinea, and Ketan Kotecha. «Cascaded Complementary Filter Architecture for Sensor Fusion in Attitude Estimation». In: *Sensors* 21.6 (Mar. 2021), p. 1937. ISSN: 1424-8220. DOI: 10.3390/s21061937. URL: http://dx.doi.org/10.3390/s21061937 (cit. on pp. 1, 13, 14).

[11] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. «Nonlinear Complementary Filters on the Special Orthogonal Group». In: *IEEE Transactions on Automatic Control* 53.5 (2008), pp. 1203–1218. ISSN: 0018-9286. DOI: 10.1109/tac.2008.923738. URL: http://dx.doi.org/10.1109/TAC.2008.923738 (cit. on pp. 2, 16, 19).

[12] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. «Estimation of IMU and MARG orientation using a gradient descent algorithm». In: *2011 IEEE International Conference on Rehabilitation Robotics*. IEEE, 2011, pp. 1–7. DOI: 10.1109/icorr.2011.5975346. URL: http://dx.doi.org/10.1109/ICORR.2011.5975346 (cit. on pp. 2, 17, 19).

[13] Dhruv Parikh, Sajil Vohra, and Maryam Kaveshgar. «Comparison of Attitude Estimation Algorithms With IMU Under External Acceleration». In: *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*. IEEE,

Dec. 2021, pp. 123–126. DOI: 10.1109/ises52644.2021.00037. URL: http://dx.doi.org/10.1109/iSES52644.2021.00037 (cit. on p. 2).

[14] A. Cavallo, A. Cirillo, P. Cirillo, G. De Maria, P. Falco, C. Natale, and S. Pirozzi. «Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation». In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 7585–7591. ISSN: 1474-6670. DOI: 10.3182/20140824-6-za-1003.01173. URL: http://dx.doi.org/10.3182/20140824-6-ZA-1003.01173 (cit. on p. 2).

[15] Hunor Simon, Áron Fehér, and Lőrinc Márton. «Comparative Analysis of Attitude Estimation Algorithms on Embedded Systems». In: *2025 IEEE 19th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, May 2025, pp. 31–36. DOI: 10.1109/saci66288.2025.11030114. URL: http://dx.doi.org/10.1109/SACI66288.2025.11030114 (cit. on p. 2).

[16] Yu Sheng, Gang Tao, and Peter Beling. «Dynamic Mutation and Adaptive Tracking Control of Omni-Directional Multirotor Systems». In: *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics, Jan. 2019. DOI: 10.2514/6.2019-1562. URL: http://dx.doi.org/10.2514/6.2019-1562 (cit. on p. 6).

[17] *ICM-20948: 9-Axis MotionTracking Device Datasheet*. [Online; accessed 24-May-2026]. TDK InvenSense. URL: https://d17t6iyxenbwp1.cloudfront.net/s3fs-public/2026-03/ds-000189-icm-20948-datasheet.pdf (cit. on pp. 13, 25).

[18] Walter Higgins. «A Comparison of Complementary and Kalman Filtering». In: *IEEE Transactions on Aerospace and Electronic Systems* AES-11.3 (May 1975), pp. 321–325. ISSN: 0018-9251. DOI: 10.1109/taes.1975.308081. URL: http://dx.doi.org/10.1109/TAES.1975.308081 (cit. on p. 14).

[19] Joan Solà. «Quaternion kinematics for the error-state Kalman filter». In: *arXiv preprint arXiv:1711.02508* (Nov. 2017) (cit. on pp. 20–22).

[20] E.J. Lefferts, F.L. Markley, and M.D. Shuster. «Kalman Filtering for Spacecraft Attitude Estimation». In: *Journal of Guidance, Control, and Dynamics* 5.5 (1982), pp. 417–429. ISSN: 1533-3884. DOI: 10.2514/3.56190. URL: http://dx.doi.org/10.2514/3.56190 (cit. on p. 20).

[21] Raspberry Pi Ltd. *Raspberry Pi Pico W Datasheet*. Release 1.0. Available at: https://pip-assets.raspberrypi.com/categories/686-raspberry-pi-pico-w/documents/RP-008312-DS-1-pico-w-datasheet.pdf. 2022 (cit. on p. 24).

[22] Asahi Kasei Microdevices. *AK09916: 3-axis Electronic Compass*. Document No. 015007392-E-02. 2015 (cit. on p. 25).

[23] michal34512. *Magnetometer-calibration*. https://github.com/michal34512/Magnetometer-calibration. Accessed: June 16, 2026 (cit. on p. 28).

[24] lurk101. *pico-littlefs: A LittleFS implementation for the Raspberry Pi Pico*. https://github.com/lurk101/pico-littlefs. Accessed: June 16, 2026 (cit. on p. 30).

[25] Sebastian O. H. Madgwick. *Open-source IMU and AHRS algorithms*. Accessed: June 16, 2026. x-io Technologies. 2010. URL: https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/ (cit. on p. 32).

[26] Roberto Valenti, Ivan Dryanovski, and Jizhong Xiao. «Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs». In: *Sensors* 15.8 (Aug. 2015), pp. 19302–19330. ISSN: 1424-8220. DOI: 10.3390/s150819302. URL: http://dx.doi.org/10.3390/s150819302 (cit. on p. 34).

[27] *Sensor Fusion and Tracking Toolbox Documentation: ahrsfilter System object*. Available online at https://www.mathworks.com/help/fusion/ref/ahrsfilter-system-object.html. The MathWorks, Inc. Natick, Massachusetts, USA (cit. on p. 49).